AN EFFICIENT BICLUSTERING ALGORITHM FOR FINDING GENES WITH SIMILAR PATTERNS IN TIME-SERIES EXPRESSION DATA[*]

SARA C. MADEIRA

INESC-ID / IST
University of Beira Interior
Rua Marquês D'Ávila e Bolama, 6200-001 Covilhã, Portugal
E-mail: smadeira@di.ubi.pt

ARLINDO L. OLIVEIRA

INESC-ID / IST
Rua Alves Redol, 9, 1000-039 Lisbon, Portugal
E-mail: aml@inesc-id.pt

Biclustering algorithms have emerged as an important tool for the discovery of local patterns in gene expression data. For the case where the expression data corresponds to time-series, efficient algorithms that work with a discretized version of the expression matrix are known. However, these algorithms assume that the biclusters to be found are perfect, in the sense that each gene in the bicluster exhibits exactly the same expression pattern along the conditions that belong to it. In this work, we propose an algorithm that identifies genes with similar, but not necessarily equal, expression patterns, over a subset of the conditions. The results demonstrate that this approach identifies biclusters biologically more significant than those discovered by other algorithms in the literature.

1. Introduction

Several non-supervised machine learning methods have been used in the analysis of gene expression data. Recently, biclustering[4], a non-supervised approach that performs simultaneous 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[4,14,1,6,10]. 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[11] is a recent proposal of an algorithm that finds and reports all maximal contiguous column coherent biclusters (CCC-Biclusters) in time linear on the size of the expression matrix by manipulating a discretized matrix using string

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

processing techniques based on suffix trees. Each expression pattern shared by a group of genes in a contiguous subset of time-points is a potentially relevant biological process.

However, discretization may limit the ability of the algorithm to discover biologically relevant patterns due to the noise inherent to most microarray experiments. To overcome this problem we present a new algorithm, *e*-CCC-Biclustering, that finds CCC-Biclusters with up to a given number of errors per gene in their expression pattern (*e*-CCC-Biclusters). These errors can, in general, be substitutions of a symbol in the expression pattern by other symbols in the alphabet (measurement errors), or restricted to the lexicographically closer discretization symbols (discretization errors).

We present results using a well known gene expression dataset that support the view that allowing errors in CCC-Biclusters improves the ability of the algorithm to discover more relevant biological processes, either by adding genes to the CCC-Bicluster that had been excluded due to errors, or by adding columns (up to the number of errors allowed) either at the left or at the right of the expression pattern of CCC-Bicluster.

The paper is organized as follows: Section 2 presents definitions needed to state the problem and construct the algorithm, as well as related work on biclustering in time-series expression data. Section 3 presents the algorithm and Section 4 describes the experimental results. Finally, Section 5 states some conclusions and outlines future work.

2. Definitions and Related Work

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[5]. 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[5]. Ukkonen's algorithm[16], 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

	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

	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

Figure 1. Toy example: (left) original expression matrix, (middle) discretized matrix and (right) 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[10]. 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. *Biclusters in Time-Series Gene Expression Data*

When analyzing time-series gene expression data we can restrict the attention to biclusters with contiguous columns[11,17,8]. This leads us to the definition of CCC-Bicluster and other relevant definitions related to it (already defined in previous work [11]), such as, trivial CCC-Biclusters and row-maximal, left-maximal, right-maximal, and maximal CCC-Biclusters:

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 $J \subseteq M \Rightarrow I = L \wedge J = M$.

Given these definitions we can now define the type of biclusters we are interested in this work, *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.

The goal of the *e*-CCC-Biclustering algorithm we propose in this work can now be defined: find and report all maximal *e*-CCC-Biclusters given a discretized version A of the original gene expression matrix A'.

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

Although several algorithms have been proposed to address the general problem of biclustering[10], to our knowledge, only three recent proposals have addressed this problem in the specific case of time-series expression data[17,8,11].

