Enhanced Biclustering on Expression Data

Jiong Yang
UIUC
jioyang@cs.uiuc.edu

Haixun Wang
IBM T. J. Watson
haixun@us.ibm.com

Wei Wang
UNC Chapel Hill
weiwang@cs.unc.edu

Philip Yu
IBM T. J. Watson
psyu@us.ibm.com

Abstract

Microarrays are one of the latest breakthroughs in experimental molecular biology, which provide a powerful tool by which the expression patterns of thousands of genes can be monitored simultaneously and are already producing huge amount of valuable data. The concept of bicluster was introduced by Cheng and Church (2000) to capture the coherence of a subset of genes and a subset of conditions. A set of heuristic algorithms were also designed to either find one bicluster or a set of biclusters, which consist of iterations of masking null values and discovered biclusters, coarse and fine node deletion, node addition, and the inclusion of inverted data. These heuristics inevitably suffer from some serious drawback. The masking of null values and discovered biclusters with random numbers may result in the phenomenon of random interference which in turn impacts the discovery of high quality biclusters. To address this issue and to further accelerate the biclustering process, we generalize the model of bicluster to incorporate null values and propose a probabilistic algorithm (FLOC) that can discover a set of k possibly overlapping biclusters simultaneously. Furthermore, this algorithm can easily be extended to support additional features that suit different requirements at virtually little cost. Experimental study on the yeast gene expression data shows that the FLOC algorithm can offer substantial improvements over the previously proposed algorithm.

1 Introduction

Microarrays are one of the latest breakthroughs in experimental molecular biology, which provide a powerful tool by which the expression patterns of thousands of genes can be monitored simultaneously and are already producing huge amount of valuable data. Analysis of such data is becoming one of the major bottlenecks in the utilization of the technology. The gene expression data are organized as matrices — tables where rows represent genes, columns represent various samples such as tissues or experimental conditions, and numbers in each cell characterize the expression level of the particular gene in the particular sample. Investigations show that more often than not, several genes contribute to the same pathway, which motivates researchers to identify a subset of genes whose expression levels rise and fall coherently under a subset of conditions, that is, they exhibit fluctuation of a similar shape when conditions change. Discovery of such clusters of genes is essential in revealing the significant connections in gene regulatory networks.

The concept of *bicluster* was introduced by Cheng and Church (2000) to capture the coherence of a subset of genes and a subset of conditions. Unlike previous methods that treat similarity as a function of pairs of genes or pairs of conditions, the bicluster model measures coherence within the subset of genes and conditions. This model may be particularly useful to disclose the involvement of a gene or a condition in multiple pathways, some of which can only be discovered under the dominance of more consistent ones. The coherence score is defined as a symmetric function of genes and conditions involved and thereby the biclustering is a process of simultaneous grouping of genes and conditions. The so called *mean squared residue* was employed and was applied to expression data transformed by a logarithm and augmented by the additive inverse. While the mean squared residue represents the variance of the selected genes and conditions with respect to the coherence, the goal of biclustering is to find biclusters with low mean squared residue. In gene expression data analysis, this goal is often accompanied with an additional requirement of reasonably large row variance. The rationale is that a low mean squared residue only indicates that the gene expression levels fluctuate approximately in unison, which also includes the constant biclusters where there is no or little fluctuation at all. These trivial biclusters may not be as interesting as the biclusters where the set of genes show strikingly similar up-regulation and down-regulation under the set of conditions. It has been proven that the problem of finding biclusters satisfying these criteria is NP-hard in general. Therefore, a set of heuristic algorithms were designed by Cheng and Church (2000) to either find one bicluster or a set of biclusters, which consist of iterations of masking null values and discovered biclusters, coarse and fine node deletion, node addition, and the inclusion of inverted data. The computational complexities are in the order of $O(MN \times (M+N) \times k)$ for discovering k biclusters where M and N are the number of conditions and the number of genes, respectively. The proposed heuristics, which have been demonstrated to be able to produce good quality biclusters, inevitably suffer from some serious drawback. The masking of null values and discovered biclusters was performed by replacing the relevant cells with random numbers. The rationale of replacing missing values with random numbers was that these random values only have a mathematical chance to form any recognizable pattern and therefore would not result in distorted biclustering. The original intention of masking discovered bicluster was to ensure that each successive run of the (deterministic) algorithm outputs a different bicluster in the case where multiple biclusters are preferred. In both cases, even though the random data is unlikely to form any fictitious pattern, there exists a substantial risk that these random numbers will interfere with the future discovery of biclusters, especially those ones that have overlap with the discovered ones. We call this phenomenon the *random interference*. Our experimental study has confirmed that this random interference will impact the biclustering result.

