Automatic Layout and Visualization of Biclusters

Gregory A. Grothaus*
Google Inc.
1600 Amphitheater Parkway
Mountain View CA 94043
ggrothau@gmail.com

Adeel Mufti
Dept. of Computer Science
660 McBryde Hall
Virginia Polytechnic Institute
and State University
Blacksburg VA 20460
amufti@vt.edu

T. M. Murali
Dept. of Computer Science
660 McBryde Hall
Virginia Polytechnic Institute
and State University
Blacksburg VA 20460
murali@cs.vt.edu

ABSTRACT

Biclustering has emerged as a powerful algorithmic tool for analyzing measurements of gene expression. A number of different methods have emerged for computing biclusters in gene expression data. Many of these algorithms may output a very large number of biclusters with varying degrees of overlap. There are no systematic algorithms that create a two-dimensional layout of the computed biclusters and display overlaps between them.

We develop a novel algorithm for laying out biclusters in a two-dimensional matrix whose rows (respectively, columns) are rows (respectively, columns) of the original dataset. We display each bicluster as a contiguous submatrix in the layout. We allow the layout to have repeated rows and/or columns from the original matrix as required, but we seek a layout of the smallest size. We also develop a web-based search interface for the user to query the layout for genes and samples of interest. We demonstrate the usefulness of our approach on gene expression data for two types of leukemia and on protein-DNA binding data for two growth conditions. The software implementing the layout algorithm is available at http://bioinformatics.cs.vt.edu/~murali/papers/bivoc.

1. INTRODUCTION

Measurement of gene expression using DNA microarrays [13; 31] have revolutionized biological and medical research. Since gene expression plays an important role in cell differentiation, development, and pathological behavior, computational analysis of DNA microarray data has the potential to assign functions to newly-discovered genes, unravel the structure of biological pathways, and assist in the development of new medicines. Biclustering has emerged as a powerful algorithmic tool for analyzing gene expression data. A bicluster in a gene expression data set is a subset of genes and a subset of conditions with the property that the selected genes are co-expressed in the selected conditions; these genes may not have any coherent patterns of expression in the other conditions in the data set. Biclusters have a number of advantages over clusters computed by more traditional algorithms such as k-means and hierarchical clustering [11]. Since a bicluster includes only a subset of genes and samples, it models condition-specific patterns of co-expression. Traditional clusters may miss such patterns since they operate in the space spanned by all the conditions. Further, many biclustering algorithms allow a gene or a sample to participate in multiple biclusters, reflecting the possibility that a gene product may be a member of multiple pathways.

A number of different methods have emerged for computing biclusters in gene expression data [6; 5; 9; 14; 17; 20; 22; 28; 32; 33; 34; 35; 36]; two papers survey these techniques [26; 37]. These algorithms use a number of different strategies to compute biclusters such as exhaustive enumeration [8; 24; 36], iterated improvement [5; 9], repeated random sampling [28], and expectation maximization [32]. An issue all these algorithms deal with is trying to avoid outputting two or more biclusters with nearly the same set of samples and/or genes. A common approach is to remove a bicluster from the output if it shares a large fraction of genes and/or samples (based on a user-defined threshold) with an already computed bicluster. In spite of these measures, biclustering algorithms may compute tens, hundreds, or even thousands of biclusters with varying degrees of overlap.

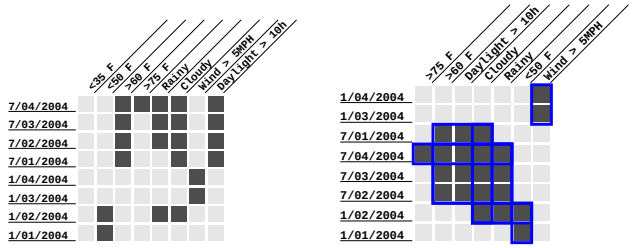
Organising, manipulating, and querying the potentially large number of biclusters computed by these algorithms is a data mining task in itself, which has not been adequately addressed. In this paper, we develop a novel algorithm for laying out biclusters in a manner that visually reveals overlaps between them. We lay out the biclusters in a two-dimensional matrix whose rows (respectively, columns) are rows (respectively, columns) of the original dataset. We display each bicluster as a contiguous submatrix in the layout. We allow the layout to have repeated rows and/or columns from the original matrix, but we seek a layout of the smallest size. In addition, we develop a web-based search interface that allows the user to query the results for genes and samples of interest and visualise the layout of the biclusters that match the search criteria.

The layout algorithm is general enough to be applied to biclusters computed in real-valued, binary, or categorical data. For instance, the combination of biclustering algorithms and our layout algorithm can be used to analyze measurements of the concentrations of other types of molecules, including proteins and metabolites. We demonstrate our approach on two types of data. First, we compute layouts for biclusters extracted from leukemia microarray data by the xMotif biclustering algorithm [16; 28]. Second, we analyze protein-DNA binding data in *S. cerevisiae* and demonstrate how

*This author performed the research at the Virginia Polytechnic Institute and State University.

biclustering in combination with the layout algorithm can visually demonstrate differences in the transcriptional regulatory network that is activated in different growth conditions.

