Finding Composite Regulatory Patterns in DNA Sequences

Eleazar Eskin[1] and Pavel A. Pevzner[2]

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

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

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 signals. A difficulty in discovering composite signals is that one of the component monad signals 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. A better approach would be to output a long ranked list (i.e., thousands) of top-scoring patterns in a hope that the monad parts belong to this list. Most existing motif finding algorithms lack the ability to produce such ranked list. In this paper we present a new algorithm for discovering composite signals and advocate the use of a pattern search algorithm like the the classical Waterman et al., 1984 [26] sample-driven algorithm for generating such ranked lists. We present a new algorithm, *MITRA (MIsmatch TRee Algorithm)*, to perform this search and present several sets of experiments over biological

and synthetic data. We demonstrate that our approach performs well for both monad and composite motifs.

1 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 signals [8, 22, 25]. This type of pattern can occur when a gene is regulated by two transcription factors. An example of a type of composite regulatory pattern is a *dyad* signal which is a pair of monad patterns that appear a fixed (or almost fixed) distance apart. A difficulty in discovering composite signals is that one of the component monad signal in the groups may be "too weak" making the composite signal difficult to find using the traditional monad-based approaches. For example, GuhaThakurta and Stormo 2001, [8], described a set of yeast *S. cerevisiae* genes 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.

A possible approach to detecting composite patterns is to attempt to detect each part (monad) of the pattern separately and then reconstruct the composite pattern. Unfortunately, the monad parts may be "too weak" to detect on their own. 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. A better approach would be to output a long ranked list (i.e., thousands) of top-scoring patterns in a hope that the monad parts belong to this list. Most existing motif finding algorithms (and stochastic optimization techniques in particular) lack the ability to produce such ranked list. In contrast, a simple exhaustive pattern search can generate such a ranked list through an exhaustive search but has a drawback of excessive time requirement.

In this paper, we present a new composite pattern discovery algorithm. The algorithm consists of two steps. In the first step, we perform a search

over candidate monad patterns and retrieve a list of several thousand best-scoring candidate monad patterns. In the second step, we perform a pairwise comparison of candidate monad patterns. In this comparison, we check the positions of the instances of candidate monads to determine if any of the pairs of monads form a composite pattern. The algorithm can both detect dyad patterns and the more general class of composite patterns. The major computational bottleneck of the algorithm is the retrieval of the list of candidate patterns. We present a new monad discovery algorithm, *MITRA (MIsmatch TRee Algorithm)*, which generates this list much more efficiently than previous methods.

Three related works approach the problem of identifying composite patterns by attempting to identify the component monad signals at the same time. The approach of GuhaThakurta and Stormo 2001, [8] uses a profile based approach. While being very useful in practice, profile based approaches do not guarantee convergence to the best dyad signal and may fail for weak dyad signals. The second related work is the dyad pattern search algorithm of [13] which searches for structured (composite) motifs using a suffix tree. However, the complexity of their algorithm limits these motifs to be relatively short compared to the ones that are discovered by MITRA. The third related work is the approach of van Helden et al. 2000 [25] which compared observed frequencies of pairs of spaced trinucleotides to the expected frequencies of the pairs. 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. We first present a set of experiments evaluating MITRA versus other monad signal finding methods. We test both monad and dyad algorithms with synthetic and biological samples. In particular we show that our algorithm automatically recover the dyad signal in *P. horikoshii* studied in Gelfand et al., 2001 [22] as well as the composite pattern in *S. cerevisiae* that can not be discovered by traditional monad discovery methods [8].

2 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).

However, the observed instances of the signal will typically be the l-mer with several mismatches in different positions. 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. Precisely this is the set of strings of length l which have maximum d positions differing from the l-mer.

We can formally define pattern search as follows. Given sample S, we want to recover all l-mers such that they occur with up to d mismatches at least k times in the sample. A variant of the problem assumes that the sample S is split into several sequences and we want to find an l-mer that occurs with at most d mismatches in k of the sequences.

There have been many approaches presented to discovery of monad signals. Among the best performing are MEME [2], CONSENSUS [10], Gibbs sampler [12, 17], random projections [5] and combinatorial based approaches [21]. 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 even guarantee to find the best-scoring monad signals.

In order to obtain a list of candidate monads, we can apply the classical Pattern Driven Approach. The PDA [20] examines all 4^l patterns of fixed length l, and compares each pattern to every l-mer in the sample. It returns all of the patterns that match l-mers with up to d mismatches. The search through the space of all 4^l patterns is done in lexical order which allows for easy parallelization of the algorithm. The pattern-driven approach guarantees to find the best scoring patterns at the expense of an overwhelming computational cost.

To bypass the problem of excessive time requirements in PDA, Waterman et al., 1984 [26] and Galas et al., 1985 [7] 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 hash table of size 4^l. Each entry of the hash table corresponds to a pattern. For each l-mer in the sample, SDA generates

the (l, d)-neighborhood of the l-mer. Each hash table entry corresponding to an element of the neighborhood is incremented by a certain amount (see Waterman et al., 1984 [26] for details). After all of the l-mers in the sample are processed, the hash table contains a score reflecting the significance of the pattern. SDA returns all patterns whose hash table entries have scores that exceed the threshold.

