# Pampa: An Improved Branch and Bound Algorithm for Planted $(l, d)$ Motif Search

Jaime Davila, Sudha Balla and Sanguthevar Rajasekaran [⋆]

CSE Department at University of Connecticut, Storrs, CT 06269 USA
`{jdavila, ballasudha, rajasek}@engr.uconn.edu.co`

**Abstract.** We consider the planted $(l, d)$ motif search problem as defined in [1] and [12], a problem that arises from the need to find transcription factor-binding sites in genomic information. We build an exact branch and bound algorithm, which has small space requirements. An implementation of this algorithm is able to tackle challenging instances as big as $(19, 7)$, cutting by half the time required by the best existing algorithm.

**Keywords:** Pattern and motif discovery, String 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 [15] for a literature survey–.

We can define it in a formal way, following [1] or [12].

**Definition 1.** *Given a set of strings $\mathcal{S} := \{s_i\}_{i=1}^{n}$ over an alphabet $\Sigma$, 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 for all $i = 1 \ldots n$, $s_i$ 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 $\mathcal{S}$, and each $x_i$ will be an* instance of the motif *for $\mathcal{S}$.*

This problem is known to be NP-complete [8] and a PTAS exists for variants of the problem known as the Common Approximate Substring and Common Approximate String [9]. 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 practical instances of this problem. Examples include Random Projection [1], MITRA [6], Winnower [12], Pattern Branching [13], Hybrid Sample and Pattern Driven Approaches [17], PMS1 [14], PMSP [4], PMSprune [5] SPELLER [16], SMILE [10], RISO [2], RISOTTO [11] CENSUS [7] and Voting [3]. Out of these algorithms,

[⋆] This research was supported in part by the NSF Grant ITR-0326155.

MITRA, CENSUS, PMS1, PMSP, PMSprune, SMILE, RISO, RISOTO and Voting are among the exact category.

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 *challenging instance* [1] as one where if the strings are selected at random, the expected number of $(l, d)$ motifs (that occur by random chance) is bigger than 1. According to this definition, $(11, 3)$, $(13, 4)$, $(15, 5)$, $(17, 6)$ and $(19, 7)$ are challenging instances.

MITRA solves this problem and the more general problem 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 [6].

The family of algorithms SPELLER, SMILE, RISO, RISOTTO solve the more general problem of finding *structured motifs* and are based on the use of suffix trees, RISOTTO being the latest and the most efficient implementation of them [11]. In the case of the planted $(l, d)$ motif search problem their theoretical space complexity is $O(n^2m)$ and their time complexity is $O(n^2m\mathcal{N}(l, d))$, where $\mathcal{N}(l, d) := \sum_{i=1}^{d} \binom{l}{i}(|\Sigma| - 1)^d$. RISOTTO is able to solve challenging instances as big 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 time complexity is $O(nml\mathcal{N}(l, d))$, improving the theoretical bounds of the previous family of algorithms. Its implementation is able to tackle instances such as $(15, 4)$. In [7], an algorithm with claimed space 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.

PMS1 solves the problem by making use of radix sorting and 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. It is able to solve instances as big as $(15, 4)$ but at the cost of using close to a 1GB of memory.

Voting is based on the idea of hashing and its space complexity is $O(\max\{|\Sigma|^l, \mathcal{N}(l, d)\})$ with 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 $d$.

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 $O(nm)$ 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 PMS1, 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.

PMSprune is a branch and bound method which is proven to have a time complexity of $O(nm^2\mathcal{N}(l, d))$ with a space complexity of $O(nm)$, 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 reported solved before in the literature.

In this paper we proposed a new algorithm called Pampa. Pampa is also a branch and bound method, which extends and improves significantly the ideas used in PMSprune by generating the search space in a more efficient way. The theoretical space and time complexity is the same as in PMSprune, but outperforms PMSprune on a practical perspective when tested on challenging instances by cutting the running time in half.

The paper is structured as follows. In section 2 we describe briefly the algorithm PMSprune and its complexity bounds. In section 3 we describe Pampa incrementally and prove its complexity bounds. Finally, in section 4 we describe some of the experimental results we have obtained, and compare the results with some of the already existing exact algorithms.

## 2 PMSprune

Before proceeding we will introduce some concepts that will allow us to define different concepts of "distance" between a string and a set of strings. This will allow us to state the $(l, d)$ motif problem in a more "geometrical" way and to describe the PMSprune in those terms. For more details on PMSprune the reader should refer to [5].

