Finding common motifs in DNA sequences: a survey

Nuno D. Mendes <ndm@algos.inesc-id.pt>

IST/UTL

Abstract

In recent years, especially after the completion of genome sequencing projects for various organisms, there has been a growing interest in the study of regulation and gene expression mechanisms. The amount of data available makes it unfeasible to pursue a manual analysis calling for some sort of automatic processing. In this context, bioinformatics tools have become more and more central to the activity of biologists.

Despite the remarkable success of these tools in some areas of application like gene finding, sequence alignment, etc, there are still problems for which no significant results have been achieved. Notably, the identification of biologically meaningful motifs in cis-regulatory regions remains an open problem.

However, many approaches have been proposed and one can find a panoply of published papers describing novel algorithms to address the problem. These algorithms can be roughly classified in two main groups: combinatorial and statistical. In this survey we will concentrate on a specific sub-group of combinatorial algorithms: algorithms based on graph theory. This interest is chiefly motivated by the fact that some recent proposals using this approach have claimed to gain efficiency by avoiding unnecessary explorations of the search space. Furthermore, we will compare their efficiency and operation with other combinatorial algorithms which use other data structures.

Keywords: cis-regulatory regions, motif finding, combinatorial algorithms, graph theory, suffix trees

1 Introduction

1.1 Some biology

There are various regulatory mechanisms for gene expression, especially in eukaryotic organisms. One of the most important mechanisms is the transcriptional regulation which is directed by proteins that specifically bind to motifs present in cis-regulatory regions. In fact, the presence of these motifs is essential for the efficient binding of the cellular transcription machinery. Different motifs can play different roles in gene expression. While some are critical for eliciting the start of transcription others recruit proteins which act as enhancers or repressors.

It follows that co-regulated genes should share some of their motifs if one is to believe that they are expressed simultaneously. It is also known that the transcription machinery will recognize binding sites even if the motifs do not occur exactly, i.e., allowing for some nucleotide substitutions. This allows us to formalize a clear but challenging problem we describe in the next section.

However, some authors have argued that the current methods to identify the very cis-regulatory regions are somewhat unreliable and that it is not always easy to confidently establish a set of co-regulated genes [1]. Most algorithms make some effort to address these issues as we will show in this paper.

So far we have implicitly referred to motifs as contiguous regions of a DNA sequence. Nonetheless, many binding sites exhibit a non-contiguous nature. This may not be a problem since we are looking for common motifs to a set of sequences regardless of whether they are structurally

related or not. However, it has been verified that the advantages of methods which search for composite motifs are twofold. On the one hand, component motifs may be too weak to be extracted in isolation, i.e., they may be poorly conserved in each sequence but may be easily identified if one looks for a composite structure and allows for more degeneration to occur. On the other hand, the imposition of a certain distance between component motifs in order to form a composite adds yet another restriction which may limit the amount of motifs extracted. This is a critical issue for algorithms that extract too many motifs and are left with the problem of deciding which of them are to be considered relevant.

1.2 The problem

Published literature uses a disparate terminology and notation to define and describe the problem of identifying binding sites in regulatory regions. We will make no attempt to establish a standard nomenclature, this has already been tried in [2] and is yet to be widely adopted.
However, to facilitate our discussion we will try to use a less formal but uniform terminology throughout the survey. The term 'motif' may cause some confusion, as it has been used freely to denote either an actual substring occurring exactly in the input sequences or the extracted pattern reported by the algorithms. We will nevertheless use it and make all efforts to disambiguate whenever we deem necessary.
We now present a very general description of a computational problem which attempts to model the biological problem described in the previous section.
Consider a set of sequences $\mathcal{S} = \{S_1, S_2, \ldots, S_t\}$. We are asked to find motifs within a range of lengths $l_{\min}, \ldots, l_{\max}$ which occur on $q \leq t$ of the presented sequences with at most e mismatches. It follows from this definition that a motif is a contiguous string that may or may not occur exactly on the given set of sequences, due to the allowed degeneration. The reason for requiring less than t occurrences of the motif is related to the fact that many input sequences may have been corrupted in the sense that they may not actually contain the motif being sought.
As we have already mentioned, it is sometimes useful to consider composite motifs which consist of two or more motifs co-occurring in non-overlapping positions at some distance of one another. We will get back to this topic during our presentation of the algorithms.

1.3 Notation

We shall refer to each contiguous string of length l on a sequence as an l-mer. Moreover, m_{ij} shall denote the l-mer starting at position i of sequence S_j, $S_j[i]$ denotes the ith character in sequence S_j and n_j denotes the length of sequence S_j.
Given two motifs of the same length l, the function $\delta : \Sigma^l \times \Sigma^l \mapsto \mathbb{N}$ denotes the Hamming distance between those two motifs.
At all times we will be assuming $\Sigma = \{A, T, G, C\}$.