For the alphabet of nucleotides the size of the (l, d)-neighborhood for any l-mer is $\sum_{i=0}^{d} \binom{l}{i} 3^i$. Each term in the sum corresponds to the number of members of the neighborhood with exactly i positions differing from the l-mer (mismatches). For simplicity of presentation, we use the notation (l, d)-pattern to denote the (l, d)-neighborhood of a specific l-mer. For example, the $(4, 2)$-pattern $AGTC$, can appear as $ACTC$ and $AGGG$ because both these l-mers are in the $(4, 2)$-neighborhood of $AGTC$, but it can not appear as $CGAT$ since it is not in the $(4, 2)$ neighborhood of $AGTC$.

The SDA approach is much faster than the pattern-driven approach but it requires a large 4^l hash table. As a result, the SDA was not practical for long patterns in mid 1980s. Moreover, it was not in the mainstream of pattern discovery algorithms in the last decade and did not result in a practical software tool in 1990s. We feel that the time has come to resurrect the Waterman et al., 1984 [26] idea with a new approach that eliminates its excessive memory requirements. Not to mention that gigabytes of RAM memory routinely available today make the memory issue less precious. Therefore, the practical values of l have significantly increased as compared to mid80s even without a memory-efficient algorithm.

The SDA approach explores the space of all neighbors of l-tuples form the sample in a consecutive fashion, i.e., at the first step it explores all neighbors of the first l-tuple from the sample, at the second step it explores all neighbors of the second l-tuple, etc. If an l-tuple P belongs to the neighborhoods of the l-tuples appearing at positions $i_1, \ldots, i_k$ in the sample then the information about P is collected at iterations $i_1, \ldots, i_k$. Since information about P is collected at k different iterations, the Waterman et al., 1984 [26] updates information about P k different 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-tuple. Since most of l-tuples explored by SDA are not interesting the information about them is useless and is later forgotten thus wasting the memory slots for such l-tuples. 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. This idea requires a new approach to navigating the space of all l -tuples explored by SDA algorithm.

In this paper we present a new *MITRA (MIsmatch TRee Algorithm)* approach that uses only a fraction of the memory of SDA. The idea of MITRA is to split the space of all possible patterns into disjoint subspaces corresponding to patterns that start with a given prefix. In most of these subspaces, we can quickly conclude that there is no pattern with k instances, and save time by not searching the subspace exhaustively. For example, if we are looking for patterns of length l in DNA sequences. 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. As we describe below, we employ strategies for determining whether the subspace contains a pattern with k instances. If we can not rule out that a subspace contains such a pattern, we split the subspace again on the next symbol and repeat. We present two variants of the MITRA which differ in the method that they rule out whether a subspace contains a high scoring pattern. The first variant, MITRA-Count, gives an algorithm that is similar to the recently developed WEEDER software [18]. WEEDER is based on Sagot, 1998 [23] approach to pattern discovery that significantly improves the space efficiency of the Waterman et al., 1984 [26] algorithm. Although there are some minor differences between MITRA-Count and the Sagot, 1998 [23] algorithm, they are very similar and can typically be applied to search of length 15 or less with up to 4 mismatches.

However, the framework of MITRA of splitting the space of possible patterns can motivate more aggressive methods for determining whether or not a subspace of patterns contains a high scoring pattern. Our MITRA-Graph algorithm uses a technique similar to Pevzner and Sze, 2000 [21] graph-theoretic approach to determining whether or not a high scoring pattern exists. For long patterns, MITRA-Graph significantly improves the efficiency of the pattern search over previous algorithms and allows for performing search for patterns of length over 30 nucleotides which is impossible with previous algorithms.

The pattern finding algorithms (like PDA) are often contrasted against the profile-based approaches (like Gibbs sampler) to motif finding and there is a point of view that the profile-based technique are more biologically relevant for finding motifs in biological samples [3]. This is probably the reason why the Waterman et al., 1984 [26] algorithm was not popular in the past

decade. In fact, there are more similarities than the differences between these two 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 would return the correct motif and would reconstruct the original profile thus indicating that the pattern-driven approaches are at least as good as the profile-based approaches for many biological samples. In fact, Sze et al., 2001 [24], recently benchmarked different motif finding algorithms and demonstrated that pattern-based approaches may perform better than profile-based methods for some difficult biological samples. Similarly, Pevzner and Sze, 2000 [21] and Buhler and Tompa, 2001 [5] demonstrated that the pattern-driven approaches perform better than the profile-based methods on simulated samples with implanted motifs. In summary, there is currently no evidence that the profile-based methods perform better than the pattern-based methods on either biological or simulated samples.

3 Mismatch Tree Algorithm

One approach to the search for composite patterns is to form a large ranked list of best-scoring monad patterns. Since SDA has excessive memory requirements, we present an efficient algorithm, MITRA, to address this problem. MITRA has two variants. The first variant, MITRA-Count, uses minimal memory and in practice can performs pattern searches with up to 5 mismatches. The second variant, MITRA-Graph, can perform much larger searches but uses more memory. The combination of both algorithms allows us to outperform previous methods as we show below.

A mismatch tree is similar to the suffix tree data structure which has a long history of applications to string matching problems [9]. A mismatch tree is similar to a suffix tree in which the paths from the root to the leaves represent not only the substrings in the data, but also all valid mismatches of these substrings. The data structure is a variation of the sparse prediction tree data structure presented in Eskin et al., 2000 [6] which built on an idea presented in [19].

Given a sequence S, and parameters l, d and k, we want to recover all of the (l,d)-patterns that occur at least k times in the sample (denoted $(l,d)-k$). It is easy to extend the algorithm to handle samples with multiple sequences. This is outlined below.

A mismatch tree is a rooted tree where each node is either an internal node or a leaf node. Each internal node has 4 branches labeled with each element of $\{A,C,G,T\}$. The maximum depth of the tree is l.

