A Linear Time Biclustering Algorithm for Time Series Gene Expression Data

Sara C. Madeira[1,2,3] and Arlindo L. Oliveira[1,2]

[1] INESC-ID, Lisbon, Portugal
[2] Technical University of Lisbon, IST, Lisbon, Portugal
[3] University of Beira Interior, Covilhã, Portugal
smadeira@di.ubi.pt, aml@inesc-id.pt

Abstract. Several non-supervised machine learning methods have been used in the analysis of gene expression data obtained from microarray experiments. Recently, biclustering, 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 goal of biclustering is to find subgroups of genes and subgroups of conditions, where the genes exhibit highly correlated behaviors. In the most common settings, biclustering is an NP-complete problem, and heuristic approaches are used to obtain sub-optimal solutions using reasonable computational resources.

In this work, we examine a particular setting of the problem, where we are concerned with finding biclusters in time series expression data. In this context, we are interested in finding biclusters with consecutive columns. For this particular version of the problem, we propose an algorithm that finds and reports all relevant biclusters in time linear on the size of the data matrix. This complexity is obtained by manipulating a discretized version of the matrix and by using string processing techniques based on suffix trees. We report results in both synthetic and real data that show the effectiveness of the approach.

1 Introduction

Recent developments in DNA chips enabled the simultaneous measure of the expression level of a large number of genes (sometimes all the genes of an organism) for a given experimental condition (sample) [11]. The samples may correspond to different time points, different environmental conditions, different organs or even different individuals. Extracting biologically relevant information from this kind of data, widely called (gene) expression data, is a challenging and very important task.

Most commonly, gene expression data is arranged in a data matrix, where each gene corresponds to one row and each condition to one column, as in Fig. 1(a). Each element of this matrix represents the expression level of a gene under a specific condition, and is represented by a real number, which is usually the logarithm of the relative abundance of the mRNA of the gene under

R. Casadio and G. Myers (Eds.): WABI 2005, LNBI 3692, pp. 39–52, 2005.

the specific condition. Gene expression matrices have been extensively analyzed in both the gene dimension and the condition dimension. These analyses correspond, respectively, to the analysis of the expression patterns of genes and to the analysis of the expression patterns of samples. A number of different objectives are pursued when this type of analysis is undertaken. Among these, relevant examples are the classification of genes, the classification of conditions and the identification of regulatory processes. Clustering techniques have been extensively applied towards these objectives. However, applying clustering algorithms to gene expression data runs into a significant difficulty: many activation patterns are common to a group of genes only under specific experimental conditions. In fact, our general understanding of cellular processes leads us to expect subsets of genes to be co-regulated and co-expressed only under certain experimental conditions, but to behave almost independently under other conditions. Discovering such local expression patterns may be the key to uncovering many genetic mechanisms that are not apparent otherwise [1]. Researchers have therefore moved past this simple idea of row or column clustering and have turned to biclustering [2], a technique that when applied to gene expression matrices identifies subgroups of genes that show similar activity patterns under a specific subset of the experimental conditions.

Many approaches to biclustering in expression data have been proposed to date [9]. In its general form, this problem is known to be NP-complete [14], and almost all the approaches presented to date are heuristic and obtain only approximate results. In a few cases, exhaustive search methods have been used, but limits are imposed on the size of the biclusters that can be found, in order to obtain reasonable runtimes. There exists, however, a particular restriction to the problem that is very important but has not been extensively explored before, and that leads to a tractable problem. This restriction is applicable when the gene expression data corresponds to snapshots in time of the expression level of the genes. Under this experimental setup, the researcher is particularly interested in biclusters with contiguous columns, that correspond to samples taken in consecutive instants of time. In this context, we show that there exists a linear time algorithm that finds all maximal contiguous column biclusters.

2 Definitions and Related Work

2.1 Biclusters in Gene Expression Data

Let A' be an n row by m column matrix, where A'_{ij} represents the expression level of gene i under condition j. In this work, we are interested in the case where the gene expression levels can be discretized to a set of symbols of interest, Σ, that represent distinctive activation levels. In the simpler case, Σ may contain only three symbols, $\{D, U, N\}$ meaning *DownRegulated*, *UpRegulated* or *NoChange*. However,in other applications, the values in matrix A' may be discretized to a larger set of values.