2 The Algorithms

2.1 An early graph-based approach: WINNOWER

2.1.1 Description

The WINNOWER algorithm [3] sets out to solve a restricted version of the problem we described in section 1.2. The problem consists of finding all motifs of length l occurring on all t sequences with at most e mismatches.
It begins by building a graph $G = (V, E)$, where each vertex corresponds to an l-mer in the input sequences. More precisely,

$$V = \{m_{ij} : 1 \leq i \leq n_j - l + 1 \wedge 1 \leq j \leq t\}$$

There is an edge between a pair of vertices m_{ij}, m_{rs} if $\delta(m_{ij}, m_{rs}) \leq 2e$ and $j \neq s$.
That is,

$$E = \{(m_{ij}, m_{rs}) : m_{ij}, m_{rs} \in V \wedge \delta(m_{ij}, m_{rs}) \leq 2e \wedge j \neq s\}$$

Note also that if

$$\delta(m_{ij}, m_{rs}) = d$$

then

$$\delta(m_{i+1,j}, m_{r+1,s}) = d - x_{ijrs} + x_{i+l,j,r+l,s}$$

where

$$x_{ijrs} = \begin{cases} 0 & \text{if } S_j[i] = S_s[r] \\ 1 & \text{otherwise} \end{cases}$$

This observation leads to an $O(N^2)$ algorithm for building the graph, where

$$N = \sum_{i=1}^{t} n_i$$

This approach implicitly defines a t-partite graph, each partition consisting of vertices generated from different input sequences. A k-partite graph is a graph where one can establish k disjoint sets of vertices (partitions) where there are no edges between vertices in the same set. This will be important below.
It is easy to see that a motif that occurs in k sequences with at most e mismatches will be represented by a k-clique in G, i.e., a set of k fully connected vertices.
This can be easily seen if we observe that given two l-mers m_{ij} and m_{rs}, there is a pattern p of length l such that $\delta(m_{ij}, p) \leq e$ and $\delta(m_{rs}, p) \leq e$ iff $\delta(m_{ij}, m_{rs}) \leq 2e$ and thus, in order to have a pattern at a distance no larger than e from any number of l-mers each pair of these must not be at a distance larger than $2e$ from each other.
However, for $k > 2$, the converse proposition is not necessarily true, that is, the existence of a k-clique does not necessarily mean that there is a pattern p at a Hamming distance no larger than e from every participating vertex. For instance, consider the 4-mers ATAT, GTGT, TTTT. Each of them is at a distance no larger than 2 from each other but there is no pattern at a unitary distance from all of them.
The problem is thus reduced to identifying k-cliques in the graph for $k = t$ which corresponds to the t occurrences of a pattern each in a different input sequence. It is known that the problem of finding a maximal k-clique in a graph for $k > 2$ is NP-complete.
The authors adopt a winnowing strategy which we describe now. The algorithm uses the notion of extendable clique. A vertex u is a *neighbour* of a clique $C = \{v_1, \ldots, v_k\}$ if $\{v_1, \ldots, v_k, u\}$ is also a clique in the graph. A clique is *extendable* if it has at least one neighbour in each partition of the graph. An edge is called *spurious* if it does not belong to any extendable clique of size k. The algorithm operates by iteratively deleting spurious edges for increasing values of k hoping that at the end only extendable cliques survive. The winnowing operations will repeat until no further changes to the graph can be made.
For $k = 1$, a vertex u is a neighbour of vertex v if $(u, v) \in E$. The approach, in this case, is to delete every vertex which does not have at least $t - 1$ different neighbours in the graph. For $k = 2$, a vertex u is a neighbour of an edge (v, w) if $\{v, w, u\}$ is a triangle in the graph. Therefore, the algorithm must make sure that every edge has $t-2$ neigbours, each from a different partition of the graph.
For $k > 2$, WINNOWER relies on the observation that for any clique of size t there are $\binom{t}{k}$ extendable cliques with k vertices and thus, every edge on a t-clique necessarily belongs to at least $\binom{t-2}{k-2}$ extendable cliques of size k. Edges which belong to less than the appointed number of extendable cliques are deemed *inconsistent* and therefore deleted. For $k = 3$ this procedure leads to do the deletion of all edges which belong to less than $t - 2$ extendable cliques.

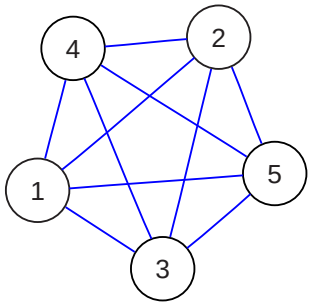

Fig. 1: A maximal 5-clique. All edges must participate in 3 different triangles

WINNOWER does not actually report any motifs. As far as the algorithm is described it is only concerned with identifying the t-cliques in the graph. As we have seen before, from the existence of these cliques it does not necessarily follow that there is a motif at a distance no larger than e from each participating vertex. Therefore, if we are interested in reporting the motif we have to analyze the clique and extract the appropriate motifs, if any.