Figure 1 displays a layout computed by our algorithm on a toy binary matrix. Figure 1(a) displays a dataset in which rows represent dates and columns represent weather conditions in Blacksburg. A cell has a one (the cell is drawn shaded) if the weather condition corresponding to the cell's column (e.g., "Rainy" or "$>$ 75° F") is true on the date corresponding to the cell's row. In this dataset, we define a bicluster to be an itemset, i.e., a subset of rows and a subset of columns with the property that the submatrix defined by these rows and columns only contains ones. We computed all the closed biclusters in this binary matrix, i.e., biclusters with the property that every row (respectively, every column) not in the bicluster contains a zero in at least one column (respectively, one row) in the bicluster. Figure 1(b) displays the layout computed by our algorithm of the seven biclusters in this dataset. .



(a) Weather conditions in Blacksburg.

(b) Layout of the biclusters in the weather data.

Figure 1: An example of a bicluster layout for weather data in Blacksburg, VA.

The bicluster layout problem, which we formally define in Section 3.1, is very similar to the hypergraph superstring problem studied by Batzoglou and Istrail study in the context of physical mapping of genomes. Batzoglou and Istrail prove that the hypergraph superstring problem is MAX-SNP Hard, i.e., it is computationally intractable to obtain a bicluster layout whose size is smaller than a constant factor of the optimal size. In this work, we present a heuristic that minimizes the size of the layout well in practice. In the special case when there is a solution involving no repeated rows or columns, the algorithm computes the layout of smallest size. Our algorithm runs in $O(mn^2 + n^2 \log n)$ where n is the number of biclusters and m is the number of rows and columns in all the biclusters; the running time of the algorithm is independent of the size of the original dataset. We lay out the rows and columns of the biclusters independently. Our algorithm to lay out the rows is similar to a bottom-up hierarchical clustering of the rows of the biclusters. At each stage, we merge two biclusters if the submatrix induced by them in the original matrix has the "consecutive ones property" (see Section 3.2). Finally we generate the two-dimensional layout by combining the row and column layouts.

2. RELATED WORK

A binary matrix has the *Consecutive Ones Property* (COP) for rows if its columns can be permuted such that all the ones in each row are consecutive [7]. See Figure 2 for an example of a matrix with the COP. Determining whether a matrix has the COP and computing the permutation of the columns that proves this property has applications in a number of areas including testing for graph planarity [7] and recognizing interval graphs [7; 18]. Booth and Leuker [7] describe a data structure called the PQ tree which they use to represent all legal permutations of column orderings in a matrix with the COP property. They prove that the PQ tree and the correct column permutation can be computed in time linear in the number of ones in the matrix.

$$\begin{pmatrix} \mathbf{1} & \mathbf{0} & 1 & 1 & 0 \\ \mathbf{0} & \mathbf{0} & 1 & 1 & 1 \\ \mathbf{1} & \mathbf{0} & 1 & 0 & 0 \\ \mathbf{1} & \mathbf{1} & 1 & 1 & 1 \\ \mathbf{0} & \mathbf{0} & 0 & 1 & 1 \end{pmatrix}$$

(a) A matrix that has the COP with the first two columns highlighted.

$$\begin{pmatrix} \mathbf{0} & \mathbf{1} & 1 & 1 & 0 \\ \mathbf{0} & \mathbf{0} & 1 & 1 & 1 \\ \mathbf{0} & \mathbf{1} & 1 & 0 & 0 \\ \mathbf{1} & \mathbf{1} & 1 & 1 & 1 \\ \mathbf{0} & \mathbf{0} & 0 & 1 & 1 \end{pmatrix}$$

(b) Swapping the first two columns of the matrix demonstrates that the matrix has the COP.

Figure 2: An illustration of the COP.

Researchers have studied a number of generalizations of the COP problem; however, most of these generalizations are NP-complete or NP-Hard. For example, seeking the column ordering for a non-COP matrix that minimizes the number of gaps between the ones in each row can be reduced to the traveling salesman problem [12]. An important generalization studied in bioinformatics is one where we are allowed to repeat as well as rearrange columns in order to ensure that the consecutive ones of every original row occur in at least one contiguous set of columns in that row. As mentioned earlier, in their study of physical mapping of chromosomes, Batzoglou and Istrail prove that this problem is MAXSNP-Hard [3]. The most common application of this generalization of the COP is physical mapping of chromosomes with probes. We can represent physical mapping data as a binary matrix where the rows represent clones (short overlapping sections of a chromosome), the columns represent DNA probes, and a cell in the matrix has a one if the corresponding probe hybridizes to the corresponding clone. Constructing a physical map of the chromosome is equivalent to finding an ordering of the probes such that all the probes matching a clone appear consecutively and the total length of the ordering is as small as possible.

Algorithms for constructing physical maps from hybridization data typically exploit the Lander-Waterman model [21], which assumes that clones are distributed uniformly across the chromosome and that probes are distributed according to independent Poisson processes. Some algorithms make additional domain-specific assumptions [3; 12; 19; 25; 27]. For instance, Batzoglou and Istrail compute an ordering whose length is at most twice the length of the optimal ordering under the requirement that each clone must contain a probe that does not hybridize to any other clone. None of these algorithms are applicable to our problem since the biclusters we want to lay out may not have the required properties.

3. ALGORITHM

We present our approach in four stages. First, we define some useful notation. Second, we introduce the PQ-tree, a data structure that is fundamental to our approach. Third, we present our layout algorithm. Finally, we discuss its implementation and the web interface.

