A Repeated Local Search Algorithm for BiClustering of Gene Expression Data

Duy Tin Truong, Roberto Battiti, and Mauro Brunato

University of Trento, Italy
{truong,battiti,brunato}@disi.unitn.it

Abstract. Given a gene expression data matrix where each cell is the expression level of a gene under a certain condition, *biclustering* is the problem of searching for a *subset* of genes that coregulate and coexpress only under a *subset* of conditions. The traditional clustering algorithms cannot be applied for biclustering as one cannot measure the similarity between genes (or rows) and conditions (or columns) by normal geometric similarities. Identifying a network of collaborating genes and a subset of experimental conditions which activate the specific network is a crucial part of the problem. In this paper, the BIClustering problem is solved through a REpeated Local Search algorithm, called **BICRELS**. The experiments on real datasets show that our algorithm is not only fast but it also significantly outperforms other state-of-the-art algorithms.

Keywords: biclustering, co-clustering, local search, gene expression, microarray.

1 Introduction

Gene expression is the process by which information from a gene is used in the synthesis of proteins. Microarray experiments provide us with the expression level of a large number of genes under different experimental conditions [2]. The conditions can be different time points, different types of tissues, or individuals, etc. The gene expression results are presented as a matrix where each gene is a row and each condition is a column. Each cell of the data matrix is the expression level of a gene under a certain condition. The expression level measures the relative abundance of a gene, usually as the logarithmic ratio between the intensities of the dyes used in the experimental process.

Given the gene expression data, one would like to find a *subset* of genes that coregulate and coexpress (think "behave in a coherent manner") only for a *subset* of conditions. This problem is called *biclustering* by Cheng and Church [4]. The objective is to find sub-matrices, i.e. maximal subgroups of genes and subgroups of conditions where the genes exhibit highly correlated activities over a range of conditions, and therefore often related to an underlying *gene regulatory network* of biological interest.

Biclustering is also known as co-clustering, bidimensional clustering, or subspace clustering, and used in other areas like marketing and collaborative recommendations, although with different underlying models. In the traditional clustering problem, only rows or columns of a data matrix are partitioned in different groups based on some geometric similarity measures like the *cosine similarity*, or *Euclidean* distance. Meanwhile, in the biclustering problem, both rows and columns are clustered simultaneously and the identification of a relevant subset of genes and a subset of experimental conditions is a prerequisite to obtain a clustering of biological interest. Traditional clustering algorithms cannot be applied for biclustering as they cannot be based on Euclidean or other geometric properties. This raises the need of developing a new class of algorithms for biclustering, which aim at identifying a relevant network of cooperating genes.

Several algorithms have been proposed for solving the biclustering problem [8]. The algorithms can return a single large and "coherent" bicluster or a set of biclusters. The bicluster "coherence" must be related to experimental process used to identify biological gene regulatory networks. An additive model of the gene expression is often used: the expression of a gene in a network is proportional to the sum of a term associated to the specific gene and a term associated to the specific experimental condition. The squared error w.r.t. this linear model, averaged over the entire bicluster expression levels, called *mean squared residue*, measures the lack of "coherence" of the network [4].

In this paper, we consider the problem of searching for a largest bicluster under the constraint that the *mean squared residue* is below a threshold. A bicluster with mean squared residue less than or equal to a threshold δ is called a δ-bicluster. The problem of finding the largest δ-bicluster is NP-hard [4].

We introduce a Repeated Local Search algorithm for BIClustering (abbreviated as **BICRELS**). Our algorithm is not only reasonably fast due to an incremental evaluation scheme, but it also significantly outperforms other state-of-the-art algorithms in both objectives, leading to larger biclusters with smaller residues.

The rest of this paper is organized as follows. In Section 2, we summarize the related work. Then, we describe formally the biclustering problem in Section 3 and our algorithm in Section 4. Finally, we report on the experimental results in Section 5.

2 Related Work

The algorithms proposed for solving the biclustering problem can be classified into different groups:

- Iterative row and column clustering combination: applying the standard clustering methods on rows and columns of the data matrix and then combining the row and column clusters to form biclusters [5].
- Divide and conquer: breaking the problem into smaller problems, solving them recursively, and combining the solutions of sub-problems to form the solution for the original problem [6].
- Greedy iterative search: removing rows or columns to reduce the bicluster residue below the threshold and adding rows or columns to increase the bicluster volume while the constraint on residue is still satisfied [4].

– Exhaustive bicluster enumeration: enumerating all possible biclusters to identify the best ones in exponential time [10].
– Distribution parameter identification: assuming the data is generated from a model and trying to fit parameters of that model by minimizing a certain criterion [7].

Not all algorithms optimize the same "coherence" criterion, therefore we only compare our algorithm with those based on the same *additive mode* and searching for the largest bicluster with residue below a threshold.

One of the first biclustering algorithms searching for the largest δ-bicluster is proposed by Cheng and Church [4]. The algorithm starts from the initial bicluster which contains all genes and conditions. Each gene or condition is considered as a node. The method iteratively deletes a set of nodes until the mean squared residue of that bicluster below the threshold. Then, a set of nodes is added to the bicluster to increase its volume until any further addition would cause the residue to exceed the threshold. This algorithm is deterministic and very fast, as a set of nodes can be deleted or added at the same time. Its complexity is $O(MN)$ where M and N are the number of genes and conditions. However, modifying a set of nodes simultaneously can also make the algorithm stuck in local minima. We refer this algorithm as **ChengChurch** in this paper.