**Definition 2.** *Given a string $x := x[1] \dots x[l]$:*

1. *We say that another string $y := y[1] \dots y[l]$ is at distance $h$ if they differ in exactly $h$ places. This is called the Hamming distance between $x$ and $y$ and we will write it as $d_H(x, y) = h$.*
2. *We define $B_d(x) := \{y : d_H(x, y) \leq d\}$. This is also referred as the $d$ vicinity or neighborhood of $x$. Notice that $|B_d(x)| = \mathcal{N}(l, d)$.*

**Definition 3.** *Given strings $s := s[1] \dots s[m]$ and $x := x[1] \dots x[l]$ with $l < m$:*

1. *We say $x \lhd_l s$ if $x$ is a subsequence of $s$, this is $s = \alpha x \beta$ (for some strings $\alpha$ and $\beta$). Equivalently we say that $x$ is an $l$-mer of $s$.*
2. *$\bar{d}_H(x, s) = \min_{r \lhd_l s} d_H(x, r)$, where $d_H$ stands for the Hamming distance.*

**Definition 4.** *Given a string $x = x[1] \dots x[l]$ and a set of strings $\mathcal{S} := \{s_1 \dots s_n\}$ with $|s_i| = m$ for $i = 1 \dots n$ and $l < m$, we denote by*

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

In the light of the previous definitions we can write the definition 1 in a more concise way. The following is an equivalent ways of saying that $x$ is an $(l, d)$ motif for $\mathcal{S} := \{s_1, \dots, s_n\}$.

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

The general strategy of PMSprune follows equation 1. That is, it evaluates the function $\bar{d}_H(\cdot, \mathcal{S})$ for the elements of $B_d(y)$ for every $l$-mer $y$ in $s_1$. If we find an $x$ such that $\bar{d}_H(x, \mathcal{S}) \leq d$ we output $x$ as the motif.

However it extends this idea with at least the following two non-trivial observations:

1. For every $l$-mer $y$ it generates $B_d(y)$ in a branch and bound manner using a tree $\mathcal{T}(y)$ whose height is at most $d$. If $x \in B_d(y)$, $x$ appears in $\mathcal{T}(y)$ at height $h := d_H(x, y)$. Furthermore if $x'$ is a child of $x$ in $\mathcal{T}(y)$ we will have that $d_H(x, x') = 1$. In Figure 1 we show an example of such a tree. Using this approach we could calculate $\bar{d}_H(\cdot, \mathcal{S})$ in an incremental way using $O(mn)$ time for every move we make on $\mathcal{T}(y)$.
2. For a node in the tree $x \in \mathcal{T}(y)$, $y \lhd_l s_1$ it uses the value of $\bar{d}_H(x, \mathcal{S})$ and $d_H(x, y)$ to prune the descendants of $x$. To do that one has to notice if $x'$ is a child of $x$ in $\mathcal{T}(y)$ then $\bar{d}(x', \mathcal{S}) = \bar{d}(x, \mathcal{S}) + \delta$ where $\delta \in \{-1, 0, 1\}$. This implies that if $\bar{d}_H(x, \mathcal{S}) = d + \Delta$ and $\Delta > d - d_H(x, y)$ we can safely prune all the descendants of $x$, since for any such descendant $x''$ we will have that $\bar{d}_H(x'', \mathcal{S}) \geq d + (\Delta - d + d_H(x, y)) > d$.

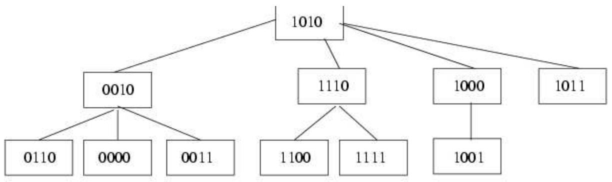

**Fig. 1.** $\mathcal{T}(1010)$ with $d = 2$ and $\Sigma = \{0, 1\}$

The previous observations can be summarized in the following algorithm.

**Algorithm** PMSprune

```
1. For each $y \lhd_l s_1$:
   (a) Traverse $\mathcal{T}(y)$ in a depth first search way evaluating $\bar{d}_H(\cdot, \mathcal{S})$.
        i. If $\bar{d}_H(x, \mathcal{S}) \leq d$ output $x$.
       ii. If $\bar{d}_H(x, \mathcal{S}) - d > d - d_H(x, y)$ prune all the descendants from $x$.
```

It is clear that we have the following result.