Each node in the mismatch tree corresponds to the subspace of patterns which prefixes match the path of the tree from the root to node. For example the node that is reached by the path ACG corresponds to the space of all patterns that have a prefix ACG. In this space $ACGTGC$ would be a possible pattern while $AGGTGC$ would not. The root node of the mismatch tree corresponds to the space of all patterns.

The basic approach used in MITRA is as follows. We first start with examining the root node. When examining a node, if we can not rule out the possibility of the subspace corresponding to the node containing a relevant pattern, we expand the node children and examine each of them. If we reach depth l, the pattern corresponding to the path from the root to the leaf corresponds to a valid pattern. Whenever we reach a node that we can rule out containing a pattern, we backtrack effectively eliminating the subtree rooted at that node from our search.

The intuition is that for many of the nodes, we may be able to rule out having a valid pattern in the subspace corresponding to the node. This allows us to avoid searching much of the pattern space that would be searched in PDA or SDA. 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.

The two variants of MITRA differ in the information they keep track of as they traverse the tree and the test they use to rule out a subtree. The first variant MITRA-Count keeps track of instances that can occur in the subspace corresponding to the node. The second variant MITRA-Graph uses an approach similar to Pevzner, Sze, 2000 [21]. If we construct a graph where each node is an instance in the sample and there is an edge connecting two instances if there is a pattern that is within the acceptable number of mismatches of both instances. In this graph, a pattern that occurs k times is a clique of size k. If we can rule out an existence of a clique, then we can rule out the existence of a pattern.

3.1 MITRA-Count

In MITRA-Count, at each node in the tree, we keep track of what instances are "valid" in the subspace of patterns that correspond to the node. An instance is "valid" if the prefix of the instance matches the prefix of the pattern space with at most d mismatches. Otherwise, no pattern in the space can possibly match the instance. Note that if there are less than k valid instances in a node, there is no pattern in the subspace associated with the node which will occur k times. This is the test we use to rule out a node.

The key to the MITRA-Count algorithm is how we generate the set of valid instances for a node. Notice that the set of valid instances for a node is a subset of the set of valid instances for the parent of the node. This is because the space of patterns corresponding to the node is a subspace of patterns corresponding to the parent node. We can efficiently generate the valid instances for a node by keeping track of the number of mismatches that it has with the prefix of the pattern space. For a valid instance in the parent of a node, there are two cases. Either the position corresponding to the branch to the child matches the instance in which case the instance is still valid in the child, or the position corresponding to the branch to the child does not match the instance. In this case, the count of mismatches for that instance increases. If the mismatch count is higher than the number of allowable mismatches, the pattern is not passed on to the child. Thus a child node's set of valid instances simply is the valid instances of the parent that either match the branch to the child or are still within an acceptable number of mismatches.

The MITRA-Count algorithm is as follows. We first examine the root node. The root of the tree contains all of the substrings with 0 mismatches. We then examine the first child, A. This child contains all of the instances that have prefix A with 0 mismatches and all of the instances that have another prefix with 1 mismatch. We continue with a depth first search until one of two things happen. If the number of instances is less than k, we backtrack since we know there is no valid pattern. If we reach depth l, then we know that the path is a valid pattern there and we 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 do not need to store any of the instances of the pattern and can backtrack in the tree, we 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 internal nodes in the tree (counting the root node) which bounds the memory usage of the algorithm. Unlike the SDA algorithm, since we do not need to keep information about each pattern during the entire algorithm, we do not need to keep all of the patterns in a large hash table.

Consider a very simple example finding the patterns of length 8 with up to 2 mismatches in the input sequence $AGTATCAGTT$. The substrings in the input sequences are $AGTATCAG$, $GTATCAGT$ and $TATCAGTT$. Clearly in a practical problem, there would be thousands of substrings. The initial tree for the this example is shown in Figure 3.1(a). Once we initialize the tree as shown in the figure, we begin traversing the tree looking for the best scoring pattern. Figure 3.1(b) shows the tree after expanding the branch A. Notice that in most of the substrings in the leaves the number of mismatches increases. As we continue expanding the tree for many of the substrings the number of mismatches will reach the maximum allowed which will cause many of substrings to not pass on the the leaf nodes. This is shown in Figure 3.1(c) where the branch G is expanded from Figure 3.1(b). The deeper we get in the tree, the fewer occurrences of patterns we observe. This results in speeding up the traversal of the tree.

Essentially, MITRA-Count performs the same search as SDA, but orders l-tuples in the search differently, similarly to the search in PDA. The mismatch tree is used as an index to the data to allow for all of the instances of a pattern to be determined efficiently. The average case complexity analysis for MITRA-Count is non trivial because it depends on the scoring function used and the nature of the data. Since MITRA-Count and SDA perform a search over the same space, it is easy to compare the two. Compared to SDA, MITRA has an advantage of minimizing the memory requirements and allowing for parallelizing at the price of the penalty of the overhead in managing the data structure.

MITRA-Count can be optimized to further reduce the search space. The idea is, compute the upper bound for the scores of the children nodes. For example, if we use the number of instances as the scoring function, the number of instances at the internal node is the upper bound for the number of instances of the child node. The idea is when MITRA-Count reaches a node with few instances, it does not always need to expand the tree. If MITRA-Count can determine that the maximum possible score of the children nodes to be below a threshold that is considered interesting, MITRA-Count can avoid examining the children node.