After the discretization process, matrix A' is transformed in matrix A and $A_{ij} \in \Sigma$ represents the discretized value of the expression level of gene i under

condition j. Figure 1(b) represents a possible discretization of the gene expression values in Fig. 1(a). In this example, an expression level was considered as *NoChange* if it falls in the range $[-0.3 : 0.3]$. The matrix A is defined by its set of rows, R, and its set of columns, C. Let $I \subseteq R$ and $J \subseteq C$ be subsets of the rows and columns, respectively. Then, $A_{IJ} = (I, J)$ denotes the sub-matrix of A that contains only the elements A_{ij} belonging to the sub-matrix with set of rows I and set of columns J. We will use A_{iC} to denote row i of matrix A and A_{Rj} to denote column j of matrix A.

Definition 1. *A bicluster is a subset of rows that exhibit similar behavior across a subset of columns, and vice-versa. The bicluster $A_{IJ} = (I, J)$ is thus a subset of rows and a subset of columns where $I = \{i_1, ..., i_k\}$ is a subset of rows ($I \subseteq R$ and $k \leq n$), and $J = \{j_1, ..., j_s\}$ is a subset of columns ($J \subseteq C$ and $s \leq m$), and can be defined as a k by s submatrix of the matrix A.*

The specific problem addressed by biclustering algorithms can now be defined. Given a data matrix, A', or its discretized version, A, the goal is to identify a set of biclusters $B_k = (I_k, J_k)$ such that each bicluster B_k satisfies some specific characteristics of homogeneity. The exact characteristics of homogeneity vary from approach to approach, and will be studied in Section 2.2.

2.2 Bicluster Types and Merit Functions

Biclustering approaches may identify many types of biclusters by analyzing directly the values in matrix A or using its discretized version [9]. However, in this paper we will deal with biclusters that exhibit coherent evolutions, characterized by a specific property of the symbols present in the discretized matrix. In particular, we are interested in finding column coherent biclusters satisfying the following definition:

Definition 2. *A column coherent bicluster (cc-bicluster), $A_{IJ} = (I, J)$, 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 $j \in J$.*

Although interesting biclusters can be identified in the discretized matrix A, they are usually ranked using merit functions computed over the original,

	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)

Fig. 1. Toy example. (a) represents the original expression matrix, (b) the discretized matrix and (c) the discretized matrix after alphabet transformation (section 3.1).

non-discretized version of the matrix, A'. To understand these metrics, consider a bicluster $A_{IJ} = (I, J)$ and let a'_{iJ} represent the mean of the ith row in the bicluster, a'_{Ij} the mean of the jth column in the bicluster and a'_{IJ} the mean of all elements in the bicluster.

Depending upon the application, it may be helpful to characterize biclusters by the degree of fluctuation in gene expression level as well as the similarity in behavior. For example, a bicluster with a low *mean squared residue* (1), $MSR(I, J)$, where $r(A'_{ij})$ are the residues (2), indicates that the expression levels fluctuate in unison [2]. This includes, however, flat biclusters with no fluctuation. In order to remove flat biclusters or identify biclusters with high degree of fluctuation in expression levels is beneficial to use the bicluster *variance* (3), $VAR(I, J)$, the *average row variance* (4), $ARV(I, J)$, and the *average column variance* (5), $ACV(I, J)$. A low $VAR(I, J)$ identifies a constant bicluster. A bicluster with high $ARV(I, J)$ and low $ACV(I, J)$ has high fluctuation on the rows and coherent columns while a bicluster with low $ARV(I, J)$ and high $ACV(I, J)$ is a bicluster with high fluctuation on the columns and coherent rows. Since a low value of $MSR(I, J)$ identifies a bicluster with coherent values [2], if the value of $ARV(I, J)$ is high and the value of $MSR(I, J)$ is low we can also identify a bicluster with high fluctuation on the rows and coherent columns.

$$MSR(I, J) = \frac{1}{|I||J|} \sum_{i \in I, j \in J} r(A'_{ij})^2 \tag{1}$$

$$r(A'_{ij}) = A'_{ij} - a'_{iJ} - a'_{Ij} + a'_{IJ} \tag{2}$$

$$VAR(I, J) = \frac{1}{|I||J|} \sum_{i \in I, j \in J} (A'_{ij} - a'_{IJ})^2 \tag{3}$$

$$ARV(I, J) = \frac{1}{|I||J|} \sum_{i \in I, j \in J} (A'_{ij} - a'_{iJ})^2 \tag{4}$$

$$ACV(I, J) = \frac{1}{|I||J|} \sum_{i \in I, j \in J} (A'_{ij} - a'_{Ij})^2 \tag{5}$$

Many heuristic approaches have been proposed for the selection of biclusters that minimize directly this type of merit functions [9]. However, the inherent difficulty of this problem when dealing with the non-discretized matrix A' and the great interest in finding coherent behaviors regardless of the exact numeric values in the data matrix, has led many authors to a formulation based on a discretized version of the gene expression matrix. Since most versions of the problem addressed by these authors are NP-complete [1,5,6,7,13,15,16] the solutions proposed are heuristic and are not guaranteed to find optimal solutions.

