



Finding Composite Regulatory Patterns in DNA Sequences

Eleazar Eskin¹ and Pavel A. Pevzner²

¹Department of Computer Science, Columbia University, New York, 10027, NY and

²Department of Computer Science and Engineering, University of California at San Diego, La Jolla, 92093-0114, CA

ABSTRACT

Pattern discovery in unaligned DNA sequences is a fundamental problem in computational biology with important applications in finding regulatory signals. Current approaches to pattern discovery focus on *monad* patterns that correspond to relatively short contiguous strings. However, many of the actual regulatory signals are *composite* patterns that are groups of monad patterns that occur near each other. A difficulty in discovering composite patterns is that one or both of the component monad patterns in the group may be “too weak”. Since the traditional monad-based motif finding algorithms usually output one (or a few) high scoring patterns, they often fail to find composite regulatory signals consisting of weak monad parts. In this paper, we present a *MITRA* (*Mismatch TRee Algorithm*) approach for discovering composite signals. We demonstrate that MITRA performs well for both monad and composite patterns by presenting experiments over biological and synthetic data.

Availability: MITRA is available at <http://www.cs.columbia.edu/compbio/mitra/>

Contact: eeskin@cs.columbia.edu

INTRODUCTION

Pattern discovery in unaligned DNA sequences is a fundamental problem in computational biology, with important applications in finding regulatory signals. Current approaches to pattern discovery focus on *monad* patterns that correspond to relatively short contiguous strings that appear (with some mismatches) surprisingly many times (in a statistically significant way). However, many of the actual regulatory signals are *composite* patterns that are groups of monad patterns that occur relatively near each other (GuhaThakurta & Stormo (2001); Gelfand *et al.* (2000); van Helden *et al.* (2000)). These patterns are often associated with co-regulated genes that share two or more transcription factors. A difficulty in discovering composite signals is that one of the component monad signals in the groups may be “too weak,” making the composite signal difficult to find using the traditional monad-based approaches. For example, GuhaThakurta

& Stormo (2001), described a set of yeast *S. cerevisiae* genes that are regulated by two transcription factors with experimentally verified sites, URS1 and UASH, that occur relatively near each other. Although URS1 is strongly conserved and easily found with traditional monad-based approaches, UASH is too weak to be discovered with these approaches.

An example of a composite pattern is a *dyad* signal, which is a pair of monad patterns that appear a fixed (or almost fixed) distance apart. A possible approach to detecting composite and dyad patterns is to find each part (monad) of the pattern separately and then reconstruct the composite pattern. Since the traditional monad-based motif-finding algorithms usually output one (or a few) high scoring pattern, they often fail to find composite regulatory signals consisting of weak monad parts. This is because one of the parts may not be significant on its own. A better approach would be to detect *both* parts of a composite pattern at the same time. In many cases, the composite pattern is strong enough to be detected even in the case its monad parts are too weak to be detected on their own.

In this paper, we present a composite pattern discovery algorithm consisting of two steps. We first cast the composite pattern discovery problem as a larger monad discovery problem by preprocessing the sample. Preprocessing creates a set of “virtual” monads (concatenations of the monad parts of the composite signal) which represent the possible instances of the composite signals. The second step of the algorithm applies an exhaustive monad discovery algorithm to the “virtual” monad problem. The “virtual” monad problem is harder than traditional monad discovery problems because the monads are typically longer and the samples are larger. We present a monad discovery algorithm, *MITRA* (*Mismatch TRee Algorithm*) that efficiently performs exhaustive search over patterns.

MITRA uses pairwise similarity information which is the basis of the WINNOWER algorithm by Pevzner & Sze (2000). and takes advantage of a new insight into pruning the pattern space that allows for more efficient use of pairwise similarity than in the WINNOWER algorithm.

MITRA is able to discover composite motifs of combined length over 30 bp unlike previous exhaustive pattern discovery methods.

Four related works approach the problem of identifying composite patterns by attempting to identify the component monad signals at the same time. The studies of composite patterns were pioneered by Marsan & Sagot (2000). MITRA borrows some insights from their work and further develops some ideas used in their suffix tree approach. The approaches of GuhaThakurta & Stormo (2001) and Liu *et al.* (2001) use profile-based approaches. While being very useful in practice, profile-based approaches do not guarantee convergence to the best dyad signal and may fail when detecting weak dyad signals. The forth related work is the RSA-dyad algorithm by van Helden *et al.* (2000), which compares observed frequencies of pairs of spaced trinucleotides to the expected frequencies of the pairs. Since MITRA is designed to discover patterns that occur with mismatches, it can potentially detect weaker dyads.

We present several tests over biological and synthetic data and demonstrate that MITRA performs well for both monad and composite patterns. In particular, we show that our algorithm automatically recovers the dyad signal in *P. horikoshii* from Gelfand *et al.* (2000) (that the RSA-dyad algorithm failed to find), as well as the composite pattern in *S. cerevisiae* that can not be discovered by traditional monad discovery methods (GuhaThakurta & Stormo (2001)). In a separate paper MITRA has been applied to automatic discovery of composite regulatory signals in completely sequenced bacterial genomes (Eskin *et al.* (2002)).

MONAD PATTERN DISCOVERY

DNA sequences are subjects to mutations and as a result regulatory signals typically occur with some mismatches from the “canonical” patterns. We can represent the canonical pattern as an l -mer (a continuous string of length l). In the case when the biological signal is represented by a profile we use the most frequent nucleotide in every position of the profile to form the canonical pattern. Although this is a crude approximation of the profile we explain, below that our algorithm is able to recover the profile using the canonical pattern as a “seed”. We use the term *pattern* or *signal* to refer to the canonical l -mer. We define the term (l, d) -neighborhood of an l -mer P to represent all possible l -mers with up to d mismatches as compared to P . For the alphabet of nucleotides the size of the (l, d) -neighborhood for any l -mer is $\sum_{i=0}^d \binom{l}{i} 3^i$. We use the term *instances* of the pattern P to refer to l -mers from the (l, d) -neighborhood of P that are present in the sample (i.e., l -mer in the sample with up to d mismatches from P). Given a set of patterns \mathcal{P} we call an l -mer *valid*

if it is an instance of a pattern $P \in \mathcal{P}$.