To address this issue and to further accelerate the biclustering process, we generalize the model of bicluster to incorporate null values and propose a probabilistic algorithm (FLOC[1]) that can discover a set of k possibly overlapping biclusters simultaneously. Furthermore, this algorithm can easily be extended to support additional features that suit different application needs at virtually little cost. Typical features include the maximum amount of overlap allowed between biclusters, the maximum/minimum size of each bicluster, the minimum overall coverage of the biclusters, and so on. The general process of FLOC consists of iterations of series of gene and condition moves (i.e., selections or deselections) aiming at achieving the best potential residue reduction. During the course of biclustering, certain move may be "blocked" temporarily if performing such move would lead to an unfavorable situation such as producing a trivial bicluster or violating one or more feature constraints. We implemented FLOC to find 100 biclusters on the same yeast data containing 2884 genes and 17 conditions with the same parameter setting as in Cheng and Church (2000) and found that the biclusters returned by FLOC, on average, have a comparable mean squared residue but a larger size than that reported by Cheng and Church (2000), largely because FLOC avoids the random interference suffered by the Cheng-Church algorithm. In addition, FLOC is able to locate these biclusters much faster than the algorithms proposed in Cheng and Church (2000).

2 The General Model of Bicluster

In this section, we formally present the *generalized* bicluster model that can handle null values in a seamless manner. (In the remaining of this paper, we use the term of biclusters to refer to the generalized biclusters.) A bicluster is defined on a gene-expression matrix. Let $\Im = \{A_1, A_2, \ldots, A_M\}$ be the set of conditions and $\Re = \{O_1, O_2, \ldots, O_N\}$ be the set of genes. The data can be viewed as an $M \times N$ matrix D of real numbers. Each entry d_{ij} in this matrix corresponds to the logarithm of the relative abundance of the mRNA of a gene O_i under a specific condition A_j, and may have a null value.

A *bicluster* essentially corresponds to a submatrix that exhibits some coherent tendency. Formally, each bicluster can be uniquely identified by the set of relevant genes and conditions. Even though allowing missing values brings great flexibility to the bicluster model, the amount of missing entries in a bicluster should be limited to some extent to avoid trivial cases. The rule of the thumb is that, despite the missing values, there should still be sufficient evidence to demonstrate the coherency. We introduce a parameter α (which is a positive number less than or equal to 1) to limit the amount of missing values for each gene and each condition in a bicluster.

	cond 1	cond 2	cond 3	cond 4
gene 1	1		3	
gene 2		4		5
gene 3		3	4	

(a) not a valid bicluster

	cond 1	cond 2	cond 3	cond 4
gene 1	1		3	3
gene 2	3	4		5
gene 3		3	4	4

(b) a bicluster

Figure 1. Missing Values in Biclusters

Definition 2.1 *For a given matrix $\Im \times \Re$ and an occupancy threshold α, a* **bicluster** *(of α occupancy) can be represented by a pair (I, J) where $I \subseteq \{1, \ldots, M\}$ is a subset of genes and $J \subseteq \{1, \ldots, N\}$ is a subset of conditions. For each gene $i \in I$, $\frac{|J'_i|}{|J|} > \alpha$ where $|J'_i|$ and $|J|$ are the number of specified conditions for gene i in the bicluster and the number of conditions in the bicluster, respectively. Similarly, for each condition $j \in J$, $\frac{|I'_j|}{|I|} > \alpha$ where $|I'_j|$ and $|I|$ are the number of specified genes under condition j in the bicluster and the number of genes in the bicluster, respectively.*

Let $\alpha = 0.6$, the submatrix in Figure 1 (a) is not a valid bicluster while the submatrix in Figure 1 (b) is a bicluster.

Definition 2.2 *The* **volume** *of a bicluster (I, J) (v_{IJ}) is defined as the number of specified entries d_{ij} such that $i \in I$ and $j \in J$.*

In the case that all entries are specified, $v_{IJ} = |I| \times |J|$ where $|I|$ and $|J|$ are the number of conditions and the number of genes participating in the bicluster, respectively. Figure 2(a) shows a gene expression matrix with ten genes (one for each rows) under five conditions (one for each column). The bicluster defined by picking $I = \{2, 3, 8\}$ and $J = \{1, 3, 5\}$ is shown in Figure 2(b). The volume of this bicluster is 9.

In order to properly accommodate various expression levels associated with each gene and each condition within a bicluster, we introduce a concept — *base*.