2.1.2 Discussion

Results show that running WINNOWER for $k = 1$ is insufficient to delete all spurious edges. In many cases, the same happens for $k = 2$. Running the algorithm for $k > 3$ is prohibitively time consuming. The time complexity given by the authors is $O(t^3N^{1.5})$ for $k = 2$ and $O(t^4N^{2.66})$ for $k = 3$.
The authors have also shown that in practice the running time for $k = 3$ is very similar for the case where $k = 2$ which is justified by the fact that the edge removal condition is much stronger in the former situation requiring thus less iterations to remove spurious edges.
As we have stated, WINNOWER addresses a restricted version of our general problem, in the sense that it searches for occurrences of the motifs in t and not $q \leq t$ input sequences and refers to a fixed value for l. The first restriction can be dealt with by relaxing the edge and vertex removal conditions but the second requires us to re-run the algorithm for any given value of l.
The authors suggest a method to estimate the value of l for a given sample. They observe that the resulting graph can be of one of three kinds:

- **empty** in which case there is no motif in the conditions of the problem
- **small graph with some large cliques** in which case some motifs may have been found
- **very large graph** in which case the edge removal efforts were insufficient to identify maximal t-cliques

The authors claim that, by analyzing the outcomes of running the algorithm for a given value of the parameters e and l, one can adjust them to extract the motifs being sought.
Another important issue refers to an assumption made by the algorithm. WINNOWER assumes that the motif does not repeat in the same input sequence. If it does, as it frequently happens with real data, the algorithm will work but it will consume an unnecessary amount of space. The pre-processing stage of the algorithm may be changed in order to avoid the multiplicity of vertices generated by identical l-mers.
In addition to the substantial amount of time and space required by the algorithm, a major drawback is that despite the fact that it can solve many practical problem instances it is not guaranteed to solve any given instance. Moreover, as we have seen, the fact that a clique is found does not necessarily mean it represents a motif in the conditions of the problem, thus an additional effort must be made to determine whether we can find a motif which does not mismatch in more than e positions with each vertex in the clique.

In respect to composite motifs, WINNOWER does not address this matter directly. However, in [4], a procedure is proposed to transform a composite motif search problem into a simple motif problem. This procedure involves pre-processing the input sequences such that for each l_1-mer of the sequences and for each $d \in \{d_{\min}, \ldots, d_{\max}\}$ an $(l_1 + l_2)$-mer is generated which is the result of concatenating the l_1-mer with the l_2-mer, d characters upstream in the sequence. Although this method only allows the extraction of composite motifs composed of two simple motifs, a more complex (and potentially much more time-consuming) pre-processing stage can enable the extraction of any number of components.
Before ending our discussion of WINNOWER we must add that in [5] it was proposed a new algorithm (cWINNOWER) which adds a constraint to the graph construction procedure hoping to reduce the number of spurious edges in the initial graph. This new algorithm is based on the observation that the condition to add an edge to the graph (that the two vertices must not mismatch in more than $2e$ positions) is too permissive. The authors then follow to propose a *consensus constraint* which consists of a more stringent condition to assert whether the link under analysis can be part of a t-clique.

2.2 Avoiding the explicit search for cliques: PRUNER

2.2.1 Description

The PRUNER algorithm [6] was proposed to address some of the limitations of WINNOWER. PRUNER starts from the same graph described for the WINNOWER algorithm and then claims to explore only a small portion of the search space whereas most contemporary algorithms, the authors say, have to traverse most of the e-mismatch neighbourhood of an l-mer under analysis. The price to pay for this approach, as we will see, is incompleteness.
The rationale behind PRUNER follows an idea introduced in [3] which suggests that one may have a different approach to distinct edges in the graph with differentiated degrees of similarity between their connecting vertices.
Before describing the algorithm itself we need to introduce the notion of set of consistent patterns and make some remarks.
The set of *consistent patterns* of two l-mers m_1 and m_2, denoted $\rho(m_1, m_2)$ is defined as follows:

$$\rho(m_1, m_2) = \{m \in \Sigma^l : \delta(m, m_1) \leq e \wedge \delta(m, m_2) \leq e\}$$

From our previous presentation of WINNOWER we can rapidly observe that, regardless of the value for e considered, if $\delta(m_1, m_2) > 2e$ then $\rho(m_1, m_2) = \emptyset$.
We can now make three observations which form the basis of the algorithm:

1. For each l-mer, one needs only to explore its set of consistent patterns with respect to every other l-mer
2. The number of l-mers which are at a distance no greater than d reduces rapidly with decreasing values of d. In particular, the authors note that the average d-mismatch neighbourhood is much smaller than the average $2d$-mismatch neighbourhood
3. The size of the set of consistent patterns for two l-mers which mismatch in d positions decreases rapidly with increasing values of d

In particular, the authors show that for any two motifs in the graph m_1, m_2 of length l, such that $e < \delta(m_1, m_2) \leq 2e$, $|\rho(m_1, m_2)| \in O(l^{\frac{e}{2}}|\Sigma|^{\frac{e}{2}})$ and the size of the set is maximal when $\delta(m_1, m_2) = e + 1$, whereas if $\delta(m_1, m_2) \leq e$, $|\rho(m_1, m_2)| \in O(l^e|\Sigma|^e)$.
Thus, PRUNER divides the edges in E into two groups, defined as follows:

- Group 1
$$E_1 = \{(m_1, m_2) \in E : e < \delta(m_1, m_2) \leq 2e\}$$
- Group 2
$$E_2 = \{(m_1, m_2) \in E : \delta(m_1, m_2) \leq e\}$$

