EFFICIENT BICLUSTERING ALGORITHMS FOR IDENTIFYING TRANSCRIPTIONAL REGULATION RELATIONSHIPS USING TIME SERIES GENE EXPRESSION DATA

SARA C. MADEIRA
Knowledge Discovery and BIOinformatics (KDBIO) team of INESC-ID, Lisbon
IST, Technical University of Lisbon
University of Beira Interior, Covilhã
Portugal
E-mail: smadeira@di.ubi.pt

JOANA P. GONÇALVES
Knowledge Discovery and BIOinformatics (KDBIO) team of INESC-ID, Lisbon
IST, Technical University of Lisbon
Portugal
E-mail: joana.spg@gmail.com

ARLINDO L. OLIVEIRA
Knowledge Discovery and BIOinformatics (KDBIO) team of INESC-ID, Lisbon
IST, Technical University of Lisbon
E-mail: aml@inesc-id.pt

Biclustering algorithms have shown to be remarkably effective in a variety of applications. Although the biclustering problem is known to be NP-complete, in the particular case of time series gene expression data analysis, efficient and complete biclustering algorithms, are known and have been used to identify biologically relevant expression patterns. However, these algorithms, namely CCC-Biclustering (a linear time algorithm to identify biclusters with perfect expression patterns) and *e*-CCC-Biclustering (a polynomial time algorithm to identify biclusters with approximate expression patterns) do not take into account the importance of time-lagged and anti-correlation relationships in the study of transcription regulation using time series expression data. Furthermore, it is known that certain type of network motifs can generate temporal programs of expression, in which genes are activated one by one in a predefined order and can thus produce time-lagged expression patterns.
In this context, we proposed extensions to these state of the art algorithms to analyze time series gene expression data in order to identify time-lagged activation together with anti-correlation while dealing directly with missing values. We present preliminary results obtained by applying the extended version of the CCC-Biclustering algorithm to the transcriptomic expression patterns occurring in *Saccharomyces cerevisiae* in response to heat stress.

1. Introduction

Several non-supervised machine learning methods have been used in the analysis of gene expression data. Recently, biclustering [3], a non-supervised approach that performs simul-

taneous clustering on the row and column dimensions of the data matrix, has been shown to be remarkably effective in a variety of applications. The advantages of biclustering (when compared to clustering) in the discovery of local expression patterns have been extensively studied and documented [2, 3, 9, 13, 15]. These expression patterns can be used to identify relevant biological processes involved in regulatory mechanisms. Although, in its general form, biclustering is NP-complete, in the case of time series expression data the interesting biclusters can be restricted to those with contiguous columns leading to a tractable problem.

In this context, CCC-Biclustering [10] and *e*-CCC-Biclustering [11] are recent proposals of efficient algorithms to find and reports all maximal contiguous column coherent biclusters, with perfect (CCC-Biclusters) expression pattern in linear time, and approximate expression pattern (*e*-CCC-Biclusters) in polynomial time, respectively, using efficient string processing techniques based on suffix trees.

However, these algorithms do not take into account important aspects of gene regulation such as anti-correlation (activation and inhibition behaviors in the same biological process) and time-lagged activation. Moreover, other authors have already point out the importance of taking time-lagged relationships into consideration since from time series gene expression data it is apparent that most genes do not regulate each other simultaneously but after a certain time lag [8]. That is, the products that a gene produces during the expression process may affect other genes' expression later. Such regulation can be divided into two types: activation and inhibition. In the activation process, an increase in certain genes' expression levels will increase some other genes' expression levels after a time lag. During the inhibition process, an increase in some genes' expression levels will result in a decrease in other genes' expression levels [8].

In a recent work from Uri Alon [1] about network motifs Uri Alon [1] talks about the temporal programs of expression that are apparent in certain network motifs, in particular, Uri Alon explains that the SIM (Single Input Module) network motif (one of the four network motifs he studies in this work), can generate temporal programs of expression, in which genes are activated one by one in a predefined order. As such, it is clear that identifying biclusters in time series expression data that can identify time-lagged activation together with anti-correlation (activation and inhibition of genes within the same biclusters) is crucial to help in the discovery of network motifs by giving a step towards the understanding of the temporal programs of gene expression and the global structure of transcriptional networks.

In this context, we propose extensions to CCC-Biclustering and *e*-Biclustering algorithms, able to maintain the computational efficiency and completeness of these algorithms, but that are now able to deal directly with missing values (not possible in the previous versions of both algorithms) and discover CCC-Biclusters/*e*-CCC-Biclusters with sign-changes and time-lags. We present preliminary results obtained by applying the algorithm to the transcriptomic expression patterns occurring in *Saccharomyces cerevisiae* in response to heat stress.

The paper is organized as follows: Section 2 presents definitions needed to state the problem and construct the algorithm; Section 3 discusses related work on biclustering

in time series expression data focusing on three recent proposals q-clustering [8], CCC-Biclustering [10] and *e*-CCC-Biclustering [11] and analyzing their positive and improvable aspects; Section 4 presents the extended version of CCC-Biclustering to deal with missing values and allow the identification of CCC-Biclusters with sign-changes, CCC-Biclusters with time-lags and CCC-Biclusters with sign-changes and time-lags; Section 5 describes the changes needed to allow the same functionalities in *e*-CCC-Biclustering. Finally, Section 6 describes preliminary experimental results and Section 7 states some conclusions and outlines future work. We have also included three appendices: Appendix B contains the CCC-Biclustering algorithms needed to highlight the changes needed to extract the new types of CCC-Biclusters; Appendix C contains the CCC-Biclustering algorithm extended to deal with time-lags and finally, Appendix A describes the methodology used to score and evaluate statistically the CCC-Biclusters identified in the results section.

2. Definitions

2.1. *Strings and Suffix Trees*

This section revises basic concepts about strings and suffix trees that will be needed throughout the paper.

Definition 2.1. A **string** S is an ordered list of symbols (over an alphabet Σ) written contiguously from left to right [6]. For any string S, $S[i..j]$ is its (contiguous) **substring** starting at position i and ending at position j. The **suffix** of S that starts at position i is $S[i..|S|]$.

Definition 2.2. The ***e*-Neighborhood** of a string S of length $|S|$ ($e \geq 0$) defined over the alphabet Σ, $N(e, S)$, is the set of strings S_i, such that: $|S| = |S_i|$ and $Hamming(S, S_i) \leq e$. This means that the Hamming distance between S and S_i is no more than e, that is, we need at most e substitutions to obtain S_i from S. The e-Neighborhood of a string S contains the following number of elements: $\nu(e, |S|) = \sum_{j=0}^{e} C_j^{|S|}(|\Sigma| - 1)^j \leq |S|^e|\Sigma|^e$.

Definition 2.3. A **suffix tree** of a string S is a rooted directed tree with exactly $|S|$ leaves, numbered 1 to $|S|$. Each internal node, other than the root, has at least two children. Each edge is labeled with a nonempty substring of S (**edge-label**), and no two edges out of a node have edge-labels beginning with the same character. Its key feature is that for any leaf i, the label of the path from the root to the leaf (**path-label**) spells out the suffix of S starting at position i. Each leaf is identified by the starting position of the suffix it corresponds to.

In order to enable the construction of a suffix tree obeying this definition when one suffix of S matches a prefix of another suffix of S, a character terminator, that does not appear anywhere else in the string, is added to its end.

Definition 2.4. A **generalized suffix tree** is a suffix tree built for a set of strings S_i. Each leaf is now identified by two numbers, one identifying the string and the other the suffix.

Suffix trees (generalized suffix trees) can be built in time linear on the size of the string (sum of the sizes of the strings), using several algorithms [6]. Ukkonen's algorithm [17],

used in this work, uses *suffix links* to achieve a linear time construction. An example of a generalized suffix tree built for the set of strings that correspond to the rows of the right matrix in Figure 1 is presented in Figure 2.

Definition 2.5. There is a **suffix link** from node v to node u, (v, u), if the path-label of node u represents a suffix of the path-label of node v and the length of the path-label of u is equal to the length of the path-label of v minus 1.

2.2. *Gene Expression Data and Matrix Discretization*

Let A' be a gene expression matrix defined by its set of rows (genes), R, and its set of columns (conditions), C. In this context, A'_{ij} represents the expression level of gene i under condition j, which is usually a real value corresponding to the logarithm of the relative abundance of mRNA in gene i under condition j. Let A'_{iC} and A'_{Rj} denote row i and column j of matrix A', respectively. Moreover, consider that $|R|$ is the number of rows and $|C|$ is the number of columns in A'.

