Extraction of structured motifs using sequence alignment and biclustering


Joana P. Gonçalves

I. INTRODUCTION

GENE regulation mechanisms control the manufacture of functional gene products, which are involved in biological processes that occur throughout the life of a cell. The major part of these mechanisms are mediated by specific proteins, transcription factors, which regulate the expression of protein coding genes by enhancing or inhibiting their transcription into RNA. Transcription factors bind to specific sites of DNA sequences, usually short stretches of DNA with 5 to 25 nucleotides long, called transcription factor binding sites. These sites are located in the so-called promoter regions, mostly within non-coding parts of DNA. Some TFs are involved in the transcription of several genes, causing them to express coherently under specific conditions during the functioning of a cell. Genes that are regulated by common TFs consequently show similar binding sites. Thus, identifying promoter regions that resemble can provide some insights about functionally related genes and their corresponding regulatory mechanisms.

Promoter regions can be modeled as sets of conserved regions, or motifs, which occur in a well-ordered and regularly spaced manner [1]. These models are referred in literature as structured, complex or composite motifs. They are usually composed of two or more conserved regions separated by non-conserved regions of unspecified length. If composed by a single conserved region, they are usually named simple motifs, motifs, or monads. Many transcription factors bind to sites composed of two or three conserved regions, which are similarly known as dyads and triads, respectively.

Identifying structured motifs is a challenging task, both because there is no prior information about the structure of the composite motif to search, and the set of sequences being analyzed is not known to be regulated by the same transcription factor(s). Several algorithms have been proposed to address the problems of identifying and extracting structured motifs from a set of sequences. Their approaches can be divided in two main categories: probabilistic and combinatorial.

The most popular probabilistic algorithms rely on an expectation-maximization iterative approach. Expectation step identifies the most likely occurrences of a motif, according to a given model. The maximization procedure then adjusts the model to the motif occurrences. The two steps are iteratively applied until the method converges to a solution or a maximum number of iterations is reached. Convergence, however, is not guaranteed to lead to a global maximum. Moreover, expectation-maximization is very sensitive to noisy data. Although the majority of probabilistic methods has only been applied to simple motif extraction [2]–[6], MEME [7] also supports the search for structured motifs.

Combinatorial methods typically enumerate all possible patterns and search for their occurrences in the sequences. They also use a branch-and-bound approach to reduce the exponential search space. However, they are not able to discriminate relevant motifs from potentially numerous random ones which match the generated patterns. Another drawback of these algorithms is the fact that they require a large number of parameters to be specified [1], [2], [8]–[13] although some of them, like MUSA [13], are less restrictive on their values.

We present an algorithm that extracts structured motifs common to sets of sequences. It combines sequence alignment to reveal conserved regions within sequences with biclustering to further group the sequences with similar motif structures. The first step consists in the identification of candidate blocks, that is, conserved regions across subsets of sequences. We follow the approach proposed by Wang et al. in [14] that uses pairwise sequence alignment to reveal conserved fragments on each pair of sequences in the dataset. Subsequently, a greedy clustering method is applied to group conserved fragments of different sequences in order to agglomerate them by their similarity and create larger blocks. The next phase comprises a biclustering algorithm to group sequences that exhibit common block structures, i.e., similar sequences of conserved regions. The method we propose uses transformed versions of the original sequences, composed only by their candidate blocks and mines them to extract approximate sequences of blocks common to subsets of sequences.

This report is outlined as follows. Section II briefly defines the problem we are trying to address and reviews over some methods for structured motif extraction. In section III we present our approach, as well as the pseudocode of the main methods. Complexity analysis is provided in Section IV. Finally, Section V outlines some conclusions and future work.

Joana P. Gonçalves is with IST, Technical University of Lisbon, and KDBIO group of INESC-ID, Rua Alves Redol 9, Apartado 13069, 1000-029 Lisbon, Portugal. E-mail: jpg@kdbio.inesc-id.pt.
II. RELATED WORK

Several methods have been proposed to address the problem of structured motif extraction from one or multiple sequences. Helden et al. [8] derive an algorithm that finds structured motifs composed by pairs of highly conserved trinucleotides spaced by a non-conserved region of fixed width. Their approach is to generate all possible pairs of spaced trinucleotides, count their occurrences across sequences and assess their statistical significance. Although this may be an efficient methodology, it is also very restrictive on the structure of the motifs to identify, which makes it more suitable to cases when sequences are known to hold this particular kind of motif.

SMILE [12] is a combinatorial algorithm which relies on a traversal of a generalized suffix tree [15], representing all suffixes of the set of sequences, to spell all occurrences of potential components of the structured motif. Each component (simple motif) search is guided by a virtual lexicographic trie which represents all possible motifs of a given length (or range of lengths). Once a component is found, the algorithm jumps in the generalized suffix tree from current to deeper nodes, named potential starts. Potential starts are all nodes within a depth that ranges from depth of current structured motif plus the minimum and maximum spaces to consider between the last component found and its subsequent one. Once the search for a structured motif is complete, the algorithm resumes the traversal of the lexicographic trie and backtracks in the generalized suffix tree to the level where the first component was found. SMILE requires the following parameters to be specified: a range for the length of each component of the structured motif and a range for the length of the gap between each pair of consecutive components. This algorithm yields a time complexity that is exponential in the number of gaps between the components of the structured motif [1].

Co-Bind [11] models dyads with Position Weight Matrices (PWMs) and finds the PWMs that maximize the joint likelihood of occurrences of the two components. Gibbs Sampling is used to select motifs and refine the PWMs for a fixed number of times. Co-Bind may miss some motifs, since not all patterns are considered. Moreover, using a fixed number of iterations does not guarantee that the method will converge to an optimal PWM.

MITRA-dyad [10] is an algorithm that maps the problem of finding structured motifs with two components (simple motifs) separated by a non-conserved region of bounded width to a more simple problem of finding a simple motif. It combines a SMILE’s tree-like structure and the Winnower’s graph construction [9] in a hybrid approach. MITRA relies on a mismatch tree, quite similar to a trie, which groups motifs starting with the same prefix. Motifs are identified by going down on the mismatch tree, like in SMILE. At each node, the algorithm uses Winnower approach and takes advantage of pairwise similarity between the two motifs to try to prune the search space. It checks if the subspace under analysis is weak, that is, if it cannot lead to the discovery of a valid pattern. In that case, the mismatch tree is pruned and the algorithm backtracks; otherwise, the search space is further divided and analyzed. MITRA authors provide no complexity analysis. It is fair to say that this algorithm will typically examine a smaller search space than SMILE, since it can decide earlier wether it should stop its search. However, it will also consume more time visiting nodes, as it has to derive and analyze a graph for each one of them.