By only evaluating consistent patterns for edges in E_1, PRUNER avoids the larger size of the set of consistent patterns for edges in E_2. While in many situations the algorithm can report or discard a pattern without even looking into edges in E_2 there are cases, as we will see, for which this becomes necessary.
Before actually describing the algorithm we define the notion of *partition-degree*, denoted by the function $\gamma : V \mapsto \mathbb{N}$. If $m \in V$, then $\gamma(m)$ denotes the number of different partitions the vertex is connected to.
After building the t-partite graph PRUNER begins by removing all vertices m such that $\gamma(m) < t-1$ (not unlike WINNOWER in the case where $k = 1$) and the corresponding incident edges. It then proceeds to consider each remaining vertex at a time.
For each $m_i \in V$, PRUNER computes the set of consistent patterns $\rho(m_i, m_j)$ for every edge $(m_i, m_j) \in E_1$. Upon analysis each of these edges is deleted, and the computed patterns are added to a list $\eta(i)$ associated with m_i.
After processing each vertex PRUNER sets out to count how many partitions a pattern in $\eta(i)$ occurs in other than the partition of which m_i is a member of. This involves maintaining, for each pattern, a bit vector of length t where each position refers to a different partition.
Each pattern will have a bit set to 1, corresponding to the partition of vertex m_j from edge $(m_i, m_j) \in E_1$ which originated the pattern.
Every list $\eta(i)$ consists of patterns of equal length on a fixed alphabet Σ, so we can sort it in linear time using radix sort. The list is then scanned and the bit vectors of repeating patterns are combined to compute the number of other partitions the pattern occurs in.
Having established the referred partition count for a pattern p, $c(p)$, one of three situations can occur. Let $R = \gamma(m_i)$ be the partition-degree of m_i after processing and removing all incident edges of E_1 and let p be the pattern under scrutiny, then we do the following:

- If $c(p) + R < t - 1$ then p can be safely discarded because the partition count for p can increase by at most R, if we compare p with each m_j of the remaining edges $(m_i, m_j) \in E_2$
- If $c(p) \geq t - 1$ then p is reported since it is clear that it already occurs in $t - 1$ other partitions
- If $t - 1 \leq c(p) + R < t - 1 - R$ then we have to compare p with every vertex which is still connected to m_i (necessarily via an edge in E_2). At this stage we have to bear in mind that our pattern p is associated with a bit vector. Thus, for every vertex m_j still connected to m_i, if $\delta(p, m_j) \leq e$ we set to 1 the bit corresponding to the partition m_j is a member of. After processing all remaining edges, if the partition count is still less than $t - 1$, p is discarded, otherwise p is reported

After processing every edge in E_1 incident upon a vertex m_i, the vertex itself can be safely removed if $R < t - 1$, because no new patterns can be reported in result of the subsequent analysis of edges in E_2 we will describe next.
When all vertices have been subjected to the procedure described above we are left with a graph containing only edges in E_2. The remaining vertices, if any, are guaranteed to have a partition-degree not less than $t - 1$. The l-mers associated with these vertices are, thus, themselves valid patterns and are therefore promptly reported. However, the algorithm will not look into consistent patterns between these remaining vertices. The authors admit that there may be valid patterns in the graph that fail to be reported. PRUNER is thus correct but not complete. The authors try to mitigate this unfortunate fact by assuming that there will be fewer than t distinct valid patterns for the input sequences. This assumption does not hold true in many practical cases.
In any case, the authors claim that at least t patterns are guaranteed to be reported by the algorithm.

2.2.2 Discussion

The authors claim that PRUNER takes $O(N^2t^2l^{\frac{e}{2}})$ time and $O(Ntl^{\frac{e}{2}}|\Sigma|^{\frac{e}{2}})$ space to operate. In [6] another version of PRUNER is presented which trades time for space yielding a time complexity

in $O(N^3t^3l^{\frac{e}{2}}|\Sigma|^{\frac{e}{2}})$ and a space complexity in $O(l^{\frac{e}{2}}|\Sigma|^{\frac{e}{2}})$.
Most remarks we produced about WINNOWER are also applicable to PRUNER, in particular what we mentioned about corrupted input sequences, the assumption that a motif will not repeat on the same sequence and the search for composite motifs. The main difference is that while WINNOWER is not guaranteed to solve any particular instance of the problem, PRUNER can at least deliver a partial solution. The major drawback of this algorithm is, indeed, its incompleteness.
Another noticeable difference between PRUNER and WINNOWER refers to the fact that PRUNER explicitly reports motifs whereas WINNOWER only identifies cliques which can ultimately prove not to represent any actual motif.