In this work, we are interested in the case where the gene expression levels in A' can be discretized to a set of symbols of interest, Σ, that represent distinct activation levels. In the simpler case, Σ may contain only two symbols, one used for *no-regulation* and other for *regulation*. Another widely used possibility, is to consider a set of three symbols, $\{D, N, U\}$, meaning *DownRegulation*, *NoChange* and *UpRegulation* In other applications, the values in matrix A' may be discretized to a larger set of symbols. After discretization, A' is transformed into matrix A and $A_{ij} \in \Sigma$ represents the discretized value of A'_{ij}.

	C1	C2	C3	C4	C5
G1	0.07	0.73	-0.54	0.45	0.25
G2	-0.34	0.46	-0.38	0.76	-0.44
G3	0.22	0.17	-0.11	0.44	-0.11
G4	0.70	0.71	-0.41	0.33	0.35

(a)

	C1	C2	C3	C4	C5
G1	N	U	D	U	N
G2	D	U	D	U	D
G3	N	N	N	U	N
G4	U	U	D	U	U

(b)

	C1	C2	C3	C4	C5
G1	N1	U2	D3	U4	N5
G2	D1	U2	D3	U4	D5
G3	N1	N2	N3	U4	N5
G4	U1	U2	D3	U4	U5

(c)

Figure 1. Toy example. (a) represents the original expression matrix, (b) the discretized matrix and (c) the discretized matrix after alphabet transformation.

Figure 1 (middle) represents a possible discretization of the expression values in the left matrix in the same figure. In this example, the alphabet $\Sigma = \{D, N, U\}$ was used and an expression level was considered as *NoChange* if it falls in the range $[-0.3, 0.3]$.

Consider now the **alphabet transformation** that consists, essentially, in appending the column number to each symbol in the matrix. This corresponds to considering a new alphabet $\Sigma' = \Sigma \times \{1, \ldots, |C|\}$, where each element Σ' is obtained by concatenating one symbol in Σ and one number in the range $\{1 \ldots |C|\}$. In order to do this we use a function $f : \Sigma \times \{1, \ldots, |C|\}$ defined by $f(a, k) = a|k$, where $a|k$ represents the character in Σ' obtained by concatenating the symbol a with the number k. For example, if $\Sigma = \{D, N, U\}$ and $|C| = 3$, then $\Sigma' = \{D1, D2, D3, N1, N2, N3, U1, U2, U3\}$. As examples, $f(D, 2) = D2$ and $f(U, 1) = U1$.

Consider also that Σ is always given in lexicographic order. The function $f_j : \Sigma \times \{j\}$ defined by $f_j(a, j) = a|j$, where $a|j$ represents the character in Σ'_j obtained by concatenating the symbol a with the number j, is used to define the possible alphabet for a specific column j. Moreover, $\Sigma'_j[p]$ is defined as the p element of Σ'_j. For instance, $\Sigma'_1 = \{D1, N1, U1\}$ is the possible set of symbols in column 1 and $\Sigma'_1[2] = N1$.

In this setting, consider also the **set of strings** $S_i = \{S_1, \ldots, S_{|R|}\}$ **obtained by mapping each row** A_{iC} **in matrix** A **to string** S_i such that $S_i[j] = f(A_{ij}, j)$. Each string S_i has exactly $|C|$ symbols which correspond to the symbols in row A_{iC}. After this transformation, the middle matrix in Figure 1 becomes the right matrix in Figure 1.

2.3. *Biclusters in Gene Expression Data*

Consider now the matrix A, corresponding to the discretized version of matrix A'. This matrix is defined by the discretized versions of the set of rows and the set of columns in A': $\{A_{iC}, 1 \leq i \leq |R|\}$ and $\{A_{Rj}, 1 \leq j \leq |C|\}$. Let $I \subseteq R$ and $J \subseteq C$ be subsets of the rows and columns, respectively. Then, $A_{IJ} = (I, J)$ is a submatrix of A that contains only the elements A_{ij} belonging to the submatrix with set of rows I and set of columns J.

Definition 2.6. A **bicluster** is a subset of rows that exhibit similar behavior across a subset of columns, and vice-versa. The bicluster A_{IJ} is thus a subset of rows and a subset of columns where $I = \{i_1, ..., i_k\}$ is a subset of the rows in R ($I \subseteq R$ and $k \leq |R|$), and $J = \{j_1, ..., j_s\}$ is a subset of the columns in C ($J \subseteq C$ and $s \leq |C|$). As such, the bicluster A_{IJ} can be defined as a k by s submatrix of matrix A.

Given this definition and a data matrix, A', or its discretized version, A, the goal of biclustering algorithms is to identify a set of biclusters $B_k = (I_k, J_k)$ such that each bicluster satisfies specific characteristics of homogeneity. These characteristics vary from approach to approach enabling the discovery of many types of biclusters by analyzing directly the values in matrix A or using its discretized version [9]. In this paper we will deal with biclusters that exhibit coherent evolutions, characterized by a specific property of the symbols in the discretized matrix. We are interested in column coherent biclusters:

Definition 2.7. A **CC-Bicluster**, column coherent bicluster, A_{IJ}, is a subset of rows $I = \{i_1, \ldots, i_k\}$ and a subset of columns $J = \{j_1, \ldots, j_s\}$ from the matrix A such that $A_{ij} = A_{lj}$, for all $i, l \in I$ and $j \in J$.

2.4. *CC-Biclusters in Time-Series Gene Expression Data*

When analyzing time series gene expression data we can restrict the attention to biclusters with contiguous columns [8, 10, 18]. This leads us to the definition of CCC-Bicluster and *e*-CCC-Bicluster, and other relevant definitions related to it (already defined in previous work [10, 11]), such as, maximal CCC-Biclusters and maximal *e*-CCC-Bicluster:

Definition 2.8. A **CCC-Bicluster**, contiguous column coherent bicluster, A_{IJ}, is a subset of rows $I = \{i_1, \ldots, i_k\}$ and a **contiguous** subset of columns $J = \{r, r+1, \ldots, s-1, s\}$ from matrix A such that $A_{ij} = A_{lj}, \forall i, l \in I$ and $j \in J$.

In this settings, each CCC-Bicluster A_{IJ} defines a string S corresponding to a contiguous **expression pattern** that is common to every row in the CCC-Bicluster, between columns r and s of matrix A. This means there exists a string $S = S_i[r...s], \forall i \in I$.

Definition 2.9. A CCC-Bicluster A_{IJ} is **trivial** if it has only one row or only one column.

Definition 2.10. A CCC-Bicluster A_{IJ} is **row-maximal** if no more rows can be added to its set of rows I while maintaining the coherence property in Def. 2.8.

Definition 2.11. A CCC-Bicluster A_{IJ} is **right-maximal** if its expression pattern S cannot be extended to the right by adding one more symbol at its end (the column contiguous to the last column of A_{IJ} cannot be added to J without removing genes from I).

Definition 2.12. A CCC-Bicluster A_{IJ} is **left-maximal** if its expression pattern S cannot be extended to the left by adding one more symbol at its beginning (the column contiguous to the first column of A_{IJ} cannot be added to J without removing genes from I).

Given the three definitions above we can intuitively say that a maximal CCC-Bicluster is a CCC-Bicluster that is row-maximal, left-maximal and right-maximal. This means that no more rows or contiguous columns (either at right or at left) can be added to it while maintaining the coherence property in Def. 2.8.

Definition 2.13. A CCC-Bicluster A_{IJ} is **maximal** if no other CCC-Bicluster exists that properly contains it, that is, if for all other CCC-Biclusters A_{LM}, $I \subseteq L$ and $J \subseteq M \Rightarrow I = L \wedge J = M$.

When we are interested in approximate expression (in the sense of allowing a certain amount of errors per gene in the expression pattern), we can now define *e*-CCC-Biclusters and maximal *e*-CCC-Biclusters:

Definition 2.14. An ***e*-CCC-Bicluster**, contiguous column coherent bicluster with e errors, A_{IJ}, is a CCC-Bicluster where all the strings S_i that define the expression patterns of each of the genes in I are in the *e*-Neighborhood of an expression pattern S that defines the e-CCC-Bicluster, that is, $S_i \in N(e, |S|), \forall i \in I$. The definition of *0*-CCC-Bicluster is equivalent to the definition of a CCC-Bicluster (Def. 2.8).

Definition 2.15. An *e*-CCC-Bicluster, A_{IJ}, is **maximal** if it is row-maximal, left-maximal and right-maximal. This means that no more rows or contiguous columns can be added to it while maintaining the coherence property in Def. 2.14.