A different approach, from Ji and Tan [4], aims at finding time-lagged biclusters in time series expression data. As in the present work, they are also interested in identifying biclusters formed by consecutive columns. They propose to use a naive algorithm that has a complexity $O(|R||C|^3)$, if all consecutive column biclusters are to be found. With an appropriate implementation (not described in the paper) 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 our proposed approach.

2.3 Biclusters in Time Series Expression Data

Finding a set of maximal biclusters that satisfy the coherence property defined in Def. 2 remains an NP-complete problem. As such, most biclustering algorithms use heuristic approaches that are not guaranteed to find optimal solutions. However, we are interested in the analysis of time series expression data, and this leads to an important restriction, on which relies the linear time algorithm we propose.

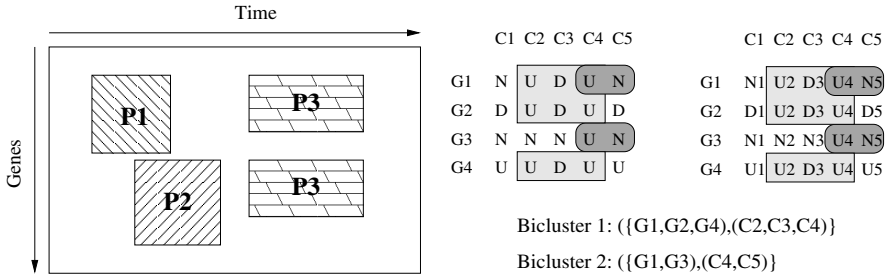
When analyzing time series expression data, with the objective of isolating coherent activity between genes in a subset of conditions, we want to restrict the attention to biclusters with contiguous columns. Other authors have already pointed out the importance of biclusters that span consecutive columns [4], and their importance in the identification of gene regulatory processes. In fact, the activation of a set of genes under specific conditions corresponds, in many cases, to the activation of a particular biological process. As time goes on, biological processes start and finish, leading to increased (or decreased) activity of sets of genes that can be identified because they form biclusters with contiguous columns, as illustrated in Fig. 2(a). In this figure, the existence of three processes (P1, P2 and P3) leads to increased activity of different sets of genes, represented by three biclusters. Note that, although the columns of each of the biclusters are contiguous, the rows are in arbitrary positions, and are represented as contiguous for P1 and P2 only for convenience. Overlapping is also allowed.

Time series expression data are often used to study dynamic biological systems and gene regulatory networks since their analysis can potentially provide more insights about biological systems [8]. In this context, the identification of biological processes that lead to the creation of biclusters, together with their relationship, is crucial for the identification of gene regulatory networks and for the classification of genes. This leads us to the definition of the type of biclusters that are of interest in this work.

Definition 3. *A contiguous column coherent bicluster (ccc-bicluster),* $A_{IJ} = (I, J)$*, 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* $j \in J$*.*

For the remainder of this work, we will refer to a contiguous column coherent bicluster simply as a ccc-bicluster. By definition, each row in matrix A is a ccc-bicluster. These are **trivial biclusters** and will not be of interest, in general. The biclusters with only one row or only one column will also be considered as trivial.

In this settings, each ccc-bicluster defines a string S that is common to every row in the ccc-bicluster, between columns r and s of matrix A. Figure 2(b) illustrates two ccc-biclusters that appear in the expression matrix in Fig. 1(a). These two ccc-biclusters are maximal, in the sense that they are not properly contained in any other ccc-biclusters. This notion will be defined more clearly later on.



(a) Three biological processes.

(b) Two ccc-biclusters in the toy example.