2.3 An approach using suffix-trees: SMILE

2.3.1 Description

The SMILE algorithm [2] is a combinatorial algorithm which uses a completely different approach from those discussed earlier. Instead of building a graph, SMILE relies on a generalized suffix-tree [7].
A generalized suffix-tree $\mathcal{T}$ is a representation of all suffixes of a set of sequences $\mathcal{S}$. It differs from a classical suffix-tree in the sense that it uses multiple termination symbols (one for each input sequence) and stores at each node a bit vector of size t indicating the sequences in $\mathcal{S}$ to which the string labelling the path from the root to the node occurs in. This bit vector for a node v is denoted by $colors_v$.
We shall begin by presenting a simpler version of the algorithm, designed to extract simple motifs and originally described in [8] and follow to discuss its extension for the extraction of composite motifs.
To avoid the necessary distinction between referring to a character which is part of the label of an arc of $\mathcal{T}$ and the node the arc is incident upon, which actually contains all the information the algorithm will need, we will think of $\mathcal{T}$ as if it were a trie. The adaptation of this trie view to a suffix-tree view is straightforward but would make our analysis unnecessarily laborious.
Before describing the operation of the algorithm we recall that the path-label of a node v corresponds to the ordered concatenation of the labels of the arcs which form the path from the root to node v.
The algorithm operates by traversing $\mathcal{T}$. This traversal is guided by a virtual lexicographic trie $\mathcal{M}$. Thus, we perform a recursive depth-first traversal of $\mathcal{M}$ which represents all possible motifs of length l. As we go down on $\mathcal{M}$ we follow all paths in $\mathcal{T}$ for which there is still hope of finding a path-label of the appropriate length mismatching in at most e positions from the motif being spelt by our traversal of $\mathcal{M}$ and as long as the combined paths being followed represent occurrences in at least $q \leq t$ input sequences. The algorithm does this by keeping track, for each followed path (each reached node v at the current depth), of the number of mismatches endured so far. At the same time, the algorithm checks whether the combination (bitwise-OR operation) of all bit vectors $colors_v$ for each node v yields a bit-vector with at least as many bits set to 1 as the value of parameter q[†].
As soon as there is no path in $\mathcal{T}$ under the said conditions we prune the sub-trie below our current node in $\mathcal{M}$ and we backtrack both in $\mathcal{M}$ and in $\mathcal{T}$. If we are able to reach a node v in $\mathcal{M}$ at depth l and still have valid paths in $\mathcal{T}$ we report the path-label of v as a valid pattern.
At this point it is worth noting that the algorithm can easily not only extract motifs of length l but also motifs within any range of lengths $l_{\min}, \ldots, l_{\max}$. To do so it suffices that our traversal of $\mathcal{M}$ proceeds as far as depth $l_{\max}$ while reporting any pattern with at least $l_{\min}$ characters that respects the aforementioned reporting requirements.
We now proceed to describe the simplest extension for the extraction of composite motifs. The procedure to extract composite motifs using SMILE is very different from the one used by

[†] In fact, in many cases, the algorithm manages to avoid the explicit verification of the $colors_v$ bit vectors by performing some extra book-keeping we will not look into in this survey

previous algorithms because no special pre-processing of the input sequences is needed. Instead, the algorithm will jump down in the suffix-tree $\mathcal{T}$ and search for the next component motif from there, in a similar fashion to what we have described for the extraction of a simple motif.
The problem at hand when extracting composite motifs using SMILE is slightly different from the one we originally described in section 1.2 in the sense that we can specify a different range of lengths for each component motif as well as a different number of allowed mismatches. It is also possible to establish a global maximum mismatch threshold.
Now, for the sake of simplicity, let us consider the extraction of a composite motif with p components, all of length l. Note that the maximum total length of the composite motif is $pl + (p-1)d_{\max}$. Likewise its minimum total length is $pl + (p-1)d_{\min}$.
It is therefore unnecessary to keep nodes in $\mathcal{T}$ outside this range of depths. This is accomplished during the construction of $\mathcal{T}$ by not incorporating suffixes shorter than the minimum length and by only adding prefixes of suffixes not longer than the maximum length. The extraction of the first component of the composite does not differ from the extraction of a simple motif, as we have already noted. But now, once the first component is found, instead of backtracking we will jump in $\mathcal{T}$ from the nodes we are at, having followed all valid paths, to nodes deeper in the suffix-tree that the authors name *potential starts*. The potential starts, in this case, are nodes at depth $l + d_{\min}$ to $l + d_{\max}$. We will, thus, enumerate nodes at different depths depending on the allowed range of distances between the components at hand. Having jumped in $\mathcal{T}$ we shall again consider all motifs of length l, by virtually traversing another lexicographic trie, but we will now continue our traversal of $\mathcal{T}$ starting at each of the potential starts previously identified. The identification of subsequent components proceeds in a similar fashion. Having extracted a composite motif, the algorithm will then proceed by resuming the traversal of $\mathcal{M}$ and by backtracking in $\mathcal{T}$ at the level the first component was found. The algorithm can easily be extended to deal with components of variable length as it is shown in [2].
Furthermore, several improvements have been made to the algorithm we described. Also in [2], the authors have noted that the structure of a suffix-tree replicates substrings at different depths. It is then possible to search for component motifs always starting from the root of the suffix-tree if only we take the time to visit the lower levels and collect some information to change the first l levels of $\mathcal{T}$. The necessary modifications to $\mathcal{T}$ can be made efficiently by following *suffix-links*. This technique was improved further by the introduction of *box-links* [9] and eventually a new algorithm, RISO, was proposed in [10] with a substantial speedup when compared to SMILE.

2.3.2 Discussion

SMILE is in fact a family of algorithms with a similar basis, designed to solve several extensions to our original problem.
The extraction of mere simple motifs has a time complexity in $O(t^2 N \mathcal{V}(e,l))$ and space complexity in $O(t^2 N)$, where

$$\mathcal{V}(e,l) = \sum_{i=0}^{e} \binom{l}{i} (|\Sigma| - 1)^i \leq l^e |\Sigma|^e$$

is the size of the e-mismatch neighbourhood of motifs of length l.
The simplest extension for the extraction of composite motifs has a time complexity in

$$O(\underbrace{p\Delta^2 \lambda_{(p-1)l+(p-1)d_{\max}} \mathcal{V}^{p-1}(e,l)}_{(p-1)\text{first components}} + \underbrace{tp\Delta\lambda_{pl+(p-1)d_{\max}} \mathcal{V}^{p}(e,l)}_{p\text{th component}})$$

and space complexity in $O(t^2 N)$ [11], where λ_d denotes the number of nodes in $\mathcal{T}$ at depth d and $\Delta = d_{\max} - d_{\min} + 1$. The ulterior improvements to the algorithm yield even better time complexities [2, 9–11].
Unlike the algorithms we described previously, SMILE guarantees both convergence and completeness. Moreover, it is embodied with a generality which enables it to address various extensions to the basic problem.