RISO [1] is also based on SMILE’s outline. It introduces a new data structure, box-links, which is the major contributor to the worst-case time and space exponential gain over SMILE’s performance for the problem of extracting structured motifs from a set of sequences. Box-links stores the information needed to jump from each component to another in a structured motif, so that the algorithm can skip the corresponding nodes in the suffix tree. In RISO, the (generalized) suffix tree is also substituted by a factor tree, which is actually a suffix tree pruned at a given depth. Combined, box-links and factor tree, produce a significant reduction in both time and space complexity.

Zhang and Zaki proposed ExMotif [16], an algorithm that extracts all frequent occurrences of a structured motif template that have a given quorum. The method represents each sequence by a set of sorted lists, one per each symbol. Each list contains the positions where the corresponding symbol occurs in the sequence. The list of each symbol occurrences across the set of sequences is obtained by grouping all lists of that symbol occurrences in each sequence of the set. ExMotif defines an additional operation, called positional join, that is able to build a list of occurrences of the concatenation of two sequences using the lists of the last and first symbols of the first and second given sequences, respectively, and the space width between them. Occurrences of simple motifs can be obtained by incremental positional joins on the symbols of the alphabet, until the length of the motif is reached. ExMotif uses a technique to avoid redundant computation and skip some positional joins. Structured motifs are extracted by doing positional joins on the lists of occurrences of the simple motifs. For the case of simple motifs, a space width of zero is used, since there are no gaps to consider between consecutive symbols of a simple motif. Positional joins for structured motifs consider space widths within the ranges defined in the structured motif template. ExMotif worst-case time complexity is $O(\log(m)N|\Sigma|^m + kN|\Sigma|^{km}) = O(kN|\Sigma|^{km})$, where $m$ is the length of the longest simple motif component in the structured motif template, $N$ is the total length of all sequences in the set, $|\Sigma|$ is the size of the alphabet and $k$ is the number of simple motif components in the structured motif template.

MUSA [13] is a combinatorial motif finder proposed by Mendes et al. that starts by generating all motif models of a given length composed by the symbols of the alphabet. It then constructs a matrix of co-occurrences to depict the $z$ tolerant score of the most common configuration of each pair of motif models in the set of sequences. Finally, the algorithm applies a biclustering approach to the matrix of co-occurrences in order to group the motif models and identify (complex) motifs. MUSA does not guarantee to extract all interesting correlations of motif models. It however identifies weak or uncommon patterns that would only be revealed by an exhaustive enumeration over the search space. MUSA yields a time complexity of $O(N^2 + O(|\Sigma|^{4\lambda}))$, where $N$ is the total length of the sequences in the set, $|\Sigma|$ is the length of the alphabet and $\lambda$ is the length of each motif model.

III. METHODS

Current methods for structured motif extraction require a large number of parameters to be specified, thus severely limiting the characteristics of the motif template. Since in most cases there is no prior information about the configuration of the motif to find, this can present a major problem. However, the
length of conserved regions, corresponding to binding sites, is easier to estimate than the width of non-conserved regions in-between. The method we propose tries to overcome this drawback by ignoring the space widths between the components of the structured motif. Also, when comparing to other probabilistic or combinatorial algorithms, it requires a quite reduced number of parameters to be specified. The approach we devise is based on the algorithm proposed by Wang et al. in [14], BlockMSA, which takes advantage of an interesting combination of global sequence alignment and biclustering to produce local multiple sequence alignment (MSA). Although local MSA and structured motif extraction may seem two different problems at first sight, their main goal in fact resembles, as both techniques aim at identifying local conserved regions across sets of sequences. Thus, we can say that the major difference between them resides in their output.

To clear out our problem, we first provide some definitions.

**Definition 1 (Sequence):** A sequence $S_k$ of length $|S_k|$ is an ordered list of symbols over an alphabet $\Sigma$ (with $|\Sigma|$ symbols) written contiguously from left to right.

**Definition 2 (Fragment):** For any sequence $S_k$ (with $|S_k|$ symbols), $S_k[i..j]$ $(1 \leq i < j \leq |S_k|)$ is a fragment, i.e., a contiguous subsequence, of sequence $S_k$ starting at position $i$ and ending at position $j$.

**Lemma 1:** A fragment of a sequence $S_k$ is also a sequence.

**Definition 3 (Set of sequences):** A set of sequences is an unordered collection of sequences $S = \{S_1, S_2, ..., S_n\}$ where $|S|$ is the number of sequences in the set.

**Definition 4 (k-block):** A $k$-block $B = \{f_1, f_2, ..., f_k\}$ represents an ungrepped alignment region conserved across a set of sequences $S$ with $k$ sequences and consists of $k$ equal length fragments, one from each of the $k$ sequences in $S$.

**Definition 5 (2-block):** A 2-block is a special case of a $k$-block, where $k = 2$, which represents an ungrepped pairwise alignment region conserved across a set of 2 sequences and consists of equal length fragments, one from each pairwise.

**Definition 6 (Block similarity score):** Given a $k$-block $b = \{f_1, f_2, ..., f_k\}$, the similarity score of block $B$ is the sum of the similarity scores of all the $\binom{k}{2}$ pairwise combinations of fragments $f_i, f_j$ $(1 \leq i < j \leq k)$ from $b$: 

$$\text{Score}_{b}(b) = \sum_{1 \leq i < j \leq k} \text{Score}(f_i, f_j),$$

where $\text{Score}(f_i, f_j)$ is the similarity score between fragments $f_i$ and $f_j$.

**Definition 7 (Fragment order):** Given two fragments $f_1 = (i_1, j_1)$ and $f_2 = (i_2, j_2)$ of a sequence $S$, where $i_k$ and $j_k$ correspond to the initial and final positions of fragment $k$ $(1 \leq k \leq 2)$, respectively, $f_1$ is said to be less than $f_2$ (written as $f_1 < f_2$) if and only if $j_1 < i_2$, that is, if the ending position of $f_1$ is less than the starting position of $f_2$.