We define the pattern discovery problem as follows. Given a sequence S , we want to find all l -mers that occur with up to d mismatches at least k times in the sample. Such l -mers are called $(l, d) - k$ patterns. A variant of the problem assumes that the sample S is split into several sequences and we want to find all l -mers that occur with up to d mismatches in at least k sequences in the sample.

There have been many approaches presented to discovery of monad signals. Among the best performing are MEME (Bailey & Elkan (1995)), CONSENSUS (Hertz & Stormo (1999)), Gibbs sampler (Lawrence *et al.* (1993); Neuwald *et al.* (1995); Hughes *et al.* (2000)), random projections (Buhler & Tompa (2001)), combinatorial based approaches (Pevzner & Sze (2000)), and MULTIPRO-FILER (Keich & Pevzner (2002b)). All these approaches focus on discovering the highest scoring signals and may not be applicable in the case where each of the pair of monad signals is not statistically significant on its own. Moreover, the existing software tools to pattern discovery involve some heuristics and/or stochastic optimization procedures and do not necessarily guarantee to find *all* best-scoring monad signals.

In order to obtain a list of candidate monads, one can apply the classical pattern-based approaches such as the Pattern Driven Approach (PDA) or Sample Driven Approach (SDA). The PDA (see Pevzner (2000)) examines all 4^l patterns of fixed length l in lexical order, compares each pattern to every l -mer in the sample, and returns all $(l, d) - k$ patterns. To bypass the problem of excessive time requirements in PDA, Waterman *et al.* (1984) and Galas *et al.* (1985) suggested an algorithm that significantly reduces the time requirements of the pattern-driven approach. They noticed that most of 4^l patterns examined in the PDA are not worth examining since neither these patterns nor their neighbors appear in the sample. Therefore, the time spent examining these patterns in the PDA is wasted and a significant speed up can be achieved by eliminating these patterns from the search. Based on this observation they designed an algorithm which we call the *Sample Driven Approach* (SDA) that only explores the l -mers appearing in the sample and their neighbors.

The SDA first initializes a table of size 4^l . Each entry of the table corresponds to a pattern. For each l -mer in the sample, SDA generates the (l, d) -neighborhood of the l -mer. Each table entry corresponding to an element of the neighborhood is incremented by a certain amount. After all of the l -mers in the sample are processed, the table contains a score reflecting the significance of the pattern. SDA returns all patterns whose table entries have scores that exceed the threshold.

The SDA approach is much faster than the pattern-driven approach but it requires a large 4^l table. As a result, the SDA was not practical for long patterns in

mid 1980s. Moreover, until recently, SDA was not in the mainstream of pattern discovery algorithms and did not result in a practical software tool. Sagot (1998) and Pavesi *et al.* (2001) resurrected the Waterman *et al.* (1984) idea with a new approach that eliminates its excessive memory requirements. At the same time, with the gigabytes of RAM memory routinely available today, the practical values of l have significantly increased as compared to 1980s even without a memory-efficient algorithm.

The SDA approach explores the space of all neighbors of l -mers from the sample in a consecutive fashion, i.e., at the first step it explores all neighbors of the first l -mer from the sample, at the second step it explores all neighbors of the second l -mer, etc. If an l -mer P belongs to the neighborhoods of the l -mers appearing at positions i_1, \dots, i_k in the sample then the information about P is collected at iterations i_1, \dots, i_k . Since information about P is collected at k different iterations, the Waterman *et al.* (1984) approach updates information about P k times during the course of the algorithm. As a result a memory slot for P is occupied during the entire course of the algorithm even if P is not an “interesting” (i.e., not a frequent) l -mer. Since most of l -mers explored by SDA are not interesting the information about them is useless and is later forgotten thus wasting the memory slots for such l -mers. A better solution would be to collect information about all instances of a pattern P at the same time. This removes the need to keep the information about a pattern in memory but requires a new approach to navigating the space of all l -mers. *MITRA* (*Mismatch TRee Algorithm*) approach runs much faster than PDA and SDA and uses only a fraction of the memory of SDA.

The pattern-finding algorithms (like PDA and MITRA) are often contrasted against the profile-based approaches (like Gibbs sampler) for motif-finding and there is a point of view that the profile-based techniques are more biologically relevant for finding motifs in biological samples (Berg & von Hippel (1998)). This is probably the reason why the Waterman *et al.* (1984) algorithm was not popular in the past decade. Sagot and colleagues were the first to rebut this opinion and to develop an efficient version of the Waterman *et al.* (1984) algorithm that was successfully applied to analyze difficult biological samples (Sagot (1998); Marsan & Sagot (2000); Vanet *et al.* (1999)). In fact, there are more similarities than differences between the pattern-based and profile-based approaches. The pattern-driven approaches, similarly to the profile approaches, generate the profiles (as a consensus of found instances of a pattern) but they explore the space of possible motifs in a different and often more efficient way than the stochastic optimization algorithms. Every profile of length l corresponds to a pattern of length l formed by the most frequent nucleotides in every position of the profile. For most biological samples, the

search with this consensus pattern as a seed would return the correct motif and would reconstruct the original profile with variable frequencies in different positions. It indicates that the pattern-driven approaches are at least as good as the profile-based approaches for many biological samples (Sze *et al.* (2002)). On the other hand, Pevzner & Sze (2000), Buhler & Tompa (2001), and Keich & Pevzner (2002b) demonstrated that the pattern-driven approaches perform better than the profile-based methods on simulated samples with implanted patterns. Of course, the “pattern-implantation” model is somehow limited and it remains to be seen whether the pattern-based algorithms deteriorate for the “profile-implantation” model. However, today there is little evidence that the profile-based methods perform any better than the pattern-based methods on either biological or simulated samples.