In fact, the ability to search for motifs within a range of lengths or to consider composite motifs with variable distances between their components in a single run and without pre-processing the input sequences is one of the major advantages of SMILE when compared to previous algorithms.

2.4 Merging WINNOWER and SMILE: MITRA

2.4.1 Description

MITRA [4] is a hybrid approach to the problem of finding simple motifs common to a set of sequences. It combines the tree-like view of SMILE with the graph construction proposed for WINNOWER, taking advantage of pairwise similarities between l-mers occurring in the input sequences. Composite motifs are dealt with using the same pre-processing stage we described for WINNOWER. MITRA relies on a different data structure: a mismatch tree, which is quite similar to a trie. The mismatch tree splits the search space of all possible motifs into disjoint subspaces. These subspaces refer to motifs which start with a given prefix.
A mismatch tree $\mathcal{M}$ is a rooted tree where each internal node v has $|\Sigma|$ branches, each labelled with a different symbol of Σ. The maximum depth of $\mathcal{M}$ is l. Each node v corresponds to the subspace of motifs $\mathcal{P}$ with a fixed prefix (defined by the path-label of v) and contains a reference to all l-mers in the input sequences which are within e mismatches from a pattern $p \in \mathcal{P}$.
Initially, $\mathcal{M}$ contains only the root node which corresponds to the space of all patterns and it is recursively expanded in a depth-first manner. While examining a node of $\mathcal{M}$, MITRA will try to assert whether the corresponding subspace contains a pattern p for which there are $q \leq t$ occurrences in different input sequences with at most e mismatches. If the subspace under analysis is deemed *weak*, i.e., inconsistent with the hypothesis that there is a pattern p under the said conditions, the mismatch tree is pruned at that point and the algorithm backtracks. If the algorithm is unable to determine whether the subspace under consideration is weak, the subspace is further divided and the analysis moves down one level. If MITRA reaches depth l then the path-label of the reached node is reported since it is clearly a valid motif and the references from this node are the corresponding l-mers occurring in the input sequences.
MITRA keeps track of all valid l-mers for each node v as we have already said. Now, note that an l-mer referred from v is valid if its prefix matches the path-label of v with at most e mismatches and that the set of valid l-mers of v is a subset of the valid l-mers referred from the parent node of v. The set of valid l-mers for a node can thus be efficiently generated by keeping track of the number of mismatches between each valid l-mer and the path-label of the node. This is achieved by generating the list of valid l-mers of v at the expense of the list of the parent of v. In particular, when expanding the parent of v, each of its valid l-mers can fit into two cases. Either the position corresponding to the label of the branch to v matches an l-mer or it does not. In the former case, the l-mer is still valid for the child. In the latter, the number of mismatches increases. If the threshold e is not surpassed the l-mer remains valid, otherwise it will not be included in the list of v.
Having described the general operation of MITRA we are left with the problem of determining, for a particular node v, whether the associated subspace is weak. In [4] two alternatives are presented.
The first is a simple test to decide whether to prune a node and consists of counting the number of valid l-mers from different input sequences (there have to be more than q such l-mers for the corresponding subspace not to be deemed weak). The resulting algorithm is named MITRA-COUNT and is tantamount to the operation of SMILE when searching for simple motifs.
The second alternative relies on the construction of a graph for each visited node of $\mathcal{M}$ and is inspired by the idea behind WINNOWER. The rationale behind this approach is to take advantage of pairwise similarities between l-mers for an early decision about the weakness of a subspace under scrutiny.
This version of MITRA is called MITRA-GRAPH and will be described hereafter.
Consider a pattern p and a set of sequences $\mathcal{S}$. We construct a graph $G(p, \mathcal{S})$ where each l-mer occurring in $\mathcal{S}$ is a vertex and there is an edge connecting two l-mers if p is within e mismatches of both and they occur in two different sequences of $\mathcal{S}$. If p is under the conditions of our

