Gene Expression Biclustering using Random Walk Strategies

Fabrizio Angiulli and Clara Pizzuti

ICAR-CNR
Via P. Bucci 41C
Università della Calabria
87036 Rende (CS), Italy
{angiulli,pizzuti}@icar.cnr.it

Abstract. A biclustering algorithm, based on a greedy technique and enriched with a local search strategy to escape poor local minima, is proposed. The algorithm starts with an initial random solution and searches for a locally optimal solution by successive transformations that improve a gain function, combining the mean squared residue, the row variance, and the size of the bicluster. Different strategies to escape local minima are introduced and compared. Experimental results on yeast and lymphoma microarray data sets show that the method is able to find significant biclusters.

1 Introduction

In the past recent years, DNA *microarray* technology has captured the attention of scientific community because of its capability of simultaneously measure the activity and interactions of thousands of genes. The relative abundance of the mRNA of a gene under a specific experimental condition (or sample) is called the *expression level* of a gene. The expression level of a large number of genes of an organism under various experimental conditions can be arranged in a data matrix, also known as *gene expression data matrix*, where rows correspond to genes and columns to conditions. Thus each entry of this matrix is a real number representing the expression level of a gene under a specific experiment. One of the objectives of gene expression data analysis is to group genes according to their expression under multiple conditions. Clustering [3, 10, 1] is an important gene expression analysis method that has been extensively used to group either genes, to search for functional similarities, or conditions, to find samples characterized by homogeneous gene expression levels. However, generally, genes are not relevant for all the experimental conditions, but groups of genes are often co-regulated and co-expressed only under specific conditions. This important observation has lead the attention towards the design of clustering methods that try to simultaneously group genes and conditions. The approach, named *biclustering*, detects subsets of genes that show similar patterns under a specific subset of experimental conditions.

Biclustering was first defined by Hartigan [6] and called *direct clustering*. His aim was to find a set of sub-matrices having zero variance, that is with constant values. This concept was then adopted by Cheng and Church [2] by introducing a similarity score, called *mean squared residue*, to measure the coherence of rows and columns in the bicluster. A group of genes is considered coherent if their expression levels varies simultaneously across a set of conditions. Biclusters with a high similarity score and, thus, with low residue, indicate that genes show similar tendency on the subset of the conditions present in the bicluster. The problem of finding biclusters with low mean squared residue, in particular maximal biclusters with scores under a fixed threshold, has been proved to be NP-hard [2] because it includes the problem of finding a maximum biclique in a bipartite graph as a special case [4]. Therefore, Cheng and Church proposed heuristic algorithms that are able to generate good quality biclusters.

In this paper a greedy search algorithm to find k biclusters with a fixed degree of overlapping is proposed. The method is enriched with an heuristic to avoid to get trapped at poor local minima. The algorithm starts with an initial random bicluster and searches for a locally optimal solution by successive transformations that improve a *gain* function. The *gain* combines the mean squared residue, the row variance, and the size of the bicluster. In order to escape poor local minima, that is low quality biclusters having negative gain in their neighborhood, random moves with given probability are executed. These moves delete or add a row/column on the base of different strategies introduced in the method. To obtain k biclusters the algorithm is executed k times by allowing to control the degree of overlapping among the biclusters. Experimental results on two well known microarray data sets, *yeast cell cycle* and *B-cell lymphoma*, show that the algorithm is able to find significant and coherent biclusters.

The paper is organized as follows. The next section defines the problem of biclustering and the notations used. In Section 3 an overview of the existing approaches to biclustering is given, section 4 describes the algorithm proposed, and, finally, section 5 reports the experiments on the two mentioned data sets.

2 Notation and Problem definition

In this section the notation used in the paper is introduced and a formal definition of bicluster is provided [2]. Let $X = \{I_1, \ldots, I_N\}$ be the set of genes and $Y = \{J_1, \ldots, J_M\}$ be the set of conditions. The data can be viewed as an $N \times M$ matrix A of real numbers. Each entry a_{ij} in A represents the relative abundance (generally its logarithm) of the mRNA of a gene I_i under a specific condition J_j.

A *bicluster* is a sub-matrix (I, J) of A, where I is a subset of the rows X of A, and J is a subset of the columns Y of A.