**Definition 8 (Block order):** Given two $k$-blocks on a set of $k$ sequences, $F = \{f_1, f_2, ..., f_k\}$ and $G = \{g_1, g_2, ..., g_k\}$, where $f_i$ and $g_i$ represent two fragments from the same sequence $i$ in the set, block $F$ is said to be less than block $G$ (written $F < G$) if and only if $f_i < g_i$ $(1 \leq i \leq k)$.

**Definition 9 (Non-overlapping blocks):** Given two $k$-blocks, $F$ and $G$, on a set of $k$ sequences, $F$ and $G$ are said to be non-overlapping if and only if $F < G$ or $G < F$, that is, if $F$ is less than $G$ or $G$ is less than $F$.

**Definition 10 (Chain):** A set of $k$-blocks on a set of $k$ sequences, $B = \{b_1, b_2, ..., b_n\}$, where each $b_i$ $(1 \leq i \leq n)$ is a $k$-block, is called a chain if, for all pairwise combinations of blocks, $(b_i, b_j)$ $(1 \leq i < j \leq n)$, $b_i$ and $b_j$ are non-overlapping blocks.

**Definition 11 (Chain similarity score):** The similarity score of a chain of $k$-blocks, $C = \{b_1, b_2, ..., b_n\}$, where each $b_i$ $(1 \leq i \leq n)$ is a $k$-block, is defined as the sum of the scores of all its $k$-blocks minus the gap penalties between them: 

$$\text{Score}_{chain}(C) = \sum_{1 \leq i < n} \text{Score}_{b}(b_i) - \text{Gap}(b_i, b_{i+1}) + \text{Score}_{b}(b_n).$$

**Definition 12 (Structured motif):** Given a set of chains on a given set of $k$ sequences, $\text{Chains} = \{C_1, C_2, ..., C_n\}$, where each $C_i$ $(1 \leq i \leq n)$ is a chain, a structured motif is the chain $C_i$ with the highest chain similarity score over the set of chains. Given these definitions, our problem is generally defined as identifying structured motifs, that is, optimal ordered sets of non-overlapping blocks composed of fragments that occur approximately, in subsets of a set of sequences. To address this problem, we define three more specific problems:

1. Identification of candidate blocks (local conserved regions within subsets of sequences).
2. Grouping of subsets of sequences which exhibit a similar structure of candidate blocks.
3. Extraction of the structured motif for each subset of sequences.

### A. Identification of candidate blocks

The first consists in the identification of local conserved regions, also called candidate blocks. These blocks are regions that show a high degree of conservation across subsets of the input sequences. More formally, the problem is defined as follows. Consider a set of sequences, $S = \{S_1, S_2, ..., S_N\}$ a length threshold $k$ and a 2-block similarity score threshold $s$. Our goal is to find structured motifs, that is, optimal ordered sets of non-overlapping blocks common to subsets of sequences in $S$. As in BlockMSA, the method uses an external sequence alignment program to generate a library of pairwise alignments for all combinations of sequence pairs. For each pairwise alignment, a window of length $k$ is slid through it. Each position of the window delimits a $k$-length sub-region, composed of $k$-length fragments from two distinct sequences, which is kept if it holds two properties: being an ungrepped sub-region and having a pairwise similarity score greater than a given threshold $s$. Each kept sub-region is a block, i.e., a local conserved region between two sequences and its ungrepped alignment provides the similarity score between the two fragments it contains. Additionally, the similarity score between all fragments not appearing in the blocks is set to zero. Pairwise blocks are then used to construct candidate blocks, which are potentially larger blocks composed of fragments from of more than two sequences. The algorithm initially chooses some seed fragments from the two closest sequences. Each seed fragment constitutes a cluster. A greedy clustering-based approach is then used to iteratively add more fragments, from distinct sequences, to each cluster. A fragment is added to a cluster if it has the maximum sum of similarity scores within the fragments already in the cluster, that is, if it shows the highest local conservation consistency. Further details on the identification of candidate blocks is available in [14].

### B. Biclustering step

The second step is the one which actually differentiates our method from BlockMSA. Both algorithms use biclustering to group sequences that exhibit similar groups of candidate blocks.
However, both the mapping of the information provided by candidate block identification to the biclustering problem and the approach which leads to bicluster extraction are quite different.

BlockMSA defines a matrix of binary elements, whose rows and columns represent candidate blocks and sequences, respectively. Each element of the matrix is set to 1 if the block in the corresponding row contains a fragment from the sequence of its corresponding column, otherwise it is set to 0. BiMax [17] biclustering algorithm is then applied to the binary matrix. It uses a divide-and-conquer approach to extract all biclusters composed of 1s than are not entirely contained in any other bicluster (inclusion-maximal biclusters). BiMax tries to prune the search space by identifying areas that contain only 0s and consequently excluding them from further inspection. At each step, three submatrices are identified. One composed of only 0s, which is then disregarded, and two other that are directly delimited by the borders of the first one. The algorithm is recursively applied to these two remaining submatrices. Recursion ends when a matrix can be no further divided, meaning it contains only 1s and represents a bicluster. Details on BiMax algorithm are provided by the authors in [17].

In order to present our biclustering step approach, new definitions arise.

**Definition 13 (e-neighborhood):** The e-neighborhood of a sequence \( S_k \) of length \( |S_k| \), defined over an alphabet \( \Sigma \) with \( |\Sigma| \) symbols, \( N(e,S_k) \), is defined as the set of sequences \( S \), such that: for each sequence \( S_i \) \( (1 \leq i \leq |S|) \) in \( S \), \( |S_k| = |S_i| \) and \( Hamming(S_k, S_i) \leq e \), where \( e \) is an integer such that \( e \geq 0 \). This means that the Hamming distance between \( S_k \) and \( S_i \) is no more than \( e \), that is, we need at most \( e \) operations to obtain \( S_i \) from \( S_k \) and vice-versa. Three operations are considered: substitutions, insertions and deletions.