[1] FLOC stands for FLexible Overlapped biClustering

(a) a data matrix (rows: genes; columns: conditions)

genes	1	2	3	4	5
1	4392	284	4108		228
2	401	281	120	275	298
3	318	280	37	277	215
4	401	292	109	580	238
5	2857	285		271	226
6	228	290	48	285	224
7	538	272	266	277	236
8	322	288	41	278	219
9	312		40	273	232
10	329	296	33	274	228

(b) a bicluster (rows: genes; columns: conditions)

genes	1	2	3	4	5
1					
2	401		120		298
3	318		37		215
4					
5					
6					
7					
8	322		41		219
9					
10					

Figure 2. An Example of Bicluster

Definition 2.3 *For a given bicluster (I, J), the* **base** *of a gene O_i is defined as the average value of O_i for all specified conditions in J, $d_{iJ} = \frac{\sum_{j \in J'_i} d_{ij}}{|J'_i|}$ where $J'_i \subseteq J$ is the set of specified conditions in J for gene O_i. Similarly, the* **base** *of a condition A_j is the average specified value of A_j taken by all genes in I, i.e., $d_{Ij} = \frac{\sum_{i \in I'_j} d_{ij}}{|I'_j|}$ where $I'_j \subseteq I$ is the set of genes whose value is specified in condition A_j. The* **base** *of the bicluster is the average value of all specified entries of the submatrix defined by (I, J), i.e., $d_{IJ} = \frac{\sum_{i \in I, j \in J} d_{ij}}{v_{IJ}}$ where v_{IJ} is the volume of the bicluster.*

For example, we have $d_{2,J} = 273$, $d_{3,J} = 190$, $d_{8,J} = 194$, and $d_{I,1} = 347$, $d_{I,3} = 66$, $d_{I,5} = 244$, and $d_{IJ} = 219$ in Figure 2(b). While d_{iJ} and d_{Ij} take care of the potential tendency that may associate with each individual gene or condition, the value of d_{IJ} set the base point of the entire bicluster. In a perfect bicluster where each gene and condition exhibits an absolutely consistent tendency[2], the value of each entry d_{ij} can be uniquely determined by its gene base d_{iJ}, its condition base d_{Ij}, and the bicluster base d_{IJ}. The difference $d_{iJ} - d_{IJ}$ is essentially the relative tendency held by gene O_i in contrast to other genes in the bicluster. This tendency should hold exactly on the entry d_{ij} as well in a perfect bicluster. That is, $d_{ij} - d_{Ij} = d_{iJ} - d_{IJ}$. Consequently, we have $d_{ij} = d_{iJ} + d_{Ij} - d_{IJ}$. Figure 2(b) is a perfect bicluster even though the values are quite far apart (which may produce poor quality bicluster(s) of the traditional meaning). For example, the entry $d_{2,1} = d_{2,J} - d_{I,1} + d_{IJ} = 273 - 347 + 219 = 401$ and this property holds for every entry in Figure 2(b).

In practice, the bicluster may not always be perfect. The concept of *residue* is thus introduced to quantify the difference between the actual value of an entry and the expected value of an entry predicted from the corresponding gene base, condition base, and the bicluster base.

Definition 2.4 *The* **residue** *of an entry d_{ij} in a bicluster is $r_{ij} = d_{ij} - d_{iJ} - d_{Ij} + d_{IJ}$ if d_{ij} is specified. Otherwise, $r_{ij} = 0$.*

It is obvious that every entry in Figure 2(b) has a zero residue. The residue indeed serves as an indicator of the degree of coherence of an entry with the remaining entries in the bicluster given the tendency of the relevant gene and the relevant condition. The lower the residue, the stronger the coherence. To assess the overall quality of a bicluster, the **residue** of the bicluster can be defined as the mean residue of all specified entries. The mean can be in the form of either arithmetic, geometric, or square mean. In this paper, we use the square mean in the assessment of the bicluster residue as in Cheng and Church (2000).

Definition 2.5 *The* **residue** *of a bicluster (I, J) is $r_{IJ} = \frac{\sum_{i \in I, j \in J} r_{ij}^2}{v_{IJ}}$ where r_{ij} is the residue of the entry d_{ij} and v_{IJ} is the volume of the bicluster.*

In the above example, the residue of the bicluster in Figure 2(b) is 0. The lower the residue, the stronger the coherence exhibited by the bicluster, and the better the quality of the bicluster. We also refer to a bicluster (I, J) as a r**-residue bicluster** if its residue $r_{IJ} \leq r$ where r is a constant number. In addition, we may prefer the *row variance* to be relatively large to reject trivial biclusters.

