EXACT ALGORITHMS FOR PLANTED MOTIF CHALLENGE PROBLEMS [*]

SANGUTHEVAR RAJASEKARAN, SUDHA BALLA, AND CHUN-HSI HUANG

Dept. of Computer Science and Engineering
Univ. of Connecticut, Storrs, CT 06269-2155, USA
E-mail: {rajasek,ballasudha,huang}@engr.uconn.edu

The problem of identifying meaningful patterns (i.e., motifs) from biological data has been studied extensively due to its paramount importance. Three versions of this problem have been identified in the literature. One of these three problems is the *planted* (l, d)*-motif problem*. Several instances of this problem have been posed as a challenge. Numerous algorithms have been proposed in the literature that address this challenge. Many of these algorithms fall under the category of approximation algorithms. In this paper we present algorithms for the *planted* (l, d)*-motif problem* that always find the correct answer(s). Our algorithms are very simple and are based on ideas that are fundamentally different from the ones employed in the literature. We believe that the techniques we introduce in this paper will find independent applications.

1. Introduction

Motif search is an important problem in biology. This problem in general requires finding short patterns of interest from voluminous data. Three variants of this motif search problem have been identified in the literature. In this paper we focus one of these problems (defined below).

Problem (Planted Motif Search (PMS)) Input are t sequences of length n each. Input also are two integers l and d. The problem is to find a motif (i.e., a sequence) M of length l. It is given that each input sequence contains a variant of M. The variants of interest are sequences that are at a hamming distance of d from M.

Numerous papers have been written in the past on the topic of motif search (PMS). Examples include Bailey and Elkan,[1] Lawrence et al.,[10] Rocke and Tompa.[14] These algorithms employ local search techniques such as Gibbs sampling, expectation optimization, etc. These algorithms may not output the planted motif always. We refer to such algorithms as *approximation algorithms*. Algorithms that always output the correct answer are referred to as *exact algorithms*.

More algorithms have been proposed for PMS by the following authors: Pevzner and Sze,[11] Buhler and Tompa.[5] The algorithm of Pevzner and Sze[11] is based on finding cliques in a graph and the algorithm of Buhler and Tompa[5] employs random projections. These

[*]This work has been supported in part by the NSF Grants CCR-9912395 and ITR-0326155.

algorithms have been experimentally demonstrated to perform well. These are approximation algorithms as well.

Algorithms for PMS can be categorized into two depending on the basic approach employed, namely, profile-based algorithms and pattern-based algorithms (see e.g., Price et al.[12]) Profilebased algorithms predict the starting positions of the occurrences of the motif in each sequence. On the other hand, pattern-based algorithms predict the motif (as a sequence of residues) itself.

Several pattern based algorithms are known. Examples include PROJECTION,[5] MULTIPROFILER,[9] MITRA,[6] and PatternBranching.[12] PatternBranching (due to Price, Ramabhadran and Pevzner[12]) starts with random seed strings and performs local searches starting from these seeds.

Examples of profile-based algorithms include CONSENSUS,[7] GibbsDNA,[10] MEME,[1] and ProfileBranching.[12] The performance of profile-based algorithms are specified with a measure called "performance coefficient". The performance coefficient gives an indication of how many positions (for the motif occurrences) have been predicted correctly. For the (15, 4) challenge problem, these algorithms have the following performance coefficients (respectively): 0.2, 0.32, 0.14, and 0.57. The run times of these algorithms for this instance are (respectively, in seconds): 40, 40, 5, and 80.

A profile-based algorithm could either be approximate or exact. Likewise a pattern-based algorithm may either be exact or approximate. Algorithms that are exact are also known as *exhaustive enumeration algorithms* in the literature.