**Definition 14 (e-motif):** An e-motif is a sequence \( m \) of length \( |m| \) which occurs exactly or within an e-neighborhood in all sequences of a given set of sequences \( S \) at potentially different starting positions.

**Definition 15 (e-motif bicluster):** An e-motif bicluster is an e-motif \( m \), a subset of sequences \( S = \{s_1, s_2, ..., s_k\} \) from a set \( S (|s| \leq |S|) \) and a set of initial and final positions \( p = \{(i_1, f_1), (i_2, f_2), ..., (i_k, f_k)\} \), such that each pair of initial and final positions, \( (i_x, f_x) \) \( (1 \leq x \leq k) \), identifies an exact or approximate (in the e-neighborhood) occurrence of e-motif \( m \) in sequence \( s_x \).

**Definition 16 (Sequence-maximal e-motif bicluster):** An e-motif bicluster is sequence-maximal if it cannot be added more sequences while maintaining the property referred in 15.

**Definition 17 (Left/Right-maximal e-motif bicluster):** An e-motif bicluster is left/right-maximal if its e-motif cannot be extended to the left/right by adding a symbol of the alphabet to its beginning/end without changing its subset of sequences.

**Definition 18 (Maximal e-motif bicluster):** An e-motif bicluster is maximal if it is sequence-maximal, left-maximal and right-maximal.

We then propose the following mapping from candidate blocks and sequences to the biclustering problem: each sequence is rewritten as the ordered set of candidate blocks it contains, disregarding the non-conserved regions in-between. In this case, the alphabet of the new set of sequences contains the identifiers of all candidate blocks revealed in the previous step. Consequently, the number of symbols in the alphabet corresponds to the number of candidate blocks.

Considering the new set of sequences, the problem of finding groups of sequences with a common group of candidate blocks, addressed by the biclustering step, is then defined as the problem of identifying e-motifs across subsets of the new set of sequences. This formulation differs from the one in BlockMSA in two essential points: the order of blocks matters and we introduce an error threshold on the groups of blocks that are common to the subsets of sequences. Despite the fact that the order of blocks is granted a priori by the process of candidate block construction, since fragment consistency is obtained through global sequence alignment, having this property applied to the biclustering problem allows us to restrict it to the identification of contiguous subsequences, in which we allow a limited number of errors: e-motifs.

The algorithm we present for step two, e-Motif-Biclustering, relies its outline and implementation on e-CCC-Biclustering [18] that efficiently extracts biclusters with approximate patterns over a set of \( N \) sequences. e-CCC-Biclustering bases itself on another method, SPELLER [19], for extracting approximate motifs from a set of \( N \) sequences using a generalized suffix tree [15]. We will not provide the definitions of suffix tree, generalized suffix tree, edge, edge-label, edge-length, string-depth, string-label or suffix link. Complete details on these concepts are provided in [18] and they also apply to this report.

e-Motif-Biclustering thus identifies and reports all maximal e-motif biclusters from a set of \( N \) sequences. Since e-motifs exact versions may not be present in the sequences, maybe only approximate ones, an e-motif is seen as an "external" object and denoted by the term model. In order to be considered a valid model, a given model \( m \) of length \(|m|\) has to verify the quorum constraint. This is to say that its exact or approximate versions, words in its e-neighborhood \( (N(e,m)) \), must occur in at least \( q \) distinct sequences, where \( 2 \leq q \leq N \). This is similar to the common motifs problem, as defined by Sagot in [19]: given a set of \( N \) sequences \( S_i (2 \leq N_i) \) and two integers \( e \geq 0 \) and \( 2 \leq q \leq N \), where \( e \) is the maximum number of errors allowed per each e-motif occurrence and \( q \) the required quorum, find all models \( m \) that appear in at least \( q \) distinct sequences of \( S_i \).

In order to solve this problem, as in [19] and [18], e-Motif-Biclustering starts by building a generalized suffix tree \( T \) for the set of sequences \( S_i \) and then, after some further preprocessing, uses this tree to spell all valid models. Valid models verify two properties [19]:

1. All prefixes of a valid model are also valid models.
2. When \( e = 0 \), spelling a model leads to one node \( v \) in \( T \) such that \( L(v) \geq q \). When \( e > 0 \), spelling a model leads to a set of nodes \( v_1, ..., v_k \) in \( T \) for which \( \sum_{j=1}^k L(v_j) \geq q \).

In this setting, and since the occurrences of a valid model are, in fact, nodes of the generalized suffix tree \( T \), they are called node-occurrences [19]. Following SPELLER approach, e-CCC-Biclustering and e-Motif-Biclustering identify valid models by extending them in the generalized suffix tree and reporting their corresponding set of node-occurrences. We present an adaptation of the definition of node-occurrence used in e-CCC-Biclustering, since this algorithm only considers substitutions and we also want to allow for insertion and deletion operations. Node-occurrences are then defined as follows [18]:

**Definition 19 (Node-occurrence):** A node-occurrence of a model \( m \) is a triple \((v,v_{err},p)\), where \( v \) is a node in the
generalized suffix tree $T$, $v_{err}$ is the number of operations needed to transform $m$ into the string-label of node $v$ and $p \geq 0$ identifies a position in the generalized suffix tree such that:

1) If $p = 0$ we are exactly at node $v$.

2) If $p > 0$ we are in $E(v)$, the edge between $father_v$ and $v$, in a point between two symbols in $label(E(v))$ such that:

\[ 1 \leq p \leq |label(E(v))| \]

$\epsilon$-Motif-Biclustering relies on three Lemmas which support valid extensions of a model $m$ to a new model $m'$ by concatenating model $m$ with a symbol $\alpha$ ($\alpha \in \Sigma$). These Lemmas are adapted from e-CCC-Biclustering after considering the broader set of edition operations (substitutions, insertions and deletions):

**Lemma 2:** $(v, v_{err}, 0)$ is a node-occurrence of model $m' = m\alpha$, if and only if:

- Deletion $(v, v_{err} - 1, 0)$ is a node-occurrence of $m$.
- Match $(father_v, v_{err}, 0)$ is a node-occurrence of $m$ and $label(E(v)) = \alpha$ or $(father_v, v_{err}, |label(E(v))| - 1)$ is a node-occurrence of $m$ and $label(E(v))[|label(E(v))|] = \alpha$.
- Substitution $(father_v, v_{err} - 1, 0)$ is a node-occurrence of $m$ and $label(E(v)) = \beta$ ($\beta \neq \alpha$) or $(v_{err} - 1, |label(E(v))| - 1)$ is a node-occurrence of $m$ and $label(E(v))[|label(E(v))|] = \beta$ ($\beta \neq \alpha$).
- Insertion $(father_v, v_{err} - 1, 0)$ is a node-occurrence of $m$ and $|label(E(v))| = 1$ or $(v_{err} - 1, |label(E(v))| - 1)$ is a node-occurrence of $m$ and $|label(E(v))| > 1$.