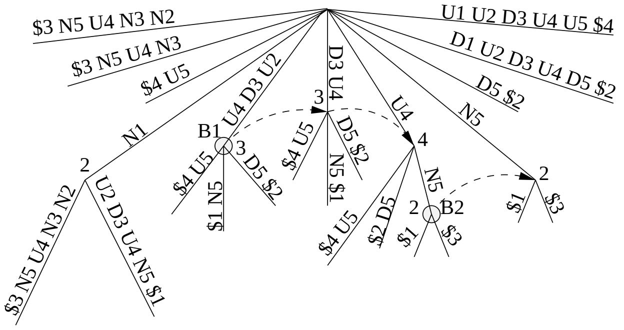
Zhang et. al[17] proposed to modify the heuristic algorithm of Cheng and Church[4], by restricting it to add and/or remove only columns that are contiguous to the partially constructed bicluster thus forcing the resulting bicluster to have only contiguous columns. Multiple biclusters are identified (as 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.

A different approach, from Ji and Tan[8], also works with a discretized data matrix. As in CCC-Biclustering[11], an $O(|R||C|)$ algorithm that has been recently proposed and that will be described in the end of this section) they are also interested in identifying biclusters formed by consecutive columns. Therefore, their idea generates exactly the same biclusters as the ones generated by CCC-Biclustering. With an appropriate implementation (not described by the authors) their sliding window approach can have its complexity reduced to $O(|R||C|^2)$, a complexity that is still of the order of $|C|$ higher than that of CCC-Biclustering. However, they propose to use a naive algorithm that, as made available by the authors[7], requires time and space exponential on the number of columns, when applied to the generation of all CCC-Biclusters. In practice, it cannot be applied to generate biclusters with more than 10 or 11 time-points.

CCC-Biclustering[11], 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 Sec. 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 2.1. *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 2.1 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 $e > 0$.



	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,G4},{C2,C3,C4})

B2=({G1,G3},{C4,C5})

Figure 2. (Left) 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). (Right) The CCC-Biclusters B1 and B2 are showned in the matrix as subsets of rows and columns (I, J). The strings m =[U2 D3 U4] and m =[U4 N5] correspond to the expression patterns of B1 and B2, respectively (called valid models/motifs in Section 3.1).

3. Finding and Reporting all Maximal *e*-CCC-Biclusters

3.1. *Finding e-CCC-Biclusters and the Common Motifs Problem*

SPELLER[13] is an algorithm that extracts common motifs from a set of N sequences using a generalized suffix tree. The motifs searched correspond to *words* which occur with at most e mismatches in $1 \leq q \leq N$ distinct sequences. The words representing the motifs may not be present exactly in the sequences (see SPELLER for details). As such a **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, that is, it must belong to the *e*-neighborhood of a word w, $N(e, w)$, in at least q distinct sequences.

The **common motifs problem** is as follows[13]: given a set of N sequences S_i ($1 \leq i \leq N$), and two integers $e \geq 0$ and $2 \leq q \leq N$, where e is the number of errors allowed and q is the required quorum, find all models m that appear in at least q distinct sequences of S_i.

SPELLER solves the problem above starting by building a generalized suffix tree T of the sequences S_i and then, after some further preprocessing, using this tree to "spell" the valid models. When $e = 0$ spelling this models leads to a node v in T such that $L(v)$ is at least q (similarly to CCC-Biclustering). When errors are allowed, spelling all the occurrences of a model m leads to a set of nodes v_1,...,v_k in T for which $\sum_{j=1}^{k} L(v_j)$ is at least q[13]. Since in SPELLER the occurrences of a model m are in fact nodes of the generalized suffix tree T, they are called node-occurrences:

Definition 3.1. A **node-occurrence** of a model m is a triple (v, v_{err}, p), where v is a node in the generalized suffix tree, v_{err} is the number of mismatches between m and the path-label from the $root$ to node v and p identifies a position in the generalized suffix tree: if $p = 0$, we are exactly at node v, if $p > 0$ we are at a point p between two symbols in $label_b$ ($1 \leq p < |label_b|$), where b is the edge between nodes $father_v$ and v.