MISMATCH TREE ALGORITHM

MITRA uses a *mismatch tree* data structure to split the space of all possible patterns into disjoint subspaces that start with a given prefix. By splitting the pattern space, MITRA keeps reducing the pattern discovery into smaller sub-problems similarly to the SPELLER algorithm (Sagot (1998)). MITRA also takes advantage of pairwise similarity between instances. These similarities can be used to construct a graph where each vertex is an l -mer in the sample and there is an edge if the two l -mers are similar (e.g., differ in no more than $2d$ positions). An $(l, d) - k$ pattern will correspond to a *clique* of size k in this graph. This type of approach is the basis of the WINNOWER algorithm (Pevzner & Sze (2000)). In fact, we show that we can impose stronger conditions on the graph for the existence of a pattern than simply a clique of size k .

Splitting Pattern Space

MITRA splits the space of all possible patterns into disjoint subspaces corresponding to patterns that start with a given prefix. A pattern is called *weak* if it has less than k (l, d) -neighbors in the sample. A subspace is called *weak* if all patterns in this subspace are weak. For most of the subspaces, we can quickly conclude that they are weak and save time by not searching the subspace exhaustively. For example, if we are looking for patterns of length l we would first split the space of all l -mers into 4 disjoint subspaces. The first subspace would be the space of all l -mers starting with A , the second subspace would be the space of all l -mers starting with C , etc. We further employ strategies for determining whether the subspace contains a $(l, d) - k$ pattern. If we can rule out that a subspace contains such a pattern, we stop searching in this subspace and release the memory slot. If we cannot rule out that a subspace contains such a pattern, we split this subspace

again on the next symbol and repeat. The key ingredient of MITRA is the method to rule out whether a subspace contains a $(l, d) - k$ pattern.

Mismatch Tree Data Structure

Mismatch trees are similar to the suffix trees and tries that have a long history of applications to string matching problems (see Gusfield (1997)). The paths from the root to the leaves in a mismatch tree represent not only the substrings in the data (like in suffix trees and tries), but also all neighbors of these substrings with up to k mismatches. The data structure is a variation of the *sparse prediction trees* from Eskin *et al.* (2000). A mismatch tree is a rooted tree where each internal node has 4 branches, each labeled with a symbol in $\{A, C, G, T\}$. The maximum depth of the tree is l . Each node in the mismatch tree corresponds to the subspace of patterns \mathcal{P} with a fixed prefix (defined by the path from the root to the node) and contains pointers to all l -mers instances from the sample that are within d mismatches from a pattern $P \in \mathcal{P}$ (valid l -mers). The tree is initialized to contain only a root node and is explored in a depth first fashion over the course of the algorithm.

MITRA starts with examining the root node of the mismatch tree that corresponds to the space of all patterns. When examining a node, MITRA tries to prove that it corresponds to a weak subspace. If we can not prove it, we expand the node's children and examine each of them. This corresponds to splitting the pattern subspace into 4 separate parts. Whenever we reach a node corresponding to a weak subspace, we backtrack, effectively eliminating the subtree rooted at that node from our search. The intuition is that many of the nodes correspond to weak subspaces and can be ruled out. This allows us to avoid searching much of the pattern space that would be searched in SDA. If we reach depth l , the l -mer corresponding to the path from the root to the leaf corresponds to an $(l, d) - k$ pattern and the pointers from this node correspond to the instances of this pattern. In practice, we do not need to explicitly maintain the mismatch tree in memory since we "virtually" traverse the mismatch tree in the depth first fashion.

MITRA keeps track of all valid l -mers at each node in the tree (i.e., instances of patterns from the subspace of patterns that correspond to the node). An l -mer is valid for a node if its prefix matches the prefix of the node with at most d mismatches. The set of valid l -mers for a node is a subset of the set of valid l -mers for the parent of the node. MITRA efficiently generates the set of valid l -mers for a node by keeping track of the number of mismatches between each valid l -mer and the prefix of the node. For a valid l -mer in the parent of a node, there are two cases. Either the position corresponding to the branch to the child matches the l -mer, or the position corresponding to the

branch to the child does not match the l -mer. In the first case, the l -mer is still valid for the child. In the second case, the count of mismatches for that l -mer increases. If the mismatch count exceeds the threshold d , the l -mer is not passed on to the child. Thus a child node's set of valid l -mer is simply the set of valid l -mers of the parent that either match the label of the branch to the child or are still within an acceptable number of mismatches of the prefix.

The MITRA algorithm is as follows. We first examine the root node that corresponds to the set \mathcal{P} of all 4^l l -mers of length l . This node points to all l -mers in the sample. We then examine the first child, A . This child points to all of the l -mers in the sample that have prefix A (with 0 mismatches) and to all of the l -mers in the sample that have a different prefix (with 1 mismatch). We continue with a depth first search and test every node to see if it corresponds to a weak subspace. If yes, we backtrack since there is no $(l, d) - k$ pattern in this subspace. If we reach depth l , then the node corresponds to an $(l, d) - k$ pattern. We then compute the score of the pattern and output the pattern along with the score if it is above some threshold that we consider interesting. Since we are finished with this pattern, we backtrack in the tree, collapse the current node, and expand the next node. Since the only expanded nodes are along the current search path, there is a maximum of l stored nodes in the tree (counting the root node) which bounds the memory usage of the algorithm. Unlike in the SDA algorithm, we do not need to keep all of the patterns in a large table.