Fig. 2. Biclusters in time series gene expression data

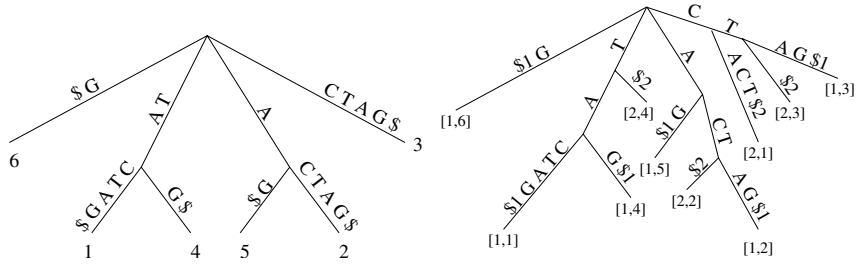
2.4 Suffix Trees

A *string* S is an ordered list of characters written contiguously from left to right [3]. For any string S, $S[i..j]$ is the (contiguous) *subtring* of S that starts at position i and ends at position j. In particular, $S[1..i]$ is the *prefix* of S that ends at position i and $S[i..|S|]$ is the *suffix* of S that starts at position i, where $|S|$ is the number of characters in S. A *suffix tree* is a data structure built over all the suffixes of a string S that exposes its internal structure. This data structure has been extensively used to solve a large number of string processing problems.

Definition 4. *A* suffix tree *of a $|S|$-character 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 and each edge is labeled with a nonempty substring of S. No two edges out of a node have edge-labels beginning with the same character. The key feature of the suffix tree is that for any leaf i, the label of the path from the root to the leaf i exactly spells out the suffix of S that starts at position i.*

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 nowhere else in the string, is added to its end. For example, the suffix tree for the string S=TACTAG is presented in Fig. 3(a). The suffix tree construction for a set of strings, called a *generalized suffix tree*, can be easily obtained by consecutively building the suffix tree for each string of the set. The leaf number of the single string suffix tree is now converted to two numbers: one identifying the string and other the starting position (suffix) in that string.

Suffix trees can be built in time that is linear on the size of the string, using several algorithms. Generalized suffix trees can be built in time linear on the sum of the sizes of the strings. Ukkonen's algorithm [18], used in this work, uses *suffix links* to achieve a linear time construction.



(a) Suffix tree for S=TACTAG.

(b) Generalized suffix tree for S_1=TACTAG and S_2=CACT.

Fig. 3. Example of a suffix tree for the string S=TACTAG and a generalized suffix tree for the strings S_1=TACTAG and S_2=CACT

Definition 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 exactly equal to the length of the path-label of v minus 1.*

3 Biclustering Time Series Expression Data

3.1 Biclusters and Suffix Trees

We can now introduce the major results of this work, that lead to the linear time biclustering algorithm. We first introduce the concept of maximal ccc-bicluster.

Definition 6. *A ccc-bicluster $A_{IJ} = (I, J)$ is maximal if no other ccc-bicluster exists that properly contains A_{IJ}, that is, if for all other ccc-biclusters $A_{LM} = (L, M)$, $I \subseteq L$ and $J \subseteq M \Rightarrow I = L \wedge J = M$.*

We will also call a ccc-bicluster *right-maximal* if it cannot be extended to the right by adding one more column at the end, and *left-maximal* if it cannot be extended to the left by adding one more column at the beginning. Stated more plainly, a ccc-bicluster is maximal if no more rows nor contiguous columns (either at the right or at the left) can be added to it while maintaining the coherence property in Def. 3.

We will now consider a new alphabet $\Sigma' = \Sigma \times \{1 \dots m\}$, where each element Σ' is obtained by concatenating one symbol in Σ and one number in the range $\{1 \dots m\}$. In order to do this alphabet transformation we use a function $f : \Sigma \times \{1 \dots m\}$ 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 $\Sigma = \{U, D, N\}$ and $m = 3$, then $\Sigma' = \{U1, U2, U3, D1, D2, D3, N1, N2, N3\}$. For this case, $f(U, 2) = U2$ and $f(D, 1) = D1$.

Consider now the set of strings $S = \{S_1, \dots, S_n\}$ obtained by mapping each row A_{iC} in matrix A to string S_i such that $S_i(j) = f(A_{ij}, j)$. Each of these strings has m characters and corresponds to the symbols in a row of matrix A after the above alphabet transformation. After this transformation, the matrix

in Fig. 1(a) corresponding to the discretized matrix in Fig. 1(b), becomes the matrix in Fig. 1(c).