**Lemma 3:** $(v, v_{err}, 1)$ is a node-occurrence of a model $m' = m\alpha$, if and only if:

- Deletion $(v, v_{err} - 1, 1)$ is a node-occurrence of $m$.
- Match $(father_v, v_{err}, 0)$ is a node-occurrence of $m$ and $label(E(v))[1] = \alpha$.
- Substitution $(father_v, v_{err} - 1, 0)$ is a node-occurrence of $m$ and $label(E(v))[1] = \beta$ ($\beta \neq \alpha$).
- Insertion $(father_v, v_{err} - 1, 0)$ is a node-occurrence of $m$ and $|label(E(v))| > 1$.

**Lemma 4:** $(v, v_{err}, p)$, where $2 \leq p < |label(E(v))|$, is a node-occurrence of a model $m' = m\alpha$, if and only if:

- Deletion $(v, v_{err} - 1, p)$ is a node-occurrence of $m$.
- Match $(v, v_{err}, p - 1)$ is a node-occurrence of $m$ and $label(E(v))[p] = \alpha$.
- Substitution $(v, v_{err} - 1, p - 1)$ is a node-occurrence of $m$ and $label(E(v))[p] = \beta$ ($\beta \neq \alpha$).
- Insertion $(v, v_{err} - 1, p - 1)$ is a node-occurrence of $m$.

The adaptation of SPELLER algorithm for the e-CCC-Biclustering is able to extract all right-maximal e-CCC-Biclusters from a transformed matrix, which is constructed by appending to each element (symbol) of the original discrete matrix the number of the corresponding column. This transformation is needed when restricting the occurrences of an $\epsilon$-motif to the same initial and final positions in the sequences, as it is the case of e-CCC-Biclustering algorithm. In $\epsilon$-Motif-Biclustering, however, we do not use a matrix-like structure, since sequences may have variable lengths. Moreover, we are interested in all $\epsilon$-motif occurrences, which may appear at distinct locations in the sequences. This is the reason why we eliminate the transformation of the alphabet from $\epsilon$-Motif-Biclustering.

In this setting, given a set of sequences, $S_i$, of size $N$ and two integers, $e \geq 0$ and $1 \leq q \leq N$, what we want to find is the set of all maximal models, $e$-motifs, that occur in at least $q$ distinct sequences of $S_i$. Together, each model $m$ and the set of its node-occurrences identify an $\epsilon$-motif bicluster.

Algorithm 1 unfolds the necessary steps to identify and report all maximal $\epsilon$-motif biclusters, $b_k$, such that each $b_k$ has at least $q_s$ sequences and the length of its $\epsilon$-motif is at least $q_m$. It is easily adapted to identify and report all maximal $\epsilon$-motif biclusters without quorum constraints by setting both values $q_s$ and $q_m$ to one. It holds three stages:

1) Computes all models corresponding to sequence-maximal and right-maximal $\epsilon$-motif biclusters. Uses a set of sequences, a generalized suffix tree and an adapted version of both computeRightMaximalBiclusters and spellModels procedures of e-CCC-Biclustering [18] and SPELLER [19] algorithms, respectively.

2) Deletes models which do not identify left-maximal $\epsilon$-motif biclusters. Uses all models corresponding to sequence-maximal and right-maximal $\epsilon$-motif biclusters computed in the previous stage and a trie.

3) Deletes models which represent the same $\epsilon$-motif bicluster. Uses all models corresponding to maximal $\epsilon$-motif biclusters computed in previous stage and an hashtable.

**Algorithm 1:** $\epsilon$-Motif-Biclustering: Algorithm to find and report all maximal $\epsilon$-motif biclusters

**Input:** matrix, $\Sigma$, $e$, $q_s$, $q_m$

1. modelsOcc ← {} 
2. computeRightMaximalBiclusters($\Sigma$, $e$, $q_s$, $q_m$, $\{S_1, ..., S_N\}$, modelsOcc) 
3. deleteNonLeftMaximalBiclusters(modelsOcc) 
4. if $e > 0$ then 
5. deleteRepeatedBiclusters(modelsOcc) 
6. reportMaximalBiclusters(modelsOcc)

1) Computing models corresponding to sequence-maximal and right-maximal $\epsilon$-motif biclusters: In this step we compute all valid models $m$ together with their node-occurrences $Occ_m$ corresponding to sequence-maximal and right-maximal $\epsilon$-motif biclusters.

Results are stored in a list, modelsOcc, which contains triples $(m, seqOcc_m, numberOfSeqOcc_m)$, where $m$ is the model, seqOcc_m is an array of bits signaling the distinct sequences in the node-occurrences of $m$, Occ_m, and numberOfSeqOcc is the number of bits set to 1 in seqOcc, which corresponds to the number of sequences where model $m$ occurs.