Many exact algorithms are known. (See e.g., Blanchette et al.,[3] Brazma et al.,[4] Sinha and Tompa,[15] Staden,[16] Tompa,[17] and van Helden et al.[18]) However, as pointed out in Buhler and Tompa,[5] these algorithms ”become impractical for the sizes involved in the challenge problem". One of the exceptions is the MITRA algorithm.[6] This algorithm is pattern-based and is exact. It solves for example the $(15, 4)$ instance in 5 minutes using 100 MB of memory.[6] This algorithm is based on the WINNOWER algorithm[11] and uses pairwise similarity information. A new pruning technique enables MITRA to be more efficient than WINNOWER. MITRA uses a mismatch tree data structure and splits the space of all possible patterns into disjoint subspaces that start with a given prefix.

2. New Results

In this paper we present pattern-based exact algorithms for the planted (l, d)-motif problem. The run time of our basic algorithm is $O\left(tn\binom{l}{d}|\Sigma|^d\frac{l}{w}\right)$. Most of the algorithms in the literature are based on exploring the neighborhood of possible patterns. Our algorithm also uses this basic approach. In addition, the existing algorithms use a subset of the following ideas: sampling, local search, pairwise similarity scoring, statistically or randomly selecting potential candidates, expectation maximization, and random projections. On the other hand, in this paper we present many ideas that are fundamentally different from the ones found in the literature. We believe that these techniques will find independent applications. The ideas we propose are very simple. We have implemented our algorithms and measured their performances. To the best of our knowledge, MITRA is the best performing exact algorithm in the literature before this paper. Remarkably, our algorithms are in general faster than MITRA. For example, for the $(15, 4)$ instance, one of our algorithms takes 217 seconds and in comparison, MITRA takes 5 minutes. It has been pointed out Price et al.[12] that the $(14, 4)$ instance is more difficult than the $(15, 4)$ instance. Our algorithm takes nearly the same time for the $(14, 4)$ instance as well. As another example, our algorithm takes less than a second for the $(11, 2)$ instance and MITRA takes a minute.[6]

When compared to MITRA, our algorithm is very simple and is based on fundamentally different concepts. We only use arrays. MITRA uses the mismatch tree data structure. No complexity analysis has been done for MITRA.

It is noteworthy here that the profile-based algorithms such as CONSENSUS, GibbsDNA, MEME, and ProfileBranching take much less time for the $(15, 4)$ instance.[12] However these algorithms fall under the approximate category and may not always output the correct answer. Some of the pattern-based algorithms (such as PROJECTION, MULTIPROFILER, and PatternBranching) also take much less time.[12] However these are approximate as well (though the success rates are close to 100%).

Some of the instances of PMS are difficult to solve as has been reported by Pevzner and Sze[11] and Rocke and Tompa.[14] For example, the following instances are difficult for the algorithms of Pevzner and Sze[11] and Rocke and Tompa[14]: $(9, 2), (11, 3), (13, 4), (15, 5), (17, 6)$. One of the reasons for this difficulty is that the above instances are expected to have spurious solutions (i.e., motifs other than the planted one). Since our algorithms are exact, we report all such motifs. For example, our algorithms solve the $(9, 2)$-instance easily in 1.43 seconds.

3. Our Algorithms

In this section we present two straight-forward algorithms. The first algorithm has the following steps: 1) Let the input sequences be $S_1, S_2, \ldots, S_t$. The length of each sequence is n. Form all possible l-mers from out of these sequences. The total number of l-mers is $\leq tn$. Call this collection of l-mers C. Let the collection of l-mers in S_1 be C'; 2) Let u be an l-mer in C'. For all $u \in C'$ generate all the patterns v such that u and v are at a hamming distance of d. The number of such patterns for a given u is $\binom{l}{d}(|\Sigma| - 1)^d$. Thus the total number of patterns generated is $O\left(n\binom{l}{d}|\Sigma|^d\right)$. Call this collection of l-mers C''. Note that C'' contains M, the desired output pattern (assuming that M does not occur in any of the input sequences); 3) For every pair of l-mers (u, v) with $u \in C$ and $v \in C''$ compute the hamming distance between u and v. Output that l-mer of C'' that has a neighbor (i.e., an l-mer at a hamming distance of d) in each one of the n input sequences. The run time of this algorithm is $O\left(tn^2 l\binom{l}{d}|\Sigma|^d\right)$. If M occurs in one of the input sequences, then this algorithm will run in time $O(t^2n^2l)$.