**Theorem 1.** *Algorithm PMSprune can be implemented in $O(nm^2\mathcal{N}(l, d))$ time and in $O(nm)$ space.*

## 3 Pampa

Pampa is based on the observation that we can describe $B_d(x)$ in a more succint way if we allow ourselves to use extended $l$-mers with "wildcards", which could represent any symbol from a fixed alphabet $\Sigma$. In this sense, an $l$-mer with $k$ wildcards will represent $|\Sigma|^k$ different $l$-mers. For example, if $\Sigma = \{A, C, G, T\}$ we have that $**A$, represents the 16 3-mers that end with $A$ and we have that $B_2(AAA) = \{*AA, A*A, AA*, **A, *A*, A**\}$. Hence the basic idea would be to extend the notion of "distance" $\bar{d}_H(\cdot, \mathcal{S})$ for the case of extended $l$-mers and evaluate this function for the neighborhoods generated in this way. More details on this idea will follow.

### 3.1 Branch and Bound Strategy: Generating the Extendend $l$-mer Space

In order to make the previous ideas more precise, we will be introducing the following definitions.

**Definition 5.** *Given an alphabet $\Sigma$ and a string $y' := y'[1] \dots y'[l]$:*

1. *We say that $y'$ is an extended $l$-mer over $\Sigma$ if for $i = 1 \dots l$, $y'[i] \in \Sigma \cup \{*\}$.*
2. *We will say that $j$ is a* degenerated position *of $y'$ if $y'[j] = *$. We will call* $\deg(y')$ *the number of degenerated positions of $y'$.*
3. *Given an $l$-mer $x$ over $\Sigma$ we say that $y \in \mathrm{gen}(y')$ if $y[i] = y'[i]$ for all the non-degenerated positions of $y'$.*

**Definition 6.** *Given an extended $l$-mer $y'$ over $\Sigma$ and a set of strings $\mathcal{S} := \{s_1 \dots s_n\}$ with $|s_i| = m$ for $i = 1 \dots n$ and $l < m$, we denote by*

$$\bar{d}_H(y', \mathcal{S}) := \min_{y \in \mathrm{gen}(y')} \bar{d}_H(y, \mathcal{S}) = \min_{y \in \mathrm{gen}(y')} \max_{i=1}^{n} \min_{r \triangleleft_l s_i} d_H(y, r).$$

Notice that if $x$ in an $(l, d)$ motif for $\mathcal{S}$ then the following statement holds.

$$\exists y \triangleleft_l s_1 : \{x, y\} \in \mathrm{gen}(y') \wedge \deg(y') \leq d \wedge \bar{d}_H(y', \mathcal{S}) \leq d \tag{2}$$

Notice that (2) is strictly an implication. We could have $y'$ with $y' \in \mathrm{gen}(y)$, $y \triangleleft_l s_1$, $\deg(y') \leq d$, $\bar{d}_H(y', \mathcal{S}) \leq d$ such that there is no $(l, d)$-motif $x$ with $x \in \mathrm{gen}(y')$.

Pampa exploits (2) by generating all the extended $l$-mers of $s_1$ and evaluating $\bar{d}_H(\cdot, \mathcal{S})$ over them. If $\bar{d}_H(\cdot, \mathcal{S} \leq d$ we continue on a second phase which will be described in section 3.2 and whose objective is to recognize if there is a real motif behind an extended $l$-mer.

**Definition 7.** *Given $y = y[1] \dots y[l]$ an string over $\Sigma$ 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 an extended $l$-mer $y'$, with $deg(y') = h$ and $y \in gen(y')$.*

2. *The root level is $y$.*
3. *Given a node $y'$ with degenerated positions $1 \leq i_1 < \cdots < i_h \leq l$, its children are the extended $l$-mers $y''$ with $gen(y') \subset gen(y'')$ with degenerated positions $1 \leq i_1 < \cdots < i_h < i_{h+1} \leq l$.*

In order to evaluate $\bar{d}_H(\cdot, \mathcal{S})$ in an efficient way as we progress in the tree, we have to make the following observations. We assume $y', y'' \in \mathcal{T}'(y)$ with $y''$ being a children node of $y'$.