Definition 2.6 *The* **row variance** *of a bicluster (I, J) is defined as $var_{I,J} = \frac{\sum_{i \in I, j \in J} (d_{ij} - d_{iJ})^2}{v_{IJ}}$.*

This accompanying score would warrant the bicluster to capture genes exhibiting fluctuating yet coherent trends under some set of conditions. The basic bicluster model can also be easily extended to support some additional features that may be very useful in many applications. (1) *The amount of overlap allowed between a pair of biclusters $Cons_o$*: Some application may require mutually exclusive biclusters while others may prefer some degree of overlap. The user can control the amount of overlap by specifying some threshold. (2) *The coverage of the biclusters $Cons_c$*: the number of genes/conditions should be covered by any of the biclusters. In some case, the user may want every gene to be covered by some bicluster. (3) *The balance between number of genes and conditions of the bicluster $Cons_b$*: the desirable ratio (or its range) between the number of genes and the number of conditions of each bicluster can be also be specified if the user prefers to find "balanced" biclusters. (4) *The volume of the final biclusters $Cons_v$*: The user can also control the volume of the final biclusters. This can be useful in the application where certain statistical significance needs to be warranted. We shall see later in this paper that our proposed algorithm can be applied with minor modification to suit the above purposes and to produce promising results.

[2]The entries of each gene (or condition) can be exactly generated by shifting the entries of other genes (or conditions) by a common offset.

3 The FLOC Algorithm

3.1 Algorithm Description

In general, the bicluster problem is NP-hard as proven by Cheng and Church (2000). Thus, finding an exact solution could be time consuming. In this section, we present a new probabilistic *move-based* algorithm called FLOC, which can efficiently and accurately approximate the k biclusters with low mean squared residues without the impact of random interference. The data is represented in the form of a matrix as shown in Figure 2(a) where the rows correspond to the genes and the columns correspond to the conditions. The FLOC biclustering algorithm starts from a set of seeds (initial biclusters) and carries out an iterative process to improve the overall quality of the biclustering. At each iteration, each row and column is moved among biclusters to produce a better biclustering in terms of lower mean squared residues. The best biclustering obtained during each iteration will serve as the initial biclustering for the next iteration. The algorithm terminates when the current iteration fails to improve the overall biclustering quality.

The FLOC algorithm has two phases (Figure 3). In the first phase, k initial biclusters are constructed. As we presented in the previous section, a bicluster contains a set of genes (rows) and a set of conditions (columns). A parameter ρ is introduced to control the size of a bicluster. For each initial bicluster, a random switch is employed to determine whether a row or column should be included. Each row and column is included in the bicluster with probability ρ. Consequently, each initial bicluster is expected to contain $M \times \rho$ rows and $N \times \rho$ columns. If the percentage of specified values in an initial cluster falls below the α threshold, then we keep generating new clusters until the percentage of specified values of all columns and rows satisfy the α threshold. In this paper, the α is chosen as follows. Let D be the original matrix, which has N rows and M columns. α is set to $\frac{|D|}{N \times M}$ where $|D|$ is the volume of D. In this case, we can guarantee that any biclusters of D has at most the same percentage of unspecified values as D itself.

The second phase is an iterative process to improve the quality of the biclusters continuously. During each iteration in the second phase, each row and each column are examined to determine its best action towards reducing the overall mean squared residue. These actions are then performed successively to improve the biclustering. An **action** is defined with respect to a row (or column) and a bicluster. There are k actions associated with each row (or column), one for each bicluster. For a given row (or column) x and a bicluster c, the **action** $Action(x, c)$ is defined as the change of membership of x with respect to c. Note that this action is uniquely defined at any stage. If x is already included in c, then $Action(x, c)$ represents the removal x from the bicluster c. Otherwise, $Action(x, c)$ denotes the addition of x to the bicluster c. Figure 4 shows a data matrix with 3 rows and 4 columns. Assume that we want to find two biclusters