The second algorithm considers every possible l-mer one at a time and checks if this l-mer is the correct motif M. There are $|\Sigma|^l$ possible l-mers. Let M' be one such l-mer. We can check if $M' = M$ as follows. Compute the hamming distance between u and M' for every $u \in C$. (Note that C is the collection of all possible l-mers in the input sequences.)

As a result we can check if M' occurs in each input sequence (at a hamming distance of d). Thus we can identify all the motifs of interest in a total of $O\left(tnl|\Sigma|^l\right)$. We get the following Lemma.

Lemma 3.1. *We can solve the planted (l, d)-motif problem in $O(tnl|\Sigma|^l)$ time.*

Now we present a different algorithm based on sorting. This sorting based algorithm (Planted Motif Search 1 (PMS1)) takes the following form.

Algorithm PMS1

(1) Generate all possible l-mers from out of each of the t input sequences. Let C_i be the collection of l-mers from out of S_i for $1 \leq i \leq t$.
(2) For all $1 \leq i \leq t$ and for all $u \in C_i$ generate all l-mers v such that u and v are at a hamming distance of d. Let the collection of l-mers corresponding to C_i be C'_i, for $1 \leq i \leq t$. The total number of patterns in any C'_i is $O\left(n\binom{l}{d}|\Sigma|^d\right)$.
(3) Sort all the l-mers in every $C'_i, 1 \leq i \leq t$. Let L_i be the sorted list corresponding to C'_i.
(4) Merge all the L_is $(1 \leq i \leq t)$ and output the generated (in step 2) l-mer that occurs in all the L_is.

The following theorem results.

Theorem 3.1. *Problem 1 can be solved in time $O\left(tn\binom{l}{d}|\Sigma|^d\frac{l}{w}\right)$ where w is the word length of the computer. A run time of $O\left(\left[tn + n\binom{l}{d}^2|\Sigma|^{2d}\right]\frac{l}{w}\right)$ is also achievable.*

4. Improved Algorithms

In this section we present techniques for improving the performance of the algorithm PMS1.

The algorithm of Buhler and Tompa[5] is based on random projections. Let the motif M of interest be an l-mer. Let C be the collection of all the l-mers from all the t input sequences. Project these l-mers along k randomly chosen positions (for some appropriate value of k). A typical value used by Buhler and Tompa[5] is 7. In other words, for every l-mer $u \in C$, generate a k-mer u' which is a subsequence of u corresponding to the k random positions chosen. (The random positions are the same for all the l-mers). We can think of each k-mer thus generated as an integer. We group the k-mers according to their integer values. (I.e., we hash all the l-mers using the k-mer of any l-mer as its hash value).

If a hashed group has at least a threshold number of l-mers in it, then there is a good chance that M will have its k-mer equal to the k-mer of this group. The threshold used by Buhler and Tompa[5] is 3. We collect all the k-mers that pass the threshold and these are processed further to arrive at the final answer M.

We now present a different algorithm for processing the potential k-mers. Let M' be any l-mer. We can check if $M' = M$ as follows. Compute the hamming distance between M' and u for every $u \in C$. At the end we will know if M' is the correct answer or not.

Thus testing if $M' = M$ takes $O(tnl)$ time. As a corollary, we get the following Lemma (c.f. Lemma 3.1).

Lemma 4.1. *Given k residues in M (and their positions in M), we can determine M in time $O(tnl|\Sigma|^{l-k})$.*

4.1. ***Improvement 1***

Lemma 4.1 can be used to improve Theorem 3.1 as follows. Note that if M occurs in every input sequence, then every substring of M also occurs in every input sequence. In particular, there are at least $l - k + 1$ k-mers (for $d \le k \le l$) such that each of these occurs in every input sequence at a hamming distance of at most d. Let Q be the collection of k-mers that can be formed out of M. There are $l - k + 1$ k-mers in Q. Each one of these k-mers will be present in each input sequence at a hamming distance of at most d.