1. $\bar{d}_H(y'', \mathcal{S}) = \bar{d}_H(y', \mathcal{S}) + \delta$ with $\delta \in \{0, -1\}$.
2. One can incrementally calculate $\bar{d}_h(y'', \mathcal{S})$ in $O(mn)$ time by noticing that it will depend only on the value of $y'[i_{h+1}]$ and the character at position $i_{h+1}$ of all the $l$-mers in $s_2, \ldots, s_n$.

We also need to notice that we can implement a pruning strategy similar to the one used in PMSprune, namely that if $y'$ appears at height $h$ in $\mathcal{T}'(y)$ with $\bar{d}_H(y', \mathcal{S}) = d + \Delta$ and $\Delta > d - h$ we can prune all of the descendants of $y'$.

Taking these observations into consideration we arrive at the first phase of Pampa.

**Algorithm** Pampa - First Phase

```
1. For each $y \lhd_l s_1$:
   (a) Traverse $\mathcal{T}'(y)$ in a depth first way evaluating $\bar{d}_H(\cdot, \mathcal{S})$ in a incremental way.
       i. If $\bar{d}_H(y', \mathcal{S}) \leq d$ save $y'$ for the Second Phase of Pampa .
       ii. If $\bar{d}_H(y', \mathcal{S}) - d > d - h$ prune all the descendants from $y$.
```

The following Theorem is evident.

**Theorem 2.** *The first phase of Pampa can be implemented in $O\left(nm^2 \sum_{i=1}^{d} \binom{l}{i}\right)$ time.*

### 3.2 Detecting False Positives

The idea which will be used in the second phase is that given an extended $l$-mer $y'$ with $deg(y') \leq d$ we will explore the space of gen$(y)$ evaluating the function $\bar{d}_H(\cdot, \mathcal{S})$ in an incremental way, to see if there are any motifs $x$, in which case $\bar{d}_h(x, \mathcal{S}) \leq d$.

**Definition 8.** *Given $y'$ an extended $l$-mer with $deg(y') = e$, we define a tree $\mathcal{G}(y')$ by the following rules:*

1. *A node at level $0 \leq h \leq e$ is an extended $l$-mer $w$, with $deg(w) = d - h$ .*
2. *The root level is $y'$.*

3. *Given a node $w$ with degenerated positions $1 \leq i_1 < \cdots < i_h \leq l$, its children are the extended $l$-mers $w'$ with gen($w'$) $\subset$ gen($w$) with degenerated positions $1 \leq i_1 < \cdots < i_{h-1} < l$. It is clear that each node has $|\Sigma|$ children.*

Following the same spirit of the algorithms described so far, we have to make the following observations. We assume $w, w' \in \mathcal{G}(y')$, with $w'$ being a child of $w$.

1. $\bar{d}_H(w', \mathcal{S}) = \bar{d}_H(w, \mathcal{S}) + \delta$ with $\delta \in \{0, 1\}$.
2. One can calculate incrementally $\bar{d}_H(w', \mathcal{S})$ in $O(mn)$ time noticing that it will depend only on $w'[i_h]$ and the value of the characters at position $i_h$ in all the $l$-mers of $s_2, \ldots, s_n$.
3. If $\bar{d}_H(w, \mathcal{S}) = d + \Delta$ where $\Delta > 0$ we can prune all the descendants of $w$.

**Algorithm** Pampa - Second Phase

```
1. Given $y'$, traverse $\mathcal{G}'(y')$ in a depth first manner evaluating $\bar{d}_H(\cdot, \mathcal{S})$ in a
   incremental way.
   (a) If $\bar{d}_H(w, \mathcal{S}) \leq d$ and deg($w$) = 0 output $w$.
   (b) If $\bar{d}_H(w, \mathcal{S}) > d$ prune all the descendants from $w$.
```

**Theorem 3.** *The second phase of Pampa takes $O(nm|\Sigma|^d)$ time per each extended $l$-mer tested.*

One could also follow another idea which depends on generating the vicinities of the closest $l$-mers to $y'$, when we restrict ourselves only to the degenerated positions of $y'$. This idea somehow resembles the one used in PMS1 [14]. In order to be more precise we introduce the following notation.