3. Related Work on Biclustering Algorithms for Time-Series Expression Data

Although several algorithms have been proposed to address the general problem of biclustering [9], to our knowledge, only four recent proposals have addressed this problem in the specific case of time series expression data [8, 10, 11, 18].

Zhang et. al [18] proposed to modify the heuristic algorithm of Cheng and Church [3], by restricting it to add and/or remove only columns that are contiguous to the partially constructed bicluster thus forcing the resulting bicluster to have only contiguous columns.

Multiple biclusters are identified (as in the approach of Cheng and Church) by masking the biclusters found so far with random values. This method has one strong limitation, however. The greedy row and column addition and removal, that is already likely to find sub-optimal biclusters in general expression data, does not work well in time series gene expression data. In fact, the restriction imposed on the columns that can be removed makes the algorithm converge, in many cases, to a local minimum, from which it does not escape.

We will now describe in detail the CCC-Biclustering [10], *e*-CCC-Biclustering [11] since we will present extensions to both algorithms to allow the discovery of biclusters with sign changes and time-lags. We will also focus on the *q*-clustering algorithm from Ji and Tan [8] since, to our knowledge it is the state of the art algorithm for finding these type of biclusters. We will also highlight the positive aspects and drawbacks of each of the approaches.

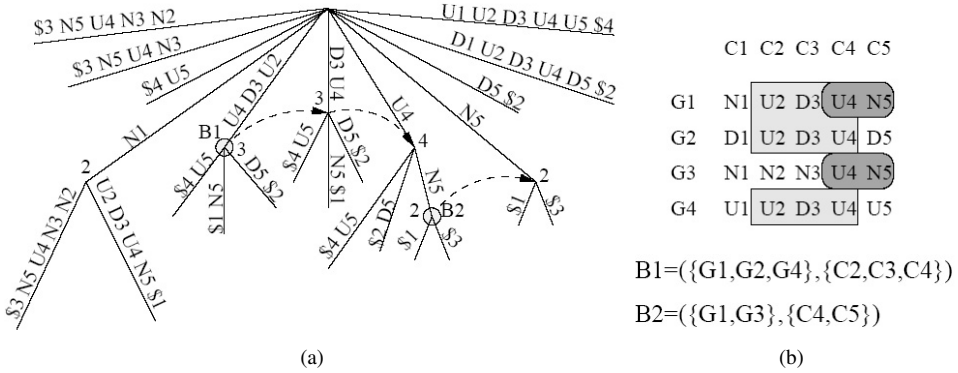
3.1. *CCC-Biclustering*

CCC-Biclustering [10], finds and reports all CCC-Biclusters in time linear on the size of the expression matrix by manipulating a discretized version A of the original matrix A' and using string processing techniques based on suffix trees. Let T be the generalized suffix tree obtained from the set of strings S obtained after the matrix transformation explained in Section 2.2. Let v be a node of T and let $P(v)$ be the **path-length** of v, that is, the number of symbols in the string that labels the path from the root to node v (**path-label**). Additionally, let $E(v)$ be the **edge-length** of v, that is the number of symbols in the edge that leads to v (**edge-label**), and $L(v)$ the number of leaves in the sub-tree rooted at v, in case v is an internal node. The CCC-Biclustering algorithm is based on the following theorem:

Theorem 3.1. *Let v be a node in the generalized suffix tree T. If v is an internal node, then v corresponds to a maximal CCC-Bicluster iff $L(v) > L(u)$ for every node u such that there is a suffix link from u to v. If v is a leaf node, then v corresponds to a maximal CCC-Bicluster iff the path-length of v, $P(v)$, is equal to $|S_i|$ and the edge-label of v has symbols other than the string terminator, that is, $E(v) > 1$. Furthermore, every maximal CCC-Bicluster in the matrix corresponds to a node v satisfying one of these conditions.*

Figure 2 illustrates that every node in the generalized suffix tree corresponds to one CCC-Bicluster. However, the rules in Theorem 3.1 have to be applied to extract only the maximal. In this case (no errors allowed) each CCC-Bicluster is *perfect*, in the sense of having no errors, and is identified by exactly one node in the suffix-tree. We will see that this is no longer true when our goal is to extract *e*-CCC-Biclusters with $e > 0$.

The **positive aspects** of CCC-Biclustering are: 1) in first place its efficiency in terms of computational complexity (in linear time by using an efficient pattern discovery algorithm based on suffix-trees); 2) in second place its completeness since it discovers of all maximal CCC-Biclusters with perfect expression patterns (in the sense of not having errors); and 3) finally allowing the use of any discretization technique.



(a) (b)

Figure 2. (a) Generalized suffix tree for the right matrix in Figure 1 used by CCC-Biclustering. The circles identify the Maximal Non-Trivial CCC-Biclusters (B1 and B2). (b) The CCC-Biclusters B1 and B2 are showed in the matrix as subsets of rows and columns (I, J). The strings [U2 D3 U4] and [U4 N5] correspond to the expression patterns of B1 and B2, respectively.

The **drawbacks** of CCC-Biclustering are 1) not considering approximate expression patterns 2) not dealing directly with missing values (the algorithm relies on a preprocessing step that either removes all genes with missing values or fills the missing values with one of several input missing values techniques such as average expression value of the gene of average of k nearest neighbors), 3) not allowing sign-changes in the same bicluster and thus ignoring potential anti-correlation between genes, and also not allowing time-lag patterns and thus ignoring time-delays in the activation of genes.

We included the CCC-Biclustering algorithm in Appendix B since it will be useful the explain the extensions proposed in the main paper.

3.2. *e-CCC-Biclustering*

e-CCC-Biclustering [11] was recently proposed by Madeira and Oliveira to overcome one of the drawbacks of CCC-Biclustering: not allowing the discovery of CCC-Biclusters with approximate patterns. This efficient algorithm is also complete and reports in polynomial time all maximal *e*-CCC-Biclusters. To achieve its goal *e*-CCC-Biclustering uses an adaptation of SPELLER [14] (an algorithm that extracts common motifs from a set of N sequences using a generalized suffix tree), together with two additional steps to achieve its goal.

Madeira and Oliveira [11] showed that SPELLER can be adapted to extract all right-maximal *e*-CCC-Biclusters from the transformed matrix A. Figure 3(a) shows the generalized suffix tree used by SPELLER and also by the first step of *e*-CCC-Biclustering $e = 1$ and two maximal *1*-CCC-Biclusters (B1 and B2). It is also possible to observe these *1*-CCC-biclusters in the matrices in Figure 3(b).

Since the adaptation of SPELLER used in the first step of the algorithm can output *e*-

CCC-Biclusters that are not left-maximal and also repeated e-CCC-biclusters, the e-CCC-Biclustering algorithm proposed by Madeira and Oliveira has three main steps (see [11] for details): STEP 1) uses an adaptation of SPELLER to extract all right maximal e-CCC-Biclusters; STEP 2) uses a trie to remove all right-maximal e-CCC-Biclusters not corresponding to left-maximal e-CCC-Biclusters and finally STEP 3) uses an hash tree to remove from the maximal e-CCC-Biclusters extracted in the previous steps those that correspond to repeated e-CCC-biclusters.

The asymptotic complexity of e-CCC-Biclustering is $O(|R|^2|C|^{1+e}|\Sigma|^e)$, when general errors are allowed (that is, any symbol in the alphabet can be replaced by one of the others). This complexity can be reduced to$O(|R|^2|C|^{1+e}|\Sigma^{Rest}|^e)$, when only restricted errors are allowed, where Σ^{Rest} is the restricted alphabet (see [11] for details).

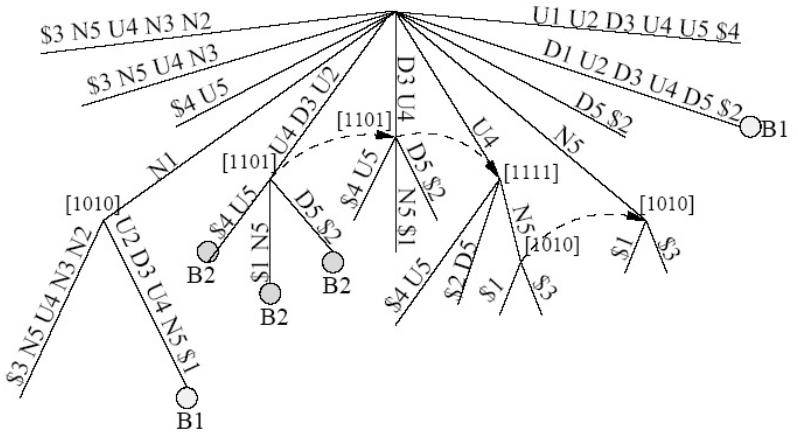
The **positive aspects** of e-CCC-Biclustering are: 1) in first place the ability to discover biclusters containing genes with approximate expression patterns; 2) in second place, its polynomial complexity for approximate pattern matching it is still optimal given the complexity of the problem; 3) in third place its completeness since it also discovers of all maximal e-CCC-Biclusters with perfect/approximate expression patterns; and 4) finally allowing the use of any discretization technique.