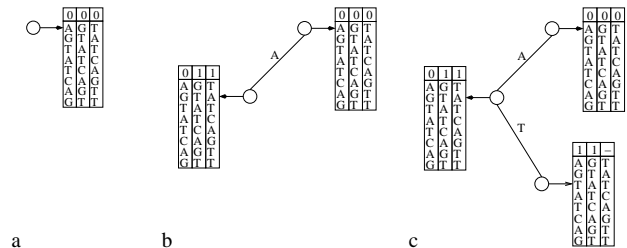


Fig. 1. A Mismatch Tree for a sequence AGTATCAGTT corresponding to a search for $(8, 1)$ motifs. The substrings (instances) (8-mers) in the input sequences are AGTATCAG, GTATCAGT and TATCAGTT. The paths from the root to the nodes define the labels of the nodes. The nodes contain a record for each substring which contains (i) the number of mismatches between the prefix of a substring in the sequence and the label of the node and (ii) a pointer to the tail of the substring. The pointer to the tail of the substring changes position as we move down the tree. (a) The tree is in its initial state. (b) The tree after expanding the path A . (c) The tree after expanding the path AT . Notice that one of the instances reached the maximum number of allowed mismatches. This instance would not be passed farther down the tree.

Consider a very simple example of finding the patterns of length 8 with up to 1 mismatch in the input sequence AGTATCAGTT shown in Figure 1.

A simple test for ruling out a node is simply checking

if it contains less than k valid l -mers. We refer to this algorithm as MITRA-Count that is in fact very similar to the Speller algorithm (Sagot *et al.* (1995); Sagot (1998)). The main difference is that Speller uses a suffix tree to store the data and keeps pointers to nodes in the suffix tree rather than pointers to the actual l -mers in the sample. In Sagot (1998) a worst-case complexity analysis for Speller was presented which also applies to MITRA-Count. In practice, both Speller and MITRA-Count will usually need to traverse a significantly smaller portion of the search space than the SDA algorithm. This explains why MITRA-Count is faster than SDA even though it pays the penalty of updating the data structure.

Incorporating Pairwise Similarity into the Sample Driven Approach

The key new ingredient of MITRA is an insight that the information about pairwise similarities between instances of the pattern can significantly speed up the sample-driven approach. We take advantage of this information using an insight from Pevzner & Sze (2000). We call this variant of the MITRA algorithm MITRA-Graph.

We will construct a graph that models this pairwise similarity. Given a pattern P and a sample S we construct a graph $G(P, S)$ where each vertex is an l -mer in the sample and there is an edge connecting two l -mers if P is within d mismatches from both l -mers. For an $(l, d) - k$ pattern P the corresponding graph contains a *clique* of size k . Given a set of patterns \mathcal{P} and a sample S , define a graph $G(\mathcal{P}, S)$ whose edge set is a union of edge sets of graphs $G(P, S)$ for $P \in \mathcal{P}$. Each vertex of $G(\mathcal{P}, S)$ is an l -mer in the sample and there is an edge connecting two l -mers if there is a pattern $P \in \mathcal{P}$ that is within d mismatches from both l -mers. If for a subspace of patterns \mathcal{P} we can rule out an existence of a clique of size k , then we can prove that the subspace is empty. This idea leads to a significantly more efficient pruning of the mismatch tree than in MITRA-Count and Speller. To implement this idea MITRA-Graph keeps list of the edges of the graph $G(\mathcal{P}, S)$ at each node of the tree and efficiently updates this list while traversing the tree.

The WINNOWER algorithm by Pevzner & Sze (2000) constructs the following graph. Each l -mer in the sample is a vertex. An edge connects two vertices if the corresponding l -mers have less than $2d$ mismatches. Note that instances of a $(l, d) - k$ pattern form a *clique* of size k in this graph. Since cliques are difficult to find, WINNOWER takes the approach of trying to remove edges that do not correspond to a clique. Once all of the “spurious” edges are removed, in many cases the problem is easy to solve since only the clique remains. A problem with the WINNOWER approach is that for subtle signals many edges would remain making it difficult to find the clique.

MITRA-Graph adapts this idea into our framework and (implicitly) constructs a graph at each node in the mismatch tree. We remove edges which we are certain are not part of a clique. If no potential clique remains, we rule out the subspace corresponding to the node and backtrack. If we can not rule out a clique, we split the subspace of patterns considered by examining the child nodes. There exists an innovative difference between the WINNOWER algorithm and MITRA-Graph. MITRA-Graph knows the prefix of the pattern while looking for a clique while WINNOWER does not. WINNOWER must be more conservative in removing edges because it is harder to rule out a clique without knowing the prefix of the pattern. Therefore, MITRA-Graph has the ability to remove edges more efficiently than WINNOWER.

Improvements over the WINNOWER

At each node of the tree, we remove edges by computing the degree of each vertex. If the degree of the vertex is less than $k - 1$, we can remove all edges that lead to the vertex since we know it is not part of a clique. We repeat this procedure until we can not remove any more edges. If the number of edges remaining is less than the minimum number of edges in the clique we can rule out the existence of a clique and backtrack.

The problem with this approach is how to efficiently construct the graph at each node since building it from scratch at every node of the tree is impractical. Instead of constructing the graph from scratch, we construct it based on the graph at the parent. Let e be an edge connecting two l -mers such that the first l -mer matches the prefix of the pattern subspace with d_1 mismatches and the second l -mer matches with d_2 mismatches. We denote the number of mismatches between the tails of the first and second l -mer m . The edge between these l -mers exists in the pattern subspace if and only if $d_1 \leq d$, $d_2 \leq d$ and $d_1 + d_2 + m \leq 2d$. In the root node since $d_1 = d_2 = 0$, an edge exists only if $m \leq 2d$ which is the equivalent graph to WINNOWER. However, with moving down the tree the condition becomes much stronger than the WINNOWER condition and typically lead to better pruning. We can compute the edges of a node based on the edges of parents of the node by keeping track of the quantities d_1 , d_2 , and m for each edge.

To summarize, the MITRA-Graph algorithm works as follows. We first compute the set of edges at the root node by performing pairwise comparisons between all l -mers. We traverse the tree in a depth first order passing on the valid edges and keeping track of the quantities d_1 , d_2 , and m . At each node we prune the graph by eliminating any edges which correspond to vertices that have degrees of less than $k - 1$. If there are less than the minimum number of edges for a clique, we backtrack. If we reach a leaf of the tree (depth l) then we output the corresponding pattern.