In addition, in every input sequence S_i, there will be at least one position i_j such that a k-mer of Q occurs starting from i_j; another k-mer of Q occurs starting from $i_j + 1$;. . .; yet another k-mer occurs starting from $i_j + l - k$. We can get an l-mer putting together these k-mers that occur starting from each such i_j.

Possibly, there could be many motifs of length k that are in the positions starting from each of $i_j, i_j + 1, \ldots, i_j + l - k$ such that all of these motifs are present in all of the input sequences (with a hamming distance of at most d). Assume that M_{i_j+r} is one motif of length k that starts from position $i_j + r$ of S_i that is also present in every input sequence (for $0 \le r \le l - k$). If the last $k - 1$ residues of M_{i_j+r} are the same as the first $k - 1$ residues of M_{i_j+r+1} (for $0 \le r \le l - k - 1$), then we can obtain an l-mer from these motifs in the obvious way. This l-mer is potentially a correct motif. Also, note that to obtain potential motifs (of length l), it suffices to process one of the input sequences (in a manner described above). Now we are ready to describe our improved algorithm.

There are two phases in the algorithm. In the first phase we identify all $(d + c)$-mers M_{d+c} (for some appropriate value c) that occur in each of the input sequences at a hamming distance of at most d. We also collect potential l-mers (as described above) in this phase. In the second phase we check, for each l-mer M' collected in the first phase, if M' is a correct answer or not. Finally we output all the correct answers.

First we observe that the algorithm PMS1 can also be used for the case when we look for a motif M that occurs in each input sequence at a hamming distance of at most d. The second observation is that if c is large enough then there wont be many spurious hits. A suggested value for c is the largest integer for which PMS1 could be run (without exceeding the computers core memory and within a reasonable amount of time).

We present more details on the two phases.

Algorithm PMS2

Phase I

Solve the planted $(d+c, d)$-motif problem on the input sequences (with a hamming distance of $\le d$, using e.g., a modified PMS1). Let R be the set of all motifs found.

Let S be one of the input sequences. (S could be an arbitrary input sequence; it could be chosen randomly as well.) Find all the occurrences of all the motifs of R in S (with a hamming distance of up to d). This can be done, e.g., as follows: form all the $(d+c)$-mers of S (keeping track of the starting position of each in S); For each $u \in S$, find all the $(d+c)$-mers v such that u and v are at a hamming distance of at most d. If R' is the collection of these $(d+c)$-mers, sort R and R' and merge them; and figure out all the occurrences of interest.

Let S be of length n. For every position i in S, let L_i be the list of all motifs of R that are in S (with a hamming distance of $\leq d$) starting from position i.

Let A be the l-mer of S that occurs starting from position i. Let M_1 be a member of L_i. If M_2 is a member of $L_{i+l-(d+c)}$ such that the last $2(d+c)-l$ characters of M_1 are the same as the first $2(d+c)-l$ characters of M_2, then we could get an l-mer B by appending the last $l-(d+c)$ residues of M_2 to M_1 (at the end). If the hamming distance between A and B is d, then B is retained as a candidate for the correct motif. We gather all such candidates and check if any of these candidates are correct motifs. Details are given below.

```
**for** $i := 1$ **to** $n - l + 1$ **do**
    **for** every $u \in L_i$ **do**
        **for** every $v \in L_{i+l-(d+c)}$ **do**
            Let the $l$-mer of $S$ starting from position $i$ be $A$. If the last
            $2(d+c)-l$ residues of $u$ are the same as the first $2(d+c)-l$
            residues of $v$, then form an $l$-mer $B$ by appending the last
            $l-(d+c)$ residues of $v$ to $u$. If the hamming distance between
            $A$ and $B$ is $d$, then add $B$ to the list $C$ of candidates.
```

Phase II

```
**for** every $v \in C$ **do**
    Check if $v$ is a correct motif in $O(tnl)$ time.
```

For any node u of G there can be at most $|\Sigma|^{l-(d+c)}$ candidate motifs. Thus the time needed to process G to get all the candidate motifs is $O\left(\sum_{i=1}^{l-(d+c)+1} |L_i||\Sigma|^{l-(d+c)} l\right)$.