The **drawbacks** of e-CCC-Biclustering are 1) not dealing directly with missing values although a missing value could in this case be easily considered as one error (the algorithm relies on a preprocessing step that either removes all genes with missing values or fills the missing values with one of several input missing values techniques such as average expression value of the gene of average of k nearest neighbors), 2) not allowing opposite patterns nor time-lag patterns.

3.3. *q-Clustering*

Ji and Tan [8] proposed the q-clustering algorithm that works with a discretized data. As CCC-Biclustering [10], they are also interested in identifying biclusters formed by consecutive columns called q-clusters. Therefore, if appropriately implemented, their idea would generate exactly the same biclusters as the ones generated by CCC-Biclustering [10] together with a set of time-lagged gene clusters that will correspond to the CCC-Biclusters with sign-changes and time-lags that the proposed extensions to CCC-Biclustering and e-CCC-Biclustering (that will be defined in the next chapters) will enable to extract in a more simple and efficient way and that will be defined in the next chapters.

However, they propose to use a naive algorithm that, in the implementation made available by the authors [7], requires time and space exponential on the number of columns, when applied to the generation of all contiguous columns coherent biclusters. In practice, it cannot be applied to generate biclusters with more than 10 or 11 time-points. As such direct comparison with this algorithm is not possible. Moreover we will show in the next chapters that the extensions proposed for CCC-Biclustering and e-CCC-Biclustering together with a statistical evaluation of the extracted patterns outperform the approaches of Ji and Tan both in terms of computational complexity and potentially interesting biological results.



(a)

	C1	C2	C3	C4	C5
G1	N1	U2	D3	U4	N5
G2	D1	U2	D3	U4	D5
G3	N1	N2	N3	U4	N5
G4	U1	U2	D3	U4	U5

	C1	C2	C3	C4	C5
G1	N1	U2	D3	U4	N5
G2	D1	U2	D3	U4	D5
G3	N1	N2	N3	U4	N5
G4	U1	U2	D3	U4	U5

B1=({G1,G2},{C1,C2,C3,C4,C5}) B2=({G1,G2,G4},{C2,C3,C4,C5})

(b)

Figure 3. (a) Generalized suffix tree for the right matrix in Figure 1 used in *e*-CCC-Biclustering when $e > 0$ (when $e = 0$ the suffix tree used is in Figure 2). The circles B1 and B2 identify two *1*-CCC-Biclusters (b) Maximal *1*-CCC-Bicluster corresponding to the valid model/motif m = [D1 U2 D3 U4 N5] (left) and maximal *1*-CCC-Bicluster corresponding to the valid model/motif m = [U2 D3 U4 D5] (right). Note that these two 1-CCC-Biclusters correspond, respectively, to two and three node-occurrences (B1 and B2) showed in the generalized suffix tree above.

The q-Clustering algorithm proposed by Ji and Tan [8] has three phases.

In phase 1, the original expression matrix is transformed into a "slope" matrix to reflect changing tendency of the genes over time using a three symbol alphabet $\Sigma = \{-1, 0, 1\}$. Phase 2, generates q-clusters that contain information about genes with a similar pattern over (any) q consecutive conditions. Finally, in phase 3, biclusters are extracted from each q-cluster and between q-clusters [8].

After the discretization performed in phase 1, phase 2 generates a set of q-clusters. Each q-cluster has the following property: all genes in the cluster have the same expression pattern over some q consecutive time points. As such, the authors aim at finding genes that share similar subsequences of length $(q - 1)$, where q is a user-defined parameter. Each q-cluster has a unique identifier, called its *q-clusterID*. The q-clusters are generated

as follows: for each row (gene) in the "slope" matrix, the authors apply a sliding window of length $(q-1)$. As each $(q-1)$ substring is examined, its *q-clusterID* is determined and the $(geneID, st)$ pairs are inserted into the corresponding q-cluster, where $geneID$ is the identifier of the gene and st is the position of the starting point of the sliding window that identifies the $(q-1)$ character substring. With an appropriate implementation, all q-clusters (with q in the range of 2 to $|C|$) can be extracted in $O(|R||C|^2)$.

Phase 3 has four tasks. The first task extracts biclusters from the q-clusters. The $(geneID, st)$ pairs in each q-cluster are first sorted by starting position, so that all $(GeneID, st)$ pairs with the same starting position st are grouped together. This step directly identifies the biclusters contained in each q-cluster, since in a q-cluster all genes with the same starting position share the same pattern under the same q conditions. The exact complexity of this phase is not easy to estimate, but will be at least $O(|R||C|^2)$, since that is the number of potential q-clusters. Although each q-cluster can have up to $O(|R|)$ elements, it may be possible to prove a tighter upper bound than the obvious $O(|R|^2|C|^2)$ for this approach. Note that, despite the optimistic complexity of $O(|R||C|^2)$ for the execution of q-clustering till this step, the CCC-Biclustering [10] algorithm is $\Theta(|C|)$ faster than the possible best implementation of the q-clustering algorithm, producing exactly the same results (when the same discretization technique is used).

Phase 3 also performs some additional computations in order to extract potential regulations from the identified biclusters and allow approximate expression patterns. We will show in the next chapter that CCC-Biclustering can be adapted to extract the same regulatory properties when we are interested in perfect expression patterns and the same applies for *e*-CCC-Biclustering when we are interested in approximate expression patterns.

The task two performed in phase 3 deals with gene relationships within a q-cluster by comparing the starting positions of biclusters obtained from the q-cluster. Since biclusters (within a q-cluster) with different starting positions share the same pattern, there is a promising time-lagged activation co-regulation relationship between these biclusters [8]. In particular, and according to Jin and Tan, given two biclusters, the one with the smaller starting position is a potential activator of the bicluster with the larger starting position. The time lag between the two activations is given by the difference in the starting positions. The authors consider that two possible relationships that need further biological study: (1) only a certain gene in one bicluster individually activates another gene in the other bicluster; (2) all or most of the genes in a bicluster collectively activate some (or all) genes in the other bicluster. Following the same idea, task three attempts to find inhibition regulations. To do so they start by finding a pair of q-clusters with opposite patterns. Such a pair of q-clusters is, according to the authors, a promising inhibition pair (The authors consider that two patterns are opposite to one another if corresponding elements between the two patterns are either both '0' or else '1' and '-1', respectively). According to their reasoning, genes/biclusters of one q-cluster with a smaller start position may inhibit genes/biclusters of the other q-cluster with a larger start position [8]. Finally, in task four, Ji and Tan handle approximate matching. They consider that similar/opposite patterns with only one or two exceptional elements may still be regarded as interesting by some users and deal with this

problem as follows: for each q-cluster, they allow changes to be made to certain positions of the pattern. The corresponding q-cluster with the changed pattern is a potential candidate for co-regulation. For inhibition regulation, find the q-cluster that has an opposite pattern from the changed pattern.

The total complexity of task 4, used by Ji and Tan to generate time-lagged co-regulated relationships between genes/genes clusters is not easy to estimate if all the tasks are performed simultaneously, although we estimate optimistically that it can never be less than $O(|R|^2|C|^2)$ when approximate matching is not carried out and would definitely be much higher in this case. We once again highlight, that the implementation of the q-clustering made available by the authors [7], requires time and space exponential on the number of columns, when applied to the generation of all possible q-clusters (equivalent to CCC-Biclusters discovered by CCC-Biclustering) even without going further to the discovery of activation/inhibition relationships between the the q-clusters.

The **positive aspects** of q-clustering (if appropriately implemented) are: 1) ability to extract biclusters with sign-changes and time-lags although indirectly by combining previously discovered q-clusters.

The **drawbacks** of q-clustering are 1) not dealing directly with missing values removing genes with missing values as a preprocessing step, 2) relying on a particular three symbol discretization matrix; 3) high computational complexity and not allowing opposite patterns nor time-lag patterns; 4) scoring of biclusters is performed using the mean squared residue score [3] when statistical scores could be used directly to evaluate the significance of the patterns (see next chapters for details); 5) filtering step based on the amount of symbols '0' found in the pattern in an attempt to filter patterns with poor changing tendency in terms of expression patterns (once again a statistical significance measure could be in the filtering step); 6) approximate matching is not performed directly and appears here as a pos-processing step that can be used to add more genes to the bicluster; 7) the length of the expression patterns discovered is limited by the value of the parameter q.