**Definition 9.** *Given $y'$ an extended $l$-mer with deg($y'$) = $h$ and with non- degenerated positions $j_1, \ldots, j_{l-h}$ and $x$ an $l$-mer, we define*

$$d_H(x, y') := \min_{w \in \text{gen}(y)} d_H(x, w) = d_H(x[j_1] \ldots x[j_{l-h}], y'[j_1] \ldots y'[j_{l-h}]).$$

Assume $y'$ is an extend $l$-mer with degenerated positions $i_1, \ldots, i_h$ and $\bar{dist}_H(y', \mathcal{S}) = d - \Delta$ with $\Delta \geq 0$. We have that $x \in \text{gen}(y')$ is an $(l, d)$ motif for $\mathcal{S}$ if:

$$x[i_1] \ldots x[i_h] \in \bigcap_{i=2}^{n} \bigcup_{r \lhd_l s_i} \{B_e(r[i_1] \ldots r[i_h]) : e = d - d_H(y', r) \wedge d_h(y', r) \leq d\} \quad (3)$$

This strategy can be very useful in cases where $d$ is small, or when we have that all the $l$-mers in several of the sequences, say $s_i$ are at distance which is very close to $d$. In that case 3 can be evaluated using a small amount of time. However, notice in the worst case, if one uses a strategy similar to PMS, one would expend time $O(\frac{h}{w} nm|\Sigma|^d)$ and space $O(\frac{h}{w} m|\Sigma|^d)$.

### 3.3 Time Complexity Results

Theorem 4 estimates the run time of the algorithm Pampa.

**Theorem 4.** *Algorithm Pampa can be implemented in $O(nm^2\mathcal{N}(l,d))$ time and in $O(nm)$ space.*

*Proof.* This follows easily from theorems 2 and 3 $\square$

We are interested in estimating the time PMSprune takes on the average. To do so, we assume that the set of strings $s_i$ is constructed by picking each character from every sequence with equal probability, and we assume also that an $l$-mer $x$ is constructed in the same way. In doing so we follow the approach used in [1], which basically assumes that the probabilty of occurence 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 1.** *Let $0 \le d \le l$ and construct two random $l$-mers $x$ and $y$ in the model previously described. The probability that $x$ and $y$ are at distance $\le d$ is*

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

**Lemma 2.** *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*

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

Using equation (4), it is possible to estimate for fixed values of $n$, $m$, $l$, a value $d'$ such that $\Pr\{\bar{d}_H(x, \mathcal{S}) \ge d'\}$ is close to 1. This implies that if one is trying to solve a random instance of the $(l, d)$ motif where one sets $d \le \frac{d'}{2}$, our algorithm does not explore the search tree because of the pruning strategy since $\bar{dist}_H(x, \mathcal{S}) = d' > d + d$ in most cases.

**Theorem 5.** *The expected time complexity of Pampa is $O(nm^2)$ for $2d < d'$.*

For example, when using $n = 600$, $m = 20$ and $l = 15$, we have that $d' = 7$. Thus on the average, the solution of $(15, 3)$ will take time proportional $O(nm^2)$.

## 4 Experimental Results

As described before, we follow the experimental setting described in [12] and [1]. In particular, 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.

Pampa, PMSprune, PMSP as well RISOTTO are coded in C, and we have run these programs on the same Linux machine with a Pentium4 2.40 GHz processor and with a core memory size of 1GB. The results of Voting are the ones reported in [3], on a machine with the same processor and a core memory size of 512MB.

We compare the results of Pampa against PMSP, RISOTTO and Voting on some challenging instances in table 1 . When we write '–' it means that the algorithm uses too much memory in the instance, took too long or its time is not reported. Pampa consistently outperforms the best previous algorithm (PMSprune) by a factor close to 2 in instances such as $(19, 7)$, $(17, 6)$ and $(15, 5)$. It also outperforms the others by a factor that gets bigger with increasing d.

| Algorithm | (11,3) | (13,4) | (15,5) | (17,6) | (19,7) |
|---|---|---|---|---|---|
| Pampa | 4s | 35s | 5m | 40m | 4h:45m |
| PMSprune | 5s | 53s | 9m | 69m | 9h:10m |
| PMSP | 7s | 152s | 35m | 12h | – |
| Voting | 9s | 108s | 22m | – | – |
| RISOTTO | 54s | 600s | 100m | – | – |

**Table 1.** Time Comparison In Challenging Problem Instances

In table 2 we can see the behavior of algorithm Pampa for several values of $(l, d)$. We choose values of $d$ such that as in Theorem 5 $2d \leq d'$, where $d'$ is an approximation of the value of $\bar{dist}_H(\cdot, \mathcal{S})$ when $\mathcal{S}$ is generated at random. Notice that we get in most case a running time of close to 0.2 seconds.