This information is computed using the procedure in Algorithm 2, similar to its homonymous in [18], which is able to:

- Use a generalized suffix tree $T_{right}$ and define node-occurrences as triples $(v, v_{err}, p)$, where $p$ is used to specify whether we are at node $v$ ($p = 0$) or in an edge $E(v)$ between nodes $v$ and $father_v$ ($p > 0$).
- Check if a valid model $m$ corresponds to a right-maximal $\epsilon$-motif bicluster. This is performed by checkRightMaximality [18] inside keepModel consequently in spellModels. It removes from the list of stored models, modelsOcc, all valid models $m$ which can be extended with a symbol $\alpha$ and whose resulting model, $\alpha m$, satisfies two properties: being also a valid model and having a set of node-occurrences, $Occ_{\alpha m}$ with as many sequences as the set of node-occurrences of its father model $m$, $Occ_m$. In this case $m$ no longer corresponds to
Algorithm 2: computeRightMaximalBiclusters procedure
Input: $\Sigma, e, qs, qm$
1 $T_{right} \leftarrow$ constructGeneralizedSuffixTree($\{S_1, ..., S_N\}$)
2 addNumberOfSeqs($T_{right}$)
3 if $e = 0$ then
4   addColorArray($T_{right}$)
5 $m \leftarrow ""$
6 $length_m = 0$
7 $father_m \leftarrow ""$
8 numberOfSeqs($father_m$) $\leftarrow 0$
9 $Occ_m \leftarrow \{\}$
10 addNodeOccurrence($\text{root}(T_{right}), 0, 0)$
11 $Ext_m \leftarrow \{\}$
12 if $e = 0$ then
13   foreach edge $E(v)$ leaving node root($T_{right}$) to node $v$
14     if $\text{label}(E(v))[1]$ is not a string terminator then
15       addSymbol($\text{label}(E(v))[1], Ext_m$)
16   else
17     foreach symbol $\alpha$ in $\Sigma$ do
18       addSymbol($\alpha, Ext_m$)
19       numberOfSeqs($father_m$) $\leftarrow 0$
20   spellModels($\Sigma, e, qs, qm, Occ_m, T_{right}, m, length_m, Occ_m, Ext_m, father_m, numberOfSeqs(father_m)$)

Algorithm 3: spellModels procedure
Input: $\Sigma, e, qs, qm, modelsOcc, T_{right}, m, length_m, Occ_m, Ext_m, father_m, numberOfSeqs(father_m)$
1 keepModel($qs, qm, modelsOcc, T_{right}, m, length_m, Occ_m, father_m, numberOfSeqs(father_m)$)
2 if $length_m < \text{maximum Model Length}$ then
3   foreach symbol $\alpha$ in $Ext_m$ do
4     maxSeq $\leftarrow 0$
5     minSeq $\leftarrow \infty$
6     if $\alpha$ is not a string terminator then
7       Colors$\alpha \leftarrow \{\}$
8     if $e > 0$ then
9       for $i$ from 1 to $N$ do
10          Colors$\alpha[i] \leftarrow 0$
11     $Ext_m \leftarrow \{\}$
12     $Occ_m \leftarrow \{\}$
13     foreach node-occ $(v, v_{err}, p)$ in $Occ_m$ do
14       removeNodeOccurrence($v, v_{err}, p, Occ_m$)
15     if $p = 0$ then
16       extendModelFromNode($T_{right}, (v, v_{err}, p, \alpha, Occ_m, Occ_m, Colors_{\alpha}, Ext_m, \text{max Seq, min Seq}, 0)$)
17     else
18       extendModelFromBranch($T_{right}, (v, v_{err}, p, \alpha, Occ_m, Occ_m, Colors_{\alpha}, Ext_m, \text{max Seq, min Seq}, 0)$)
19     if modelHasQuorum($\text{max Seq, min Seq}, Colors_{\alpha}, q_s$) then
20       spellModels($\Sigma, e, qs, qm, modelsOcc, T_{right}, ma, length_m + 1, Occ_m, Ext_m, m, numberOfSeqsOcc$)

a right-maximal $e$-motif bicluster, since its $e$-motif can be extended to the right with $\alpha$ without losing sequences.

Algorithm 3 presents spellModels, the procedure which actually spells valid models by traversing the generalized suffix tree and identifying node-occurrences. We will not provide further details on procedures that describe very specific tasks, as modelHasQuorum, or whose adaptation involves minor changes over its homonymous from [18], as keepModel.

Extensions of valid models, through the addition of a symbol, are performed by extendModelFromNode and extendModelFromBranch. Pseudocode for both procedures is given in Algorithms 4 and 5, respectively. The first one extends a valid model from a given node, while the latter applies to the case where the extension follows from a position inside an edge (branch), while descending from one node to another in the generalized suffix tree.

Two additional procedures complete the set of operations needed to spell valid models: tryExtension and extendModel. tryExtension (Algorithm 6) tries to extend a given model $m$ with a symbol $\alpha$, having $\beta$ as the actual symbol occurrence in the tree. tryExtension is also given the node-occurrence from which the model has to be extended, $(v, v_{err}, p)$, and the position it will potentially reach in the tree, as the pair $son, p_{son}$. It considers four distinct extension operations: match, substitution, deletion and insertion. These only affect the error calculation procedure, findMinimumError, as one can see in Algorithm 7.

removeChildrenNodeOcss recursively removes all node-occurrences from $Occ_m$ which identify a position in the tree below than the one given by the pair $son, p_{son}$.

Extension is finally accomplished by extendModel. Algorithm 8 shows how this procedure updates the relevant information for subsequent model extensions. It also calls either extendFromNode or extendFromBranch procedure, when further extensions have to be considered.

2) Deleting models not corresponding to left-maximal $e$-motif biclusters: Step two of $e$-Motif-Biclustering removes from the models stored in $modelsOcc$ those that do not correspond to left-maximal $e$-motif biclusters. Models are removed from $modelsOcc$ by first building a trie with the reverse patterns of all (right-maximal) models $m$ and storing the number of sequences in $numberOfSeqOcc_m$ its corresponding node [18]. A non left-maximal node is then any node in the trie that has at least one child with as many sequences as itself. Non left-maximal nodes are marked doing a depth-first traversal of the trie and computing, for each node, the maximum value among all its children [18]. Models whose corresponding node in the trie is marked as non left-maximal are finally removed from $modelsOcc$.