4. Extended CCC-Biclustering

4.1. *Dealing with missing values*

Consider now a modification of the toy example presented in Figure 1 where some of the values in the for genes are missing and a fifth gene was added:

	C1	C2	C3	C4	C5
G1	-	0.73	-0.54	0.45	0.25
G2	-0.34	0.46	-	0.76	-
G3	-	-	-	0.44	-0.11
G4	0.70	-	-0.41	0.33	0.35
G5	-0.34	-	0.73	-0.41	-0.54

(a)

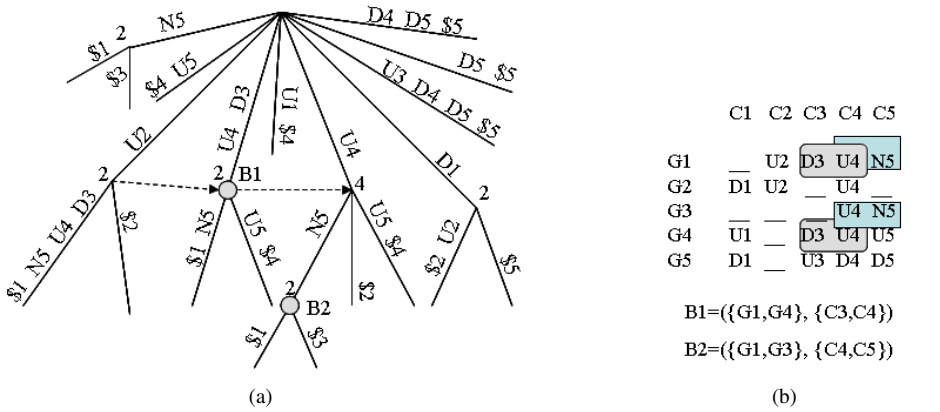
	C1	C2	C3	C4	C5
G1	-	U	D	U	N
G2	D	U	-	U	D
G3	-	-	-	U	N
G4	U	-	D	U	U
G5	D	-	U	D	D

(b)

	C1	C2	C3	C4	C5
G1	-	U2	D3	U4	N5
G2	D1	U2	-	U4	D5
G3	-	-	-	U4	N5
G4	U1	-	D3	U4	U5
G5	D1	-	U3	D4	D5

(c)

Figure 4. Extended Toy Example. (a) represents the original expression matrix, (b) the discretized matrix and (c) the discretized matrix after alphabet transformation.



(a) (b)

Figure 5. (a) Generalized suffix tree for the right matrix in Figure 4 used by CCC-Biclustering extended to deal with missing values. The circles identify the Maximal Non-Trivial CCC-Biclusters (B1 and B2). (b) The CCC-Biclusters B1 and B2 are showed in the matrix as subsets of rows and columns (I, J). The strings [D3 U4] and [U4 N5] correspond to the expression patterns of B1 and B2, respectively.

In order to deal with missing values directly in the CCC-Biclustering algorithm one just have to change the way the strings S_i corresponding to each of the rows are inserted in the suffix tree. For example, the string corresponding to gene G2 in the toy example in Figure 1 would be S_2 = [D1 U2 D3 U4 D5 \$2], \$2 is the string terminator identifying gene G2. Now looking at the extended toy example in Figure 4, and in order to jump the missing values that gene G2 now has, there will be inserted two strings for gene G2 (one for each contiguous set of symbols without missing values) with the same string terminator: S_{2_1} = [D1 U2 \$2] and S_{2_2} = [U4 \$2]. Figure 5 shows the suffix tree used by CCC-Biclustering and the maximal non-trivial CCC-Biclusters extracted by dealing directly with the missing values in the algorithm. The complexity of the algorithm remains $O|R||C|$.

There are only two changes relatively to the original CCC-Biclustering algorithm 1 transcribed for clarity in Appendix B:

(1) in line 1, the mapping of each row A_{iC} to a string S_i using function f: $S_i \leftarrow f(A_{iC}), i \in \{1, \ldots, |R|\}$ must be performed as explained above in order to jump the gaps. That is: each S_i is now split in a set of r substrings of S_i, $\{S_{i_1}, \ldots, S_{i_r}\}$, one for each set of contiguous symbols without missing values. Note that all the substrings of S_i are inserted in the suffix tree T using the same terminator.

(2) in line 14, $P(v)! = |S_i|$ should be replaced by $P(v)! = |S_{i_r}|$, since a leaf might now correspond to a maximal CCC-Bicluster if it corresponds to one of the substrings of row i (if row i had missing values) and not only to the entire row i as was required before.

4.2. *Discovering CCC-Biclusters with sign-changes*

Before presenting the extensions needed in CCC-Biclustering to allow the discovery of CCC-Biclusters with sign changes we first define formally this type of bicluster (already

classified using this terminology in the biclustering survey of Madeira and Oliveira [9]):

Definition 4.1. A **CCC-Bicluster with sign-changes**, contiguous column coherent bicluster with sign-changes, A_{IJ}, is a subset of rows $I = \{i_1, \ldots, i_k\}$ and a **contiguous** subset of columns $J = \{r, r+1, \ldots, s-1, s\}$ from matrix A such that $A_{ij} = A_{lj}$ **or** $A_{ij} = A_{lj}^{-1}, \forall i, l \in I$ and $j \in J$, where A_{lj}^{-1} is the opposite pattern of A_{lj}.

After the necessary changes (explained below) to allow the discovery of this type of patterns the asymptotic complexity of CCC-Biclustering remains $O(|R||C|)$ if we do not care about the fact of generating repeated CCC-Biclusters. If we want to avoid the extraction of repetitions we can easily do so by using an hash table to store the CCC-Biclusters extracted (same genes, same first and last column means same CCC-Bicluster). This will, however, increase the complexity to $O(|R|^2|C|)$.

In order to extract CCC-Bicluster with sign-changes we need now to build the suffix tree with the set of strings S_i but also to insert the opposite patterns of these strings: S_i^{-1}. Some other changes are needed when marking the nodes as non-valid. Lets look at the detailed changes:

(1) line 1: the mapping of each row A_{iC} to a string S_i using function f: $S_i \leftarrow f(A_{iC}), i \in \{1, \ldots, |R|\}$ must be performed as in the previous section (if we want to allow genes with missing values to be analyzed). The following step should be added: Map each row A_{iC}^{-1}, where A_{iC}^{-1} is the opposite pattern of A_{iC} to a string S_i^{-1} using function f.

(2) line 2: Build a generalized suffix tree T for the set of strings $\{S_1, \ldots, S_{|R|}\}$ and insert the opposite patterns $\{S_1^{-1}, \ldots, S_{|R|}^{-1}\}$

(3) line 11: if the internal node has only leafs whose terminators correspond to rows S_i^{-1} it should also be marked as invalid since it corresponds to a "artificial" maximal CCC-Bicluster found in the rows with opposite patterns.

4) line 14: same change as in the previous section if we want to allow genes with missing values. If the leaf terminator correspond to rows S_i^{-1} it should also be marked as invalid.

Figure 6 shows the suffix tree used by CCC-Biclustering and the maximal non-trivial CCC-Biclusters and maximal non-trivial CCC-Biclusters with sign-changes extracted in one run of the algorithm by dealing directly with the missing values in the algorithm.

4.3. *Discovering CCC-Biclusters with time-lags*

Similarly to what we did in the previous section, and before presented the extensions needed in CCC-Biclustering to allow the discovery of CCC-Biclusters with time-lags, we first define formally this type of CCC-Bicluster:

Definition 4.2. A **CCC-Bicluster with time-lags**, contiguous column coherent bicluster with time-lags, A_{IJ}, is a subset of rows $I = \{i_1, \ldots, i_k\}$ and a **contiguous** subset of columns $J = \{r, r+1, \ldots, s-1, s\}$ from matrix A such that $A_{ij} = A_{lj}$ **or** $A_{ij} = A_{l(j+lag)}, \forall i, l \in I$ and $j \in J$, where lag is the time-lag and can vary from 1 to $|C| - 1$.