The MITRA-Graph pruning condition is very efficient. For example, in the (15, 4) challenge problem from Pevzner & Sze (2000), we typically only need to traverse the nodes at depth 3 before we can rule them out. So although MITRA-Graph has a higher overhead per node than MITRA-Count, it typically searches a much smaller space.

DISCOVERING DYAD SIGNALS

For dyad signals, we are interested in discovering two monads that occur a certain length apart. We use the notation $(l_1 - (s_1, s_2) - l_2, d) - k$ pattern to denote a dyad signal which consists of two monads (a pattern of length l_1 and pattern of length l_2) separated by at least s_1 nucleotides and at most s_2 nucleotides which occurs at least k times in the sample. The key feature of MITRA-Dyad is a possibility to discover dyad patterns with extremely weak monads. This is achieved by reducing the search for $(l_1 - (s_1, s_2) - l_2, d) - k$ dyad pattern to the search $(l_1 + l_2, d)$ monad pattern.

The MITRA-Dyad algorithm casts the dyad discovery problem into a monad discovery problem by preprocessing the input and creating a “virtual” sample to solve the $(l_1 + l_2, d) - k$ monad pattern discovery problem in this sample. The solution to this monad problem in the virtual sample is the solution to the dyad discovery problem.

Specifically the preprocessing is as follows. For each l_1 -mer in the sample and for each $s \in [s_1, s_2]$ we create an $l_1 + l_2$ -mer which is the l_1 -mer concatenated with the l_2 -mer upstream s nucleotides of the l_1 -mer. Note that the number of elements in the “virtual” monad sample will be approximately $(s_2 - s_1 + 1)$ times larger than the original sample. An $(l_1 + l_2, d) - k$ pattern in the “virtual” monad sample will correspond to a $(l_1 - (s_1, s_2) - l_2, d) - k$ pattern in the original sample and we can easily map the solution from the monad problem to the dyad problem.

An important feature of MITRA-Dyad is an ability to search for long patterns (see the Tests section). We remark that the Marsan & Sagot (2000) algorithm may have an advantage while searching for shorter patterns due to the use of suffix trees.

If the range $s_2 - s_1 + 1$ of acceptable distances between monad parts in a composite pattern is large, the MITRA-Dyad algorithm becomes inefficient. A simple approach to detect these patterns is to generate a long ranked list of candidate monad patterns using MITRA and then check each occurrence from each pair from the list to see if they occur within the acceptable distance. The composite pattern is detected if this ranked list is long enough to contain both monads that form the composite pattern.

TESTS

Scoring Patterns

Scoring is a central issue in motif discovery. The scoring functions evaluate the multiple alignment formed by the instances of the motif. They vary from a trivial one like the number of instances of the pattern in the sample to a more involved like distance from consensus, sum-of-pairs, entropy-based scores and others (see Pevzner (2000)).

MITRA is flexible with respect to the particular scoring function used since it first selects many candidate patterns and provides an ability to further evaluate each pattern (and the resulting multiple alignment) with any scoring function. While most of the motif search approaches have a fixed scoring function, (Bailey & Elkan (1995); Buhler & Tompa (2001); Lawrence *et al.* (1993); Neuwald *et al.* (1995); Pevzner & Sze (2000)) MITRA can be adapted to accommodate nearly any scoring function.

One problem faced by pattern based approaches is that some patterns are over represented because of nucleotide bias and low complexity regions. In general patterns that consist of common nucleotides in a sample will occur significantly more often simply by chance than patterns that consist of rare nucleotides. Similarly, low complexity patterns are often over-represented because of low complexity regions that may occur in part of the sample.

A possible approach is to set a low threshold k and then score all of the patterns that are returned with a scoring function to determine which patterns are statistically significant. A problem is that there are too many over-represented patterns to make this approach feasible. If the threshold is low enough then the output is flooded with the over-represented patterns. If the threshold is high enough to reduce the output size, the only patterns in the output are the over-represented patterns.

To solve this problem we use a *dynamic* threshold that is a function of the pattern. k is increased or decreased depending on the specific pattern. Typically, we increase the threshold for over-represented patterns such as patterns consisting of common nucleotides and low complexity patterns (see Eskin *et al.* (2002) for details).

Simulated Data

Pevzner & Sze (2000) defined the challenge problem which many of the best motif discovery methods had difficulties with. The challenge problem corresponds to finding a (15, 4) monad signal of length 15 with 4 “random” mismatches implanted at random positions in $t = 20$ randomly generated sequences of length $n = 600$. Pevzner & Sze (2000) designed the algorithms to solve the challenge problem but failed to solve the very difficult problem of finding (14,4) motifs. Although Buhler & Tompa (2001), and Keich & Pevzner (2002b) designed the

algorithms to find $(14, 4)$ motifs, a more difficult problem of finding motifs that only occur in 13 out of 20 sequences in the sample (corrupted sample) is almost impossible to solve. Such motif will be buried under an avalanche of on average 1363 “random” motifs (Keich & Pevzner (2002a)b). Therefore, the problem of finding a dyad motif consisting of two $(14,4)$ monad motifs occurring in 13 of 20 samples cannot be reduced to two separate problems of finding two $(14,4)$ monads. We define the *dyad challenge* problem in which 13 of $t = 20$ sequences of length $n = 600$ contain a dyad signal formed by a pair of $(14, 4)$ monad signals separated by 20 nucleotides.

To evaluate MITRA on synthetic data, we generated a sample of t random sequences of length n drawn from a uniform nucleotide distribution. For each sample, we generated a random l -mer which represents the target signal. We implanted the l -mer into each of the t sequences at random positions, each time with d mismatches. In a more biologically relevant case of “corrupted” samples (Pevzner & Sze (2000)), we implant the l -mer into $k < t$ sequences (in this case $t - k$ sequences do not contain the signal). This is a more difficult variation of the challenge problem.