problem there will be a q-clique in G. Now consider a set of patterns $\mathcal{P}$. We define $G(\mathcal{P}, \mathcal{S})$ as the graph whose set of edges is the union of the corresponding sets of every $G(p, \mathcal{S})$ having $p \in \mathcal{P}$. It is clear that its set of vertices is also generated from the l-mers present in $\mathcal{S}$ and that there will be an edge connecting two l-mers if there is a pattern $p \in \mathcal{P}$ within e mismatches of both and provided they occur in different input sequences.
If we can guarantee that there is no q-clique in this graph we can confidently determine that the corresponding subspace is weak.
There is one main difference between this approach and the way WINNOWER works: in this case we are not concerned with finding cliques but rather with determining whether they can exist at all. However, there are some resemblances. MITRA-GRAPH too will winnow edges that cannot be part of a q-clique, but it will invest lesser efforts in doing so. The winnowing strategy, which is also repeated until no more edges can be removed, is restricted to removing vertices (and respective incident edges) whose partition-degree is less than $q - 1$. If the number of remaining edges in the graph is insufficient to form a q-clique we can rule out its existence, if not, the algorithm will proceed by further dividing the subspace.
Now, if we actually needed to build a graph for each visited node in $\mathcal{M}$ this could be too much of a price to pay. Instead of always building a graph from scratch for each node v the authors take advantage of the graph built for its parent node u. Let (m_1, m_2) be an edge of the graph of u, p be the path-label of v and d be length of p. If $\delta(m_1, p) = e_1$, $\delta(m_2, p) = e_2$ and the last $l - d$ characters of m_1 and m_2 mismatch in ε positions then (m_1, m_2) will belong to the graph of v iff $e_1 \leq e$, $e_2 \leq e$ and $e_1 + e_2 + \varepsilon \leq 2e$. Note that for the root node, where $e_1 = e_2 = l - d = 0$, the condition is $\varepsilon = \delta(m_1, m_2) \leq 2e$ which is equivalent to the one used to build the graph in WINNOWER.

2.4.2 Discussion

The original description of MITRA is concerned with finding repeated motifs in a single sequence, but we have effectively described its adaptation to the problem we have been focusing on in this survey, which is concerned with finding motifs common to a set of sequences, by performing a construction of the graphs similar to the one we described in our presentation of WINNOWER.
The MITRA-COUNT algorithm is quite similar to SMILE when searching for simple motifs and their time complexities are identical for that restricted problem.
The MITRA-GRAPH algorithm is also clearly inspired by the suffix-tree traversal performed by SMILE but it will typically examine a smaller search space since it can detect earlier whether it should continue to go deeper in the tree by taking advantage of information about pairwise similarities between l-mers. It will thus visit less nodes albeit consuming more time for each visit because it needs to derive and analyze the corresponding graph.
It is certainly not easy to offer a time complexity bound for MITRA-GRAPH and the authors have presented none so we can do no better than make the above qualitative analysis.

2.5 Other approaches

Several other combinatorial approaches to this problem continue to be presented in published literature but we considered these to be the most interesting and innovative. In fact, the other combinatorial algorithms we looked into are either variations of the ones we presented [12], algorithms based on pattern enumeration [13,14] or algorithms which only solve a very restricted version of our problem [15].

3 Conclusions

In this survey we have described some of the most popular combinatorial algorithms to solve the problem of identifying common motifs in a set of sequences given some degree of allowed degeneration.

	Time	Space
WINNOWER	$O(t^4 N^{2.66})^\star$	n/a
PRUNER	$O(N^2 t^2 l^{e/2})$	$O(Ntl^{e/2}\|\Sigma\|^{e/2})$
SMILE	$O(t^2 N l^e \|\Sigma\|^e)^\dagger$	$O(t^2 N)$
MITRA-GRAPH	n/a	n/a

$\star$ for $k = 3$ $\dagger$ a tighter bound was given on section 2.3.2

Tab. 1: Comparative complexities for the presented algorithms

SMILE and MITRA-GRAPH are clearly the most interesting amongst those presented. WINNOWER has the disadvantage of not being able to guarantee it will find a solution and PRUNER is unable to guarantee completeness. SMILE is unique in its approach to composite motifs which is far superior to the pre-processing stage suggested for all other algorithms whereas MITRA-GRAPH has the virtue of combining different contributions from WINNOWER and SMILE to make an interesting use of all the information available in order to avoid unnecessary explorations of the search space. In any event, SMILE may outperform MITRA-GRAPH for smaller motifs since the overhead of processing the graphs for each node of the mismatch tree will probably be compensating only for larger values of l. On the other hand, SMILE is more sensitive to high levels of degeneration because given a value for e all nodes in $\mathcal{T}$ and $\mathcal{M}$ down to level e will always be visited.

SMILE has yet another interesting feature which is the ability to accept a flexible parameters specification, including ranges of lengths for motifs as well as variable distances between components of a composite motif, which are processed in a single run. Since we generally do not know the characteristics of the motifs being sought we cannot know *a priori* the exact parameters to use in our search. No general methods have been presented to address this issue but some exploratory approaches based on statistical analysis could be used.

From this sort of survey one would expect a thorough analysis of the effectiveness and efficiency of the presented algorithms but such an endeavour, in this case, meets several difficulties which we will now discuss.

In [16] it is said that the performance of a motif finder is determined by its reliability (how well does it find motifs?) and its complexity (at what cost?). In this context, two important questions arise. How should we formally define the reliability of a motif finder? Should we be content with worst-case time complexities?

Concerning the measure of reliability, some efforts have been made. In particular, the definition of *Challenge Problems* with planted-motifs [3, 4] which specify the characteristics of a set of random sequences where several motifs are planted with errors has triggered a community wide effort to comply with what had been proposed and most contemporary literature always compares the performance of the algorithms with respect to the challenge problems. However, little is said or known about the biological significance of the challenge problems in what concerns the identification of motifs in cis-regulatory regions. In addition, cis-regulatory regions are usually far from random. Thus, the ability to solve a challenge problem efficiently does not entail a reliable algorithm for identifying biologically relevant motifs.