The goal of SPELLER is to identify all valid models by extending them in the generalized suffix tree and to report the results using their set of node-occurrences. Note that in SPELLER, a node-occurrence is defined by a pair (v, v_{err}) and not by a triple (v, v_{err}, p) (for simplicity, the algorithm was exemplified in an uncompacted version of the generalized suffix tree, that is, a trie). However, as pointed out by the author, when using a generalized suffix tree, as we do, we need to know whether we are at a node v or at edge b between two nodes. Moreover, when we traverse T with a symbol α we also need to know if we get to a node v or if we stay inside an edge b. We use p to deal with these questions.

Consider that m is a model, α is a symbol in Σ', v is a node in T, $father_v$ is its father, b is the edge between $father_v$ and v and $label_b$ is the edge-label of b with length $|label_b|$. The algorithm we propose is based on the following Lemmas (adapted from SPELLER):

Lemma 3.1. $(v, v_{err}, 0)$ *is a node-occurrence of a model* $m' = m\alpha$*, if, and only if:* ***(1)*** $(father_v, v_{err}, 0)$ *is a node-occurrence of* m *and* $label_b$ *is* α ***or*** $(v, v_{err}, |label_b| - 1)$ *is a node-occurrence of* m *and the last symbol in* $label_b$ *is* α ***(match); (2)*** $(father_v, v_{err} - 1, 0)$ *is a node-occurrence of* m *and* $label_b$ *is* $\beta \neq \alpha$ ***or*** $(v, v_{err} - 1, |label_b| - 1)$ *is a node-occurrence of* m *and the last symbol in* $label_b$ *is* $\beta \neq \alpha$ ***(substitution)****.*

Lemma 3.2. $(v, v_{err}, 1)$ *is a node-occurrence of a model* $m' = m\alpha$*, if, and only if:* ***(1)*** $(father_v, v_{err}, 0)$ *is a node-occurrence of* m *and the first symbol in* $label_b$ *is* α ***(match); (2)*** $(father_v, v_{err} - 1, 0)$ *is a node-occurrence of* m *and* $label_b[1] = \beta \neq \alpha$ ***(substitution)****.*

Lemma 3.3. (v, v_{err}, p)*,* $2 \leq p < |label_b|$ *is a node-occurrence of a model* $m' = m\alpha$*, if, and only if:* ***(1)*** $(v, v_{err}, p - 1)$ *is a node-occurrence of* m *and* $label_b[p] = \alpha$ ***(match); (2)*** $(v, v_{err} - 1, p - 1)$ *is a node-occurrence of* m *and* $label_b[p] = \beta \neq \alpha$ ***(substitution)****.*

SPELLER can be adapted to extract all right-maximal *e*-CCC-Biclusters from the transformed matrix A. In fact, given the set of $|R|$ strings S_i of Section 2.2, $e \geq 0$ and $1 \leq q \leq |R|$, what we want to find is the set of all models m (identifying expression patterns) that are present in at least q distinct rows S_i **starting and ending at the same columns**. The set of node-occurrences of each model m and the model itself identifies one *e*-CCC-Bicluster with a maximum length $|C|$. Furthermore, it is possible to find all maximal *e*-CCC-Biclusters (without restricting the number of genes) by setting q to 1.

Figure 3 shows the generalized suffix tree used by SPELLER when $q = 1$ and $e = 1$ and two maximal *1*-CCC-Biclusters (B1 and B2) identified by two valid models. It is also possible to observe this fact in the matrix in Figure 4 where it is also clear that a model can be valid without being right/left maximal. Additionally, several valid models may identify the same e-CCC-Bicluster, when $e \geq 1$. For example, m=[N1 U2 D3] is valid but it is not right-maximal, m=[N2 D3 U4 N5] is also valid but it is not left-maximal, and finally the models m = [D1 U2 D3 U4 D5] and m=[D1 U2 D3 U5 N5] are both valid but identify the same *1*-CCC-Bicluster (B1). Similarly, m = [U2 D3 U4 D5], m = [U2 D3 U4 N5] and m = [U2 D3 U4 U5] are all valid models that represent B2.