Let T be the generalized suffix tree obtained from the set of strings S. Let v be a node of T and let $P(v)$ be the path-length of v, that is, the number of characters in the string that labels the path from the root to node v. Additionally, let $B(v)$ be the branch-length of v and let $L(v)$ denote the number of leaves in the sub-tree rooted at v, in case v is an internal node. It is easy to verify that every internal node of the generalized suffix tree T corresponds to one ccc-bicluster of the matrix A. This is so because an internal node v in T corresponds to a given substring that is common to every row that has a leaf rooted in v. Therefore, node v defines a ccc-bicluster that has $P(v)$ columns and a number of rows equal to $L(v)$. It is also true that all the leaves except the ones whose path label is simply a terminator also identify ccc-biclusters.

Since these ccc-biclusters may not be maximal, we will now present with only sketches of proofs the two lemmas that lead to our the main theorem.

Lemma 1. *Every right-maximal ccc-bicluster corresponds to one node in T.*

Proof. Let B be a ccc-bicluster that cannot be extended to the right by adding a column at the right, that is, a right-maximal ccc-bicluster. Since B is a ccc-bicluster, every row in B shares the substring that defines B. Since B is right maximal, at least one of the rows in B must have a character that differs from the character in the other rows, in the first column to the right that is not in B. Therefore, there is a node in T that matches B and the path-label of that node is the string that defines B. □

Lemma 2. *Let node v_1 correspond to a ccc-bicluster B_1 and node v_2 correspond to a ccc-bicluster B_2. Then, if there is a suffix link from node v_1 to node v_2, bicluster B_2 contains one less column than bicluster B_1.*

Proof. Follows directly from the definition of suffix links. □

From these lemmas, we can now derive the theorem that is our main result.

Theorem 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 bicluster iff the path-length of v, $P(v)$, is equal to $|S_i|$ and the label of the branch that leads to v has characters other than the terminator, that is, $B(v)$ is greater than one. Furthermore, every maximal ccc-bicluster in the matrix corresponds to a node v satisfying one of these conditions.*

Proof. Let B be a maximal ccc-bicluster and S the string that defines B. Now, S must lead to a node v (by Lemma 1). If node v is an internal node and does not have an incoming suffix link, the conditions of the theorem are met. Since B is also left-maximal, every node u that defines a bicluster B' with one more column than B (by Lemma 2) must have $L(v) > L(u)$, since B' cannot contain all the rows in B (otherwise, B would not be left-maximal). Therefore, it is

sufficient to check that every internal node u that has a suffix link directed at v has $L(u) < L(v)$ to ensure that node v corresponds to a maximal ccc-bicluster.

If node v is a leaf node the conditions of the theorem are met. In fact, if B is maximal the path-length of v must be equal to $|S|$, otherwise, B would not be left-maximal since it could be extended to the left until all the characters at its left in S had been added. Furthermore, the path-label of v cannot be only a string terminator, that is, $B(v)$ must be greater than one, otherwise B would also not be maximal. In one hand, if the parent of v was the root, then B would not be left maximal since it could be extended to the left by adding at least the last character of S. On the other hand, if the parent of v was an internal node, then B would not be maximal either since it could be extended by adding to it the rows that correspond to the remaining leaves of the sub-tree rooted at the internal node that is the parent of v. □

Figure 4 illustrates the generalized suffix tree obtained from the strings that correspond to the rows of the matrix in Fig. 1(c). For clarity, this figure does not contain the leaves that represent string terminators that are direct daughters of the root. Each non-terminal node, other than the root, is labeled with the value of $L(v)$, the number of leaves in its subtree. Also shown in this tree are the suffix links between nodes. For clarity, the suffix links that end at the root are not shown.

This figure also shows that there are six internal nodes, other than the root. Each one of these nodes corresponds to one ccc-bicluster. Furthermore, each leaf node with branch-length greater than one is also a ccc-bicluster. However, some of these ccc-biclusters are trivial, since they represent biclusters with only one column (nodes with branch-labels $N1$, $U4$ and $N5$, since they have branch-labels with only one character), or are represented by leaves (trivial ccc-biclusters with only one row even when they are maximal). Others are non-maximal (nodes with branch-labels $D3U4$ and $N5$), since they have an incoming suffix link from a node with the same number of leaves. As such, only the internal nodes with branch-labels $U2D3U4$ and $U4N5$ identify maximal, non-trivial ccc-biclusters. These