3.2 MITRA-Graph

The MITRA framework of splitting the space of patterns into disjoint subspaces that are defined by pattern prefixes allows for other approaches to ruling out whether or not a subspace contains a pattern. MITRA-Graph uses a graph theory based approach to this problem.

In Pevzner, Sze, 2000 [21], the WINNOWER algorithm was presented. WINNOWER constructs the following graph. Each instance (l-mer) in the sample is a node. An edge connects two nodes if the corresponding l-mers have less than $2d$ mismatches. Note that if a pattern exists, the instances of the pattern form a clique of size k in this graph. Since in general, cliques take exponential time to compute, WINNOWER takes the approach of trying to quickly find the edges that do not correspond to a clique and remove them. Once all of the "spurious" edges are removed, in many cases the problem is trivial 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 and slowing down the alignment

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 clique remains, we know we can 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. Therefore, MITRA-Graph has the ability to remove edges more easily than WINNOWER.

At each node, we remove edges by computing the out degree of each node. For any elements of the clique, the out degree must be at least k. If the out degree of the node is less than k, we can remove all edges that lead to the node 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 ($\binom{k}{2}$) we can rule out the existence of a clique and backtrack.

The main difficulty is how to efficiently construct the graph at each node. Computing the pairwise mismatches is impractical because it is too expensive. However, we can use a similar approach to MITRA-Count and instead

of constructing the graph from scratch, construct it based on the graph at the parent. Note the following observation. Let e be an edge such that the first instance of the edge matches the prefix of the pattern subspace with d_1 mismatches and the second instance matches with d_2 mismatches. We denote the number of mismatches of the tails of the first and second instance m. The edge 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 stronger than the WINNOWER condition and may 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.

The MITRA-Graph algorithm works as follows. We first compute the set of edges at the root node by performing a pairwise comparison between instances. 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 nodes that have an out degree of less than k. If there are less than the minimum number of edges for a clique, we backtrack.

The MITRA-Graph allows us to examine long patterns. The pruning condition is very strong and usually we can very quickly rule out a node. For example, in the $(15, 4)$ challenge problem from Pevzner and Sze [21], we typically only need to traverse the nodes at depth 3 before we can rule them out. So although we have a higher overhead per node than MITRA-Count, we typically have to search a much smaller space.

3.3 Extension to MITRA

The algorithms described above were presented to solve the problem of detecting a pattern that occurs k times in a single sequence. They can be easily adapted to the problem of detecting a pattern that occurs in k distinct sequences. The only modification to the algorithm is in the test to see whether or not we can rule out the existence of a pattern in the subspace.

In both variants of MITRA, we keep track of the source of each instance. In MITRA-Count, at each node, we check to see if there is an instance from k different subsequences. If this is not the case, then we rule out the existence of a pattern in the corresponding subspace. In MITRA-Graph, instead of the standard graph, we construct a multipartite graph where the nodes are

grouped based on their source. Instead of computing the out degree of a node, we compute to how many different sources it is connected to by edges. If the number of sources is less than k we can remove the edges. Notice that the pruning conditions for the Multiple Sequence problem are stronger which corresponds to a smaller search space.

4 Discovering Dyad Signals

For dyad regulatory signals, we are interested in discovering two monad signals that occur a fixed length apart. We use the notation $(l_1, d_1) - s - (l_2, d_2)$-pattern to denote a dyad signal which consists of two monad signals (a (l_1, d_1)-pattern and a (l_2, d_2)-pattern) separated by distance s.

The MITRA-Dyad algorithm discovers dyad signals by a pairwise comparison of high scoring monad signals. Given that we are looking for a highest scoring $(l_1, d_1) - s - (l_2, d_2)$-patterns, we first use MITRA to retrieve the ranked lists of the several thousands highest scoring (l_1, d_1)-patterns and (l_2, d_2)-patterns. Note that if $l_1 = l_2$ and $d_1 = d_2$ the two lists will be the same.

This approach is in fact more complicated than it sounds since the resulting list of top-scoring monad signals is usually "contaminated" by the variations of the same patterns and the question arises on how to filter this list [4]. If a "strong" pattern p appears (with up to d mutations) in the sample then variations of p often occur in the same positions with up to d mutations. A problem with using the top several thousand candidate patterns in this list directly is that many of the high scoring signals are small variations of each other. This observation explains why the list of high-scoring patterns is often "contaminated" by very similar patterns. We examine the physical positions of the candidate signals to prune our signals. We use a simple heuristic to enforce that the reported patterns are in fact different signals. We eliminate a candidate pattern P from the list if more than a certain portion (80% in our tests) of the instances have the same positions in the sample as the instances of another pattern that is scored even higher than P.

The dyad discovery algorithm runs in $O(m^2)$ time with respect to the number m of candidate monad signals evaluated. Typically, we examine several thousand candidate monad signals. In a naive implementation we examine all pairs of instances by sorting the instances based on their positions and then comparing the instances of the two patterns by examining

their sorted lists of instances. This part of the algorithm can be improved by using a hash table to store the instances, but since it is not a computational bottleneck compared to running MITRA, this was not necessary for our experiments.

5 Tests

In [21], Pevzner and Sze defined the challenge problem which many of the best signal discovery methods had difficulties with. The challenge problem corresponds to finding a monad signal of length 15 with 4 "random" mismatches implanted at random positions in $t = 20$ randomly generated sequences of length $n = 600$. Although Pevzner and Sze, 2000 [21], Buhler and Tompa, 2001 [5], Keich and Pevzner, 2001 [11] designed the algorithms to solve the challenge problem the more difficult problem of finding (14,4) motifs that occur in only 13 of the samples is almost theoretically impossible to solve. Keich and Pevzner, 2001 [11] demonstrated that the (14,4)-motif occurring in 13 of the 20 sequences will be buried under an avalanche of on average 1363 "random" motifs. 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 separate (14,4) monad signals.