The next section explains how SPELLER was adapted to extract exactly one valid model for each maximal *e*-CCC-Bicluster.

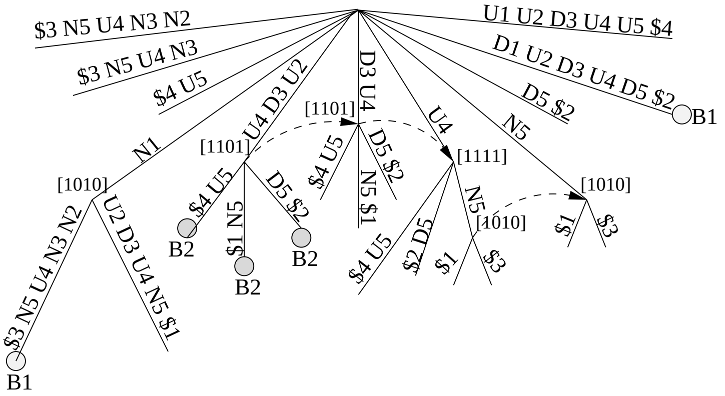
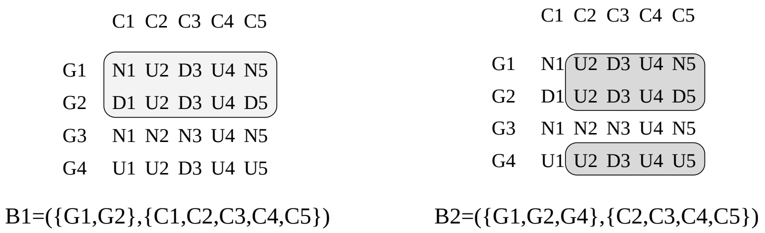


Figure 3. Generalized suffix tree for the right matrix in Figure 1 used in *e*-CCC-Biclustering when $e > 0$ (when $e = 0$ the suffix tree used is in Figure 2). The circles B1 and B2 identify two *1*-CCC-Biclusters (see Figure 4)



	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})

	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

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

Figure 4. (Left) Maximal *1*-CCC-Bicluster corresponding to the valid model/motif m = [D1 U2 D3 U4 D5]. (Right) Maximal *1*-CCC-Bicluster corresponding to the valid model/motif m = [U2 D3 U4 D5]. Note that these two 1-CCC-Biclusters correspond, respectively, to two and three node-occurrences (B1 and B2) in Figure 3.

3.2. *Algorithm Description*

This section presents *e*-CCC-Biclustering (see Algorithm 1) and describes its main steps.

The first step stores all valid models, m, and its node-occurrences, Occ_m, that correspond to right-maximal *e*-CCC-Biclusters in the list $modelsOcc$. In order to do this it uses an adaptation of SPELLER with two basic changes:

(1) Check if a model m corresponds to a right-maximal CCC-Bicluster using the procedure CHECKRIGHTMAXIMALITY which works as follows: if one of the models that result from the extension of a model m with a symbol α, $m\alpha$, is also a valid model and has as many genes in its node-occurrences as its father m, then m does not correspond to a right maximal CCC-Bicluster. As such, it is removed from the stored models. In order to compute the genes in the node-occurrences of a model m (returned as a bit vector, $genesOcc_m$), the function COMPUTEGENES(Occ_m) performs a bitwise or between the bit vectors $colors_v$ of all node-occurrences (v, v_{err}, p) of m in case $e > 0$, and uses the function NUMBEROFLEAVES(v) in case $e = 0$ (in this case Occ_m has only one node-

occurrence). The function NUMBEROFGENES($genesOcc_m$) is then used to compute the number of genes.