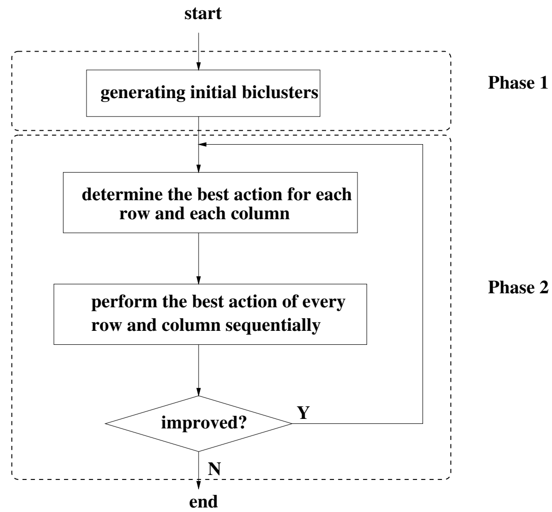

Figure 3. The Flowchart of the FLOC Algorithm

and their current status is indicated by the dash lines. Bicluster 1 contains row 1, 2 and column 1, 2; whereas bicluster 2 contains row 2, 3 and column 1, 2, 3. Each row (or column) in Figure 4 is then associated with two actions, one for each bicluster. For example, the actions associated with column 3 are (1) inserting into bicluster 1, and (2) deleting from bicluster 2. The better action among these two need to be identified and performed. In general, if the data matrix contains N rows and M columns, then $N + M$ actions will be performed during each iteration, one for each row (or column). We will discuss shortly that sometimes an action may be blocked temporarily during an iteration due to the violation of some constraint (by the action).

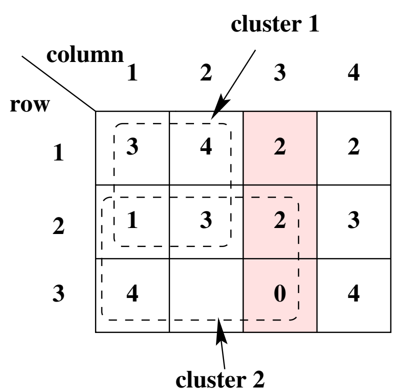

Figure 4. An Example of the Actions

Since there are k biclusters, the number of potential actions associated with the row (or column) x is k. Among these k actions, the action that brings most improvement needs to be identified. To assess the amount of improvement that can be brought by an action, we introduce a new concept called **gain**. Since our objective is to find biclusters with low residue ($< r$), the **gain** of an action $Action(x, c)$ is defined as a function of the *relative reduction* of c's residue and the *relative enlargement* of c's volume as a consequence of performing $Action(x, c)$.

Definition 3.1 *Given a residue threshold r, the* **gain** *of an action $Action(x, c)$ is defined as $Gain(x, c) = \frac{r_c - r_{c'}}{\frac{r^2}{r_c}} + \frac{v_{c'} - v_c}{v_c}$ where $r_c, r_{c'}$ are the residues of bicluster c and the bicluster, c', obtained by performing $Action(x, c)$ on c, respectively. Similarly, v_c and $v_{c'}$ are the volumes of c and c', respectively.*

When c has a much smaller residue than the threshold r ($r_c << r$), the gain measurement would favor those actions that enlarge c, especially if c has a small volume. This encourages the FLOC algorithm to find large biclusters with tolerable residues. On the other hand, when c's residue is larger than r ($r_c << r$), the measurement of gain inclines to keep the residue of c under control, especially if c already has a sufficiently large volume. Obviously, a positive gain indicates that performing $Action(x, c)$ has a potential to produce a better bicluster while a negative gain suggests that such an action would be likely to degrade the bicluster quality. Therefore, the intermediate goal during the course of biclustering becomes, for each row (or column) x, to find and perform the action with the highest gain. In the above example, the residue of bicluster 1 and bicluster 2 are $\frac{1}{16}$ and $\frac{7}{6}$, respectively. Let $r = 1$ and consider the two actions associated with column 3: (1) inserting into bicluster 1 and (2) deleting from bicluster 2. The resulting bicluster after inserting column 3 into bicluster 1 would contain row 1,2 and column 1,2,3, and its residue is $\frac{1}{6}$. Therefore, the gain of inserting column 3 into bicluster 1 is $\frac{\frac{1}{16} - \frac{1}{6}}{\frac{1^2}{1/16}} + \frac{2}{4} = -\frac{5}{768} + \frac{1}{2} = \frac{379}{768}$. Via similar computation, the gain of deleting column 3 from bicluster 2 is $\frac{\frac{7}{6} - 1}{\frac{1^2}{7/6}} - \frac{1}{3} = \frac{7}{36} - \frac{1}{3} = -\frac{5}{36}$. Consequently, the first action is chosen as the best action for column 3. Note that the best action for a row or column might be negative. Such negative action(s) will still be performed. The rationale is that the (temporary) degradation of the bicluster quality may lead to an ultimate (bigger) improvement. We will explain shortly that such action will not take into effect if the bicluster quality fails to improve by the end of the iteration. Nevertheless, the highest gain of any action associated with a given row (or column) is positive in many cases and will directly contribute to the improvement of the bicluster quality. For example, the highest gain of any action associated with column 3 is $\frac{379}{768}$.