3.1 Definitions

We denote the input matrix by D and use R and C to denote the set of rows and columns of D, respectively. Given subsets $R' \subseteq R$ and $C' \subseteq C$, we define a *bicluster* $B(R', C')$ to be the sub-matrix of D spanned by the rows in R' and the columns in C'.[1] A *layout* $\mathcal{L}(\mathcal{R}, \mathcal{C})$ of the matrix D is a two-dimensional matrix specified as follows:

1. $\mathcal{R}$ is the ordered list of rows of $\mathcal{L}$ with the property that each element of $\mathcal{R}$ is an element of R; a row in R can appear multiple times in $\mathcal{R}$.

2. $\mathcal{C}$ is the ordered list of of columns of $\mathcal{L}$ with the property that each element of $\mathcal{C}$ is an element of C; a column in C can appear multiple times in $\mathcal{L}$.

3. $\mathcal{L}_{ij}$, the element in the ith row of $\mathcal{R}$ and the jth column of $\mathcal{C}$ is equal to $D_{i'j'}$, where i' is the row of D corresponding to the the ith row of $\mathcal{L}$ and j' is the column of D corresponding to the jth column of $\mathcal{R}$.

The *size* of $\mathcal{L}$ is $|\mathcal{R}||\mathcal{C}|$. It is appropriate to consider $\mathcal{L}$ to be a layout of D since $\mathcal{L}$ specifies an order for the rows and columns of D. In the example in Figure 1(b), the layout does not contain any repeated rows or columns. The layout does not contain the column titled "< 35 F" that is in the original matrix either.

A bicluster $B(R', C')$ is *contiguous* in a layout $\mathcal{L}(\mathcal{R}, \mathcal{C})$ if and only if the elements of R' (respectively, C') appear consecutively at least once in $\mathcal{R}$ (respectively, $\mathcal{C}$). We say that the layout $\mathcal{L}(\mathcal{R}, \mathcal{C})$ is *valid* with respect to a set of biclusters S if every bicluster $B \in S$ is contiguous in $\mathcal{L}(\mathcal{R}, \mathcal{C})$. For example, the layout in Figure 1(b) is valid with respect to the bicluster ({7/04/2004, 7/03/2004, 7/02/2004 }, { > 60F, Daylight > 10h, Cloudy, Rainy }) since the bicluster spans rows four to six and columns two to five in the layout. We now formally define the *bicluster layout* problem: Given a matrix D and a set S of biclusters in D, find a layout $\mathcal{L}$ of D such that $\mathcal{L}$ is valid with respect S and $\mathcal{L}$ has the smallest size among all valid layouts of D.

3.2 PQ Trees

Booth and Leuker [7] developed a data structure called the PQ tree, which they used to compute a column ordering that proves that that a binary matrix M has the COP. To define the PQ tree, it is convenient to reformulate the COP problem as follows: Let U be the set of columns of M. Let r be the number of rows in M. For each $i, 1 \leq i \leq r$, define the set S_i to be the set of columns in U that have a one in row i. We seek a permutation of the elements of U that satisfies r *restrictions*, where restriction $i, 1 \leq i \leq r$ requires that the elements of S_i be consecutive in the permutation.

[1]This simple definition is sufficient for this paper. An algorithm that computes biclusters in gene expression data is likely to use a more complex definition relevant to the patterns to be mined.

A PQ tree can represent all legal permutations of U that satisfy the restrictions $\{S_i, 1 \leq i \leq r\}$. Each leaf of the PQ tree corresponds to a column in U. The PQ tree contains two types of internal nodes: P-nodes and Q-nodes. The children of a P-node can be permuted in any way while still satisfying the restrictions. A valid permutation of the children of a Q-node is either the order in which they appear in the PQ tree or the reversal of this order. A PQ tree supports the REDUCE operation. This operation inserts a restriction S into a PQ tree T. The operation modifies T such that T satisfies S in addition to all the previous restrictions inserted into T. The REDUCE operation fails if there are no legal permutations of U that can satisfy S and the previously inserted restrictions. The REDUCE operation takes time linear in $|S|$. Figure 3 displays a PQ tree on four elements $\{a, b, c, d\}$ after two REDUCE operations: REDUCE(T,$\{a, c\}$) and REDUCE(T,$\{b, c\}$). Inserting the restriction $\{c, d\}$ into the tree next will result in a failed REDUCE operation.

To solve the COP problem, start with an empty PQ tree T. For each $i, 1 \leq i \leq r$, invoke the operation REDUCE(T, S_i). To obtain an ordering that satisfies the restrictions, perform a breadth-first traversal of T starting at the root. At each internal node of T, visit the children of the node in an order specified by the type of the node. At a leaf node of T, append the column corresponding to the leaf to the required ordering.

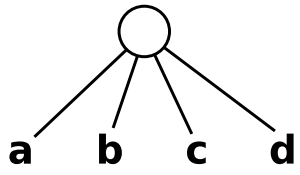
3.3 The Bicluster Layout Algorithm

We are now ready to describe our algorithm for the bicluster layout problem. To minimize the size of $\mathcal{L}$, we can minimize the length of $\mathcal{R}$ and the length of $\mathcal{C}$ independently. Therefore, we construct the layout $\mathcal{L}$ by determining $\mathcal{R}$ and $\mathcal{C}$ independently. In the rest of this section, we describe the algorithm to construct $\mathcal{C}$, the ordered list of the columns in the layout $\mathcal{L}$. We can compute $\mathcal{R}$, the ordered list of rows in the layout, analogously.