Note that we can chose to fix the value of the time-lag, lag to a particular value of interest or extract all CCC-Bicluster with time-lags independently of the time-lag if we want

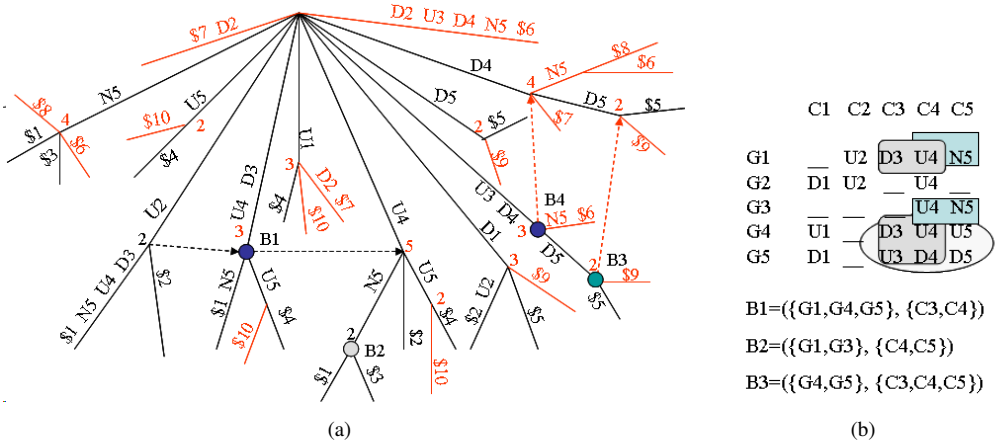


Figure 6. (a) Generalized suffix tree for the right matrix in Figure 4 used by CCC-Biclustering extended to deal with missing values and identify CCC-Biclusters with sign-changes. The circles identify the Maximal Non-Trivial CCC-Biclusters (B1, B2, B3 and B4). Since B2 and B4 correspond to the same CCC-Bicluster with sign-changes B4 is eliminated (b) The CCC-Biclusters B1, B2 and B3 are shown in the matrix as subsets of rows and columns (I, J).

the algorithm to be exhaustive. We present here the exhaustive version of the algorithm which enables us to maintain the completeness of CCC- Biclustering.

Figure 7 shows the suffix tree used by CCC-Biclustering and the maximal non- trivial maximal CCC-Biclusters and maximal CCC-Biclusters with time-lags extracted in one run of the algorithm by dealing directly with the missing values in the algorithm.

Since in order to extract CCC-Biclusters with time-lags are a lot more than in the previous versions we present the modified algorithm in Appendix C.

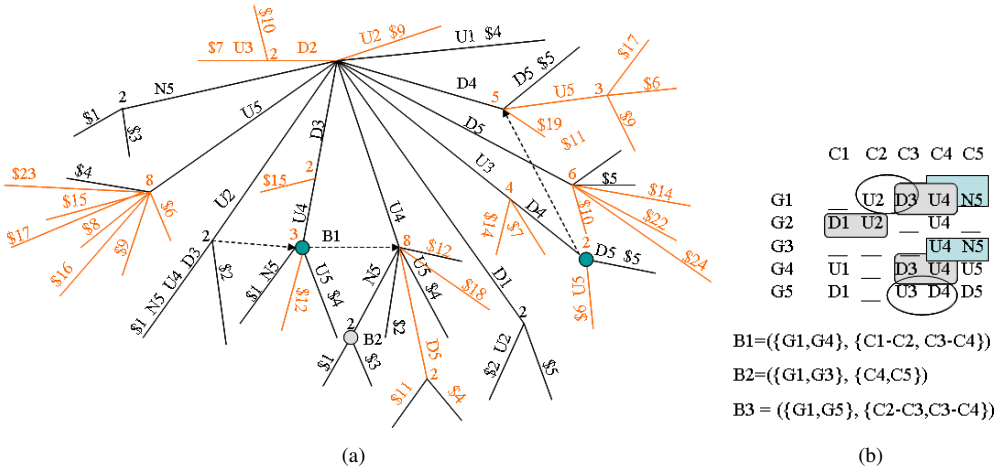
4.4. *Putting all together: Discovering CCC-Biclusters with sign-changes and time-lags while dealing with missing values when necessary*

The final step is to allow CCC-Biclusters to have simultaneously sign-changes and time-lags. Figure 8 shows the results obtained using the matrix in the extended toy example in Figure 4.

We now define formally what is a CCC-Bicluster with sign-changes and time-lags:

Definition 4.3. A **CCC-Bicluster with sign-changes and time-lags**, contiguous column coherent bicluster with sign-changes and time-lags, A_{IJ}, is a subset of rows $I = \{i_1, \ldots, i_k\}$ and a **contiguous** subset of columns $J = \{r, r+1, \ldots, s-1, s\}$ from matrix A such that $A_{ij} = A_{lj}$ **or** $A_{ij} = A_{lj}^{-1}$ **or** $A_{ij} = A_{l(j+lag)}$ **or** $A_{ij} = A_{l(j+lag)}^{-1}, \forall i, l \in I$ and $j \in J$, where lag is the time-lag and can vary from 1 to $|C|-1$ and A_{lj}^{-1} and $A_{l(j+lag)}^{-1}$ are the opposite pattern of A_{lj} and $A_{l(j+lag)}$, respectively.

Its easy to adapt the algorithm described in Appendix C to allow sign-changes and time-



(a) (b)

Figure 7. (a) Generalized suffix tree for the right matrix in Figure 4 used by CCC-Biclustering extended to deal with missing values and identify CCC-Biclusters with time-lags. The circles identify the Maximal Non-Trivial CCC-Biclusters (B1 and B2). (b) The CCC-Biclusters B1 and B2 are shown in the matrix as subsets of rows and columns (I, J). Note that B2 has a time-lag equal to 2 between the activation of the gene G2 and the activation of the remaining genes in the CCC-Bicluster G1 and G4. CCC-Bicluster B3 has a similar behavior since it has a time-lag equal to 1 between the activation of gene G1 and gene G5.

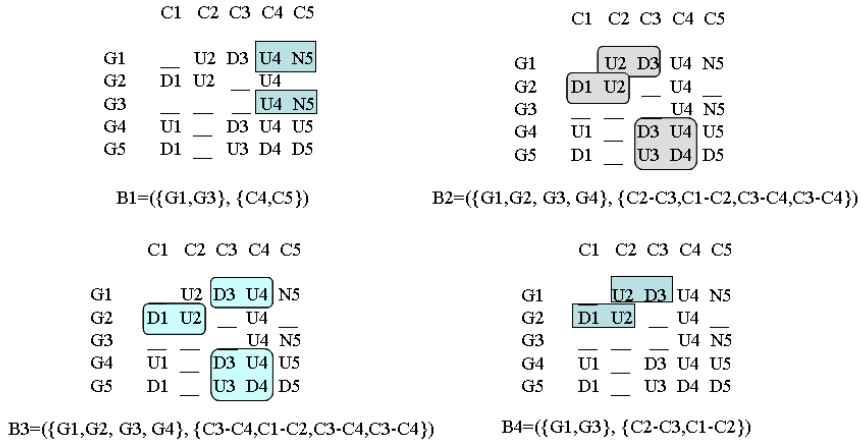


Figure 8. Maximal non-trivial CCC-Biclusters with sign-changes and/or time-lags discovered in the extended toy example in Figure 4

lags simultaneously. For lack of space we do not present it here but the key is to mark the nodes after the original matrix is added to tree as "Invalid before sign-changes", insert the matrix with opposite patterns, mark the nodes as "Invalid before time-lags" and then insert

the original matrix with time-lags and the matrix with opposite patterns in the suffix tree. After this it is only necessary to mark the nodes as "Invalid after time-lags" and report the CCC-Biclusters with sign-changes and time-lags identified by the nodes marked as "Valid after time-lags".

5. Extended *e*-CCC-Biclustering

For lack of space we do not present here the details of the extension of *e*-CCC-Biclustering to extract *e*-CCC-Biclusters with sign-changes, *e*-CCC-Biclusters with time-lags and *e*-CCC-Biclusters with sign-changes and time-lags. However this is trivially done by inserting in the suffix tree used in the first step of the algorithm the modified versions of the strings to be inserted as explained in the Section 4.

To deal directly with missing values is straightforward and can be perform in two ways: 1) consider missing values as valid errors or 2) insert the rows in the suffix tree used in the first step of the algorithm as explained in Section 4.1.

6. Experimental Results

To obtain preliminary results in real data, we used CCC-Biclustering and a dataset from Gasch et al. [4], concerning the yeast response to heat shock. This dataset comprises 5 different time-points along the first hour of exposure to 37^{o}C (0', 5', 15', 30' and 60'). The first time-point is an average of three replicates of time zero. The dataset was normalized to mean 0 and unit standard deviation and the three symbol discretization technique proposed by Ji and Tan [8] was adapted in order to be able to deal with missing values.