(2) The extensions, Ext_m, of a given model m are restricted according to the level of the model in the suffix tree. For example, if $\Sigma = \{D, N, U\}$ and model m is being extended descending from the root, m can only be $m = D1$, $m = N1$ or $m = U1$, and the possible symbols α with which it can be extended to $m\alpha$ are in $\Sigma'_2 = \{D2, N2, U2\}$.

Algorithm 1: Algorithm to Find and Report all Maximal e-CCC-Biclusters

```
input: $A$, $\Sigma$, $e$, $q$
$S \longleftarrow \{S_1, \ldots, S_{|R|}\}$, $S_i[j]$ = $f(A_{ij}, j)$, $1 \leq i \leq |R|$ and $1 \leq j \leq |C|$
$T_{right} \longleftarrow$ CONSTRUCTGENERALIZEDSUFFIXTREE ($S$)
ADDNUMBEROFLEAVES($T_{right}$) //Adds $L(v)$ to each node $v$ in $T_{right}$.
$m \longleftarrow$ “” //model $m$ is a string $[m_1 ... m_{|m|}]$
$length_m \longleftarrow 0$
$father_m \longleftarrow$ “” //$father_m$ is a string $[m_1 ... m_{|m|-1}]$
$numberOfGenesOcc_{father_m} \longleftarrow 0$
$Occ_m \longleftarrow \{\text{ROOT}(T_{right}), 0, 0)\}$ //Set of node-occurrences of model $m$.
$modelsOcc \longleftarrow \{\}$ //List of ($m$, $Occ_m$, $genesOcc_m$, $numberOfGenesOcc_m$).
if $e = 0$ then
    $Ext_m \longleftarrow \{\}$
    forall edges $b$ leaving node ROOT($T_{right}$) do
        if $label_b[1]$ is not a string terminator then
            $Ext_m \longleftarrow Ext_m \cup label_b[1]$ //$Ext_m$ is the set of possible symbols $\alpha$ to
              extend the model $m$.
else
    ADDCOLORARRAY($T_{right}$) //Adds $colors_v$ to each node $v$ in $T_{right}$
    //$colors_v[i] = 1$, if there is a leaf in the subtree rooted at $v$ that is a suffix of $S_i$.
    //$colors_v[i] = 0$, otherwise.
    $Ext_m \longleftarrow \Sigma'$
SPELLMODELS ($\Sigma$, $e$, $q$, $length_m$, $m$, $Occ_m$, $Ext_m$, $father_m$,
$numberOfGenesOcc_{father_m}$, $modelsOcc$)
DELETEMODELSNOTLEFTMAXIMALBICLUSTERS($modelsOcc$)
if $e > 0$ then
    DELETEMODELSREPRESENTINGSAMEBICLUSTERS($modelsOcc$)
REPORTMAXIMALBICLUSTERS($modelsOcc$)
```

The second step removes from the models stored in $modelsOcc$ (right-maximal *e*-CCC-Biclusters) those not corresponding to left maximal *e*-CCC-Biclusters. Non left-maximal biclusters are removed by first building a trie with the reversed patterns of all models m and storing the number of genes in Occ_m in its corresponding node in the trie. After this, it is sufficient to mark as non-maximal any node in the trie that has at least one child with as many genes as itself. This is easily achieved by performing a DFS of the trie and computing, for each node, the maximum value of all its children.

In the case where errors are allowed, different models may identify the same e-CCC-Bicluster. The third step uses an hash tree to remove from $modelsOcc$ (maximal *e*-CCC-Biclusters) repeated *e*-CCC-biclusters. The idea is that all models m with equal first and last columns and set of genes represent the same maximal CCC-Bicluster.

Finally, all maximal *e*-CCC-Biclusters in $modelsOcc$ are reported.

3.2.1. *e-CCC-Biclustering with Restricted Errors*

The *e*-CCC-Biclustering algorithm presented above allows general errors, that is substitutions of the symbols A_{ij} of the CCC-Bicluster $B = (I, J)$ by any of the symbols in the alphabet Σ'_j except A_{ij}. This kind of errors are specially relevant to identify measurement errors that occurred during the microarray experiments. However, if we are specially interested in identifying discretization errors we can consider restricted errors, that is, substitutions of the symbols A_{ij} by the lexicographically closer symbols in Σ'_j.