Yang et al.[12] propose a probabilistic algorithm named **FLOC** (Flexible Overlapped Clusters) which can discover a set of K biclusters in one run. The algorithm starts from a set of random initial biclusters. Each initial bicluster is formed by selecting randomly a subset of rows and a subset of columns from the dataset, such that the bicluster residue is below a predefined threshold. Then, it iteratively performs the best action for each row and column to improve the bicluster quality. The actions are deleting or adding a row or a column to one of K biclusters. The best action is the one that gives the highest improvement in a gain function which is the sum of the reduction ratio in mean squared residue and the increase ratio in volume. As two objectives (volume and residue) are considered at the same time, **FLOC** can return biclusters with very small volume while their residues are much lower than the threshold. Besides, **FLOC** is very sensitive to the initial biclusters and its complexity is $O((M+N)^2 \times K \times p)$ where M, and N are the number of genes and conditions, K is the number of biclusters, and p is the number of iterations the algorithm runs until convergence.

Bleuler et al.[3] introduce a single-objective genetic framework for solving the biclustering problem. In their framework, a bicluster is presented as a binary string with the length of $M + N$ where M and N are the number of genes and conditions, respectively. Normal uniform crossover and bit mutation operators are performed on the population. The minimized objective is the inverse of the bicluster volume if the residue is below the threshold. Otherwise, the algorithm minimizes the residue. However, the authors conclude that without the help of local searchers, the genetic algorithm cannot produce the bicluster which is larger than the one returned by the **ChengChurch** algorithm. Therefore, they hybridize the genetic algorithm with the local search algorithm of Cheng and Church, i.e., each instance in the population is improved by the local searcher before moving to the next generation. We denote this algorithm as **SOGA** (Single Objective Genetic Algorithm). The complexity of **SOGA** is $O(TPMN)$ where T is the number of generations, P is the population size, M and N are the number of genes and conditions, respectively.

Mitra et al.[9] also propose a multi-objective genetic algorithm for biclustering (denoted as **MOGA** in this paper). The authors focus on searching for the largest bicluster and present each bicluster as a binary string. **MOGA** maximizes the volume and the residue simultaneously. When a bicluster has the residue exceeding the threshold, its residue is set to zero. In other words, the quality of a solution violating constraints is considered as zero. Similarly to the case of **SOGA**, the **ChengChuch** algorithm is adapted as the local searcher for **MOGA**, i.e. each instance in the population is improved by the local searcher before moving to the next generation. The complexity of **MOGA** is $O(TP^2MN)$ where T is the number of generations, P is the population size, M and N are the number of genes and conditions, respectively.

3 The Biclustering Problem

We follow the same notation used in [4]. The biological motivation for the model is that, in a gene regulatory network, the gene expression level is proportional to a sum of a term characterizing the gene plus a term characterizing the experimental condition which is activating the specific network. Let's note that, if logarithms of the original measures are taken, the model is multiplicative in the original measures. Fig.1 shows an example of a bicluster with 9 genes and 6 conditions.

Let X be the set of genes and Y be the set of conditions. Let a_{ij} be the element of the expression matrix A representing the expression level of the gene i under

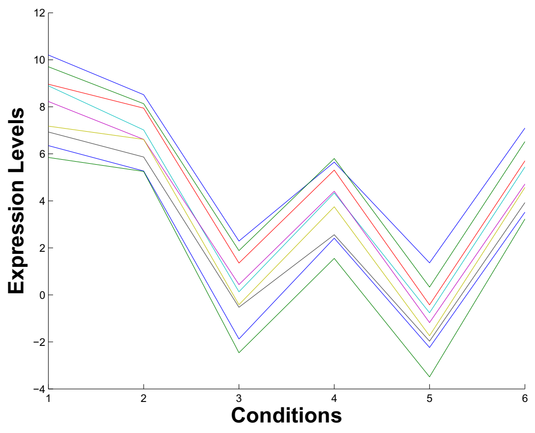


Fig. 1. An example of a bicluster with 9 genes and 6 conditions

condition j. Let $I \subset X$ and $J \subset J$ be subsets of genes and conditions. The pair (I, J) specifies a submatrix A_{IJ} with the following *mean squared residue* score:

$$MSR(I,J) = \frac{1}{|I||J|} \sum_{i \in I, j \in J} (a_{ij} - a_{iJ} - a_{Ij} - a_{IJ})^2 \tag{1}$$

where

$$a_{iJ} = \frac{1}{|J|} \sum_{j \in J} a_{ij} \tag{2}$$

$$a_{Ij} = \frac{1}{|I|} \sum_{i \in I} a_{ij} \tag{3}$$

$$a_{IJ} = \frac{1}{|I||J|} \sum_{i \in I} \sum_{j \in J} a_{ij} = \frac{1}{|I|} \sum_{i \in I} a_{iJ} = \frac{1}{|J|} \sum_{j \in J} a_{Ij} \tag{4}$$