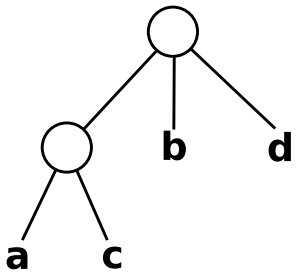
We describe the algorithm in two stages. We first transform the problem of constructing $\mathcal{C}$ to a generalization of the COP problem. We then present an algorithm to solve this transformed problem. This transformation allows us to describe our algorithm in terms of operations on PQ trees. The PQ tree cannot solve this generalization directly since the matrix we construct may not have the COP.

We start by constructing a new binary matrix M that represents the columns of the biclusters in S. Each column on M corresponds to a column of the input matrix D. M contains one row for each bicluster in S; thus, M has n rows. The entry M_{ij} is 1 if the ith bicluster in S contains the column j in D; otherwise, M_{ij} is 0. We can now reformulate the problem of constructing $\mathcal{C}$ as follows: find the shortest linear ordering $\mathcal{C}$ of the columns of M such that $\mathcal{C}$ can contain repeated columns of M and for every row of M, the columns containing the ones in that row appear consecutively at least once in $\mathcal{C}$.

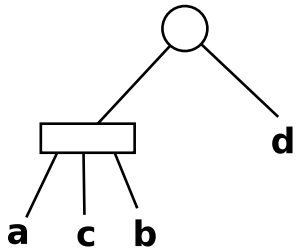
Before describing the algorithm, we define some more notation. The leaves of each PQ tree constructed by the algorithm correspond to a subset of the columns of M. We use C_T to denote the set of columns in a PQ tree T. Given two PQ trees T and T', let $\sigma(T, T')$ denote the set similarity $\frac{|C_T \cap C_{T'}|}{|C_T \cup C_{T'}|}$ between the columns in T and T'. Our algorithm executes the following steps:



(a) Initial PQ tree T formed from set $\{a, b, c, d\}$.



(b) The PQ tree T after the REDUCE(T,$\{a, c\}$) operation, requiring that a and c be consecutive.



(c) The PQ tree T after the REDUCE(T,$\{b, c\}$) operation, requiring that b and c be consecutive.

Figure 3: An example of a PQ tree. Circles represent P nodes and rectangles represent Q nodes. Valid permutations represented by the tree in Figure 3(c) are the sequences *acbd*, *bcad*, *dacb*, and *dbca*.

1. For each row i of M, $1 \leq i \leq n$, construct a PQ tree T_i consisting of a single P-node, whose children are the columns in M that contain ones in row i of M. Let $\mathcal{T}$ be the set of these n PQ trees.
2. For every pair $1 \leq i < j \leq n$, compute the set similarity $\sigma(T_i, T_j)$.
3. Compute Σ, the list of values in $\{\sigma(T_i, T_J), 1 \leq i < j \leq n\}$ sorted in descending order.
4. Repeat the following steps until Σ is empty:
   (a) Remove the largest element from Σ. Let T and T' be the PQ trees in $\mathcal{T}$ with this similarity value.
   (b) Set $T'' = T$.
   (c) For each restriction r inserted into T', invoke the operation REDUCE(T'', r). If any reduce operation fails, go to Step 4a.
   (d) Delete T and T' from $\mathcal{T}$.
   (e) For each tree $U \in \mathcal{T}$, insert $\sigma(U, T'')$ into Σ.
   (f) Insert T'' into $\mathcal{T}$.
5. For each PQ tree T in $\mathcal{T}$, traverse T to compute a valid permutation of the columns in C_T.
6. Output the column layout formed by concatenating (in any order) the permutations computed in Step 5.

The algorithm starts by storing each row of M (recall that each row of M corresponds to a bicluster) in a separate PQ tree in the set $\mathcal{T}$ (Step 1). Next, the algorithm performs a series of REDUCE operations to hierarchically cluster the rows of M. Inductively, each PQ tree in $\mathcal{T}$ corresponds to a set of rows of M with the property that the submatrix of M defined by these rows has the COP. To decide which two sets of rows to merge next, in Step 4a, the algorithm picks the two PQ trees T and T' in $\mathcal{T}$ that are the most similar and attempts to merge them. To effect the merger, the algorithm adds the restrictions added to one of these PQ trees to the other PQ tree (Step 4c). If this step succeeds, the algorithm deletes T and T' from $\mathcal{T}$, inserts the similarities between the new PQ tree T'' and each of the remaining PQ trees in $\mathcal{T}$ into Σ, and inserts T'' into $\mathcal{T}$ (Steps 4d–4f).

In Step 4c, the failure of a REDUCE operation means that the restrictions in T are not compatible with the restrictions imposed by T'. Hence, the submatrix of M induced by the union of rows in T and in T' does not have the COP. An example of such a situation is when T corresponds to the tree in Figure 3(c) and T' contains the restriction $\{c, d\}$. In this case, the algorithm aborts the merger of T and T' and moves on to the next most similar pair of PQ trees. Due to such conflicts, $\mathcal{T}$ may contain more than one PQ tree when the algorithm completes. Finally, generating the required layout is a simple matter of traversing each PQ tree in $\mathcal{T}$ (Step 5) as described in Section 3.2 and concatenating the resulting permutations into a single order (Step 6). A column of M appears as many times in this order as there are PQ trees in $\mathcal{T}$ that includes this column.