Let a_{iJ} denote the mean of the ith row of the bicluster (I, J), a_{Ij} the mean of the jth column of (I, J), and a_{IJ} the mean of all the elements in the bicluster. More formally,

$$a_{iJ} = \frac{1}{|J|} \sum_{j \in J} a_{ij}, \quad a_{Ij} = \frac{1}{|I|} \sum_{i \in I} a_{ij}, \quad a_{IJ} = \frac{1}{|I||J|} \sum_{i \in I, j \in J} a_{ij}.$$

The *volume* v_{IJ} of a bicluster (I, J) is the number of entries a_{ij} such that $i \in I$ and $j \in J$, that is $v_{IJ} = |I| \times |J|$.

The *residue* r_{ij} of an element a_{ij} is defined as $r_{ij} = a_{ij} - a_{iJ} - a_{Ij} + a_{IJ}$. The residue of an element provides the difference between the actual value of a_{ij} and its expected value predicted from its row, column, and bicluster mean. The residue of an element reveals its degree of coherence with the other entries of the bicluster it belongs to. The lower the residue, the higher the coherence. The quality of a bicluster can be thus evaluated by computing *the mean squared residue* r_{IJ}, i.e. the sum of all the squared residues of its elements:

$$r_{IJ} = \frac{\sum_{i \in I, j \in J} (r_{ij})^2}{v_{IJ}}.$$

The *mean squared residue* of a bicluster, as otlined by Cheng and Church in [2], provides the similarity score of a bicluster. Given a threshold $\delta \geq 0$, a sub-matrix (I, J) is said a *δ-bicluster*, if $r_{IJ} < \delta$. The aim is then to find large biclusters with scores below a fixed threshold δ. However, low residue biclusters should be accompanied with a sufficient variation of the gene values with respect to the row mean value, otherwise trivial biclusters having almost all constant values could be determined. To this end the row variance var_{IJ} of a bicluster (I, J) is defined as

$$var_{IJ} = \frac{\sum_{i \in I, j \in J} (a_{ij} - a_{iJ})^2}{v_{IJ}}.$$

The final goal is to obtain large biclusters, with a relatively high variance, and with mean squared residue lower than a given threshold δ.

3 Related work

A comprehensive survey on biclustering algorithms for biological data analysis can be found in [8]. In the following the main existing proposals will be described. As already mentioned in the introduction, Hartigan [6] first suggested a partition based algorithm, called *direct clustering*, that splits the data matrix to find sub-matrices having zero variance, that is with constant values. Hartigan used the variance of a bicluster to evaluate its quality and his aim was to obtain constant sub-matrices. However, he proposed to modify his algorithm to find biclusters with coherent values in rows and columns. Cheng and Church [2] were the first who introduced the new paradigm of biclustering to gene expression data analysis. They proposed some greedy search heuristics that generate sub-optimal biclusters satisfying the condition of having the mean squared residue below a threshold δ. The heuristics start with the original data matrix and add or delete rows and columns. The algorithms assume that the data matrix doesn't contain missing values and can find one or k biclusters. In the latter case, in order to avoid to reobtain the same biclusters, the values of those elements a_{ij} that have already been inserted in a bicluster are substituted with random numbers. Yang et al. [11] extended the definition of δ-bicluster to cope with missing values and to avoid problems caused by random numbers. In fact, they

experimented that random numbers in the methods of Cheng and Church can interfere with the discovery of new biclusters, in particular for those that overlap with those already obtained. They defined a probabilistic *move-based* algorithm FLOC (FLexible Overlapped biClustering) that generalizes the concept of mean squared residue and based on the concept of *action* and *gain*. Getz et al. [5] presented the Coupled Two-Way Clustering algorithm that uses a hierarchical clustering method separately on each dimension. Clusters of rows are used as conditions for column clustering and vice-versa. Lazzeroni and Owen [7] introduced the plaid model, where the concept of layers (bicluster) is used to compute the values of the elements in the data matrix. The data matrix is described as a linear function of layers corresponding to its biclusters. Tanay et al. presented SAMBA (Statistical-algorithmic Method for Bicluster Analysis), a biclustering algorithm that combines graph theory and statistics. The data matrix is represented as a bipartite graph where the nodes are conditions and genes, and edges denote significant expression changes. Vertex pairs are associated with a weight, and heavy subgraphs correspond to significant biclusters. Cho et al. [**?**] propose two iterative co-clustering algorithms that use two similar squared residue measures, and based on the k-means clustering method. They formulate the problem of minimizing the residue as trace optimization problems that provide a spectral relaxation, used to initialize their methods.