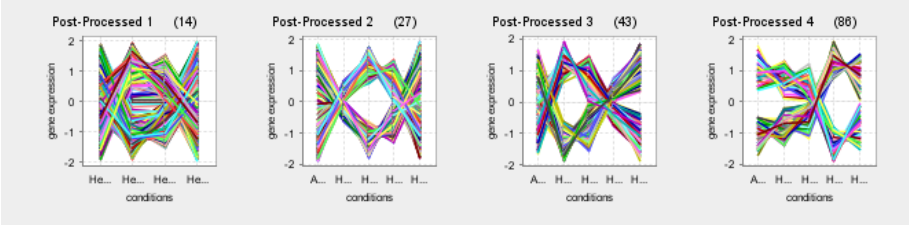
Since we were interested in CCC-Biclusters with high statistical significance (see Appendix A) the set of maximal non-trivial CCC-Biclusters discovered in each experimental setup was then sorted in ascending order according to the statistical *p*-value described in Appendix A. From these we only report those considered as highly significant at the 1% level after applying the Bonferroni correction for multiple testing. In order to avoid the analysis of highly overlapping CCC-Biclusters, we then compute the similarities between the sorted biclusters using the Jaccard similarity score also described in Appendix A, as described in Appendix A.1, and filtered CCC-Biclusters with similarity greater than 50%.

We show the expression patterns of those CCC-Biclusters surviving these strict statistically based selection criteria. In future work we will analyze the biological relevance of these expression patterns using Gene Ontology annotations together with information about transcriptional regulation available in the YEASTRACT database [16].

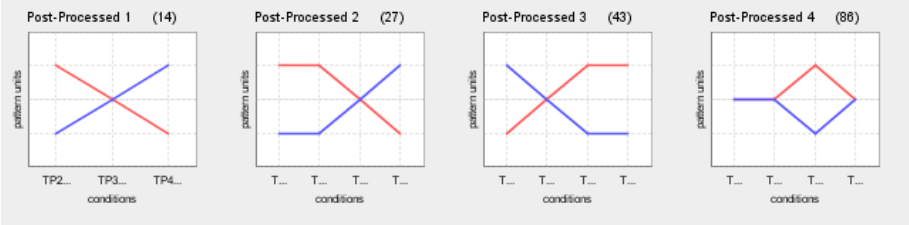
6.1. *Discovering CCC-Biclusters with sign-changes*

CCC-Biclustering as described in Section 4.2 identified 89 maximal non-trivial CCC-Biclusters. From these only 17 were considered as highly significant after the Bonferroni correction. After filtering those with similarity greater than 50% we obtained 4 CCC-Biclusters with sign-changes whose expression patterns are shown in Figure 9. These fig-

ures were produced using a software named BiGGEsTS [5] we are developing and whose first version will soon be released.



(a) Expression Values



(b) Expression Patterns

Figure 9. Top 4 CCC-Biclusters with sign-changes.

7. Conclusions and Future Work

In this work we proposed extensions to CCC-Biclustering and *e*-CCC-Biclustering algorithms in order to identify time-lagged activation together with anti-correlation while dealing directly with missing values. The proposed algorithms are still efficient (given the complexity of the problem) and complete (since they generate all maximal CCC-Biclusters with sign-changes and time-lags).

We presented preliminary results obtained by applying the extended version of the CCC-Biclustering algorithm to the transcriptomic expression patterns occurring in *Saccharomyces cerevisiae* in response to heat stress. We show the expression patterns of those CCC-Biclusters surviving strict statistically based selection criteria. In future work, we will analyze the biological relevance of these expression patterns using Gene Ontology annotations together with information about transcriptional regulation available in the YEASTRACT database [16].

As future work we plan to extend these algorithms to deal with gaps in the expression patterns. Being able to deal with gaps will certainly help to design an algorithm able to process multiple time series and extract CCC-Biclusters captured by finding coherent (but not necessarily the same) expression patterns in several time series experiments. Moreover, biclustering a compendium of problem related time series together will certainly return

more statistically and biologically relevant expression patterns by putting together more information about the same problem and also filtering noisy experiments.

Acknowledgements

This work was partially supported by projects POSI/SRI/47778/2002, BioGrid and POSI/EIA/57398/2004, DBYeast, financed by FCT, Fundação para a CiÍncia e Tecnologia, and the POSI program.

References

1. Uri Alon. Network motifs: theory and experimental approaches. *Nature Reviews Genetics*, 8(6), 2007.
2. A. Ben-Dor, B. Chor, R. Karp, and Z. Yakhini. Discovering local structure in gene expression data: The order–preserving submatrix problem. In *Proc. of the 6th International Conference on Computacional Biology*, pages 49–57, 2002.
3. Y. Cheng and G. M. Church. Biclustering of expression data. In *Proc. of the 8th International Conference on Intelligent Systems for Molecular Biology*, pages 93–103, 2000.
4. A. P. Gasch, P. T. Spellman, C. M. Kao, O. Carmel-Harel, M. B. Eisen, G. Storz, D. Botstein, and P. O. Brown. Genomic expression programs in the response of yeast cells to environmental changes. *Molecular Biology of the Cell*, 11:4241–4257, 2000.
5. J. P. Gonçalves, S. C. Madeira, and A. L. Oliveira. BiGGEsTS: BiclusterinG Gene Expression Time-Series. *Application Note (to be submitted)*, 2007.
6. D. Gusfield. *Algorithms on strings, trees, and sequences*. Computer Science and Computational Biology Series. Cambridge University Press, 1997.
7. L. Ji and K. Tan. Identifying time-lagged gene clusters using gene expression data - supplementary information. http://www.comp.nus.edu.sg/∼jiliping/p2.htm, [September 20, 2006].
8. L. Ji and K. Tan. Identifying time-lagged gene clusters using gene expression data. *Bioinformatics*, 21(4):509–516, 2005.
9. S. C. Madeira and A. L. Oliveira. Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(1):24–45, January–March 2004.
10. S. C. Madeira and A. L. Oliveira. A linear time algorithm for biclustering time series expression data. In *Proc. of 5th Workshop on Algorithms in Bioinformatics*, pages 39–52. Springer Verlag, LNCS/LNBI 3692, 2005.
11. S. C. Madeira and A. L. Oliveira. An efficient biclustering algorithm for finding genes with similar patterns in time-series expression data. In *Proc. of the 5th Asia Pacific Bioinformatics Conference (to appear)*, 2007.
12. S. C. Madeira, M. C. Teixeira, I. Sá-Correia, and A. L. Oliveira. Identification of regulatory modules in time-series gene expression data using a linear time biclustering algorithm. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (submitted)*, 2007.
13. I. Van Mechelen, H. H. Bock, and P. De Boeck. Two-mode clustering methods: a structured overview. *Statistical Methods in Medical Research*, 13(5):979–981, 2004.
14. M.-F. Sagot. Spelling approximate repeated or common motifs using a suffix tree. In *Proc. of Latin'98*, pages 111–127. Springer Verlag, LNCS 1380, 1998.
15. A. Tanay, R. Sharan, and R. Shamir. Discovering statistically significant biclusters in gene expression data. *Bioinformatics*, 18(1):136–144, 2002.
16. M. C. Teixeira, P. Monteiro, P. Jain, S. Tenreiro, A. R. Fernandes, N. P. Mira, M. Alenquer, A. T. Freitas, A. L. Oliveira, and I. Sá-Correia. The YEASTRACT database: a tool for the analysis

of transcription regulatory associations in saccharomyces cerevisiae. *Nucleic Acids Research*, 34:D446–D451, January 2006.
17. E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14:249–260, 1995.
18. Y. Zhang, H. Zha, and C. H. Chu. A time-series biclustering algorithm for revealing co-regulated genes. In *Proc. of the 5th IEEE International Conference on Information Technology: Coding and Computing*, pages 32–37, 2005.

Appendix A. Scoring of CCC-Biclusters using Statistical Significance and Similarity Measures

Madeira and Oliveira [12] proposed to score CCC-Biclusters using a criterion, able to combine or apply iteratively the following criteria: (1) statistical significance of expression pattern, (2) similarity measure (able to deal with highly overlapping biclusters) and (3) measures of biological significance (functional enrichment and co-regulation).

We will use the same criterion in this work.

In the remaining of this section we will describe in more detail how they perform the computation of a similarity measure between two CCC-Biclusters and the statistical significance of a CCC-Bicluster of a given size and with a given expression pattern.