For example, when general errors are allowed, Σ = $\{D, N, U\}$, and m = [U2 D3 U4 D5], symbol D5 can be substituted by N5 and U5 in Σ'_5 leading to the *1*-CCC-Bicluster B2=({G1,G2,G4},{C2,C3,C4,C5}) in Figure 3 and Figure 4(b). However, if only restricted errors were allowed, symbol D5 could only be substituted by N5 leading to the *1*-CCC-Bicluster B=({G1,G2},{C2,C3,C4,C5}).

In general, when restricted errors are considered, the allowed substitutions for any symbol A_{ij} are in $\Sigma'^{Rest}_j = \{\Sigma'_j[p-1], \ldots, \Sigma'_j[p-z], \Sigma'_j[p+1], \ldots, \Sigma'_j[p+z]\}$, where p is the position of A_{ij} in Σ'_j and $z \in \{1, ..., |C|\}$. If $z = |C|$ then the errors are not restricted.

It is easy to modify Algorithm 1 to restrict the allowed errors.

3.2.2. *Complexity Analysis*

The construction of T_{right} and the computation of $L(v)$ for all its nodes take $O(|R||C|)$ time each, using the Ukkonen's algorithm with appropriate data structures, and a DFS, respectively. Adding the color array to all nodes in T_{right} (needed only when $e > 0$) takes $O(|R|^2|C|)$ time, and the remaining procedures in the algorithm take $O(|R|^2|C|^{1+e}|\Sigma|^e)$ time each. Therefore, the complexity of e-CCC-Biclustering is $O(|R|^2|C|^{1+e}|\Sigma|^e)$, when general errors are allowed, and $O(|R|^2|C|^{1+e}|\Sigma^{Rest}|^e)$, in the case of restricted errors.

When $e = 0$, Theorem 2.1 and CCC-Biclustering[11] can be used to obtain $O(|R||C|)$.

4. Experimental Results

In order to validate the quality of the results of the e-CCC-Biclustering algorithm we used the yeast cell-cycle dataset publicly available[3], described by Tavazoie et al.[15] and processed by Cheng and Church[4]. This dataset contains the expression profiles of more than 6000 yeast genes measured at 17 time-points over two complete cell cycles. We used 2884 genes selected by Cheng and Church[4] (as in Tavazoie et al.[15]) and removed the genes with missing values. The matrix with the remaining 2268 genes was discretized using 3 *equal* *frequency intervals*. We have also used *smoothing* as a preprocessing step to discretization with a window of size 5 and the set of weights $\beta_k = \{0.05, 0.2, 0.5, 0.2, 0.05\}$.

$$A_{ij} = \sum_{k=-w}^{+w} \beta_{j+k} A'_{i(j+k)} \tag{1}$$

Smoothing has been used by a number of authors in order to reduce the effect of experimental errors in the gene expression levels[2,9,11]. In order to minimize these errors each expression value A'_{ij} in matrix A' is smoothed using Equation (1), where $2w + 1$ is the window size. The values β_k are the weights given to the expressions values around A'_{ij} in a window of size $2w + 1$. These values control how much smoothing is applied to the data.

After the discretization process, we applied *e*-CCC-Biclustering with $e = 1$ with restricted errors and $z = 1$ (see Section 3.2.1) to the discretized matrix described above. We have also computed the results when $e = 0$. We argue that allowing errors in the pattern of the *0*-CCC-biclusters found by *0*-CCC-Biclustering should improve the biological significance of the biclusters by minimizing the effect of possible discretization errors. In fact, in the specific case of allowing 1 error in the pattern of a *0*-CCC-Bicluster one of the following three situations can happen: (1) the *1*-CCC-Bicluster remains equal to the *0*-CCC-Bicluster; (2) one or more genes excluded from the *0*-CCC-Bicluster (due to a single error) could be added to the *1*-CCC-Bicluster; or (3) the pattern of the *0*-CCC-Bicluster could be extended by adding one column either at its beginning or at its end (leading to a *1*-CCC-Bicluster with as many genes as the *0*-CCC-Bicluster but with one more column).