To compute the gain of a particular action, the residue of the resulting bicluster (if the action was taken) needs to be computed. The straightforward way to compute the residue after each action is to recompute from scratch. This involves the computation of each gene base, each condition base, and bicluster base, and finally the bicluster residue. A more efficient method is to recompute only those gene and condition bases affected by the action. This can be done efficiently (in an incremental manner) if the gene bases and condition bases of the bicluster are maintained along with the bicluster base throughout the course. This technique effectively reduces the time complexity from $O(N \times M)$ to $O(N + M)$ where N and M are the number of rows and the number of columns of the data matrix, respectively.

After the best action is identified for every row (or column), these $N + M$ actions are then performed sequentially. The best biclustering obtained during the last iteration, denoted by $best\_biclustering$, is used as the initial biclustering of the current iteration. Let $Biclustering_i$ be the set of biclusters after applying the first i actions. After applying all actions, we would obtain $M + N$ sets of biclusterings. Among them, if any biclustering with all r-biclusters[3] has a larger aggregated volume than that of $best\_biclustering$, then there is an improvement in the current iteration. The biclustering with the minimum average residue is stored in $best\_biclustering$ and the process continues to the next iteration. Otherwise, there is no improvement in the current iteration and the process terminates. The biclustering stored in $best\_biclustering$ is then returned as the final result.

At each iteration, the set of actions are performed according to a so called **random weighted order**. Informally, actions with positive gains will be given higher probabilities to be executed earlier than actions with negative gains. To generate the random weighted order on a list of actions, a series of action swapping is performed, where each time two actions are randomly chosen and their positions are swapped with a certain probability computed from the gains associated with these two actions. Let's consider two actions a_i and a_j and a_i is in front of a_j. The gain of the two actions are g_i and g_j, respectively. The probability $p(i, j)$ of swapping of a_i and a_j is a function of to $g_j - g_i$. Let R be the difference between the maximum gain and minimum gain of all actions. Then $p(i, j) = 0.5 + \frac{g_j - g_i}{2 \times R}$. The rule of thumb is that if the action in the front has a larger gain than the one in the back, then the swap is less likely to occur. When a_j has the maximum gain and a_i has the minimum gain, then the probability of swapping a_i and a_j is 1. On the other hand, if g_j is the minimum gain and g_i is the maximum gain, then $p(i, j) = 0$. In the case that $g_i = g_j$, $p(i, j) = 0.5$. This random swapping procedure repeats g times. We experiment with various value of g and found that the randomness of the list is satisfactory where $g \geq 2 \times (M + N)$. Thus, we chose $g = 2 \times (M + N)$ to generate a (weighted) random sequence in this paper. Note that the randomness introduced to the action ordering is important because it allows FLOC to overcome some local optimal solutions. As a result, the weighted random order can provide promissing final biclustering due to the fact that the weighted random order favors actions with large gains while still allow enough room for the algorithm to surpass local optimum..

It is obvious that no random data is introduced to the data matrix by the FLOC algorithm and hence the random inter-

[3]It is possible that the biclustering at some stage contains some bicluster with residue larger than r. The biclustering at this stage will not be considered even the aggregated volume is larger than that of $best\_biclustering$.

ference can be entirely avoided. In addition, any unspecified entry can be accommodated seamlessly when computing action gains.

3.2 Complexity Analysis

In the first phase, a set of k biclusters (seeds) are generated. Thus, the time complexity of the first phase is $O(k \times (N + M))$ where N and M are the number of rows and columns of the matrix D while k is the number of the biclusters. The second phase is a series of iterations. During each iteration, each possible action of each row (or column) needs to be considered. There are k possible actions for a given row (or column). Thus, $(N + M) \times k$ actions have to be considered. In turn, the overall time complexity to evaluate all of these actions is $O((N + M)^2 \times k)$. The time complexity to perform an action is the same as to compute the gain of that action which is $O(N + M)$. There are $(N + M)$ actions to perform. Thus, the overall time complexity for an iteration is $O((N + M)^2 \times k)$, which implies that the complexity of the FLOC algorithm is $O((N + M)^2 \times k \times p)$ where p is the number of iterations till termination. Note that FLOC has less computational complexity than the Cheng-Church algorithm as it is typically the case where $p << N + M$.