| $l$ | $d$ | Time | $d$ | Time | $d$ | Time |
|---|---|---|---|---|---|---|
| 13 | 2 | 0.2s | 1 | 0.2s | | |
| 14 | 2 | 0.2s | 1 | 0.2s | | |
| 15 | 3 | 0.2s | 2 | 0.2s | 1 | 0.2s |
| 16 | 3 | 0.2s | 2 | 0.2s | 1 | 0.2s |
| 17 | 4 | 0.9s | 3 | 0.5s | 2 | 0.5s |
| 18 | 4 | 0.2s | 3 | 0.2s | 2 | 0.2s |
| 19 | 5 | 2.0s | 4 | 0.2s | 3 | 0.2s |

**Table 2.** Time Comparison of Pampa for different $(l, d)$ instances

## 5 Conclusions

In this paper we have given an algorithm for the planted motif problem that outperforms the previously known algorithms.

## References

1. J. Buhler and M. Tompa, Finding motifs using random projections, *Proceedings of the Fifth Annual International Conference on Computational Molecular Biology (RECOMB)*, April 2001.
2. Alexandra M. Carvalho, Ana T. Freitas, Arlindo L. Oliveira and Marie-France Sagot, A highly scalable algorithm for the extraction of cis-regulatory regions, *Proceedings of the Third Asia Pacific Bioinformatics Conference (APBC2005)*, volume 1 of Advances in Bioinformatics and Computational Biology, pp. 273-282. Imperial College Press, 2005.
3. F Y.L. Chin and H C.M. Leung, Voting Algorithms for Discovering Long Motifs, *Proceedings of the Third Asia-Pacific Bioinformatics Conference (APBC2005),* , pp. 261-271, January 2005.
4. J. Davila, S. Balla and S. Rajasekaran, Space And Time Efficient Algorithms for Planted Motif Search. *Proceedings of the Second International Workshop on Bioinformatics Research and Applications (IWBRA 2006),* May 2006. pp. 822-829.
5. J. Davila, and S. Rajasekaran, A Practical Branch and Bound Algorithm for Planted $(l, d)$ Motif Search. UConn BECAT/CSE Technical Report, BECAT/CSE-TR-06-4, February 2006.
6. E. Eskin and P. Pevzner, Finding composite regulatory patterns in DNA sequences, *Bioinformatics* S1, 2002, pp. 354-363.
7. P A. Evans, A. Smith. Toward Optimal Motif Enumeration.*Proceedings of Algorithms and Data Structures, 8th International Workshop, WADS*, 2003 , pp. 47-58.
8. P A. Evans, A. Smith and H. Todd Wareham. On the complexity of finding common approximate substrings. *Theoretical Computer Science* 306, 2003, pp. 407-430.
9. M. Li, B. Ma and L. Wang. On the Closest String and Substring Problems. *Journal of the ACM*, Vol. 49, No. 2, March 2002, pp. 157-171.
10. L. Marsan and M.-F. Sagot. Extracting structured motifs using a suffix-tree. - Algorithms and application to promoter consensus identification. *Proceedings RECOMB'2000* , Tokyo. ACM Press.
11. Nadia Pisanti, Alexandra M. Carvalho, Laurent Marsan and Marie-France Sagot, RISOTTO: Fast extraction of motifs with mismatches, *Proceedings of the 7th Latin American Theoretical Informatics Symposium (LATIN2006)*, volume 3887 of Lecture Notes in Computer Science, pp. 757-768. Springer-Verlag, 2006.
12. 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.
13. A. Price , S. Ramabhadran, P. Pevzner. 2003. Finding subtle motifs by branching from sample strings. *Bioinformatics supplementary edition, Proceedings of the Second European Conference on Computational Biology (ECCB-2003).*
14. 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.
15. S. Rajasekaran, Algorithms for Motif Search, in *Handbook of Computational Molecular Biology*, edited by S. Aluru, Chapman&Hall/CRC, 2006, 37-1–37-21.
16. M.F. Sagot. Spelling approximate repeated or common motifs using a suffix tree. *Proceedings of the Third Latin American Theoretical Informatics Symposium (LATIN1998)* , pp. 111-127, 1998, Springer-Verlag.
17. S. Sze, S. Lu, J. Chen, Integrating Sample-Driven and Pattern-Driven Approaches in Motif Finding. *Algorithms in Bioinformatics: 4th International Workshop (WABI 2004)* , 2004.