In order to validate the biological relevance of the *e*-CCC-Biclusters discovered we used the Gene Ontology (GO) annotations and associations files together with Ontologizer 2.0[12]. The goal was to evaluate the biological significance by computing p-values obtained when the hyper-geometric distribution is used to model the probability of observing at least k genes, from a CCC-Bicluster with $|I|$ genes, by chance, in a GO category containing c genes from the total number of genes ($|R|$) in each dataset. We used the functions from the three GO categories, biological process, molecular function and cell component.

We used the following procedure to validate the thesis that allowing errors in the CCC-Biclusters can improve the quality of the results: for each *0*-CCC-Bicluster with at least 4 genes and 2 conditions, we computed the number of GO functions enriched, in a statistically significant way, after Bonferroni correction, and stored this value together with the p-value of the most enriched function, that is, the lower GO p-value. The *0*-CCC-Biclusters were sorted according to their statistical significance. This significance was computed by evaluating the probability that a bicluster of that size appeared by chance in a matrix where the symbols are generated by a Markov chain, whose transition probabilities are obtained from the values in the matrix. Biclusters that were redundant, since their similarity[a] to biclusters already reported was above 50%, were removed from the analysis. We have then computed the *1*-neighborhood of the pattern of the top 10 *0*-CCC-Biclusters.[b]

[a] The similarity was measured by the number of common genes and conditions.

[b] This computation, although simple, is not described in this paper, since the purpose of this analysis is to evaluate the relative quality of *0*-CCC-Biclusters and *1*-CCC-Biclusters regarding their biological significance.

Table 1. Comparison between the top 10 *0*-CCC-Bicluster (sorted according to the statistical p-value) and the best *1*-CCC-Bicluster found by the 1-CCC-Biclustering algorithm whose pattern is a *1*-neighbor of the CCC-Bicluster without errors. Column 6 shows the best p-value computed using the hypergeometric distribution and the Gene Ontology, column 7 shows the Bonferroni correction of the previous value and finally column 8 shows the number of GO functions that are significantly enriched after Bonferroni correction (corrected p-value smaller or equal to 0.01).

ID	e	PATTERN	CONDITIONS	#GENES	P-VALUE	CORRECTION	#FUNCTIONS
2237	0	DDNNNDD	11-17	164	2.8E-12	2.6E-9	11
58537	1	**DDNNDDD**	11-17	**531**	1.1E-16	1.2E-17	17
767	0	UNDDDNNNDD	8-17	81	2.2E-11	1.5E-8	9
18544	1	UUDDDNNNDD	8-17	**108**	1.7E-14	1.3E-11	15
3041	0	NNNDD	13-17	340	1.2E-11	1.2E-8	9
63752	1	**DNNNDD**	**12-17**	**343**	3.5E-16	4.8E-13	17
260	0	UUUNDDDNNNDD	6-17	57	1.4E-8	1.7E.7	7
11361	1	UUUNDDDNNNDD	6-17	**144**	6.1E-13	5.2E-10	12
3220	0	NDD	15-17	772	3.5E-8	7.1E-5	3
68222	1	**DDD**	15-17	**1321**	2.2E-10	5.4E-7	7
3071	0	UNNDD	13-17	164	1.0E-5	1.1E-1	**0**
63773	1	**DUNNDD**	**12-17**	**249**	1.1E-9	1.4E-6	12
3035	0	NDDDD	13-17	161	3.6E-4	3.7E-1	**0**
66652	1	**NNDDD**	13-17	**608**	2.2E-9	4.0E-6	11
3217	0	DDD	15-17	492	2.2E-4	3.9E-1	**0**
68222	1	DDD	15-17	**1321**	2.2E-10	5.4E-7	7
627	0	NNDDDNNNDD	8-17	65	8.1E-4	4.1E-1	**0**
24875	1	NNDDDNNNDD	8-17	**165**	6.5E-14	6.0E-11	14
3191	0	UUND	14-17	183	1.0E-3	9.3E-1	**0**
66883	1	**UUUND**	**13-17**	**311**	4.6E-7	5.9E-4	6