We arrive at the following Theorem.

Theorem 4.1. *Problem 1 can be solved in time* $O\left(tn\sum_{i=0}^{d}\binom{d+c}{i}|\Sigma|^i \frac{d+c}{w} + ztnl + \sum_{i=1}^{l-(d+c)+1}|L_i||\Sigma|^{l-(d+c)}\, l\right)$ *where z is the number of potential l-mers collected in the first phase and w is the word length of the computer. If $d \leq \lfloor l/2 \rfloor$, then the run time is* $O\left(tn\binom{d+c}{d}|\Sigma|^d \frac{d+c}{w} + ztnl + \sum_{i=1}^{l-(d+c)+1}|L_i||\Sigma|^{l-(d+c)}\, l\right)$.

An Alternative Algorithm. We can modify the above algorithm as follows. We first find the collection R of all the $(d+c)$-mers that are present in every input sequence at a hamming distance of at most d as before. In the above version, we pick only one sequence S and find all the candidate motifs arising out of S. An alternative is to find the candidate motifs from each sequence and get the intersection of these sets. Let A_i be the set of candidates from S_i $(1 \leq i \leq t)$. Let $A = \bigcap_{i=1}^{t} A_i$. We output A.

4.2. *Further Improvements*

We have devised three techniques to improve the performance of PMS2 further. For example, one of these improvements enables one to handle large values of d. Let $d' = \lfloor d/2 \rfloor$. Let M be the motif of interest with $|M| = l = 2l'$ for some integer l'. Let M_1 refer to the first half of M and M_2 to the second half. We know that M occurs in every input sequence. Let $S = s_1, s_2, \ldots, s_n$ be an arbitrary input sequence. Let the occurrence of M (with a hamming distance of d) in S start at position i. Let $S' = s_i, s_{i+1}, \ldots, s_{i+l'-1}$ and $S'' = s_{i+l'}, \ldots, s_{i+l-1}$.

Then, clearly, either 1) the hamming distance between M_1 and S' is at most d' or 2) the hamming distance between M_2 and S'' is at most d'. Also, note that in every input sequence either M_1 occurs with a hamming distance of at most d' or M_2 occurs with a hamming distance of at most d'. As a result, in at least t' sequences (where $t' = \lceil t/2 \rceil$) either M_1 occurs with a hamming distance of at most d' or M_2 occurs with a hamming distance of at most d'. We have developed an algorithm (called PMS3) based on this observation. Details are omitted due to space constraints.

5. Experimental Details

In this section we provide details on implementing our algorithms and the results of our implementation. We have implemented PMS1 and PMS2. As in prior works, we use $t = 20$ and $n = 600$. The input sequences were generated randomly. The motif M was generated at random. Its occurrences in the sequences as well as the starting positions were generated at random. Our algorithms have also been tested on the biological data supplied by Blanchette.[2]

5.1. *Saving Memory*

The way PMS1 is described, we first form all possible l-mers from out of all the input sequences, generate all relevant neighbors of these l-mers, sort and merge all of them to identify the generated l-mer(s) found in all the sequences. We can modify the algorithm as follows so as to reduce the memory used.

Algorithm PMS1A

Generate all possible l-mers from out of the first input sequence S_1. Let C_1 be the collection of these l-mers. For all $u \in C_1$ generate all l-mers v such that u and v are at a hamming distance of d. Sort the collection of these l-mers and let L be the sorted collection.

```
for $i := 2$ to $t$ do
```

(1) Generate all possible l-mers from out of the input sequence S_i. Let C_i be the collection of these l-mers.
(2) For all $u \in C_i$ generate all l-mers v such that u and v are at a hamming distance of d. Let the collection of these l-mers be C'_i.
(3) Sort all the l-mers in C'_i. Let L_i be the sorted list.
(4) Merge L_i and L and keep the intersection in L. I.e., set $L := L \cap L_i$.