The biclustering problem is the problem of searching for a bicluster (I, J) such that its volume is maximized and its mean squared residue is below a threshold. Formally, the biclustering problem is defined as:

$$(I', J') = \underset{I \subset X, J \subset Y}{\operatorname{argmax}} |I||J| \tag{5}$$

subject to

$$MSR(I,J) \leq \delta \tag{6}$$

4 A Repeated Local Search Algorithm for Biclustering

Let *rowMSR*(i) and *colMSR*(j) are the mean squared residues of row i and column j w.r.t a bicluster (I, J), respectively.

$$rowMSR(i) = \frac{1}{|J|} \sum_{j \in J} (a_{ij} - a_{iJ} - a_{Ij} + a_{IJ})^2 \tag{7}$$

$$colMSR(j) = \frac{1}{|I|} \sum_{i \in I} (a_{ij} - a_{iJ} - a_{Ij} + a_{IJ})^2 \tag{8}$$

The pseudo code of our local search algorithm for biclustering (**BICRELS**) is shown in Algorithm 1. We first generate the initial set of biclusters by combing gene and condition clusters. In detail, we partition the gene set into 100 clusters by applying **K-Means** (with the *cosine similarity* distance) on the column-normalized data where each gene is an instance, and each condition is a feature. The column-normalized data is obtained from the original data by subtracting the mean value from each column and then dividing the results by its sample standard deviation. Similarly, we divide the conditions into *ceil*$(N/10)$ clusters on the row-normalized data where N is the number of conditions and *ceil*(x) returns the nearest integer equal to or greater than x. Then, we pick randomly a

Algorithm 1. BICRELS

```
Input  : data matrix $A$, residue threshold $\delta$
Output: A bicluster $(I, J)$
1  begin
2      $pool$ = create a set of initial biclusters.
3      $biclusterSet = \emptyset$
4      for $i = 1$ to $numberOfRestarts$ do
5          $bicluster$ = pick randomly a bicluster from $pool$.
6          Remove $bicluster$ from $pool$.
7          $bicluster = replaceNodes(bicluster)$
8          $bicluster = deleteNodes(bicluster, \delta)$
9          repeat
10             $bicluster = replaceNodes(bicluster)$
11             $bicluster = addNodes(bicluster, \delta)$
12         until $no\ change$ ;
13         $bicluster = deleteNodes(bicluster, \delta)$
14         $biclusterSet = biclusterSet \cup \{bicluster\}$
15 end
16 return $bicluster \in biclusterSet$ with maximum size
```

cluster of gene and a cluster of condition to form a bicluster. In the experiments of this paper, we create 100 biclusters for the initial bicluster set.

The local search procedure from line 4 to 14 of the algorithm is restarted for a number of runs to explore different local minima. Note that the normalized data is used only to create the initial bicluster set, and the local search procedure is run on the original data. In each run, the algorithm first picks randomly a bicluster from the initial bicluster set. That bicluster is then removed from the initial set. Next, the algorithm reduces the residue of that bicluster by the procedure *replaceNodes* in Algorithm 2. This procedure shrinks the residue of a bicluster by replacing the column (or row) with the highest residue in that bicluster by a column (or row) not in that bicluster with the smallest residue if the replacement can reduce the bicluster residue. The replacement process is repeated until no columns or rows are replaced. After shrinking the bicluster residue by replacing rows or columns, if the residue is still greater than the threshold, some rows or columns are deleted in the *deleteNodes* procedure of Algorithm 3 (which is the single-node deletion procedure proposed by Cheng and Church [4]). The *deleteNodes* procedure keeps deleting the columns and rows with highest mean residues until the residue of the bicluster drops below the threshold. Note that, although both the *replaceNodes* and *deleteNodes* procedure can decrease the residue, the *replaceNodes* procedure keeps the bicluster size unchanged whereas the *deleteNodes* procedure also reduces the bicluster size.

Now, the bicluster residue is guaranteed to be lower than or equal to the threshold. The algorithm starts optimizing the bicluster by repeating two steps: *replaceNodes* and *addNodes* until convergence. The main idea is that while fixing the volume, we try to reduce the residue of the bicluster and then while

Algorithm 2. replaceNodes

```
Input  : $(I, J)$ are the sets of rows and columns
Output: $(I', J')$ with smaller or equal residue
1  begin
2      repeat
3          // Replace columns
4          repeat
5              $maxJ = \operatorname{argmax}_{j \in J} colMSR(j)$
6              $minJ = \operatorname{argmin}_{j \in Y \setminus J} colMSR(j)$
7              $J' = J \cup \{minJ\} \setminus \{maxJ\}$
8              if $MSR(I, J) > MSR(I, J')$ then
9                  $J = J'$
10         until $J$ is not modified ;
11         // Replace rows
12         repeat
13             $maxI = \operatorname{argmax}_{i \in I} rowMSR(i)$
14             $minI = \operatorname{argmin}_{i \in X \setminus I} rowMSR(i)$
15             $I' = I \cup \{minI\} \setminus \{maxI\}$
16             if $MSR(I, J) > MSR(I', J)$ then
17                 $I = I'$
18         until $I$ is not modified ;
19     until $I, J$ are not modified ;
20 end
21 return $(I, J)$
```

keeping the residue below the threshold, we try to increase the bicluster volume. The *addNodes* procedure is presented in Algorithm 4. This procedure iteratively adds a column or a row with the smallest residue until the residue of the bicluster exceeds the threshold. Finally, to guarantee that the bicluster mean squared residue is less than or equal to the threshold, we perform the *deleteNodes* procedure before returning the bicluster. As the number of rows and columns is finite, the loop of two steps *replaceNodes* and *addNodes* always terminates after a finite number of iterations (which is less than or equal to $(|X| + |Y|)$).