In this evaluation framework, we assume that the parameters (lengths and number of mismatches) of the signals that we are looking for are known. Although this is not a reasonable assumption in practice, it is reasonable for an evaluation methodology since the methods can be extended to iterate over reasonable settings of the parameters in practice.

By varying l and d , we generated the motif finding problems of different complexity and evaluated the performance of PDA, SDA, and MITRA (Table 1). Pevzner & Sze (2000) showed that for $n = 600$ and $t = 20$ MEME, CONSENSUS, and GibbsDNA fail for $(11, 3)$, $(12, 3)$, $(13, 4)$, $(14, 4)$, $(15, 4)$, $(16, 5)$, $(17, 5)$, and $(18, 6)$ problems. A new random projections algorithm Buhler & Tompa (2001) is able to solve very difficult problems of $(14, 4)$, $(16, 5)$, and $(18, 6)$. MITRA was able to solve all of the problems presented in the table including the $(14, 4)$ problem. In addition, MITRA (as well as SDA and PDA) can solve the “corrupted” version of these problems where the pattern occurs in some but not all of the sequences.

To evaluate MITRA-Dyad we tested it on the dyad challenge problem. We randomly generated t sequences of length n and implanted a dyad signal into k of the sequences. The dyad signal we inserted was a pair of (l, d) signals separated by 20 bases. For our experiments we used the following parameters $n = 600$, $t = 20$, $k = 13$, $l = 14$, and $d = 4$. MITRA was able to recover the pattern by searching for a $(14 - (20, 20) - 14, 8) - 13$ pattern. It corresponds to approximately finding a $(28, 8)$ monad pattern (Table 1). We also tested the method of van Helden

$(l, d) - k$	PDA	SDA	M-Count	M-Graph
	CPU/MEM	CPU/MEM	CPU/MEM	CPU/MEM
$(11,2)-20$	270/2	1/4	1/5	1/5
$(12,3)-20$	1200/2	1/15	1/5	4/100
$(13,3)-20$	<i>9000/2</i>	5/65	2/5	2/40
$(14,4)-20$	—/—	10/250	4/5	10/210
$(15,4)-20$	—/—	20/1050	5/5	5/100
$(16,5)-20$	—/—	<i>40/4200</i>	25/5	20/400
$(18,6)-20$	—/—	—/—	250/5	40/650
$(28,8)-20$	—/—	—/—	—/—	4/50
$(30,9)-20$	—/—	—/—	—/—	5/90

Table 1. The performance of PDA, SDA, MITRA-Count and MITRA-Graph on synthetic data. The CPU time is given in minutes and the memory usage MEM is given in megabytes. In all experiments, $n = 600$, $t = 20$ and the signal occurs in all of the sequences ($k = 20$). Blank entries “—” or entries in italics denote the inability for the algorithm to solve the challenge problem because of a lack of memory or CPU resources. The italics entries are estimates of the resources necessary to solve the problem. All experiments were performed on machine with a Pentium III 750 GHz processor and 1 GB of RAM.

Sequence	Length (bases)	Num Seq.	MITRA Predictions	Ref.
preproinsulin	7689	4	CCTCAGCCCC	(A)
DHFR	800	4	TGCAATTTCCGCCAAAC	(B)
metallothionein	6823	4	TGCGCCGG	(C)
c-fos	3695	5	CCATATTAGACA	(D)
yeast ECB	5000	5	TTCCCATTAAGGAAA	(E)

Table 2. The performance of MITRA for biological samples with monad motifs from Buhler & Tompa (2001). For each sample, the prediction of MITRA is shown. The nucleotides in the predicted patterns that match the actual binding site are in bold. References: (A) preproinsulin promoter region motif Wingender *et al.* (1996). (B) DHFR non-TATA transcription start signal Means & Farnhan (1990). (C) MREa promoter Anderson *et al.* (1987). (D) *c-fos* serum response element Natsan & Gilman (1995). (E) yeast early cell cycle box McNery *et al.* (1997).

et al. (2000) using their RSA-dyad web server on the same sample. Their method failed to detect the dyad.

Monad Motifs in DNA Sequences

To evaluate MITRA on biological samples, we applied it to upstream regions of orthologous genes with known motifs from Buhler & Tompa (2001). Since *a priori* we do not know the motif length, we simply iterate over possible lengths. We can either do this directly, or a simple variant of MITRA can compute patterns of all length at the same time. By scoring not only the patterns at the leaves, but the internal nodes as well, we can compute patterns of multiple lengths in a single run of MITRA.

Table 2 contains a summary of the data as well as results of MITRA’s prediction. In each of the cases, MITRA was able to recover the correct motif. Moreover, in 4 out of 5 samples MITRA was able to find another strong motif that was not documented in Buhler & Tompa (2001).

Composite Motifs in DNA Sequences

We applied MITRA-Dyad to two sets of biological samples where there are known composite regulatory signals. The first biological sample is formed from the upstream regions involved in purine metabolism from

three *Pyrococcus* genomes studies in Gelfand *et al.* (2000). The signal is a dyad consisting of two monad patterns that occur at a distance varying from 22 to 23.

To detect these signals, we applied MITRA-Dyad. We were able to detect the dyad as shown in Table 3 by searching for $(20 - (22, 23) - 20, 10)$ patterns. We also applied the RSA-dyad algorithm of van Helden *et al.* (2000) to the same sample. Their method registered dyads inside many of the component monads since they are long signals and the ends of the signals are strongly conserved. Since each component monad is 20 bases long, dyad patterns in the form $(3 - (14, 14) - 3, 0)$ would match each component monad. However, this clearly is not the dyad that we are looking for, but merely a side effect of having a long monad signal in the sample. Their method was not able to detect that the middle portion of these monads are also conserved. In addition, their method was not able to detect that the two conserved regions formed a long dyad signal. Note that to find this signal, Gelfand *et al.* (2000) made a conjecture that it is palindromic. MITRA-Dyad does not require such an assumption and is able to find non-palindromic signals as well.