Appendix A.1. *Similarity Measure*

In order to compute the similarity score between two CCC-Biclusters, $B_1 = (I_1, J_1)$ and $B_2 = (I_2, J_2)$, Madeira and Oliveira [12] use the Jaccard Index (also called similarity coefficient or score) and use this score to measure the overlapping between CCC-Biclusters both in terms of genes and conditions and is defined as follows:

$$J(B_1, B_2) = J((I_1, J_1), (I_2, J_2)) = \frac{|B_1 \bigcap B_2|}{|B_1 \bigcup B_2|} = \frac{|B_{11}|}{|B_{01}|+|B_{10}|+|B_{11}|}, \qquad \text{(A.1)}$$

where $B_{11} = \{(i,j) : (i,j) \in B_1 \bigwedge (i,j) \in B_2\}$, $B_{10} = \{(i,j) : (i,j) \in B_1 \bigwedge (i,j) \notin B_2\}$, $B_{01} = \{(i,j) : (i,j) \notin B_1 \bigwedge (i,j) \in B_2\}$ and $B_{00} = \{(i,j) : (i,j) \notin B_1 \bigwedge (i,j) \notin B_2\}$, for the genes $i \in I_1 \bigcup I_2$ and the conditions $j \in J_1 \bigcup J_2$. This means that according to this similarity score, they are measuring the intersections between the elements in two CCC-Biclusters (genes and conditions), which in fact is the size of the submatrix corresponding to the intersection between the submatrices representing the CCC-Biclusters B_1 and B_2 (rectangle intersection).

Similarly, the gene similarity and condition similarity is computed, respectively, as follows: $J(I_1, I_2) = \frac{|I_1 \bigcap I_2|}{|I_1 \bigcup I_2|}$ and $J(J_1, J_2) = \frac{|J_1 \bigcap J_2|}{|J_1 \bigcup J_2|}$.

Note that, in practice, and since $|B_1| = |I_1| \times |J_1|$ and $|B_2| = |I_2| \times |J_2|$, the similarity score as defined in Equation A.1 can be computed easily using the fact that $|B_1 \bigcap B_2| = |I_1 \bigcap I_2| \times |J_1 \bigcap J_2|$ and $|B_1 \bigcup B_2| = |B_1| + |B_2| - |B_1 \bigcap B_2|$.

Appendix A.2. *Statistical Significance*

Madeira and Oliveira [12] proposed to measure the statistical significance of a CCC-bicluster B of size $|I| \times |J|$, where I is the set of genes and J is the set of contiguous time-points, and expression pattern p_B, against the null hypothesis, which is, in this case, the probability of a CCC-Bicluster of the considered size and expression pattern occurring by chance in a randomly generated gene expression matrix with $|G|$ genes and $|C|$ time-points. This value is obtained by computing the *tail of the binomial distribution*, P, which gives the probability of an event with probability p occurring k or more times in n independent trials, $P = \sum_{j=k}^{n} p_j (1-p)^{n-j}$.

In this context, the statistical significance of a CCC-Bicluster B, is the p-value(B), computed by obtaining the probability of a random occurrence of the expression pattern

p_B $k = |I| - 1$ times in $n = |G| - 1$ independent trials, where I is the number of genes in B and $|G|$ is the total number of genes in the gene expression matrix. We use the simplifying assumption that the probability of occurrence of a specific expression pattern p_B is adequately modeled by a first order Markov Chain, with state transition probabilities obtained from the values in the corresponding columns in the matrix. For example, if $B = (\{G1, G2, G4\}, \{C2, C3, C4\})$ (CCC-Bicluster B_1 in Figure 2), $p_B = P(U2D3U4) = P(U2)P(D3|U2)P(U4|D3)$, where $P(U2) = \frac{|U2|}{|G|}$, $P(D3|U2) = \frac{P(U2D3)}{P(U2)} = \frac{|U2D3|}{|U2|}$ and $P(U4|D3) = \frac{P(D3U4)}{P(D3)} = \frac{|D3U4|}{|D3|}$. These probabilities are, in this case, computed using the gene expression matrix after alphabet transformation in Figure 1. The values $|U2|$, $|U2D3|$, $|D3|$ and $|D3U4|$ correspond, respectively, to the number of occurrences of symbol $U2$, the number of transitions from symbol $U2$ to symbol $D3$, the number of occurrences of symbol $D3$, and the number of transitions from symbol $D3$ to symbol $U4$ (see Figure 4).

We adapted the computation of these p-values and used them to evaluate the statistical significance of CCC-Biclusters with sign-changes, CCC-Biclusters with time-lags and CCC-Biclusters with sign-changes and time-lags.

Appendix B. CCC-Biclustering

We present here the CCC-Biclustering algorithm since it will be useful the explain the extensions proposed in the main paper.

Algorithm 1: Find and Report all Maximal CCC-Biclusters

input: Discretized gene expression matrix A

Map each row A_{iC} to a string S_i using function f: $S_i \leftarrow f(A_{iC}), i \in \{1, \ldots, |R|\}$.
Build a generalized suffix tree T for the set of strings $\{S_1, \ldots, S_{|R|}\}$.
for *each node* $v \in T$ **do**
    Mark v as "Valid".
    Compute the string-length $P(v)$.
    if v *is a leaf node* **then**
        Compute the edge-length $E(v)$.
for *each internal node* $v \in T$ **do**
    Compute the number of leaves $L(v)$ in the subtree rooted at v.
for *each node* $v \in T$ **do**
    if v *is an internal node* ***and*** *there is a suffix link* $(v, s(v))$ ***and*** $L(v) >= L(s(v))$ **then**
        Mark node $s(v)$ as "Invalid".
    else
        if v *is a leaf node* ***and*** *(*$P(v)! = |S_i|$ ***or*** $E(v) = 1$*)* **then**
            Mark node v as "Invalid".
for *each node* $v \in T$ **do**
    if v *is marked as "Valid"* **then**
        Report the CCC-Bicluster that corresponds to v.

Appendix C. CCC-Biclustering with Time-Lags

Algorithm 2: Find and Report all Maximal CCC-Biclusters with time-lags

input: Discretized gene expression matrix A

Map each row A_{iC} to a string S_i using function f: $S_i \leftarrow f(A_{iC}), i \in \{1, \ldots, |R|\}$.
Build a generalized suffix tree T for the set of strings $\{S_1, \ldots, S_{|R|}\}$.
for *each node* $v \in T$ **do**
    Mark v as "Valid before time-lags".
    Compute the string-length $P(v)$.
    if v *is a leaf node* **then**
        Compute the edge-length $E(v)$.
for *each internal node* $v \in T$ **do**
    Compute the number of leaves $L(v)$ in the subtree rooted at v.
for *each node* $v \in T$ **do**
    if v *is an internal node* ***and*** *there is a suffix link* $(v, s(v))$ ***and*** $L(v) >= L(s(v))$ **then**
        Mark node $s(v)$ as "Invalid before time-lags".
    else
        if v *is a leaf node* ***and*** *(*$P(v)! = |S_i|$ ***or*** $E(v) = 1$*)* **then**
            Mark node v as "Invalid before time-lags".
for *each lag* $\in \{1, \ldots, |C| - 1\}$ **do**
    Add to the generalized suffix tree T the set of strings $\{S_1 + lag, \ldots, S_{|R|} + lag\}$ corresponding to the original strings S_i shifted lag times to the right.
for *each node* $v \in T$ **do**
    Mark v as "Valid after time-lags".
    Compute the string-length $P(v)$.
    if v *is a leaf node* **then**
        Compute the edge-length $E(v)$.
for *each internal node* $v \in T$ **do**
    Compute the number of leaves $L(v)$ in the subtree rooted at v.
for *each node* $v \in T$ **do**
    if v *is an internal node* ***and*** *there is a suffix link* $(v, s(v))$ ***and*** $L(v) >= L(s(v))$ ***or*** v *has only leaf nodes with terminators added with the lagged strings* **then**
        Mark node $s(v)$ as "Invalid after time-lags".
    else
        if v *is a leaf node* ***and*** *(*$P(v)! = |S_i|$ ***or*** $E(v) = 1$*)* ***or*** *the string terminator of* $P(v)$ *is one of the lagged strings* **then**
            Mark node v as "Invalid after time-lags".
for *each node* $v \in T$ **do**
    if v *is marked as "Valid after time-lags"* **then**
        for *each CCC-Biclusters with time-lags identified by* v **do**
            if *lag = 0* **then**
                if v *is "Valid before time-lags"* **then**
                    Report the CCC-Bicluster with lag = 0.
            else
                Report the CCC-Bicluster (lag > 0).