Algorithm Complexity. The most expensive steps in three procedures *replaceNodes*, *deleteNodes*, and *addNodes* are the computation of $MSR(I, J)$ and all $rowMSR(i)$, $colMSR(j)$ which have the complexity of $O(|X||Y|)$ where $|X|$ is the number of rows, and $|Y|$ is the number of columns. Therefore, the complexity of these three procedures as well as the whole algorithm is also $O(|X||Y|)$. However, as each iteration, only one row or column is modified, the incremental update strategy can be applied to reduce the complexity of computing $MSR(I, J)$ from $O(|X||Y|)$ to $O(max(|X|, |Y|))$. Besides, the cost of updating a_{iJ}, a_{Ij}, a_{IJ}

Algorithm 3. deleteNodes

```
Input  : $(I, J)$ are the sets of rows and columns, threshold $\delta$
Output: $(I', J')$ with residue smaller than threshold $\delta$
1  begin
2      while $MSR(I, J) > \delta$ do
3          $maxJ = \underset{j \in J}{\text{argmax}}\, colMSR(j)$
4          $maxI = \underset{i \in I}{\text{argmax}}\, rowMSR(i)$
5          if $colMSR(maxJ) > rowMSR(maxI)$ then
6              $J = J \setminus \{maxJ\}$
7          else
8              $I = I \setminus \{maxI\}$
9  end
10 return $(I, J)$
```

Algorithm 4. addNodes

```
Input  : $(I, J)$ are the sets of rows and columns, threshold $\delta$
Output: $(I', J')$ with greater or equal size
1  begin
2      while $MSR(I, J) < \delta$ do
3          $minJ = \underset{j \in Y \setminus J}{\text{argmin}}\, colMSR(j)$
4          $minI = \underset{i \in X \setminus I}{\text{argmin}}\, rowMSR(i)$
5          if $colMSR(minJ) < rowMSR(minI)$ then
6              $J = J \cup \{minJ\}$
7          else
8              $I = I \cup \{minI\}$
9  end
10 return $I, J$
```

necessary for the computation of all $rowMSR(i)$, $colMSR(j)$ can also be reduced from $O(|X||Y|)$ to $O(max(|X|, |Y|))$.

Incremental Update. As can be seen from Equation 7, before computing the value of $rowMSR(i)$ we need to update the values of a_{iJ}, a_{Ij}, a_{IJ}. There are six cases where a bicluster (I, J) can be modified: add or delete a row or column, replace a row (or a column) by another row (or column).

When we add, delete or replace a row from a bicluster (I, J), all columns $j \in J$ are unchanged, thus all mean rows a_{iJ} (where $i \in I$) are also unaffected. Therefore, we only need to update the mean columns a_{Ij} and the overall average value a_{IJ}. The update procedure for a_{Ij}, a_{IJ} in each case is presented as follows.

a) Adding a row $r \in X \backslash I$ to the bicluster (I, J): In this case, besides updating the mean columns a_{Ij} and the overall average value a_{IJ}, we also need to compute the new mean row a_{rJ}:

$$a_{rJ} = \frac{1}{|J|} \sum_{j \in J} a_{rj} \tag{9}$$

$$a_{Ij} = \frac{1}{|I|+1}(a_{Ij} * |I| + a_{rj}) \tag{10}$$

$$a_{IJ} = \frac{1}{|I|+1}(a_{IJ} * |I| + a_{rJ}) \tag{11}$$

Then, we update $I = I \cup \{r\}$. The complexity of computing a_{rJ} is $O(|Y|)$. Because each a_{Ij} is updated with the complexity of $O(1)$, the complexity of updating all a_{Ij} is $O(|Y|)$ (as $J \subset Y$).

b) Deleting a row $r \in I$ from the bicluster (I, J): In this case, we only need to update the mean columns a_{Ij} and the overall average value a_{IJ}:

$$a_{Ij} = \frac{1}{|I|-1}(a_{Ij} * |I| - a_{rj}) \tag{12}$$

$$a_{IJ} = \frac{1}{|I|-1}(a_{IJ} * |I| - a_{rJ}) \tag{13}$$

Then, we update $I = I \setminus \{r\}$. Because each a_{Ij} is updated with the complexity of $O(1)$, the complexity of updating all a_{Ij} is $O(|Y|)$ (as $J \subset Y$).

c) Replacing a row $r_1 \in I$ by a row $r_2 \in X \setminus I$: In this case, besides updating the mean columns a_{Ij} and the overall average value a_{IJ}, we also need to compute the new mean row a_{r_2J}:

$$a_{r_2J} = \frac{1}{|J|} \sum_{j \in J} a_{r_2j} \tag{14}$$

$$a_{Ij} = \frac{1}{|I|}(a_{Ij} * |I| - a_{r_1j} + a_{r_2}j) \tag{15}$$

$$a_{IJ} = \frac{1}{|I|}(a_{IJ} * |I| - a_{r_1J} + a_{r_2J}) \tag{16}$$

Then, we update $I = I \backslash \{r_1\} \cup \{r_2\}$. The complexity of computing a_{r_2J} is $O(|Y|)$. Because each a_{Ij} is updated with the complexity of $O(1)$, the complexity of updating all a_{Ij} is $O(|Y|)$ (as $J \subset Y$).