3.3 Additional Features

As mentioned previously, the bicluster model can support many optional constraint specified by the user. In order to enforce the constraint, the basic FLOC algorithm needs to be modified. In the first phase where the set of initial biclusters are generated, the produced biclusters have to comply with the specified constraint. During the second phase where the iterative improvement is carried out, some action may be "blocked" temporarily (e.g., the gain is assigned to $-\infty$) during an iteration if it will result in the violation of some constraint. Only those actions that fully comply with the constraint will be performed. For example, if an action would cause the percentage of specified values for a gene or a condition in a bi-cluster falls below α, then the gain of this action will be assigned to $-\infty$. Furthermore, if all actions for a row or column are associated with $-\infty$ gain, then no action of this row or column will be performed at this iteration. It is obvious that the result produced by this modified version of the FLOC algorithm is guaranteed to satisfy the specified constraint.

4 Experimental Results

The FLOC algorithm is implemented with C programming language and is executed on an IBM AIX machine. We compare our FLOC algorithm with the Cheng-Church algorithm (CC algorithm) proposed by Cheng and Church (2000) on the yeast micro array containing 2884 genes under 17 conditions. Table 1 shows the performance of the two algorithms. Both algorithms are used to find 100 largest biclusters whose residue is less than 300.

At a glance, the FLOC algorithm is able to locate larger biclusters with smaller residue in substantially less amount of computation time. This is due to the fact that the FLOC algorithm is able to find highly coherent genes and condition which will lead to less residue. On the other hand, since random data is used to replace the discovered bicluster, the Cheng-Church algorithm tends to find smaller cluster with less coherence, i.e., larger residue. For almost all biclusters discovered by the Cheng-Church algorithm, they are subclusters of some bicluster discovered by the FLOC algorithm. Due to space limitations, we show two examples in this paper. Figure 5 shows two biclusters output by the FLOC algorithm, which entirely encompass some bicluster reported by the Cheng-Church algorithm. In Figure 5(a), the fine curves represent the expression levels of genes in bicluster 66 in Cheng and Church (2000) while the bold curve is the additional gene (YOR074C) discovered by FLOC, which also exhibits a similar behavior as the rest. It is interesting to know that the residue *decreases* with the inclusion of this gene because the addition of highly coherent matches will reduce the residue. Figure 5(b) presents another bicluster that contains two more conditions (condition 0 and condition 9) and six more genes (YAR007C, YAR008W, YBR089W, YDR097C, YJL187C, and YKL113C) than bicluster 95 reported by the Cheng-Church algorithm. The residue of this bicluster is 279.85 comparing to residue 311.7 of the bicluster reported by Cheng-Church algorithm.

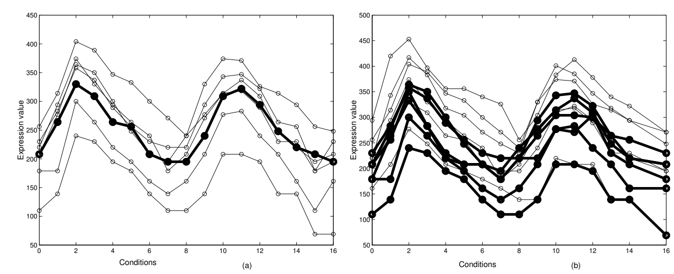

Figure 5. Discovered Biclusters

After a thorough study, we found that the deficiency of the Cheng-Church algorithm results from a combined effect of the random interference and the vulnerability of the Cheng-Church algorithm to local optimum. As the Cheng-Church algorithm proceeds, more biclusters are located and are masked with random data, and the phenomenon of random interference becomes more severe, which impacts the discovery of large biclusters. For instance, the missed genes by Cheng-Church algorithm in Figure 5 is due to the fact that all (or majority) of the conditions on the missing genes have been included in some previous discovered biclusters. As a result, the exact value is replaced by some random numbers, and in turn, they are not included in the new bicluster. On the other hand, the order of bicluster discovery has little importance in our FLOC algorithm. Therefore, FLOC is able

Table 1. Performance comparison of FLOC and CC algorithms

	avg. residue	avg. volume	avg. gene num.	avg. cond. num.	time
CC algorithm	204.293	1576.98	167	12	12 min
FLOC algorithm	187.543	1825.78	195	12.8	6.7 min

to include these genes in the discovered bicluster. This can be further confirmed by Figure 6 which show the trends (after smoothing) of residues and volumes presented by the 100 biclusters from the Cheng-Church algorithm. Note that this results in smaller average volume over the discovered biclusters comparing to FLOC. Nevertheless the average residue under FLOC is still smaller as the additional genes are high quality matches. We believe that the random interference is the prime reason to cause this phenomenon.