3) Deleting models representing the same $e$-motif biclusters: When approximate occurrences of a model are allowed, as in the case of $e$-Motif-Biclustering, distinct valid models may identify the same $e$-motif bicluster. Following the approach outlined by Madeira and Oliveira in [18], procedure deleteRepeatedBicluster uses an hashtable to remove all models that, although maximal, correspond to repeated biclusters. However, this procedure has not yet been outlined nor implemented for $e$-Motif-Biclustering.
Algorithm 4: extendModelFromNode procedure

Input: $T_{right}, (v, v_{err}, p), \alpha, Occm, Occma, Colorsma, Extma, maxSeq, minSeq, level

1 if level = 0 then
2 if $v_{err} < e$ then
3 extendModel($T_{right}, \alpha, Occm, Occma,$
4 $(v, v_{err} + 1, p), Colorsma, Extma, maxSeq,$
5 $minSeq, level, false)$
4 if $v$ is an internal node then
5 foreach child son of node $v$ do
6 if label($E(son))$[1] is not a string terminator then
7 if |label($E(son))$| > 1 then
8 $p_{son} ← 1$
9 else
10 $p_{son} ← 0$
11 tryExtension($T_{right}, (v, v_{err}, p), son, p_{son}, \beta, \alpha,$
12 $Occm, Occma, Colorsma, Extma, maxSeq,$
13 $minSeq, level)$

Algorithm 5: extendModelFromBranch procedure

Input: $T_{right}, (v, v_{err}, p), \alpha, Occm, Occma, Colorsma,$
1 maxSeq, minSeq, level

1 if level = 0 then
2 if $v_{err} < e$ then
3 extendModel($T_{right}, \alpha, Occm, Occma,$
4 $(v, v_{err} + 1, p), Colorsma, Extma, maxSeq,$
5 $minSeq, level, false)$
4 if label($E(v))$[p + 1] is not a string terminator then
5 if |label($E(v))$| > p then
6 $p_{new} = p + 1$
7 else
8 $p_{new} = 0$
9 tryExtension($T_{right}, (v, v_{err}, p), son, p_{son}, \beta, \alpha,$
10 $Occm, Occma, Colorsma, Extma, maxSeq,$
11 $minSeq, level)$

Algorithm 6: tryExtension procedure

Input: $T_{right}, (v, v_{err}, p), son, p_{son}, \beta, \alpha, Occm, Occma,$
1 Colorsma, Extma, maxSeq, minSeq, level

1 if Occm is not empty then
2 $(x, x_{err}, p_z) ← getNextNodeOccurrence(Occm)$
3 if $x = son and p_z = p_{son}$ then
4 removeNodeOccurrence($x, x_{err}, p_z), Occm$)
5 $sonerr ← findMinimumError(\minErr, v_{err} + 1,$
6 $v_{err} + 1, \beta, \alpha)$
7 if $sonerr ≤ e$ then
8 extendModel($T_{right}, \alpha, Occm, Occma,$
9 $(son, sonerr, p_{son}), Colorsma, Extma,$
10 $maxSeq, minSeq, level, true)$
11 else
12 removeChildrenNodeOccs$(Occm, son, p_{son})$
13 return
14 sonerr $← findMinimumError(\minErr, v_{err} + 1, v_{err} + 1, \beta, \alpha)$
15 $sonerr ← minSeq, level, false)$
12 extendModel($T_{right}, \alpha, Occm, Occma,$
13 $(son, sonerr, p_{son}), Colorsma, Extma,$
14 $maxSeq, minSeq, level, false)$

Algorithm 7: findMinimumError procedure

Input: matchError, substitutionError, insertionError, deletionError, \beta, \alpha

1 $\minErr ← insertionError$
2 $\minErr ← min(\minErr, deletionError)$
3 if $\beta = \alpha$ then
4 $\minErr ← min(\minErr, matchError)$
5 else
6 $\minErr ← min(\minErr, substitutionError)$
7 return $\minErr$

C. Block assembly and post-processing

$e$-Motif-Biclustering extracts all maximal $e$-motif biclusters. Each $e$-motif bicluster groups a subset of sequences which exhibit a similar sequence of candidate blocks. In this final step of the algorithm we apply the methods proposed by Wang et al. in [14] to optimize and assemble blocks. The blocks from each bicluster are extended to both sides until their score similarity falls below a predefined threshold. Blocks are also merged if they are within a relatively short distance. After extending and merging, the algorithm identifies an optimal set of non-overlapping blocks, which is actually the structured motif. Wang et al. use a single source direct acyclic graph shortest path algorithm to find the optimal chain (structured motif).

IV. Complexity analysis

The algorithm consists of three main procedures. First step, the identification of candidate blocks, takes $O(N^2a^2) + O(Nfa)$ time, where $N$ is the number of sequences, $f$ is the number of seed fragments and $a$ is the average sequence length in an unaligned region [14]. $O(N^2a^2)$ is the time for pairwise alignment and $O(Nfa)$ the time for subsequent candidate block construction. The procedure which rewrites the original sequences into sequences of candidate blocks takes $O(BN)$, where $B$ is the number of candidate blocks, by using a data structure that, for each sequence, maps each fragment starting position to the corresponding candidate block it belongs to. As new elements are inserted, this data structure guarantees that the map is always in ascending key order, that is, in ascending order of fragments starting positions.

In the biclustering step, the generalized suffix tree $T_{right}$ can be constructed in $O(L)$ using Ukkonen’s algorithm and standard data structures [18], where $L$ is the total length of the sequences in the set. Adding the number of leaves to each node in $T_{right}$ is performed by a depth-first traversal of $T_{right}$, which is linear on the number of nodes of $T_{right}$ and therefore takes $O(L)$ time. Adding the color array, a bitset of length $N$ needed when $e \geq 0$, to each of the $O(L)$ nodes in $T_{right}$ takes $O(LN)$ time. At this time, we have accounted for all operations that are executed before `computeRightMaximalBiclusters`.

All the remaining operations before procedure `spellModels`
Algorithm 8: extendModel procedure

\begin{algorithmic}
  \State \textbf{Input:} $T_{right}$, $\alpha$, $Occ_m$, $Occ_ma$, $Color_{sm}$, $(n, n_{err}, p)$, $Ext_{ma}$, $\text{maxSeq}$, $\text{minSeq}$, level $doRecursion$
  \State addNodeOccurrence\((n, n_{err}, p), Color_{sm}\)
  \State updateMaxSeqMinSeq\((\text{maxSeq}, \text{minSeq}, n)\)
  \If{$e > 0$}
    \State updateColors\((Color_{sm}, n)\)
  \EndIf
  \If{$n_{err} = e$}
    \State $doRecursion = \text{true}$
  \EndIf
  \If{$p > 0$}
    \If{label\((E(n))[p + 1]\) is not a string terminator}
      \State addSymbol\((\text{label}(E(n))[p + 1], Ext_{ma})\)
    \EndIf
    \If{$doRecursion = \text{true}$}
      \State extendModelFromBranch\((T_{right}, (n, n_{err}, p), \alpha, Occ_m, Occ_ma, Color_{sm}, Ext_{ma}, \text{maxSeq}, \text{minSeq}, \text{level} + 1)\)
    \EndIf
  \Else
    \State addAllSymbols\((\Sigma, Ext_{ma})\)
    \If{$doRecursion = \text{true}$}
      \State extendModelFromNode\((T_{right}, (n, n_{err}, p), \alpha, Occ_m, Occ_ma, Color_{sm}, Ext_{ma}, \text{maxSeq}, \text{minSeq}, \text{level} + 1)\)
    \EndIf
  \EndIf
\end{algorithmic}