We now analyze the running time of the algorithm. Let m be the number of ones in the matrix M. In Step 1, computing the PQ trees takes $O(m)$ time. Computing the similarity between a pair of PQ trees takes $O(c)$ time, where c is the number of columns of M. In Step 2 and 3, computing and sorting the $O(n^2)$ similarity values takes $O(cn^2 + n^2 \log n)$ time. We execute Step 4 $O(n^2)$ times. The running time of each iteration is proportional to the size of the new PQ tree constructed. A naive upper bound on this size is m, the total number of columns in all the biclusters. Hence, the total running time of Step 4 is $O(mn^2)$. Finally, traversing all the PQ trees in $\mathcal{T}$ and concatenating the permutations takes $O(m)$ time. Keeping in mind that $c \leq m$, the total running time of the algorithm is $O(mn^2 + n^2 \log n)$. The space occupied by the algorithm is $O(m + n^2)$, with $O(m)$ space taken to store all the biclusters and the PQ trees and $O(n^2)$ required for Σ, the sorted list of similarities.

3.4 Implementation and Web Interface

We implemented the layout algorithm in C++ and tested it on a 2.8GHz Pentium computer running the Fedora Core 3 operating system. Our software contains two executable programs. The first executable, `layout`, implements the layout algorithm. It takes a text file describing the biclusters as input and outputs the layout as a text file list of rows and columns. The second executable, `drawlayout`, uses this text file and the original data set as input and produces an image corresponding to the layout.

If the input data contains a large number of biclusters, the layout may contain too many rows and/or columns for the user to navigate with ease. To alleviate this problem, we have also developed a simple web-based interface that allows the user to upload a file containing computed biclusters and a file containing the original data, and search the biclusters with the names of rows and columns. The interface invokes `layout` and `drawlayout` on the biclusters that contain the query rows/columns and highlights the matching biclusters, rows, and columns in the resulting layout. The interface allows the user to specify whether the data is real-valued or binary, whether the layout should contain only the matching biclusters, and whether the query should be a conjunction or disjunction of the search terms.

4. EXPERIMENTAL RESULTS

4.1 Synthetic Data

We created synthetic datasets with different numbers of rows and columns. For each dataset, we generated biclusters by sampling subsets of rows and columns. For this experiment, we randomly generate the number of rows and columns and identifiers for the rows and columns; we did not need to generate values for the cells of the matrices. For each set of biclusters, we recorded the time required to run our layout algorithm and the number of rows and columns in the computed layout. For each layout, we estimated the *layout efficiency* of the layout as the ratio of the size of the layout to the size of the dataset. Lower values of efficiency are better than higher values, since they indicate that the algorithm is able to exploit overlaps between biclusters. For each choice of number of rows in the dataset, number of columns in the dataset, and number of biclusters, we averaged the results for 100 runs. Tables 1 and 2 display our results. Efficiency values may be less than one, e.g., when some rows or columns in the dataset do not belong to any bicluster.

Table 1: Execution times (in seconds) for the layout algorithm on synthetic matrices.

#biclusters	#rows + #columns in the dataset				
	10	30	50	70	90
20	0.168	0.328	0.462	0.52	0.532
40	1.23	2.514	3.046	3.574	4.008
60	4.074	7.992	11.238	11.71	12.81
80	9.484	19.586	25.546	29.652	29.446
100	17.982	37.966	48.418	50.916	56.112

Table 2: Efficiency values of the layout algorithm on synthetic matrices.

# biclusters	#rows + #columns in the dataset				
	10	30	50	70	90
20	0.184	0.842	1.316	1.254	1.428
40	0.304	1.16	1.632	2.04	2.074
60	0.398	1.496	2.262	2.26	2.508
80	0.512	1.65	2.358	2.726	2.698
100	0.48	1.808	2.582	2.686	2.996

4.2 Transcriptional Regulation in *S. cerevisiae*

To demonstrate the ability of our visualization algorithm to highlight differences between biclusters in similar datasets, we analyzed datasets of transcriptional regulation in two experimental conditions in *S. cerevisiae* [2; 23]. Each dataset is a binary matrix whose columns represent transcription factors and whose rows represent genes in *S. cerevisiae*. A matrix entry contains a one if a ChIP-on-chip experiment indicates that the transcription factor binds to the promoter of the gene. An important problem that arises in the analysis of this data is determining if a set of genes are collectively regulated by a set of transcription factors and whether this combinatorial regulation changes when the cell is exposed to stress. Although ChIP-on-chip data is noisy and significant effort may be needed to clean it up, the analysis we present next demonstrates that a combination of biclustering and our layout algorithm has the potential to yield biologically useful results.