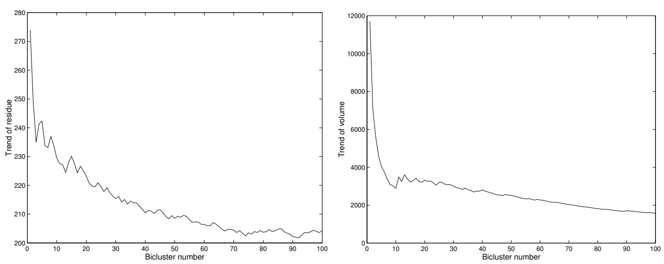

Figure 6. Trend of Cheng-Church algorithm

5 Conclusions

In this paper, we proposed a generalized model of bicluster to address potential problems when dealing with gene expression data with missing values. Our observation of the random interference phenomenon, which is inherit to the "masking" operation in the Cheng-Church algorithm, motivates us to devise and deploy a novel algorithm, FLOC, to efficiently discover biclusters on gene expression data. The key feature of the FLOC algorithm is to trigger temporary "blocking" of certain action if such action has a potential to violate any criterion. This mechanism not only enables the FLOC algorithm to deliver better result faster than the Cheng-Church algorithm, but also enriches the bicluster model by supporting additional feature constraint at nearly no extra cost.

References

[1] Aach, J., Rindone, W., and Church, G. (2000) Systematic management and analysis of yeast gene expression data. *Genome Research*, 10, 431-445.

[2] Alizadeh, A. et al. (2000) Distinct types of diffuse large B-cell lymphoma identified by gene expression profiling. *Nature*, 403, 503-510.

[3] Ben-Dor, A. and Yakhini, Z. (1999) Clustering gene expression patterns. *Proc. ACM RECOMB*, 33-43.

[4] Ben-Dor, A., Friedman, N., and Yakhini, Z. (2001) Class discovery in gene expression data. *Proc. ACM RECOMB*, 31-38.

[5] Bussemaker, H., Li, H., and Siggia, E. (2001) Regulatory element detection using correlation with expression. *Nature Genetics*, 27, 167-171.

[6] Califano, A., Stolovitzky, G., and Tu, Y. (2000) Analysis of gene expression microarrays for phenotype classification. *Proc. ISMB*.

[7] Cheng, Y. and Church, G. (2000) Biclustering of expression data, *Proc. ISMB*.

[8] Cho, R., Campbell, M., et al. (1998) A genome-wide transcriptional analysis of the mitotic cell cycle. *Mol. Cell*, 2, 65-73.

[9] Eisen M. and BrownP. (1999) DNA arrays for analysis of gene expression. *Methods in Enzymology*, 303, 179-205.

[10] Eisen, M., Spellman, P., et al. (1998) Clustering analysis and display of genome-wide expression patterns. *Proc. Natl. Acad. Sci. USA*, 96, 14863-14868.

[11] Hartuv, E., Schmitt, A., Lange, J., Meier-Ewert, S., Lehrach, H., and Shamir, R. (1999) An algorithm for clustering cDNAs for gene expression analysis using short oligonucleotide fingerprints. *Proc. ACM RECOMB*, 188-197.

[12] Kaufmann, L. and Rousseuw, P. (1990) Finding groups in data – an introduction to cluster analysis, *Wiley series in Probability and Mathematical Statistics*.

[13] Mirkin, B. (1996) *Mathematical Classification and Clustering*, Kluwer.

[14] Segal, E., Taskar, B., Gasch, A., Friedman, N., and Koller, D. (2001) Rich probabilistic models for gene expression. *Bioinformatics*, 17, 243-252.

[15] Sharan, R. and Shamir, R. (2000) CLICK: a clustering algorithm with applications to gene expression analysis. *Proc. ISMB*.

[16] Tavazoie, S., Hughes, J., Campbell, M., Cho, R., and Church, G. (1999) Systematic determination of genetic network architecture. *Nature Genetics*, 22, 281-285.

[17] Xing, E. and Karp, R. (2001) CLIFF: clustering of high-dimensional microarray data via iterative feature filtering using normalized cuts. *Bioinformatics*, 17, 306-315.

[18] Yeast Micro Data Set, available at http://arep.med.harvard.edu/network_discovery.

[19] Zien, A., Aigner, T., Zimmer, R., and Lengauer, T. (2001) Centralization: a new method for the normalization of gene expression data. *Bioinformatics*, 17, 323-331.