Name	Dyads found by MITRA-Dyad
purC	TTTGCCAGATATATGTCTAA---(23)--TTTTACATAAACATGGTGAA
purF	TTCACCATGTTTATGTAAAA---(23)--TTAGACATATATCTGGCAAA
purT	TTAAACATATTTATGTAAAA---(22)--TTTAAACATTTATACGTCAAT
purE	ATTAGCACATATATGTAGAA---(23)--ATTGACATTAATGCTAGG
purD	GTTAACACGTTTATGTAAAC---(23)--TTTGACTTAAATATGGTGAT
purA	ATTAACATAGCCCTGTCAAAA---(23)--CTTTACTTACCCTTTGGTAA
purB	ATTTCTACAAATATGTCAAAA---(23)--TTTACCCTGAAAAATGGTGAT
purL-II	ATTGACATTTCTTTGTCAAAA---(22)--TTTTACATTTTCTGGCAAA
cons.	ATTAACATATATATGTACAAA-(22,23)-TTTTACATATATATGGTAAA

Table 3. Dyad signals from *P. horikoshii* (Gelfand *et al.* (2000)) predicted by MITRA-Dyad. The last row shows the consensus pattern which is generated by choosing the most frequent nucleotide from the instances of the pattern at each position.

We also analyzed the samples with composite regulatory signals studied by GuhaThakurta & Stormo (2001). These samples consist of four sets of *S. cerevisiae* genes which are regulated by two transcription factors where the two transcription factors binding sites occur near each other. In three of the sets, both monad signals are strong

Gene ID	URS1	UASH	Dist.
YDR285W	TCGGCGGCTAAAT	GATTCGGAAGTAAA	20
YER044C-A	TGGGCGGCTAAAT	TCTTTCGGAGTCATA	23
YER179W	AAATAGCCGCCCA	TTGTGTGGAGAGATA	32
YHR014W	AAATAGCCGCCGA	TAATTAGGAGTATA	19
YNL210W	TTTTAGCCGCCGA	GGTTTTGTAGTTCTA	37
MITRA-Dyad	TAGGCGCCTA-(9,27)-TTTGGAG		

Table 4. URS1 and UASH motifs from GuhaThakurta & Stormo (2001). Binding sites for the gene upstream from yeast are shown where the two components of the composite pattern occur within 50 bases. Distances between binding sites are given. A prediction that overlaps with the actual site is considered correct. Six sequences (not shown) were not analyzed because the URS1 site and UASH site is more than 50 bases apart. The last row shows the top scoring pattern of MITRA-Dyad. The top 3 ranked patterns were minor variants of the shown pattern.

enough to be identified on its own using a standard motif finding program such as CONSENSUS, MEME, ANN-Spec and Gibbs sampler (Hertz & Stormo (1999); Bailey & Elkan (1995); Workman & Stormo (1999); Lawrence *et al.* (1993)). The two monad signals for these sets were detected by running the programs to first detect the stronger of the two monads. The instances of the stronger monad were then deleted from sequences and the program was run over the sequences a second time to identify the second monad (GuhaThakurta & Stormo (2001)).

In the fourth set of genes, one of the regulatory signals is extremely weak, making it difficult to find with a standard motif finding algorithm. The fourth set of genes is a set of 11 genes which are regulated by both the URS1 and UASH transcription factors. For 10 of these genes, the two binding sites are located within the upstream region -300 to -1. In 5 of the genes, the binding sites are within 50 bases of each other. Following the experimental setup of GuhaThakurta & Stormo (2001), we analyzed these five upstream regions. In these sequences, the URS1 signal is very strong while the UASH signal is very weak. The URS1 signal is a $(10, 2) - 5$ signal. If we were looking for just the URS1 pattern, MITRA discovered 453 of these types of signals and the actual binding site was the highest ranking signal. On the other hand, the UASH is a $(7, 1) - 4$ signal. MITRA discovered 1452 of these signals and the actual binding site was the 810th ranking signal. This signal is so weak that it is impossible to discern the binding site from a random match on its own. To detect the composite regulatory signal, we applied MITRA-Dyad. We searched for $(10 - (15, 40) - 7, 3)$ patterns in the samples. The result is shown in Table 4.

Conclusion

In 1984 Waterman *et al.* (1984) presented the SDA algorithm for discovering motifs that searched the space of all neighbors of the substrings in the data. Although this algorithm is very fast in practice it requires a significant amount of memory (Sagot (1998)). In this paper we have presented a new motif finding algorithm MITRA that uses the same idea but bypasses the excessive memory requirements of SDA. MITRA uses a new approach to pruning the search space which improves over previous algorithms.

The MITRA algorithm can be extended to handle insertions and deletions in addition to mismatches. For MITRA-Graph, instead of storing the number of mismatches between two tails of instances, we store their minimum edit distance. MITRA can also be extended to handle wild card symbols using the data structure defined in Eskin *et al.* (2000). A similar technique can be applied to handle the symbols in other meta-alphabets corresponding to pairs of letters like purines.

ACKNOWLEDGMENTS

Thanks to M. Gelfand, U. Keich and S. H. Sze for useful discussions and many comments that significantly improved the manuscript. We would like to thank J. Buhler, M. Gelfand and G. Stormo for sharing biological samples. In addition we would like to thank G. Stormo for also sharing his manuscript prior to publication.