The two protein-DNA datasets we study correspond to growth of *S. cerevisiae* cells in rich medium [23] and to growth under exposure to rapamycin [2], a condition that mimics nutrient starvation. We restricted our attention to transcription factors studied in both papers. We ran our implementation of the *Apriori* algorithm [1] that computes closed itemsets on both these datasets, applied our layout algorithm on biclusters with at least two genes and at least two transcription factors, and obtained the layout in Figure 4(a). Biclusters obtained from the data under growth in rich medium are shown as blue boxes and rapamycin-induced biclusters are shown as red boxes. A cell in the figure is dark grey (respectively, light grey) if the transcription factor binds to the gene's promotor in both (respectively, one) condition. The image strikingly demonstrates that under exposure to rapamycin, the transcriptional network activated in the cell is very different from the normal activation network. The rich medium data contains only four biclusters involving these transcription factors while the rapamycin data contains 38 biclusters. We conclude that very few genes are co-regulated by the same set of transcription factors in both conditions.

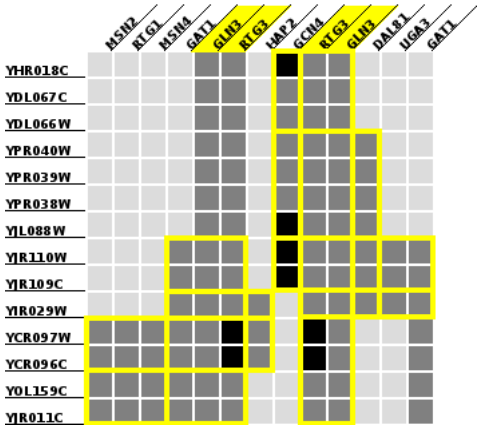
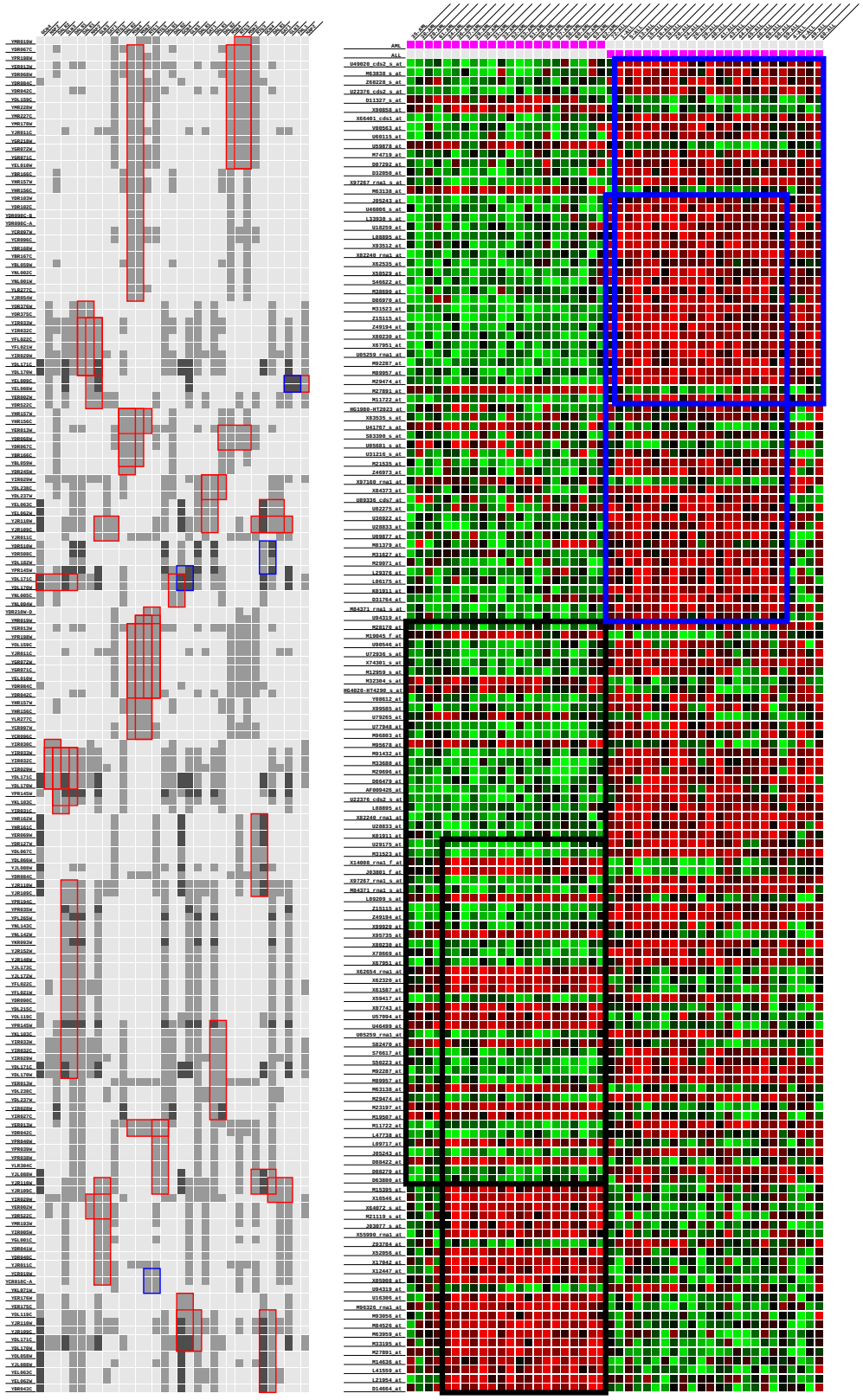


Figure 5: Genes combinatorially controlled by GLN3 and RTG3.

To illustrate the use of our web interface, we used it to search for biclusters that included the transcription factors RTG3 and GLN3. RTG3 is a transcription factor that forms a com-



(a) Combinatorial control of transcription in *S. cerevisiae*.