L now has the motif(s) of interest.

5.2. *Implementation of PMS1*

We represent every l-mer as a sequence of integers. If $r_1, r_2, \ldots, r_l$ is an l-mer, it is represented as $(i_1, i_2, \ldots, i_q)$ where $i_1, i_2, \ldots, i_q$ are integers, each integer corresponding to a sequence of successive residues. When $|\Sigma| = 4$, we need two bits per residue. Thus a sequence of residues can be represented as an integer in the usual way. For example, we can associate g with 00, c with 01, t with 10 and a with 11. In this case at will be represented as the integer 1110 (i.e., 14). For instance when $l = 16$, if the size of an integer (on the machine of interest) is 32 bits, then each l-mer is stored as an integer.

5.3. *Implementation of PMS2*

PMS1A is used to identify the collection R of all the $(d + c)$-mers present in all the input sequences. One of the input sequences S is chosen arbitrarily. For every position i in $S(1 \leq i \leq n)$, L_i is the list of all the $(d + c)$-mers in R that occur in S (with a hamming distance of $\leq d$) starting from i.

We keep each $L_i(1 \leq i \leq n + l - 1)$ in (lexicographically) sorted order.

One of the basic operations we have to perform on every $u \in L_i$ is to check if there is an entry $v \in L_{i+l-(d+c)}$ such that u and v form a candidate motif (of length l). The search for v in $L_{i+l-(d+c)}$ is done with a binary search on $L_{i+l-(d+c)}$.

l	d	Time (Sec.)	l	d	Time (Sec.)	l	d	Time (Sec.)
9	2	1.44						
10	2	0.84						
11	2	0.78	11	3	19.84			
12	2	0.84	12	3	15.53			
13	2	0.70	13	3	20.98	13	4	228.94
14	2	1.05	14	3	20.38	14	4	226.83
15	2	1.33	15	3	20.53	15	4	217.34
16	2	2.61	16	3	21.20	16	4	216.92
17	2	2.56	17	3	20.89	17	4	216.08
18	2	2.64	18	3	20.50	18	4	217.75
19	2	2.80	19	3	20.22	19	4	216.30
20	2	2.69	20	3	20.31	20	4	217.08

Two different versions of MITRA are reported in Eskin and Pevzner,[6] namely MITRA-Count and MITRA-Graph. We provide some of the run times presented in Eskin and Pevzner[6] for the purpose of comparisons. For the $(11, 2)$ instance, MITRA-Count and MITRA-Graph take one minute each. On the other hand, our algorithm takes less than a second for this instance. For the $(12, 3)$ instance MITRA-Count and MITRA-Graph take one minute and four minutes, respectively. For the same instance our algorithm takes 15.53 seconds. For the $(14, 4)$ instance MITRA-Count takes 4 minutes and MITRA-Graph takes 10 minutes. Our algorithm takes 226.83 seconds.

5.4. *The Cases of* $d = 5$ *and* $d = 6$

Our experimental data shown in the previous section deal with $d \leq 4$. When $d = 5$ or $d = 6$, the memory needs of PMS1 and PMS2 exceed the core memory size of the machine used. (We have employed a Pentium4 2.4 GHz machine with a core memory of 1GB.) We are developing out-ofcore algorithms for these two cases. Our estimates indicate that $d = 5$ is solvable in around 20 minutes and $d = 6$ is solvable in a few hours.

When PMS3 is employed we estimate that the instances $(l, 5)$ can be solved in a few seconds when $l \geq 22$. Also, the instances $(l, 6)$ can be solved in a few minutes when $l \geq 26$. These cases are solvable without employing out-of-core techniques.

6. Extensions

The planted (l, d)-motif problem as has been defined (in Pevzner and Sze[11] for example) requires discovering a motif M that occurs in every input sequence at a hamming distance of examctly d. Varitations of this problem can be conceived of. We cosider two variants in this section.

Problem 1(a). Input are t sequences each of length n. The problem is to identify a motif M of length l. It is given that each input sequence has a substring of length l such that the hamming distance between this substring and M is **at most** d.