Similarly, when we add, delete or replace a column from a bicluster (I, J), all rows $i \in I$ are unchanged, thus all mean columns a_{Ij} (where $j \in J$) are also unaffected. Therefore, we only need to update the mean rows a_{iJ} and the overall average value a_{IJ}. The update formulas can be derived similarly as in the cases of rows. The update complexity for all mean rows a_{iJ} is $O(|X|)$. In other words, the complexity of updating a_{iJ}, a_{Ij} and a_{IJ} in all cases is $O(max(|X|, |Y|)$.

In addition, in each iteration of three procedures *replaceNodes*, *deleteNodes*, and *addNodes*, we also need to recompute the mean squared residue of the updated bicluster. Let (I', J') be the updated bicluster obtained from the bicluster (I, J) by applying one of six operations: add, delete or replace rows or columns. To reduce the complexity of computing $MSR(I, J)$ from $O(|X||Y|)$ to $O(max(|X|, |Y|))$, we first derive another way to compute $MSR(I, J)$:

$$MSR(I,J) = \frac{1}{|I||J|}\sum_{i\in I}\sum_{j\in J}(a_{ij} - a_{iJ} - a_{Ij} - a_{IJ})^2 \tag{17}$$

$$= \frac{1}{|I||J|}\big(\sum_{i\in I}\sum_{j\in J} a_{ij}^2 + |I||J|a_{IJ}^2 - |I|\sum_{j\in J} a_{Ij}^2 - |J|\sum_{i\in I} a_{iJ}^2\big) \tag{18}$$

It can be seen that the complexities of four terms in the bracket on the right hand side of Equation 18 are $O(|X||Y|)$, $O(1)$, $O(|Y|)$ and $O(|X|)$, respectively (as $I \subset X, J \subset Y$). However, the first term can be updated efficiently as follows. Let $sumAll(I,J) = \sum\limits_{i\in I}\sum\limits_{j\in J} a_{ij}^2$, we can compute efficiently $sumAll(I', J')$ according to one of six cases:

a) If the updated bicluster (I', J') is obtained by adding a row r to the bicluster (I, J):

$$sumAll(I', J') = sumAll(I \cup \{r\}, J) \tag{19}$$

$$= \sum_{i\in I'}\sum_{j\in J} a_{ij}^2 \tag{20}$$

$$= \sum_{i\in I}\sum_{j\in J} a_{ij}^2 + \sum_{j\in J} a_{rj}^2 \tag{21}$$

$$= sumAll(I, J) + \sum_{j\in J} a_{rj}^2 \tag{22}$$

b) If the updated bicluster (I', J') is obtained by deleting a row r from the bicluster (I, J):

$$sumAll(I', J') = sumAll(I \setminus \{r\}, J) \tag{23}$$

$$= sumAll(I, J) - \sum_{j\in J} a_{rj}^2 \tag{24}$$

c) If the updated bicluster (I', J') is obtained by replacing a row $r_1 \in I$ by a row $r_2 \in X \setminus I$ from the bicluster (I, J):

$$sumAll(I', J') = sumAll(I \setminus \{r_1\} \cup \{r_2\}, J) \tag{25}$$

$$= sumAll(I, J) - \sum_{j\in J} a_{r_1 j}^2 + \sum_{j\in J} a_{r_2 j}^2 \tag{26}$$

d) If the updated bicluster (I', J') is obtained by adding a column c to the bicluster (I, J):

$$\begin{aligned} sumAll(I', J') &= sumAll(I, J \cup \{c\}) && (27)\\ & && (28)\\ &= sumAll(I, J) + \sum_{i \in I} a_{ic}^2 && (29) \end{aligned}$$

e) If the updated bicluster (I', J') is obtained by removing a column c from the bicluster (I, J):

$$\begin{aligned} sumAll(I', J') &= sumAll(I, J \setminus \{c\}) && (30)\\ & && (31)\\ &= sumAll(I, J) - \sum_{i \in I} a_{ic}^2 && (32) \end{aligned}$$

f) If the updated bicluster (I', J') is obtained by replacing a column $c_1 \in J$ by a column $c_2 \in Y \setminus J$ from the bicluster (I, J):

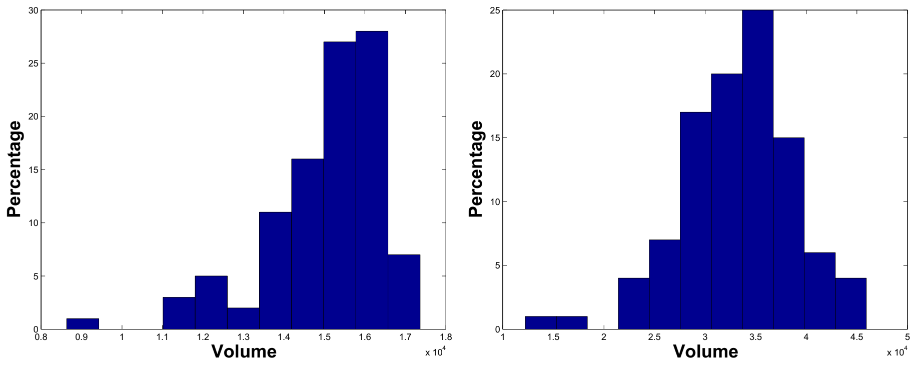
$$\begin{aligned} sumAll(I', J') &= sumAll(I, J \setminus \{c_1\} \cup \{c_2\}) && (33)\\ & && (34)\\ &= sumAll(I, J) - \sum_{i \in I} a_{ic_1}^2 + \sum_{i \in I} a_{ic_2}^2 && (35) \end{aligned}$$

In all cases, the update of $sumAll(I', J')$ has the complexity of $O(|X|)$ or $O(|Y|)$. In other words, the complexity of computing $MSR(I', J')$ form $MSR(I, J)$ is $O(max(|X|, |Y|)$.