(b) Biclusters in gene expression data for ALL and AML.

Figure 4: Visualizations of the layouts computed by our algorithm.

plex with RTG1 to activate the retrograde (RTG) and target of rapamycin (TOR) pathways [10; 30]. GLN3 encodes a transcription factor that is phosphorylated and localised to the cytoplasm when the cell is grown in nitrogen-rich media. Rapamycin treatment can induce the dephosphorylation and subsequent activation of GLN3 [4]. Figure 5 displays the layout of all the biclusters containing these two transcription factors. We note that each bicluster also includes either the transcription factor GAT1 or the transcription factor GCN4. GAT1 is a transcriptional activator of genes involved in nitrogen catabolite repression; the activity and localization of these genes is regulated by nitrogen limitation. GCN4 is another transcription activator that is a master regulator of gene expression during amino acid starvation in *S. cerevisiae* and is activated in multiple stress responses [29]. Thus, it is not suprising that GAT1 and GCN4 co-regulate genes with GLN3 and RTG3. The functional annotations of the set of nine genes targeted by GCN4, GLN3, and RTG3 are enriched in the Gene Ontology biological process "glutamine family amino acid biosynthesis" with a p-value of 2×10^{-8} (based on the hypergeometric distribution), indicating that this pathway may be activated by the three transcription factors upon rapamycin treatment.

4.3 Classification of Leukemias

Golub et al. [15] studied global expression patterns of 45 patients diagnosed with Acute Lymphoblastic Leukemia (ALL) and 27 patients diagnosed with Acute Myeloid Leukemia (AML). We ran the xMotif algorithm [16; 28] to compute biclusters in this dataset. We ensured that computed biclusters contain samples from at most one class. We selected four representative biclusters from the results to visualize. Figure 4(b) displays the layout. Each column corresponds to a sample; the two columns at the top with purple cells indicate the type of leukemia. We map the expression values of each gene into a range from green to red, with green (respectively, red) corresponding to the smallest (respectively, largest) expression value of that gene. The biclusters outlined in black correspond to AML samples and those outlined in blue to ALL samples. This layout visually highlights similarities and differences between the biclusters found in samples for the same and for different types of leukemia. We have used such biclusters as the basis for constructing a classifier that distinguishes between different diseases and tissues [16].

5. CONCLUSIONS

The biomedical community has access to large quantities of publicly-available gene expression datasets. Biclustering has emerged as a powerful methodology for analyzing these datasets. In this paper, we have introduced a novel algorithm for laying out biclusters in a two-dimensional matrix so as to reveal the overlaps and relationships between the biclusters. The algorithm performs efficiently in practice. We have demonstrated the applicability of the algorithm to two important problems in bioinformatics using both binary and real-valued data. An easy-to-use web interface distributed with the layout software allows the user to query and navigate layouts that are too large to study manually. Biclustering is useful not just for processing gene expression data but for any dataset that measures the relationships between two different types of data, for example, genes and functions, transcription factors and promotors, microRNAs and their target mRNAs, genes and diseases, etc. Thus, our algorithm has the potential to be useful for a wide variety of bioinformatic applications.

6. REFERENCES

[1] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the Twentieth International Conference on Very Large Databases*, pages 487–499, Santiago, Chile, 1994.

[2] Z. Bar-Joseph, G. K. Gerber, T. I. Lee, N. J. Rinaldi, J. Y. Yoo, F. Robert, D. B. Gordon, E. Fraenkel, T. S. Jaakkola, R. A. Young, and D. K. Gifford. Computational discovery of gene modules and regulatory networks. *Nat Biotechnol*, 21(11):1337–42, 2003.

[3] S. Batzoglou and S. Istrail. Physical mapping with repeated probes: The hypergraph superstring problem. *Journal of Discrete Algorithms*, 1:51–76, 2000.

[4] T. Beck and M. N. Hall. The TOR signalling pathway controls nuclear localization of nutrient-regulated transcription factors. *Nature*, 402(6762):689–92, 1999.

[5] S. Bergmann, J. Ihmels, and N. Barkai. Iterative signature algorithm for the analysis of large-scale gene expression data. *Phys Rev E Stat Nonlin Soft Matter Phys*, 67(3 Pt 1):031902, 2003.

[6] S. Bergmann, J. Ihmels, and N. Barkai. Similarities and differences in genome-wide expression data of six organisms. *PLoS Biol*, 2(1):E9, 2003.

[7] K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and planarity using pq-tree algorithms. *J. Comput. Sys. Sci.*, 13:335–379, 1976.

[8] A. Califano, G. Stolovitzky, and Y. Tu. Analysis of gene expression microarrays for phenotype classification. *Proc Int Conf Intell Syst Mol Biol*, 8:75–85, 2000.

[9] Y. Cheng and G. Church. Biclustering of expression data. *Proc Int Conf Intell Syst Mol Biol*, 8:93–103, 2000.

[10] J. L. Crespo, T. Powers, B. Fowler, and M. N. Hall. The TOR-controlled transcription activators GLN3, RTG1, and RTG3 are regulated in response to intracellular levels of glutamine. *Proc Natl Acad Sci U S A*, 99(10):6784–9, 2002.

[11] M. Eisen, P. Spellman, P. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proc. Natl. Acad. Sci. USA*, 95:14863–14868, 1998.