Measures of reliability with respect to documented binding sites are not commonly found and will usually refer to different data sets gathered from different organisms making any comparison amongst algorithms impossible or, at the least, not very impressive.

The question of reliability is also closely connected to that of complexity. Worst-case running time bounds are usually associated with random input sequences or some other pessimistic assumptions which never occur in practice. So, the only significant way of comparing the performance of motif finders is to test them against standard input sequences, either synthetic or real, for which one knows which motifs should be expected. Anyone could, in theory, take this task upon himself but many implementations are not publicly available, therefore one is currently restricted to qualitative remarks about the algorithms one chooses to analyze.

For exact algorithms (those which are guaranteed to report all motifs in the conditions of the problem) another related question arises. How do we distinguish the biologically significant patterns from those which happen to also meet the extraction requirements, but are not known to be of any biological significance (in some cases they are the large majority of the extracted motifs)? To address this problem many authors have proposed a panoply of scoring functions. In this regard, the authors of MITRA refer to a specifiable scoring function which would further strengthen the extraction requirements, whereas the proponents of SMILE refer to a statistical analysis which scores the extracted motifs in a post-processing stage.
Another issue is how a biologist should interpret an extracted pattern which never occurs exactly in the input sequences. From a computer science perspective it is clear that such a motif is, in fact, representing its e-mismatch neighbourhood. But is it, in itself, a motif the transcription machinery would recognize? To answer this question one would need more insight into the specifics of DNA-protein interactions, in particular, into the origin of the imperfect specificity of the transcription machinery. It might as well happen that our current model is too crude or too inaccurate. In truth, the choice of the Hamming distance in detriment of any other measure of similarity is somewhat arbitrary and it is most probably founded upon arguments of simplicity or elegance. In addition, it is also not very clear whether the transcription machinery accepts mismatches uniformly across the motifs as it is implicitly assumed in the algorithms we presented.
These issues may be behind the recent interest in other, possibly more expressive, ways of defining what constitutes a pattern the transcription machinery will recognize [17].

References

[1] Solan Z, Horn D, Ruppin E, Edelman S, Lapidot M, Kaplan S, Garten Y, Pilpel Y: **Motif Extraction from Promoter Regions of *S. cerevisiae***. *ISMB 2004*. [Submitted].

[2] Marsan L, Sagot MF: **Algorithms for extracting structured motifs using a suffix tree with application to promoter and regulatory site consensus identification**. *J. Comput. Bio.* 2000, **7**(3/4):345–360.

[3] Pevzner PA, Sze SH: **Combinatorial Approaches to Finding Subtle Signals in DNA Sequences**. In *Proc. of the Eighth International Conference on Intelligent Systems for Molecular Biology* 2000:269–278.

[4] Eskin E, Pevzner PA: **Finding composite regulatory patterns in DNA sequences**. *Bioinformatics* 2002, **18**:354–363.

[5] Liang S: **cWINNOWER Algorithm for Finding Fuzzy DNA Motifs**. In *Proceedings of the CSB'03*, IEEE Computer Society 2003.

[6] Satya RV, Mukherjee A: **New Algorithms for Finding Monad Patterns in DNA Sequences**. In *Proceedings of SPIRE 2004*. Edited by Apostolico A, Melucci M, Springer-Verlag 2004:273–285.

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

[8] Sagot MF: **Spelling Approximate Repeated or Common Motifs Using a Suffix Tree**. In *Lecture Notes in Computer Science*, Volume 1380, Springer-Verlag 1998:111–127.

[9] Carvalho AM, Freitas AT, Oliveira AL, Sagot MF: **Efficient extraction of structured motifs using box links**. In *Eleventh Symposium on String Processing and Information Retrieval*, Springer 2004:267–268.

[10] Carvalho AM, Freitas AT, Oliveira AL, Sagot MF: **A highly scalable algorithm for the extraction of cis-regulatory regions**. In *Proceedings of the 3rd Asia Pacific Bioinformatics Conference*, Imperial College Press 2005:273–282.

[11] Carvalho AM: **Efficient Algorithms for Structured Motifs Extraction in DNA Sequences**. *Master's thesis*, IST/UTL 2004.

[12] Yang X, Rajapakse JC: **Graphical Approach to Weak Motif Recognition**. In *Proceedings of the 15th International Conference on Genome Informatics* 2004.

[13] Rajasekaran S, Balla S, Huang CH: **Exact Algorithms for Planted Motif Challenge Problems**. In *Proceedings of the 3rd Asia Pacific Bioinformatics Conference*, Imperial College Press 2005:249–259.

[14] Rajasekaran S, Balla S, Huang C, Thapar V: **Exact Algorithms for Motif Search**. In *Proceedings of the 3rd Asia Pacific Bioinformatics Conference*, Imperial College Press 2005:239–248.

[15] Chin FYL, Leung HCM: **Voting Algorithms for Discovering Long Motifs**. In *Proceedings of the 3rd Asia Pacific Bioinformatics Conference*, Imperial College Press 2005:261–271.

[16] Keich U, Pevzner PA: **Finding Motifs in the Twilight Zone**. In *Proceedings of RECOMB'02* 2002.

[17] Eskin E: **From Profiles to Patterns and Back Again: A Branch and Bound Algorithm for Finding Near Optimal Motif Profiles**. In *Proceedings of RECOMB'04* 2004.

Contents