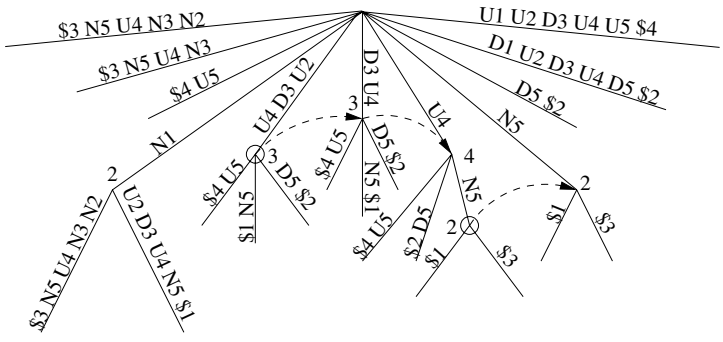


Fig. 4. Generalized suffix tree for the matrix in Fig. 1(c)

nodes correspond to the maximal ccc-biclusters $(\{G1, G2, G4\}, \{C2, C3, C4\})$ and $(\{G1, G3\}, \{C4, C5\})$ in Fig. 2(b).

Note that the rows in each ccc-bicluster are obtained from the terminators in the leaves in the subtree of each node v, while the columns in each ccc-bicluster are obtained from the value of $P(v)$ and the information on the branch-label that connects v to its parent. In fact, the value of $P(v)$ and the first character of the path label of v is needed to identify the set of columns that belong to the bicluster.

3.2 A Linear Time Algorithm for Finding and Reporting ccc-Biclusters

Theorem 1 directly implies that there is an algorithm that finds and reports all maximal ccc-biclusters in a discretized and transformed gene expression matrix A in time linear on the size of the matrix (see Alg. 1). With appropriate data structures at the nodes, the suffix tree construction is linear on the size of the input matrix, using Ukkonen' algorithm [18]. The remaining steps of our algorithm are also linear since they are performed using depth first searches (*dfs*) on the suffix tree. A more detailed analysis shows that the increase in the alphabet size does not have an impact on this linear time complexity. In fact, only two types of nodes have more than $|\Sigma|$ children: the root node and nodes that have as children only leaf nodes. In both cases, it is easy to devise a data structure that enables constant time manipulation of these nodes.

Algorithm 1. Algorithm for finding and reporting all maximal ccc-biclusters

```
 1: procedure FIND AND REPORT ALL MAXIMAL CCC-BICLUSTERS(A)
 2:     Map each row i in matrix A, A_iC, to a string S_i using function f: S_i ← f(A_iC).
 3:     Build a generalized suffix tree, T, for the set of strings S.
 4:     for all nodes v ∈ T do
 5:         Compute the path-length and the branch-length of v: P(v) and B(v).
 6:         Mark v as "Valid".
 7:     end for
 8:     for all internal nodes v ∈ T do
 9:         Compute the number of leaves in the sub-tree rooted at v: L(v).
10:     end for
11:     for all nodes v ∈ T do
12:         if (v is an internal node and there is a suffix link (v, u) and L(v) >= L(u))
                or (v is a leaf node and (P(v)! = |S_i| or B(v) = 1)) then
13:             Mark u as "Invalid".
14:         end if
15:     end for
16:     for all nodes v ∈ T do
17:         if v is "Valid" then
18:             Report the ccc-bicluster that corresponds to v.
19:         end if
20:     end for
21: end procedure
```