Since the number of models in $modelOcc$ is $O\left(LN^fB^e\right)$ and the size of the models is bounded by $O\left(L\right)$, the trie $T_{left}$ used in the procedure for deleting models corresponding to non left-maximal biclusters can be constructed and manipulated in $O\left(LN^{1+e}B^e\right)$. Detailed description of this complexity is provided by Madeira and Oliveira.

We will not present here the complexity of deleting models that represent the same e-motif bicluster, since the method is not implemented yet. In e-CCC-Biclustering approach this step would take $O\left(LN^fB^e\right)$ time. The procedure to report all maximal e-motif biclusters is also $O\left(LN^fB^e\right)$.

Finally, the block assembly and post-processing step. Wang et al. do not provide a complexity analysis on the block merging and extending procedures. The algorithm to identify the optimal set of non-overlapping blocks, the structured motif, is $O\left(n^2\right)$ for each set of $n$ blocks in a bicluster, leading to a worst-case time of $O\left(3n^2\right)$, where $\beta$ is the number of e-motif biclusters extracted by the algorithm.

V. Conclusions and Future Work

In our approach we use the candidate block construction outlined by Wang et al. However, it is not clear to us if global pairwise sequence alignment can be used to successfully identify local conserved regions and this is, in fact, the main motivation for the existence of local alignment methods. Also the greedy clustering method used to group 2-blocks into candidate blocks does not lead to an optimal solution. e-Motif-Biclustering, the algorithm proposed for the biclustering step, can be used to identify candidate blocks if applied to the original sequences. As one may notice, each e-motif actually represents a local conserved region across a subset of sequences, i.e., an e-motif is a candidate block. e-Motif-Biclustering holds a worst-case time complexity of $O\left(L_0N_0^f\Sigma^g\right)$, where $L_0$ is the total length of the original sequences, $N$ is the number of sequences, $l_0$ is the length of the longest original sequence and $\Sigma$ is the length of the original alphabet of the sequences, $\Sigma$. Although this approach may be more costly, against the $O\left(N^2L_0^f\right)$ time proposal in [14], $f$ the number of seed fragments, it is guaranteed to find all local conserved regions within the set of sequences while the combined approach of pairwise alignment and greedy clustering is not. Moreover, e-Motif-Biclustering does not impose a restriction on the width of the conserved regions, which is a major advantage over the approach of Wang et al., considering that we do not have prior information about the configuration of conserved regions. Without this restriction, our algorithm is then able to maximize the length of each conserved region, thus avoiding an excessive and unnecessary number of overlapping candidate blocks. This leads to two other important considerations:

1) The number of identified candidate blocks will be considerably lower.

2) The post-processing operation which merges and extends overlapping blocks can be discarded.

For the biclustering step, BiMax, the biclustering algorithm used by Wang et al., has a time complexity of $O\left(BN\min(B, N)\beta\right)$, where $B$ is the number of candidate blocks, $N$ is the number of sequences and $\beta$ is the number of biclusters. Our approach takes $O\left(L_0N_0^fL_r^fB^e\right)$ time, where $L_r$ is the total length of the rewritten sequences and $l_r$ the length of the longest sequence. In this setting BiMax does not allow errors and does not consider the order of the candidate blocks. It should also be noticed that $L \leq BN$, since e-Motif-Biclustering uses a rewritten version of the sequences instead of a complete matrix, thus automatically ignoring the elements which would be 0s in BiMax approach. Moreover, a given block appears at most once in each sequence, which leads to the $BN$ bound when every block occurs in all sequences.

Wang et al. use a block assembly step to merge and extend the groups of blocks for each subset of sequences and an additional post-processing operation to find the structured motifs. This final
step cannot be avoided in this approach, since the extraction of candidate blocks does not maximize the length of each conserved region, allowing for overlapping blocks. However, as we mentioned before, the method to identify pairwise conserved regions and construct candidate blocks can be substituted for the extraction of e-motifs, which will be then considered as candidate blocks by subsequent steps. In this new setting, it is unnecessary to merge and extend blocks, since they are maximal. As we eliminate these steps, it is also possible to eliminate the search for an optimal set of blocks, which would only be required if the configuration of the blocks changed. Since we are not interested in progressively aligning the sequences, as in [14], we can have overlapping subsets of sequences across the several e-motifs, that is, we do not need the subsets of sequences to be disjoint.

As future work, it would be interesting to derive a full implementation of the methods proposed in this report and analyze their results when applied to real data. One should also investigate and develop the idea of applying the proposed e-motif extraction method also to the problem of identifying candidate blocks and not only to the biclustering step. In fact, the approach we derive in this report can be generalized to use any method for the extraction of approximate (simple) motifs in multiple sequences. The application of these methods to the identification of candidate blocks is straightforward, as we have mentioned before. Their use for the extraction of structured motifs requires only that the sequences are rewritten as the sequences of their corresponding candidate blocks, after the first step, thus mapping the structured motif problem into an approximate (simple) motif extraction problem.

REFERENCES

This document is a report which describes the work developed by the author during the Applied Algorithms course of the Ph.D Programme in Information Systems and Computer Engineering at Instituto Superior Técnico, Technical University of Lisbon, under the supervision of Professor Arlindo L. Oliveira.

ACKNOWLEDGMENT

The author would like to thank Professor Arlindo L. Oliveira for the ideas and supervision of this work and Sara C. Madeira, who very kindly provided the source code, pseudocode and complexity analysis details of the e-CCC-Biclustering algorithm.