[12] D. W. F. Alizadeh, R.M. Karp and G. Zweig. Physical mapping of chromosomes using unique probes. *Journal of Computational Biology*, 2:159–184, 1995.

[13] S. Fodor, R. Rava, X. Huang, A. Pease, C. Holmes, and C. Adams. Multiplexed biochemical assays with biological chips. *Nature*, 364(6437):555–6, 1993.

[14] G. Getz, E. Levine, and E. Domany. Coupled two-way clustering of dna microarray data. *Proc. Natl Acad. Sci. USA*, 97:12079–12084, 2000.

[15] T. Golub, D. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. Mesirov, H. Coller, M. Loh, J. Downing, M. Caligiuri, C. Bloomfield, and E. S. Lander. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science*, 286:531–537, 1999.

[16] G. Grothaus and T. M. Murali. Biologically-interpretable disease and tissue classification based on dna microarray data. In preparation, 2006.

[17] J. A. Hartigan. Direct clustering of a data matrix. *J. Amer. Statis.*, 67:123–129, 1972.

[18] W.-L. Hsu. A simple test for the consecutive ones property. *J. Algorithms*, 43(1):1–16, 2002.

[19] T. Jiang and R. Karp. Mapping clones with a given ordering or interleaving. *Algorithmica*, 21:262–284, 1998.

[20] Y. Kluger, R. Basri, J. Chang, and M. Gerstein. Spectral biclustering of microarray data: coclustering genes and conditions. *Genome Res*, 13(4):703–16, 2003.

[21] E. Lander and M. Waterman. Genomic mapping by fingerprinting random clones: A mathematical analysis. *Genomics*, 2:231–239, 1988.

[22] L. Lazzeroni and A. Owen. Plaid models for gene expression data. *Statistica Sinica*, 12:61–86, 2002.

[23] T. I. Lee, N. J. Rinaldi, F. Robert, D. T. Odom, Z. Bar-Joseph, G. K. Gerber, N. M. Hannett, C. T. Harbison, C. M. Thompson, I. Simon, J. Zeitlinger, E. G. Jennings, H. L. Murray, D. B. Gordon, B. Ren, J. J. Wyrick, J.-B. Tagne, T. L. Volkert, E. Fraenkel, D. K. Gifford, and R. A. Young. Transcriptional Regulatory Networks in Saccharomyces cerevisiae. *Science*, 298(5594):799–804, 2002.

[24] J. Lepre, J. Rice, Y. Tu, and G. Stolovitzky. Genes@Work: an efficient algorithm for pattern discovery and multivariate feature selection in gene expression data. *Bioinformatics*, 20(7):1033–44, 2004.

[25] W.-F. Lu and W.-L. Hsu. A test for the consecutive ones property on noisy data-application to physical mapping and sequence assembly. *Journal of Computational Biology*, 10(5):709–735, 2003.

[26] S. C. Madeira and A. L. Oliveira. Biclustering algorithms for biological data analysis: A survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2004. to appear.

[27] G. Mayraz and R. Shamir. Construction of physical maps from oligonucleotide fingerprints data. *RECOMB*, 1999.

[28] T. M. Murali and S. Kasif. Extracting conserved gene expression motifs from gene expression data. In *Proceedings of the Pacific Symposium on Biocomputing*, 2003. 77–88.

[29] K. Natarajan, M. R. Meyer, B. M. Jackson, D. Slade, C. Roberts, A. G. Hinnebusch, and M. J. Marton. Transcriptional profiling shows that Gcn4p is a master regulator of gene expression during amino acid starvation in yeast. *Mol Cell Biol*, 21(13):4347–68, 2001.

[30] B. A. Rothermel, J. L. Thornton, and R. A. Butow. Rtg3p, a basic helix-loop-helix/leucine zipper protein that functions in mitochondrial-induced changes in gene expression, contains independent activation domains. *J Biol Chem*, 272(32):19801–7, 1997.

[31] M. Schena, D. Shalon, R. Davis, and P. Brown. Quantitative monitoring of gene expression patterns with a complementary dna microarray. *Science*, 270(5235):467–70, 1995.

[32] E. Segal, A. Battle, and D. Koller. Decomposing gene expression into cellular processes. *Pac Symp Biocomput*, pages 89–100, 2003.

[33] E. Segal, M. Shapira, A. Regev, D. Botstein, D. Koller, and N. Friedman. Module networks: identifying regulatory modules and their condition-specific regulators from gene expression data. *Nat Genet*, 34(2):166–76, 2003.

[34] Q. Sheng, Y. Moreau, and B. De Moor. Biclustering microarray data by Gibbs sampling. *Bioinformatics*, 19 Suppl 2:II196–II205, 2003.

[35] A. Tanay, R. Sharan, M. Kupiec, and R. Shamir. Revealing modularity and organization in the yeast molecular network by integrated analysis of highly heterogeneous genomewide data. *Proc Natl Acad Sci U S A*, 101(9):2981–6, 2004.

[36] A. Tanay, R. Sharan, and R. Shamir. Discovering statistically significant biclusters in gene expression data. In *Proceedings of ISMB 2002*, pages S136–S144.

[37] A. Tanay, R. Sharan, and R. Shamir. *Handbook of Bioinformatics*, chapter Biclustering Algorithms: A Survey. 2004.