4 Algorithm Description

In this section we present *RandomWalkBiclustering*, a biclustering algorithm based on a greedy technique enriched with a local search strategy to escape poor local minima. The basic schema of our method derives from the WSAT algorithm of Selman et al. for the *Satisfiability problem* [9], opportunely modified to deal with the biclustering problem. The algorithm starts with an initial random bicluster $B = (I, J)$ and searches for a δ-bicluster by successive transformations of B, until a gain function is improved. The transformations consist in the change of membership (called flip or move) of the row/column that leads to the largest increase of the gain function. The *gain* function combines mean squared residue, row variance, and size of the bicluster by means of user-provided weights w_{res}, w_{var}, and w_{vol}. More formally, let

$$\Delta res = \frac{res_{old} - res_{new}}{res_{old}}, \quad \Delta var = \frac{var_{old} - var_{new}}{var_{old}}, \quad \Delta vol = \frac{vol_{old} - vol_{new}}{vol_{old}},$$

be the relative changes of residue, row variance, and volume when a row/column is added/removed, where res_{old}, var_{old}, and vol_{old} (resp. res_{new}, var_{new}, and vol_{new}) are respectively the values of the residue, row variance and volume of B before (after) the move. Then the function *gain* is defined as

$$gain = w_{res}(2^{\Delta res} - 1) - w_{var}(2^{\Delta var} - 1) - w_{vol}(2^{\Delta vol} - 1),$$

with $w_{res}+w_{var}+w_{vol} = 1$. This function assumes values in the interval $[-1, +1]$. In fact, relative changes Δres, Δvar, and Δvol range in the interval $[-\infty, +1]$,

Algorithm RandomWalkBiclustering
Input:
- $matrix$: a gene-expression matrix
- δ: stop when this value of residue is reached (0=stop at local minima)
- max_flips: maximum number of iterations allowed
- $method$: type of random move
- p: probability of a random move (0 = no random move)
- $w_{res}, w_{var}, w_{vol}$: weight associated to the residue, row variance, and volume resp.
- row_{min}, row_{max}: minimum and maximum number of rows allowed in the bicluster
- col_{min}, col_{max}: minimum and maximum number of columns allowed in the bicluster

Method:

```
generate at random a bicluster that does not violate the constraints on the number
of rows and columns
set flips = 0, res = +∞, local_minima = false
while flips < max_flips and δ < res and not local_minima
    flips = flips + 1
    if a random generated number is less than p then
        execute a random move according to the method chosen, that does not
        violate the constraints on the number of rows and columns, and update
        the residue value res
    else
        let m be the move, that does not violate the constraints on the number
        of rows and columns, with the maximum gain gain
        if gain > 0 then
            execute the move m and update the residue value res
        else
            set local_minima = true
return the bicluster computed
```

Fig. 1. The *RandomWalkBiclustering* algorithm.

consequently the terms $2^{\Delta res}$, $2^{\Delta var}$, and $2^{\Delta vol}$ range in the interval $[0, 2]$, and the whole function is assumes values between -1 and $+1$. The weights w_{res}, w_{var}, and w_{vol} provide a trade-off among the relative changes of residue, row variance, and volume. When $w_{res} = 1$ (and thus $w_{var} = w_{vol} = 0$), the algorithm searches for a *minimum residue bicluster*, since the gain monotonically increases with the residue of the bicluster. Decreasing w_{res} and increasing w_{var} and w_{vol}, biclusters with higher row variance and larger volume can be obtained. Notice that when the residue after a flip diminishes, and the row variance and volume increase, Δres is positive, while Δvar and Δvol are negative. Thus, when the gain function is positive, *RandomWalkBiclustering* is biased towards large biclusters with a relatively high variance, and low residue. A negative gain, on the contrary, means a deterioration of the bicluster because there could have been an augmentation of the residue or a decrease of the row variance or volume. During its execution, in order to avoid get trapped into poor local minima (i.e. low quality biclusters with negative gain in their neighborhood), instead of performing the flip maximizing

the gain, with a user-provided probability p the algorithm is allowed to execute a random move. We introduced three types of random moves:

- NOISE: with probability p, choose at random a row/column of the matrix and add/remove it to/from B;
- REMOVE: with probability p, choose at random a row/column of B and remove it from B;
- REMOVE-MAX: with probability p, select the row/column of B scoring the maximum value of residue, and remove it from B.