REFERENCES

- Anderson, R. D., Taplitz, S. J., Wong, S., Bristol, G., Larkin, B. & Hershman, H. R. (1987). Metal-dependent binding of a factor in vivo to the metal-responsive elements of the methallothionein I gene promoter. *Molecular and Cell Biology*, **7**, 3574–3581.
- Bailey, T. L. & Elkan, C. (1995). Unsupervised learning of multiple motifs in biopolymers using expectation maximization. *Machine Learning*, **21**, 51.
- Berg, O. & von Hippel, P. H. (1998). Selection of DNA binding sites by regulatory proteins. *Trends Biochem Sci.*, **13**, 207–211.
- Buhler, J. & Tompa, M. (2001). Finding motifs using random projections. In *Proceedings of the Fifth Annual International Conference on Computational Molecular Biology (RECOMB01)*. pp. 69–76.
- Eskin, E., Gelfand, M. & Pevzner, P. (2002). Genome wide analysis of bacterial promoter regions (submitted).
- Eskin, E., Grundy, W. N. & Singer, Y. (2000). Protein family classification using sparse Markov transducers. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*. AAAI Press, Menlo Park, CA, pp. 134–145.
- Galas, D. J., Eggert, M. & Waterman, M. S. (1985). Rigorous pattern-recognition methods for DNA sequences. *Journal of Molecular Biology*, **186**, 117–128.
- Gelfand, M., Koonin, E. & Mironov, A. (2000). Prediction of transcription regulatory sites in archaea by a comparative genomic approach. *Nucleic Acids Research*, **28**, 695–705.
- GuhaThakurta, D. & Stormo, G. D. (2001). Identifying target sites for cooperatively binding factors. *Bioinformatics*, **17**, 608–621.
- Gusfield, D. (1997). Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology. Cambridge University Press, New York.
- Hertz, G. & Stormo, G. (1999). Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics*, **15**, 563–577.
- Hughes, J., Estep, P., Tavazoie, S. & Church, G. (2000). Computational identification of cis-regulatory elements associated with groups of functionally related genes in *Saccharomyces cerevisiae*. *J. Mol. Biol.*, **10**, 1205–1214.
- Keich, U. & Pevzner, P. (2002a). Subtle motifs: defining the limits of motif finding algorithms. *Bioinformatics*.
- Keich, U. & Pevzner, P. A. (2002b). Finding motifs in the twilight zone. In *Proceedings of the 6th International Conference on Computational Molecular Biology (RECOMB 2002)*.
- Lawrence, C. E., Altschul, S. F., Boguski, M. S., Liu, J. S., Neuwald, A. F. & Wootton, J. C. (1993). Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. *Science*, **262**, 208–214.
- Liu, X., Brutlag, D. & Liu, J. (2001). Bioprospector: Discovering conserved dna motifs in upstream regulatory regions of co-expressed genes. In *Proceedings of the 2001 Pacific Symposium on Biocomputing*, volume 6. pp. 127–138.
- Marsan, L. & Sagot, M. (2000). Algorithms for extracting structured motifs using a suffix tree with applications to promoter and regulatory site consensus identification. *Journal of Computational Biology*, **7**, 345–360.
- McInery, C. J., Partridge, J. F., Mikesell, G. E., Creemer, D. P. & Breeden, L. L. (1997). A novel mcm1-dependent element in *swi4*, *cln3*, *cdc6*, *cdc47* promoter activates *m/g₁*-specific transcription. *Genes and Development*, **11**, 1277–1288.
- Means, A. L. & Farnham, P. G. (1990). Transcription initiation from the dihydrofolate reductase is positioned by *hip1* binding at the initiation site. *Molecular and Cell Biology*, **10**, 653–651.
- Natsan, S. & Gilman, M. (1995). Yyl facilitates the association of serum response factor with the c-fos serum response element. *Molecular and Cell Biology*, **15**, 5975–5982.
- Neuwald, A., Liu, J. & Lawrence, C. (1995). Gibbs motif sampling: Detection of bacterial outer membrane repeats. *Protein Science*, **4**, 1618–1632.
- Pavesi, G., Mauri, G. & Pesole, G. (2001). An algorithm for finding signals of unknown length in DNA sequences. *Bioinformatics*, **17**, S207–S214. Proceedings of the Ninth International Conference on Intelligent Systems for Molecular Biology.
- Pevzner, P. A. (2000). Computational Molecular Biology: An Algorithmic Approach. The MIT Press.
- Pevzner, P. A. & Sze, S. (2000). Combinatorial approaches to finding subtle signals in DNA sequences. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*. pp. 269–278.
- Sagot, M. (1998). Spelling approximate or repeated motifs using a suffix tree. *Lecture Notes in Computer Science*, **1380**, 111–127.
- Sagot, M.-F., Escalier, V., Viari, A. & Soldano, H. (1995). Searching for repeated words in a text allowing for mismatches and gaps. In Baeza-Yates, R. & Manber, U., (eds.) *Proceedings of the Second South American Workshop on String Processing*. Viñas del Mar, Chile, pp. 87–100.
- Sze, S., Gelfand, M. S. & Pevzner, P. A. (2002). Finding subtle motifs in DNA sequences. In *Proceedings of Pacific Symposium on Biocomputing*. pp. 235–246.
- van Helden, J., Rios, A. F. & Collado-Vides, J. (2000). Discovering regulatory elements in non-coding sequences by analysis of spaced dyads. *Nucleic Acids Research*, **28**, 1808–1818.
- Vanet, A., Marsan, L. & Sagot, M. (1999). Promoter sequences and algorithmical methods for identifying them. *Research in Microbiology*, **150**, 779–799.
- Waterman, M., Arratia, R. & Galas, D. (1984). Pattern recognition in several sequences: consensus and alignm ent. *Bulletin of Mathematical Biology*, **46**, 515–527.
- Wingender, E., Dietze, P., Karas, H. & Knuppel, R. (1996). TRANSFAC: a database on transcription factors and their DNA binding sites. *Nucleic Acids Research*, **24**, 238–241.
- Workman, C. & Stormo, G. (1999). ANN-Spec: A method for discovering transcription factor binding sites with improved specificity. In *Proceedings of Pacific Symposium on Biocomputing*. pp. 464–475.