For these signals, we define a related problem, the *dyad challenge* problem in which 12 of $t = 20$ sequences of length $n = 600$ contains a dyad signal formed by a pair of monad signals of length 14 with 4 mismatches separated by 20 bases. The difficulty of the challenge problem is that many of the $(14, 4)$ monad signals occur by chance.

We perform two types of experiments using our algorithms. In the first set of experiments, Below, we show how MITRA and MITRA-Dyad perform on the synthetic challenge problems and biological samples.

5.1 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 [20].

An advantage of MITRA is that it 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, [2, 5, 12, 17, 21] MITRA can be easily adapted to accommodate *any* scoring function. For many scoring functions (like distance from consensus or sum-of-pairs), because MITRA performs the equivalent to an exhaustive search (although significantly more efficient), it will guarantee that we will return the highest scoring pattern unlike some other motif finding algorithms.

In our experiments we use two scoring functions. The first scoring function is simply the number of occurrences within a certain number of mismatches. For example, if we are looking $(15, 4)$-patterns, the score would simply be the number of instances of a given pattern with up to 4 mismatches in the data. We use this scoring function for evaluation over synthetic data.

For biological samples, we use the *SP-STAR* score as defined in [21]. The SP-STAR score compares the pairwise instances of the pattern to derive the score. A similarity function between two l-mers is defined as follows. For every pair of instances of the pattern P, P_i and P_j, we compute the Hamming distance $H(P_i, P_j)$ between the two instances. The score for a pattern is then the sum of pairwise Hamming distances. For instances of a (l, d)-pattern $P_1, ..., P_n$ this is just $\sum_{i,j:i \neq j} H(P_i, P_j)$. For reasons described in [21] this is a reasonable scoring function for biological samples. Recently Sze et al., 2001 [24] demonstrated that this score is efficient for finding motifs in many difficult biological samples and described how to use SP-STAR for biased samples using an entropy based modification to the scoring scheme.

5.2 Simulated Data

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 [21], 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. We perform the evaluation to compare the algorithms running times and memory usage.

Note that in this evaluation framework, we are assuming that the methods

know the exact parameters (lengths and number of mismatches) of the signals that we are looking for. 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. We discuss this when we examine biological samples.

By varying l, d, m, and n, we generated the motif finding problems of different complexity and to evaluate the performance of PDA, SDA, and MITRA. Pevzner and Sze [21] 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 [5] is able to solve very difficult problems of $(14, 4)$, $(16, 5)$, and $(18, 6)$. Table 5.2 shows the performance of PDA, SDA, and MITRA. The WEEDER algorithm [23, 18] is similar to MITRA-Count and we expect it to perform similarly. Note that in all cases, either MITRA-Count or MITRA-Graph are either comparable or outperform PDA and SDA. In the cases when PDA takes too much time to complete the search, we estimated its running time based on timing a portion of its search. For SDA approach, we were not able to run pattern search for $l > 15$ because of memory considerations. Even at this point the implementation was tricky because we use only 1/2 a byte to store each entry of the hash table and compress a string of length 15 into a 32 bit integer. This allows us to compare the identity of two stings very efficiently because it is just one CPU operation. At length 16, this breaks down and the memory requirements and running time increase even further.

One of the variants of MITRA was able to solve all of the problems presented in the table including the $(14, 4)$ problem. Note that MITRA (as well as PDA and SDA) guarantees to return the top-scoring solutions while other methods do not provide such guarantee to return a list of thousand top-scoring signals that is needed for dyad motif finding. 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. This is used in solving the dyad challenge problem.

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 $n = 600$, $t = 20$, $k = 12$, $l = 14$, and $d = 4$.

After applying MITRA to our data set with $k = 13$ threshold, we obtained 1252 candidate monad signals. In fact by using the analysis of Keich and

			PDA		SDA		MITRA-Count		MITRA-Graph		
l	d	k	CPU	MEM	CPU	MEM	CPU	MEM	CPU	MEM	Edge %
11	2	20	270	2	1	4	1	5	1	5	0.72%
12	3	20	1200	2	1	15	1	5	4	100	5.16%
13	3	20	*9000*	*2*	5	65	2	5	2	40	2.39%
14	4	20	–	–	10	250	4	5	10	210	10.59%
15	4	20	–	–	20	1050	5	5	5	100	5.37%
16	5	20	–	–	*40*	*4200*	25	5	20	400	18.01%
18	6	20	–	–	–	–	*250*	*5*	40	650	26.83%
28	8	20	–	–	–	–	–	–	4	50	2.79%
30	9	20	–	–	–	–	–	–	5	90	4.81%

Table 1: The performance of PDA, SDA, MITRA-Count and MITRA-Graph on several versions of the challenge problem. The CPU time is given in minutes and the memory usage is given in megabytes. In all experiments, $n = 600$, $t = 20$ and the signal occurs in either all of the sequences ($k = 20$) or in 15 of them ($k = 15$). The CPU time for PDA is estimated for $l > 12$ because of the long computational time necessary. 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 as estimates of the resources necessary to solve the problem. The last column gives the percentage of the possibles edges that exist in the graph at the root node of MITRA-Graph. Note that the number of edges is a better indicator for determining the memory and CPU usage than (l, d).