Table 1 reports these results. Each row contains one *0*-CCC-Bicluster followed by the best *1*-CCC-Bicluster, as measured by the number of GO functions enriched. These results show that the *1*-CCC-Biclusters tend to have higher biological significance, since their sets of genes are more significantly enriched than those of the best *0*-CCC-Biclusters. Even in cases where the *0*-CCC-Bicluster does not pass the statistical test for biological significance, there exists a *1*-CCC-Bicluster in its *1*-neighborhood that has biological significance.

For lack of space, we do not report here additional results that support the view that *e*-CCC-Biclusters are biologically more relevant than perfect CCC-Biclusters.

5. Conclusions and Future Work

In this work, we presented and analyzed a new algorithm, *e*-CCC-Biclustering, for the identification of groups of genes that exhibit similar activities in a subset of conditions, in time-series expression data. The algorithm finds and reports, in time polynomial on the size of the matrix, all *e*-CCC-Biclusters that correspond to these groups of genes. By selecting the *e*-CCC-Biclusters that are statistically more significant, it is possible to identify potentially relevant biological processes. The algorithm avoids the limitations that previous methods exhibit since they cannot consider genes that have small deviations from the central pattern of expression. Moreover, the results demonstrate that this approach identifies biclusters that are biologically more significant than those identified by existing algorithms.

In future work, we plan to build a graphical user interface to e-CCC-Biclustering, make it available to the scientific community, and use the algorithm to identify regulatory modules (sets of co-regulated genes that share a common function) in gene regulatory networks.

References

1. 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.
2. T. Chen, V. Filkov, and S. Skiena. Identifying gene regulatory networks from experimental data. In *Proceedings of the 3rd International Conference on Research in Computational Molecular Biology*, pages 99–103, 1999.
3. Y. Cheng and G. M. Church. Biclustering of expression data - supplementary information. http://arep.med.harvard.edu/biclustering/, [July 15, 2006].
4. 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.
5. D. Gusfield. *Algorithms on strings, trees, and sequences*. Computer Science and Computational Biology Series. Cambridge University Press, 1997.
6. P. De Boeck I. Van Mechelen, H. H. Bock. Two-mode clustering methods: a structured overview. *Statistical Methods in Medical Research*, 13(5):979–981, 2004.
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, [July 15, 2006].
8. L. Ji and K. Tan. Identifying time-lagged gene clusters using gene expression data. *Bioinformatics*, 21(4):509–516, 2005.
9. A. Kwon, H. Hoos, and R. Ng. Inference of transcriptional regulation relationships from gene expression data. *Bioinformatics*, 19(8):905–912, 2003.
10. 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, 2004.
11. S. C. Madeira and A. L. Oliveira. A linear time biclustering algorithm for time series expression data. In *Proc. of 5th Workshop on Algorithms in Bioinformatics*, pages 39–52. Springer Verlag, LNCS/LNBI 3692, 2005.
12. U. Bohme B. Beattie P. N. Robinson, A. Wollstein. Ontologizing gene-expression microarray data: characterizing clusters with gene ontology. *Bioinformatics*, 20(6):979–981, 2004.
13. 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.
14. A. Tanay, R. Sharan, and R. Shamir. Discovering statistically significant biclusters in gene expression data. *Bioinformatics*, 18(1):136–144, 2002.
15. S. Tavazoie, J. D. Hughes, M. J. Campbell, R. J. Cho, and G. M. Church. Systematic determination of genetic network architecture. *Nature Genetics*, 22:281–285, 1999.
16. E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14:249–260, 1995.
17. 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.