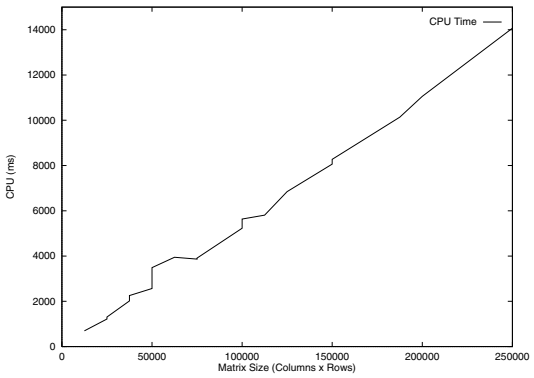


Fig. 5. CPU time versus size of the synthetic input data

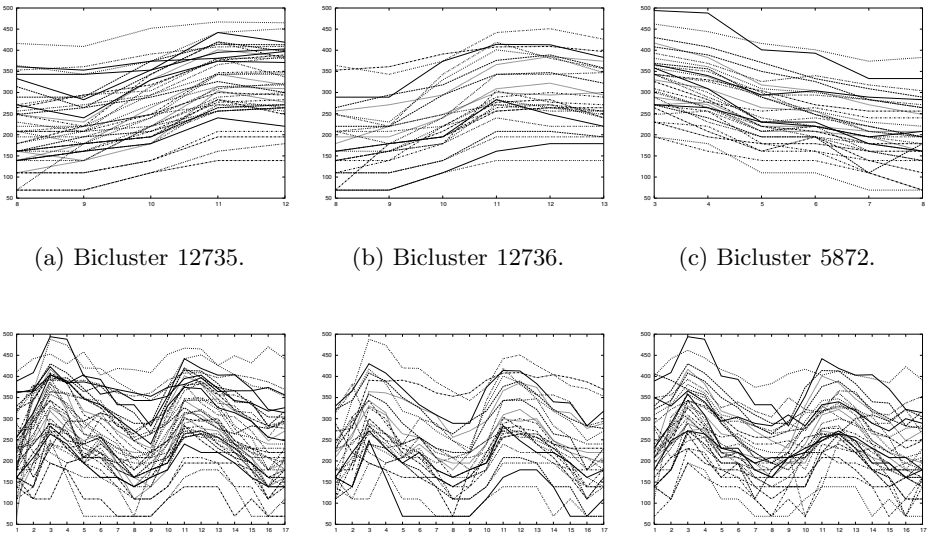
4 Experimental Results

In order to validate the approach, we performed experiments with both synthetic and real data, using a prototype implementation of the algorithm, coded in Java, and a 3GHz Pentium-4 machine, running Linux with 1GB of memory.

To evaluate the efficiency of the algorithm, and validate experimentally the predicted linear time complexity, we generated matrices with random values, on which 10 biclusters were hidden, with dimensions ranging from $15-25$ rows and $8-12$ columns. The size of the matrices varied from 250×50 (rows $\times$ columns) up to 1000×250. We used a three character alphabet, $\Sigma = \{U, D, N\}$. In all cases, we recovered the *planted* ccc-biclusters, together with a large number of artifacts that result from random coincidences in the data matrix. Figure 5 shows a plot of the variation of the CPU time with the size of the input data matrix. A clear linear relationship over several orders of magnitude is apparent from this plot. It is also clear that the algorithm runs in less than 15 seconds even in the larger synthetic matrices used.

To validate the approach with real data, we used time-series from the yeast cell-cycle dataset described by Tavazoie et al. [17] which 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 [2] as in [17] and removed the genes with missing values. The matrix with the remaining 2268 genes was discretized by gene to an alphabet $\Sigma = \{D, U, N\}$ using an equal bin frequency discretization.

The resulting matrix was then processed by our algorithm and 14728 maximal non-trivial ccc-biclusters were reported in 13.5 seconds. From these 825 had more than 4 conditions and at least 25 genes. Since we were interested in ccc-biclusters with high values of $ARV(I, J)$ and low values of $MSR(I, J)$, this last set was then ordered in descending order according to the value of $(ARV(I, J) - MSR(I, J))$ (see Sec. 2.2). The computation of the values of the metrics and the ordering of the biclusters cannot be done, in general, in time linear on the size of



(a) Bicluster 12735. (b) Bicluster 12736. (c) Bicluster 5872.

(d) Genes in bicluster 12735. (e) Genes in bicluster 12736. (f) Genes in bicluster 5872.

Fig. 6. Expression level of genes in the top three ccc-biclusters

Table 1. Biological relevance of the top three ccc-biclusters

Bicluster ID	*Genes*	*Conditions*	*GO-Category*	*GO-Level*	*P-Value*
12735	41	5 (8-12)	Cell Cycle	4	2.9E-05
12736	26	6 (8-13)	DNA replication	6,7	9.2E-03
5872	33	6 (3-8)	DNA metabolism	6,5	2.0E-05
			DNA replication	7,6	2.5E-05
			DNA repair	6,7	9.1E-05
			response to DNA damage stimulus	9	2.4E-04
			response to endogenous stimulus	9	2.7E-04
			biopolymer metabolism	4	4.3E-03
			negative regulation of DNA transposition	9,10	8.0E-03
			regulation of DNA transposition	8,9	8.0E-03
			lagging strand elongation	9,10	8.8E-03

the matrix. In practice, we observed that in real data these steps take less time than the bicluster generation steps.