Sequence	Length (bases)	Num Sequences	MITRA Prediction	Prediction (l,d,k)	Strong Motifs	Published motif	Ref.
preproinsulin	7689	4	CCTCAGCCCCC	(11,0)-3	2	CCTCAGCCCC	(A)
DHFR	800	4	CGGCTGCAATTTCGCGCCAAACTTG	(25,2)-3	1	ATTTCnnGCCA	(B)
metallothionein	6823	4	TGCGCCCGG	(9,0)-3	1	TGCRCYCGG	(C)
c-fos	3695	5	CCATATTAGGACA	(13,2)-4	2	CCATATTAGAGACTCT	(D)
yeast ECB	5000	5	TTTCCCATTAAGGAAA	(16,3)-5	0	TTtCCcnntnaGGAAA	(E)

Table 2: Results for MITRA applied to 5 biological samples with monad subtle from [5]. For each sample, the prediction of MITRA is shown as well as the $(l, d) - k$ parameters of the predicted signal. In some cases, there were several strong motifs, the number of which is listed. The shown motif is the one closest to the biological sample. References: (A) preproinsulin promoter region motif[27]. (B) DHFR non-TATA transcription start signal [15]. (C) MREa promoter [1]. (D)*c-fos* serum response element [16]. (E) yeast early cell cycle box [14].

Pevzner 2001 [11] we can expect to observe about 1363 $(14, 4)$ signals in 13 of the 20 sequences. Once we performed a pairwise comparison looking for instances of these monads that occurred 20 bases apart only 25817 pairs of the candidate monads had any valid instances which is a tiny fraction of the total possible pairs of patterns. Of those dyads and only 10 had more than 2 instances an only one had more than 5 instances. That one dyad pattern was the correct signal.

5.3 Monad Motifs in DNA Sequences

To evaluate MITRA on biological samples, we applied it to upstream regions of orthologous genes where there exists a known motif. Since *a priori* we do not know the exact length of the biological sample, 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. The instances at internal nodes correspond to instances of patterns of the depth of the node. By scoring not only the patterns at the leaves, but the internal nodes as well, we can compute patterns of multiple lengths at the same time.

We use the SP-STAR scoring function that takes into account nucleotide bias (Sze et al., 2001 [24]) and score corrupted samples using the method described in [21]. The specific data we use is the data that was examined in [5]. Table 5.3 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.

Name	Spec	Position	Sequence
purC	PH239	-59	TTTGCCAGATATATGTCTAA-(23)-TTTTACATAAACATGGTGAA
purF	PH240	-65	TTCACCATGTTTATGTAAAA-(23)-TTAGACATATATCTGGCAAA
purT	PH318	-68	TTAAACATATTTATGTTAAA-(22)-TTTAACATTTATACGTCAAT
purE	PH320	-92	ATTAGCACATATATGTAGAA-(23)-ATTGACATTAAATTGCTAGG
purD	PH323	-69	GTTAACACGTTTATGTAAAC-(23)-TTTGACTTAAATATGGTGAT
purA	PH438	-69	ATTAACATAGCCCTGTCAAA-(23)-CTTTACTTACCCTTTGGTAA
purB	PH852	-65	ATTTCTACAAATATGTCAAA-(23)-TTTACCGTGAAAATGGTGAT
purL-II	PH1953	-76	ATTGACATTTCTTTGTCAAA-(22)-TTTTACATTTTTCTGGCAAA
predicted dyad pattern			TTTTACATATATATGTTAAA-(22)-TTTAACATATATATGTAAAA

Table 3: Dyad Biological Sample (from [22]). A dyad signal from *P. horikoshii*. The last row shows the pattern predicted by MITRA-Dyad's predicted.

		URS1		UASH		
Gene ID	Name	Position	Mapped Site	Position	Mapped Site	Distance
YDR285W	ZIP1	-22	TCGGCGGCTAAAT	-42	GATTCGGAAGTAAA	20
YER044C-A	MEI4	-98	TGGGCGGCTAAAT	-121	TCTTTCGGAGTCATA	23
YER179W	DMC1	-143	AAATAGCCGCCCA	-175	TTGTGTGGAGAGATA	32
YHR014W	SPO13	-100	AAATAGCCGCCGA	-119	TAATTAGGAGTATA	19
YNL210W	MER1	-115	TTTTAGCCGCCGA	-152	GGTTTTGTAGTTCTA	37
MITRA-Dyad Predicted Pattern:			TAGGCGCCTA-(9,.27)-TTTGGAG			

Table 4: DNA binding information for URS1 and UASH. This is the same experimental setup from [8]. Binding sites and positions for the gene upstreams from yeast are shown where the two components of the composite pattern occur within 50 bases. All positions are relative to the annoatatd translation start sites of the respective genes. Distances between binding sites are given. A prediction that overlaps with the actual site is considered correct. The correct predictions are marked with *. Six sequences (not shown) were not analyzed because the URS1 site and UASH site is more than 50 bases apart. This table was taken from [8]. The last row shows the top scoring pattern of MITRA-Dyad. The top 3 ranked patterns were minor variants of the shown pattern.

5.4 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 that was examined in Gelfand, Koonin, and Mironov 2001 [22]. The signal is a dyad signal which consists of two monad patterns that occur at a distance varying from 21 and 24.

To detect these signals, we applied MITRA-Dyad. When scoring candidate dyads, we only considered instances where the signals were within the correct distance. We were able to detect the dyad as shown in Table 5.4.