5 Experiments

A preliminary set of experiments has been dedicated to analyze the distribution of final results found after individual runs of our local search technique. It is the distribution of local maxima values found after starting from a randomly picked initial seed bicluster. To estimate the distribution of bicluster volume obtained by **BICRELS**, we run **BICREL** with 100 restart times and plot the histograms of bicluster volume obtained by the individual local search processes on two datasets in Fig.2a and Fig.2b. In both cases, we observe a non-negligible probability for values of bicluster volumes that are close to the optimal one. This was the main motivation for adding a restart technique: by starting from different initial random seed biclusters the algorithm is sampling from this distribution and the best sample is reported at the end of the repetitions.

Considering now the complete **BICREL**, we compare it with four other state-of-the-art algorithms **ChengChurch** [4], **SOGA** (Single-objective GA) [3], **MOGA** (Multi-objective GA) [9], and **FLOC** [12].



(a) Volume Distribution on the *Yeast* dataset ($\delta = 300$)

(b) Volume Distribution on the *Lymphoma* dataset ($\delta = 1200$)

Fig. 2. Bicluster Volume Distribution of 100 restart times of **BICRELS**

5.1 Experimental Setup

The parameter α of **ChengChurch** is set to its default value ($\alpha = 1.2$). The number of initial rows and columns in the **FLOC** algorithm is set to 9 and 2, respectively (as in the **FLOC** paper [12], the authors did not describe which parameters are suitable for **FLOC**, thus we did some experiments to determine the suitable parameters for **FLOC**). The number of biclusters produced by **FLOC** in each run is set to 10. The parameters of **SOGA** and **MOGA** are set to their default parameters. All algorithms are implemented in Matlab and run on the same machine with Ubuntu 12.04 operating system, CPU Intel(R)Xeon(R)X3363@2.83GHz, RAM 8GiB. **FLOC**, **SOGA** and **MOGA** are run for 10 times to eliminate the randomness effect. **BICRELS** is run once but restarted for 10 times by setting its parameter *numberOfRestarts* to 10. **ChengChurch** is a deterministic algorithm and run only once.

In the experiments, we use two datasets *Yeast* [11] and *Lymphoma* [1]. The *Yeast* dataset consists of 2884 genes and 17 conditions. The *Lymphoma* dataset has 4026 genes and 96 conditions. These datasets are preprocessed by Cheng and Church[1]. Missing values of these datasets are processed as in [4]. The residue threshold of all algorithms on the *Yeast* and *Lymphoma* dataset are set to 300 and 1200 as in previous papers [4,3,9]. However, in some cases, some algorithms like **FLOC** can be stuck in local minima and return biclusters with very small volumes whereas their residues are much lower than the threshold. Therefore, for a fair comparison, we set different residue thresholds in our algorithm to produce biclusters with similar residues as in those of the other algorithms.

[1] `http://arep.med.harvard.edu/biclustering`

5.2 Experimental Results

The maximum volumes of biclusters produced by all algorithms on two datasets *Yeast* and *Lymphoma* are presented in Table 1a, Table 2a, Fig.3a, and Fig.3b.

It can be observed that **BICRELS** significantly outperforms the other algorithms on the two datasets in both objectives (larger in volume and smaller in residue). In addition, although setting the same residue threshold, **FLOC** can get stuck at local minima and can only produce very small biclusters.

Table 1b and Table 2b show the statistical information on the bicluster volume obtained by five algorithms. The average bicluster volume of our algorithm

Table 1. The comparison of five algorithms on the *Yeast* dataset

Algorithm	Max Volume ($\|I\| \times \|J\|$)	MSR
BICRELS ($\delta = 300$)	16577 (1507×11)	299.93
ChengChurch ($\delta = 300$)	12012 (1001×12)	237.33
BICRELS ($\delta = 237$)	12114 (1346×9)	236.95
SOGA ($\delta = 300$)	13050 (1305×10)	286.04
BICRELS ($\delta = 286$)	15580 (1558×10)	285.97
MOGA ($\delta = 300$)	8480 (848×10)	299.12
BICRELS ($\delta = 299$)	16511 (1501×11)	298.87
FLOC ($\delta = 300$)	942 (314×3)	143.20
BICRELS ($\delta = 143$)	5362 (766×7)	142.88