Problem 1(b). Input are t sequences each of length n. The problem is to find all motifs M of length l. A motif M should be output if it occurs in at least ϵt of the input sequences at a hamming distance of d. Here ϵ is a fraction specified as a part of the input. (This variant has been considered in Buhler and Tompa.[5] They use a value of 1/2 for ϵ).

We have developed algorithms for the above variants, details of which have been omitted due to space constraints.

7. Conclusions

In this paper we have presented exact algorithms for the planted (l, d)-motif problem. Our algorithms are in general faster than MITRA (the best known prior exact algorithm). However our algorithms are very simple and are based on different ideas. The techniques we introduce in this paper are of independent interest. The development of efficient parallel algorithms for the planted motif problem is an interesting open problem. We believe that the techniques introduced in this paper could yield superior results when combined with existing techniques. We plan to explore this possibility.

Acknowledgements

We thank Blanchette[2] for supplying us with the biological data used in our experiment on determining transcription factor binding sites.

References

1. T. L. Bailey and C. Elkan. Unsupervised learning of multiple motifs in biopolymers using expectation maximization. *Machine Learning* 21(1-2), 1995, pp. 51-80.
2. M. Blanchette. Algorithms for phylogenetic footprinting. *Proc. Fifth Annual International Conference on Computational Molecular Biology*, 2001.
3. M. Blanchette, B. Schwikowski, and M. Tompa. An exact algorithm to identify motifs in orthologous sequences from multiple species. *Proc. Eighth International Conference on Intelligent Systems for Molecular Biology*, 2000, pp. 37-45.
4. A. Brazma, I. Jonassen, J. Vilo, and E. Ukkonen. Predicting gene regulatory elements in silico on a genomic scale. *Genome Research* 15, 1998, pp. 1202-1215.
5. J. Buhler and M. Tompa. Finding motifs using random projections. *Proc. Fifth Annual International Conference on Computational Molecular Biology (RECOMB)*, April 2001.
6. E. Eskin and P. Pevzner. Finding composite regulatory patterns in DNA sequences. *Bioinformatics* S1, 2002, pp. 354-363.
7. G. Hertz and G. Stormo. Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics* 15, 1999, pp. 563-577.
8. E. Horowitz, S. Sahni, and S. Rajasekaran. *Computer Algorithms*, W. H. Freeman Press, 1998.
9. U. Keich and P. Pevzner. Finding motifs in the twilight zone. *Bioinformatics* 18, 2002, pp. 1374-1381.
10. C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wootton. Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. *Science* 262, 1993, pp. 208-214.
11. 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.
12. A. Price, S. Ramabhadran and P. A. Pevzner. Finding subtle motifs by branching from sample strings. *Bioinformatics* 1(1), 2003, pp. 1-7.
13. S. Rajasekaran, S. Balla, C.-H. Huang, V. Thapar, M. Gryk, M. Maciejewski, and M. Schiller. Exact algorithms for motif search. *Proc. Asia-Pacific Bioinformatics Conference*, 2005.
14. E. Rocke and M. Tompa. An algorithm for finding novel gapped motifs in DNA sequences. *Proc. Second International Conference on Computational Molecular Biology (RECOMB)*, 1998, pp. 228-233.
15. S. Sinha and M. Tompa. A statistical method for finding transcription factor binding sites. *Proc. Eighth International Conference on Intelligent Systems for Molecular Biology*, 2000, pp. 344-354.
16. R. Staden. Methods for discovering novel motifs in nucleic acid sequences. *Computer Applications in the Biosciences* 5(4), 1989, pp. 293-298.
17. M. Tompa. An exact method for finding short motifs in sequences, with application to the ribo-

some binding site problem. *Proc. Seventh International Conference on Intelligent Systems for Molecular Biology*, 1999, pp. 262-271.

18. J. van Helden, B. Andre, and J. Collado-Vides. Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies. *Journal of Molecular Biology* 281(5), 1998, pp. 827-842.