The second set of biological samples are the composite regulatory signals

that were analyzed by GuhaThakurta and Stormo 2001, [8]. These samples consist of four sets of *S.cerevisiae* genes which are regulated by two transcription factor binding sites where the two transcription factors occur near each other. In three of the sets, both monad signals are strong enough to be identified on its own using a standard motif finding program such as CONSENSUS, MEME, ANN-Spec and Gibbs sampler [10, 2, 28, 12]. 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 [8].

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 [8], we analyzed these five upstream regions. In these sequences, the URS1 signal is very strong while the UASH signal is very week. The URS1 signal is a $(10, 2) - 5$ signal. 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 these composite regulatory signals, we applied MITRA-Dyad. When scoring candidate composite signals, we considered any pair of instances that occurred within 50 bases in the same order. We ranked the pairs of patterns by how many of their instances satisfied this conditions and broke ties by computing the sum of the components SP-STAR scores. The top three ranking composite patterns were in fact the URS1-UASH signal.

5.5 Conclusion

In 1984 Waterman et al. [26] presented 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 [23]). In this paper we have presented MITRA, a new motif finding algorithm that uses the same idea but bypasses the ex-

cessive memory requirements of SDA. Our main contribution is the use of a new approach to pruning the search space which improves over previous algorithms. We apply the algorithm to discovering composite signals and present a new algorithm for discovering dyad signals, MITRA-Dyad.

The MITRA algorithn can be extended in several ways. MITRA can easily be extended to handle insertions and deletions in addition to mismatches. For MITRA-Count, this can be done using the similar extension presented in Sagot, 1998 [23]. For MITRA-Graph, this is a little trickier, but the similar technique applies. Briefly, instead of storing the number of mismatches between two tails of instances, we instead store their minimum edit distance. We update the edit distance for a child node by doing one step of dynamic programming looking at the case if the current symbol is optimally an insertion, a deletion or a mismatch.

MITRA can also be extended to handle wildcard symbols or symbols that represent pairs of alphabet symbols in the patterns. An efficient way to represent patterns that contain wildcard symbols is to use the data structure defined in [6]. Pairs of alphabet symbols can be implemented the same way.

MITRA can also be modified to handle a more reasonable notion of number of mismatches for biased samples. In a biased sample, the current model of mismatches is inadequate because rare symbols and common symbols get treated identically. A more reasonable model would be to use a substitution matrix defined over the symbols. Instead of keeping track of mismatches, we would instead keep track of the substitution based score of the pattern compared to the instance. Instead of a maximum mismatch count, we would impose a maximal score.

Future work also involves improving the scoring functions of the algorithms to achieve more useful results from a biological perspective. The program is available from E.E. and the web server will be setup in the near future.

6 Acknowledgements

Thanks to U. Keich and S.H. Sze for useful discussions. We would like to thank J. Buhler and G. Stormo for sharing biological samples. In additon we would like to thank G. Stormo for also sharing his manuscript prior to publication.

References

[1] R. D. Anderson, S. J. Taplitz, S. Wong, G. Bristol, B. Larkin, and H. R. Hershman. Metal-dependent binding of a factor in vivo to the metal-responsive elements of the methallothionein 1 gene promoter. *Mollecular and Cell Biology*, 7:3574–3581, 1987.

[2] T. L. Bailey and C. Elkan. Unsupervised learning of multiple motifs in biopolymers using expectation maximization. *Machine Learning*, 21:51, 1995.

[3] O.G. Berg and P. H. von Hippel. Selection of DNA binding sites by regulatory proteins. *Trends Biochem Sci.*, 13(6):207–11, Jun 1998.

[4] M. Blanchette, B. Schwikowski, and M. Tompa. An exact algorithm to identify motifs in orthologous sequences from multiple species. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 37–45, 2000.

[5] J. Buhler and M. Tompa. Finding motifs using random projections. In *Proceedings of the Fifth Annual International Conference on Computational Molecular Biology (RECOMB01)*, pages 69–76, 2001.

[6] E. Eskin, W. N. Grundy, and Y. Singer. Protein family classification using sparse markov transducers. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 134–145, Menlo Park, CA, 2000. AAAI Press.

[7] D. J. Galas, M. Eggert, and M. S. Waterman. Rigorous pattern–recognition methods for DNA sequences. *Journal of Molecular Biology*, 186:117–128, 1985.

[8] D. GuhaThakurta and G. D. Stormo. Identifying target sites for cooperatively binding factors. *Bioinformatics*, 17:608–621, 2001.

[9] D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology.* Cambridge University Press, New York, 1997.

[10] G. Hertz and G. Stormo. Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. 15(7/8):563–577, 1999.

[11] U. Keich and P. A. Pevzner. Finding motifs in the twilight zone, 2001.

[12] 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*, 262:208–214, 1993.

[13] L. Marsan and M. Sagot. Alogorithms for extracting structured motifs using a suffix tree with applications to promoter and regulatory site consensus identification. *Journal of Computational Biology*, 7:345–360, 2000.

[14] C. J. McInery, J. F. Partridge, G. E. Mikesell, D. P. Creemer, and L. L. Breeden. A novel mcm1-dependent element in swi4, cln3, cdc6, cdc47 promoter activates m/g_1-specific transcription. *Genes and Development*, 11:1277–1288, 1997.

[15] A. L. Means and P. G. Farnhan. Transcription initiation from the dihydrofolate reductase is positioned by *hip1* binding at the initiation site. *Molecular and Cell Biology*, 10:653–651, 1990.

[16] S. Natsan and M. Gilman. Yyl facilitates the association of serum response factor with the c-fos serum response element. *Molecular and Cell Biology*, 15:5975–5982, 1995.