(a) The largest biclusters obtained from five algorithms

Algorithm	Max Volume	Min Volume	Average Volume (± Std)
BICRELS ($\delta = 300$)	16577	11473	15103.80 (±1567.04)
ChengChurch ($\delta = 300$)	12012	12012	12012.00 (±0.00)
BICRELS ($\delta = 237$)	12114	7865	10695.40 (±1470.11)
SOGA ($\delta = 300$)	13050	1443	7745.16 (±2701.19)
BICRELS ($\delta = 286$)	15580	10659	14030.80 (±1434.14)
MOGA ($\delta = 300$)	8480	3520	7271.09 (±803.33)
BICRELS ($\delta = 299$)	16511	11418	15044.00 (±1564.90)
FLOC ($\delta = 300$)	942	484	589.40 (±156.83)
BICRELS ($\delta = 143$)	5362	2709	4345.60 (±938.51)

(b) Statistical information on the bicluster volume of five algorithms

Algorithm	Runtime
BICRELS ($\delta = 300$)	2.26
ChengChurch ($\delta = 300$)	0.12
BICRELS ($\delta = 237$)	1.87
SOGA ($\delta = 300$)	15.08
BICRELS ($\delta = 286$)	2.13
MOGA ($\delta = 300$)	19.52
BICRELS ($\delta = 299$)	2.24
FLOC ($\delta = 300$)	615.85
BICRELS ($\delta = 143$)	1.12

(c) Average runtime (in seconds) of five algorithms

Table 2. The comparison of five algorithms on the *Lymphoma* dataset

Algorithm	Max Volume ($\|I\| \times \|J\|$)	MSR
BICRELS ($\delta = 1200$)	43907 (1909 × 23)	1199.50
ChengChurch ($\delta = 1200$)	39026 (1027 × 38)	1101.52
BICRELS ($\delta = 1101$)	39307 (1709 × 23)	1100.56
SOGA ($\delta = 1200$)	35820 (995 × 36)	1187.24
BICRELS ($\delta = 1187$)	42274 (1838 × 23)	1186.72
MOGA ($\delta = 1200$)	39032 (1394 × 28)	1191.62
BICRELS ($\delta = 1191$)	42458 (1846 × 23)	1190.98
FLOC ($\delta = 1200$)	572 (143 × 4)	363.22
BICRELS ($\delta = 363$)	6440 (920 × 7)	362.84

(a) The largest biclusters obtained by five algorithms

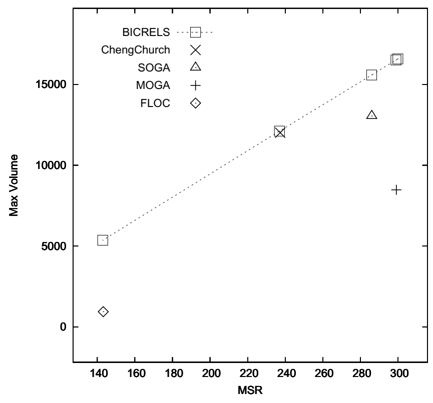
Algorithm	Max Volume	Min Volume	Average Volume (± Std)
BICRELS ($\delta = 1200$)	43907	29887	33932.50 (±4394.52)
ChengChurch ($\delta = 1200$)	39026	39026	39026.00 (±0.00)
BICRELS ($\delta = 1101$)	39307	25968	30064.10 (±4333.37)
SOGA ($\delta = 1200$)	35820	2014	23983.98 (±8441.05)
BICRELS ($\delta = 1187$)	42274	28964	33230.90 (±4191.15)
MOGA ($\delta = 1200$)	39032	30875	35451.80 (±1970.40)
BICRELS ($\delta = 1191$)	42458	29055	33348.60 (±4220.95)
FLOC ($\delta = 1200$)	572	282	354.40 (±112.25)
BICRELS ($\delta = 363$)	6440	1314	3779.00 (±1500.02)

(b) Statistical information on bicluster volume of five algorithms

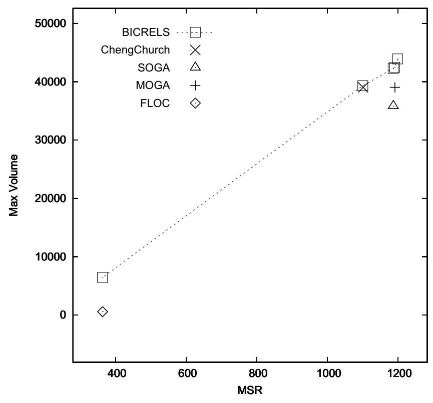
Algorithm	Runtime
BICRELS ($\delta = 1200$)	13.37
ChengChurch ($\delta = 1200$)	0.34
BICRELS ($\delta = 1101$)	12.73
SOGA ($\delta = 1200$)	76.67
BICRELS ($\delta = 1187$)	13.02
MOGA ($\delta = 1200$)	75.37
BICRELS ($\delta = 1191$)	13.12
FLOC ($\delta = 1200$)	345.93
BICRELS ($\delta = 363$)	2.54

(c) Average runtime (in seconds) of five algorithms

is much larger than that of the other algorithms. In order to study the relationship between the number of restart times and the maximum volume, we run **BICRELS** with 10 different random seeds, and in each run **BICRELS** is restarted for 100 times. Fig.4a and Fig.4b show the performance curves of our algorithm on two datasets. It can be seen that our algorithm reaches very good final results within about 30 restarts. Especially, on the *Yeast* dataset, our algorithm always converges to the optimal solution after 41 restarts. **BICRELS** also enjoys the *anytime* property: it can be terminated at any time after a given number of restarts (greater than zero) delivering the best solution found so far. This characteristic endows it with more flexibility to trade off CPU time w.r.t. solution quality.