Thus, the NOISE is a purely random strategy that picks a row/column from the overall matrix, and not only from the bicluster, and adds or removes the row/column to the bicluster if it belongs or it does not belong to it. The REMOVE strategy removes at random a row/column already present in the bicluster, thus it could accidentally delete a worthless gene/condition from the current solution, and the REMOVE-MAX removes that row/column already present in the bicluster having the highest value of the residue, i.e. mostly contributing to worsen the gain. Figure 3 shows the algorithm *RandomWalkBiclustering*. The algorithm receives in input a gene-expression matrix, a threshold value (δ) for the residue of the bicluster, the maximum number of times (max_flips) that a flip can be done, the kind of random move the algorithm can choose ($method$), the probability (p) of executing a random move, the weight to assign to residue (w_{res}), variance (w_{var}), and volume (w_{vol}), and some optional constraint (row_{min}, row_{max}, col_{min}, col_{max}) on the size of the bicluster to find. The flips are repeated until either a preset of maximum number of flips (max_fips) is reached, or a δ-bicluster is found, or the solution can not ulteriorly be improved (get trapped into a local minima). Until the stop condition is not reached, it executes a random move with probability p, and a greedy move with probability $(1 - p)$. In order to compute k biclusters, we execute k times the algorithm *Random-WalkBiclustering* by fixing two frequency thresholds, f_{row} and f_{col}, that allow to control the degree of overlapping among the biclusters. The former binds a generic row to participate to at most $k \cdot f_{row}$ biclusters among the k to be found. Analogously, f_{col} limits the presence of a column in at most $k \cdot f_{col}$ biclusters. During the k executions of the algorithm, whenever a row/column exceeds the corresponding frequency threshold, it is removed from the matrix and not taken into account any more in the subsequent executions.

Computational complexity. The temporal cost of the algorithm is upper bounded by

$$max\_flips \times C_u \times [(1 - p) \times (N + M) + p]$$

where C_u is the cost of computing the new residue and the new row variance of the bicluster after performing a move. In order to reduce the complexity of C_u, we maintain, together with the current bicluster $B = (I, J)$, the mean values a_{iJ} and a_{Ij}, for each $i \in I$, the summation $\sum_{j \in J} a_{ij}^2$, and the total sum of the row variances. The computation of the new residue of each element involves recomputing the $|I| + |J|$ mean values a_{iJ} $(1 \leq i \leq |I|)$ and a_{Ij} $(1 \leq j \leq |J|)$ after performing the move. This can be done efficiently, in time $\max\{|I|, |J|\}$, by

exploiting the values maintained together with the current bicluster. Computing the residue res_{new} of the new bicluster, requires the computation of the squares of its element residues, a time proportional to the volume of the new bicluster. We note that, usually, $|I||J| \ll NM$. Ccomputing the new row variances can be done in a fast way by exploiting the summations $\sum_{j \in J} a_{ij}^2$ already stored. Indeed, if a column is added or removed, the new row variances can be obtained quickly by evaluating the $|I|$ expressions $\frac{1}{|J|}\sum_{ij}(a_{ij}^2) - a_{iJ}^2$ $(1 \leq i \leq |I|)$. For example, if the qth column is added, in order to compute the new variance of row i, the following expression must be evaluated:

$$\frac{1}{|J|+1}\left(\sum_{j \in J}(a_{ij}^2) + a_{iq}^2\right) - \left(\frac{|J|a_{iJ}+a_{iq}}{|J|+1}\right)^2.$$

Analogously if a column is removed. Otherwise, if a row is added (removed resp.) the corresponding row variance must be computed and added (subtracted resp.) to the overall sum of row variances. Before concluding, we note that the cost of a random move is negligible, as it consists in generating a random number, when the NOISE or REMOVE strategies are selected, while the row/column with the maximum residue, selected by the REMOVE-MAX strategy, is computed, with no additional time requirements, during the update of the residue of the bicluster at the end of each iteration, and, hence, it is always immediately available.