We present some preliminary evidence of the biological significance of these results by analyzing (for lack of space) only the top three ccc-biclusters according to this criterion. To access the biological relevance of the biclusters we used the Gene Ontology (GO) and the p-values obtained from the hypergeometric distri-

bution to model the probability of observing at least k genes, from a bicluster with $|I|$ genes, by chance in a category containing c genes from the 2268 genes in the dataset. For this, we used the functions from the three GO categories, biological process, molecular function and cell component at level higher than 3. Table 1 shows p-values computed using the GOToolBox [10] and the utilities from the YEASTRACT database [12].

In order to show that the generated ccc-biclusters have biological significance, shown by statistically significant enrichment in one or more GO categories, we report the categories in which the (Bonferroni corrected [11]) p-values are below 0.01. Figure 6 presents the expression levels of these ccc-biclusters and shows how biclustering is able to identify highly correlated expression patterns of genes, under a given subset of conditions. Note that the highly correlated activity under this subset of columns does not necessarily translate into highly correlated activity under all conditions.

5 Conclusions

In this work, we presented a linear time algorithm for the identification of all maximal contiguous column biclusters in time series expression data. By discretizing the gene expression values, and manipulating the strings that correspond to each row using string processing techniques, we have been able to demonstrate that there is a correspondence between the maximal ccc-biclusters and the nodes of the generalized suffix tree that represents the rows (genes) in the matrix. This simple correspondence lead to a very efficient algorithm for the extraction of ccc-biclusters, that runs in a few seconds even for matrices with thousands of genes and hundreds of conditions. We have demonstrated the correctness of the algorithm and sketched the complexity analysis. We have also presented experimental results with synthetic data and preliminary results with real data from yeast. This work opened several promising directions for future research. Among these are the discovery of imperfect ccc-biclusters (ccc-biclusters allowing a given number of errors) and the development of methods for the identification of regulatory networks.

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 Ciência e Tecnologia, and the POSI program.

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. 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.

3. D. Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology*. Cambridge University Press, 1997.
4. L. Ji and K. Tan. Identifying time-lagged gene clusters using gene expression data. *Bioinformatics*, 21(4):509–516, 2005.
5. M. Koyuturk, W. Szpankowski, and Ananth Grama. Biclustering gene-feature matrices for statistically significant dense patterns. In *Proc. of the 8th Annual International Conference on Research in Computational Molecular Biology*, pages 480–484, 2004.
6. J. Liu, W. Wang, and J. Yang. Biclustering in gene expression data by tendency. In *Proc. of the 3rd International IEEE Computer Society Computational Systems Bioinformatics Conference*, pages 182–193, 2004.
7. S. Lonardi, W. Szpankowski, and Q. Yang. Finding biclusters by random projections. In *Proc. of the 15th Annual Symposium on Combinatorial Pattern Matching,Springer*, pages 102–116, 2004.
8. Y. Luan and H. Li. Clustering of time-course gene expression data using a mixed-effects model with b-splines. *Bioinformatics*, 19(4):474–482, 2003.
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, Jan-Mar 2004.
10. D. Martin, C. Brun, E. Remy, P. Mouren, D. Thieffry, and B. Jacq. Gotoolbox: functional investigation of gene datasets based on gene ontology. *Genome Biology*, 5(12):R101, 2004.
11. G. McLachlan, K. Do, and C. Ambroise. *Analysing microarray gene expression data*. Wiley, 2004.
12. P. Monteiro, M. C. Teixeira, P. Jain, S. Tenreiro, A. R. Fernandes, N. Mira, M. Alenquer, A. T. Freitas, A. L. Oliveira, and Isabel Sá-Correia. Yeast search for transcriptional regulators and consensus tracking (yeastract). *http://www.yeastract.com*, 2005.
13. T. M. Murali and S. Kasif. Extracting conserved gene expression motifs from gene expression data. In *Proc. of the Pacific Symposium on Biocomputing*, volume 8, pages 77–88, 2003.
14. R. Peeters. The maximum edge biclique problem is NP-complete. *Discrete Applied Mathematics*, 131(3):651–654, 2003.
15. Q. Sheng, Y. Moreau, and B. De Moor. Biclustering microarray data by Gibbs sampling. In *Bioinformatics*, volume 19 (Suppl. 2), pages 196–205, 2003.
16. A. Tanay, R. Sharan, and R. Shamir. Discovering statistically significant biclusters in gene expression data. In *Bioinformatics*, volume 18 (Suppl. 1), pages S136–S144, 2002.
17. 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.
18. E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14:249–260, 1995.