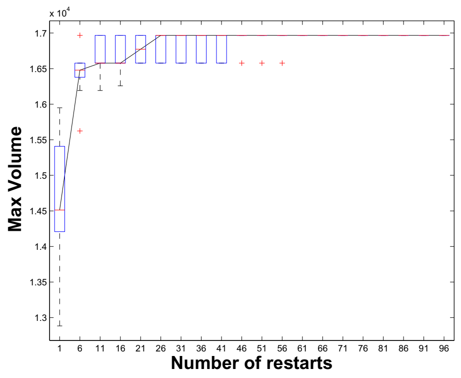


(a) On the *Yeast* dataset

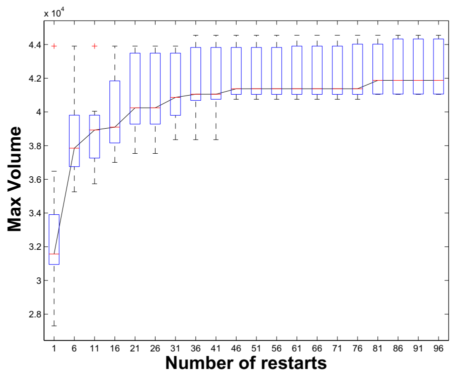


(b) On the *Lymphoma* dataset

Fig. 3. Performance comparison of five algorithms on two datasets. Let's remind that MSR has to be minimized, while volume has to be maximized. To improve visibility **BICRELS** results are connected by segments.



(a) On the *Yeast* dataset ($\delta = 300$)



(b) On the *Lymphoma* dataset ($\delta = 1200$)

Fig. 4. The performance curves of **BICRELS** on two datasets. Each boxplot of 10 different runs with the same number of restarts shows the maximum, minimum volume and lower, upper quartile together with median. The observations which are considered as outliers are presented as red crosses.

As for the comparison on the runtime, except for the **ChengChurch** algorithm, our algorithm is faster than the other algorithms as shown in Table 1c and Table 2c. Our algorithm **BICRELS** is slower than the **ChengChurch** algorithm because in each iteration, **BICRELS** only adds one row or column whereas **ChengChurch** can add a set of rows or columns. The difference between runtime of our algorithm and **ChengChurch** is the computational cost that we pay for the improvement in the solution quality. Because of the anytime property of **BICRELS**, an early termination after a smaller number of restarts is the obvious way to reduce CPU time for a lower average solution quality.

6 Conclusion

In this paper, we proposed a repeated local search algorithm for biclustering, called **BICRELS**. We also suggested an efficient incremental update scheme to speed up the algorithm. Although our algorithm is simple, it is reasonably fast and it significantly outperforms the other state-of-the-art algorithms on two real-world datasets. Finally, as our algorithm has the any-time property, it provides users with the flexibility in trading off CPU time w.r.t. solution quality.

References

1. Alizadeh, A.A., Eisen, M.B., Davis, R.E., Ma, C., Lossos, I.S., Rosenwald, A., Boldrick, J.C., Sabet, H., Tran, T., Yu, X., et al.: Distinct types of diffuse large b-cell lymphoma identified by gene expression profiling. Nature 403(6769), 503–511 (2000)
2. Baldi, P., Hatfield, G.W.: DNA Microarrays and Gene Expression: From Experiments to Data Analysis and Modelling. Cambridge University Press (2002)
3. Bleuler, S., Prelic, A., Zitzler, E.: An ea framework for biclustering of gene expression data. In: Congress on Evolutionary Computation, CEC 2004, vol. 1, pp. 166–173. IEEE (2004)
4. Cheng, Y., Church, G.M.: Biclustering of expression data. In: Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology, vol. 8, pp. 93–103 (2000)
5. Getz, G., Gal, H., Kela, I., Notterman, D.A., Domany, E.: Coupled two-way clustering analysis of breast cancer and colon cancer gene expression data. Bioinformatics 19(9), 1079–1089 (2003)
6. Hartigan, J.A.: Direct clustering of a data matrix. Journal of the American Statistical Association 67(337), 123–129 (1972)
7. Lazzeroni, L., Owen, A., et al.: Plaid models for gene expression data. Statistica Sinica 12(1), 61–86 (2002)
8. Madeira, S.C., Oliveira, A.L.: Biclustering algorithms for biological data analysis: a survey. IEEE/ACM Transactions on Computational Biology and Bioinformatics 1(1), 24–45 (2004)
9. Mitra, S., Banka, H.: Multi-objective evolutionary biclustering of gene expression data. Pattern Recognition 39(12), 2464–2477 (2006)
10. Tanay, A., Sharan, R., Shamir, R.: Discovering statistically significant biclusters in gene expression data. Bioinformatics 18(suppl. 1), S136–S144 (2002)
11. Tavazoie, S., Hughes, J.D., Campbell, M.J., Cho, R.J., Church, G.M.: Systematic determination of genetic network architecture. Nature Genetics 22, 281–285 (1999)
12. Yang, J., Wang, H., Wang, W., Yu, P.: Enhanced biclustering on expression data. In: Proceedings of the Third IEEE Symposium on Bioinformatics and Bioengineering, pp. 321–327. IEEE (2003)