[17] A. Neuwald, J. Liu, and C. Lawrence. Gibbs motif sampling: Detection of bacterial outer membrane repeats. *Protein Science*, 4:1618–1632, 1995.

[18] G. Pavesi, G. Mauri, and G. Pesole. An algorithm for finding signals of unknown length in dna sequences. *Bioinformatics*, 17(S1):S207–S214, July 2001. Proceedings of the Ninth International Conference on Intelligent Systems for Molecular Biology.

[19] F. Pereira and Y. Singer. An efficient extension to mixture techniques for prediction and decision trees. *Machine Learning*, 36(3):183–199, 1999.

[20] P. A. Pevzner. *Computational Molecular Biology: An Algorithmic Approach*. The MIT Press, 2000.

[21] P. A. Pevzner and S. Sze. Combinatorial approaches to finding subtle signals in dna sequences. In *Proceedings of the Eighth International*

Conference on Intelligent Systems for Molecular Biology, pages 269–278, 2000.

[22] Gelfand M. S., Koonin E. V., and A. A. Mironov. Prediction of transcription regulatory sites in archaea by a comparative genomic approach. *Nucleic Acids Research*, 28(3):695–705, 2000.

[23] M. Sagot. Spelling approximate or repeated motifs using a suffix tree. *Lecture Notes in Computer Science*, 1380:111–127, 1998.

[24] S. Sze, M. S. Gelfand, and P. A. Pevzner. Finding subtle motifs in DNA sequences (submitted), 2001.

[25] J. van Helden, A. F. Rios, and J. Collado-Vides. Discovering regulatory elements in non-coding sequences by analysis of spaced dyads. *Nucleic Acids Research*, 28(8):1808–188, Apr 2000.

[26] M.S. Waterman, R. Arratia, and D.J. Galas. Pattern recognition in several sequences: consensus and alignm ent. *Bulletin of Mathematical Biology*, 46:515–527, 1984.

[27] E. Wingender, P. Dietze, H. Karas, and R. Knuppel. Transfac: a database on transcription factors and their dna binding sites. *Nucleic Acids Research*, 24(1):238–241, 1996.

[28] C.T. Workman and G.D. Stormo. ANN-Spec: A method for discovering transcription factor binding sites with improved specificity. In *Proceedings of Pacific Symposium on Biocomputing*, volume 25, pages 464–475, 1999.

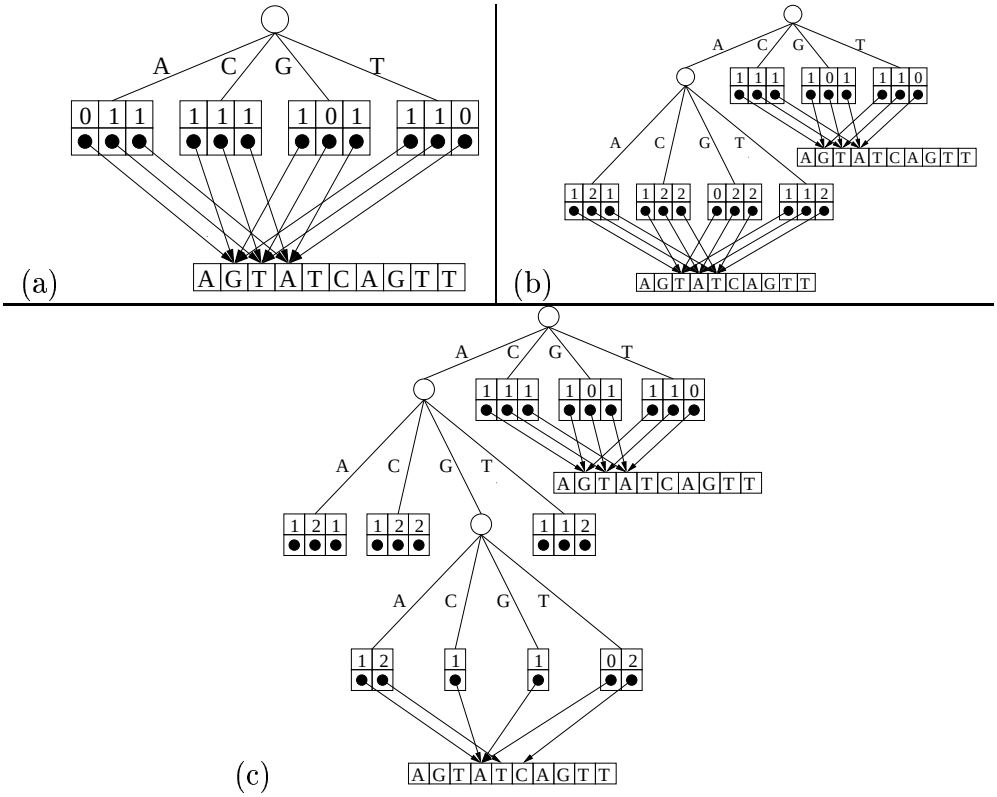

Figure 1: A Mismatch Tree with updated with $AGTATCAG$, $GTATCAGT$ and $TATCAGTT$ allowing 2 mismatches. The leaf nodes contain the number of mismatches of the prefix of a substring to the path from the root the current node and a pointer to the tail of the sequence. (a) The tree is in its initial state. (b) The tree is shown after expanding the path A. (c) The tree is shown after expanding the path AG. Notice that many of the substrings reach the maximum number of allowed mismatches and were not passed down to their children nodes. For clarity, not all of the pointers are drawn in (c).