5 Experimental Results

In this section we give experimental results to show the behavior of the *RandomWalkBiclustering* algorithm. We selected two well known gene expression data sets, the *Yeast Saccharomyces cerevisiae* cell cycle expression data set, and the human B-cell *Lymphoma* data set. The preprocessed gene expression matrices can be obtained from [2] at `http://arep.med.harvard.edu/biclustering`. The yeast cell cycle data set contains 2884 genes and 17 conditions. The human lymphoma data set has 4026 genes and 96 conditions. The algorithm has been implemented in C, and all the experiments have been performed on a Pentium Mobile 1700MHz based machine. The experiments aimed at comparing the three random move strategies when different probabilities and input parameters are given and to discuss the advantages of each of them. In particular, we computed $k = 100$ biclusters varying the probability p of a random move in the interval $[0.1, 0.6]$, for two different configurations of the weights, i.e. $\mathbf{w_1} = (w_{res}, w_{var}, w_{vol}) = (1, 0, 0)$ (dashed lines in Figure 2) and $\mathbf{w_2} = (w_{res}, w_{var}, w_{vol}) = (0.5, 0.3, 0.2)$ (solid lines in Figure 2). Notice that $w_{res} = 1$ and $w_{var} = w_{vol} = 0$, means that the gain function is completely determined by the residue value. We set max_flips to 100, δ to 0, and the frequency thresholds to $f_{row} = 10\%$ and $f_{col} = 100\%$, i.e. a row can participate in at most 10 biclusters, while a column can appear in all the 100 biclusters. The initial random generated biclusters are of size 14×14 for the Yeast data set and of size 20×20 for the Lymphoma data set, while we constrained biclusters to have at least $row_{min} = 10$ rows and $col_{min} = 10$ columns. Figure 2 shows the behavior of the algorithm on the two above mentioned data sets. From the top to the bottom, the figures show the average residue, row variance and volume of the 100

biclusters computed, the average number of flips performed by the method, and the average execution time. Figures on the left concern the Yeast data set, and figures on the right the Lymphoma data set. We can observe that, as regards the residue, the REMOVE-MAX method performs better than the two others, as expected. In fact, its random move consists in removing the gene/condition having the highest residue. Furthermore, increasing the random move probability p improves the value of the residue. The residue of the NOISE method, instead, deteriorates when the probability increases. The REMOVE strategy, on the Yeast data set is better than the NOISE one, but worse than the REMOVE-MAX. On the Lymphoma data set, the value of the residue increases until $p = 0.3$ but then it decreases. The residue scored for parameters $\mathbf{w_1}$ (dashed lines) is lower with respect to that obtained for $\mathbf{w_2}$ (solid lines), for the two strategies NOISE and REMOVE, while, for REMOVE-MAX the difference is negligible. As regards the variance, we can note that the variance of REMOVE is greater than that of NOISE, and that of NOISE is greater than that of REMOVE-MAX for both $\mathbf{w_1}$ e $\mathbf{w_2}$. This is of course expected, since in the former case we do not consider the variance in the gain function in order to obtain the biclusters, while in the latter the weight of the variance is almost as important as that of the residue (0.3 w.r.t 0.5). Analogous considerations hold for the volume, whose value is higher for $\mathbf{w_2}$. Furthermore, the volume is almost constant for the NOISE strategy, because the probability of adding or removing an element in the bicluster is more or less the same, but it decreases for the REMOVE and REMOVE-MAX strategies. These two strategies tend to discovery biclusters having the same size when the probability p increases. As for the average number of flips, we can note that 100 flips are never sufficient for the NOISE method to reach a local minimum, while the other two methods do not execute all the 100 flips. In particular, the RANDOM-MAX strategy is the fastest since it is that which needs less flips before stopping. As regards the execution time, the algorithm is faster for $\mathbf{w_1}$ w.r.t $\mathbf{w_2}$, but, in general, the execution time decreases when the probability p increases and they are almost the same for higher values of p because the number of random moves augments for both. Finally some consideration on the quality of the biclusters obtained. We noticed that the NOISE strategy, which works in a purely random way, gives biclusters with lower gain and it requires more execution time. On the contrary, REMOVE-MAX is positively biased by the removal of those elements in the bicluster having the worst residue, thus it is able to obtain biclusters with higher values of residue and volume, while the REMOVE strategy extract biclusters with higher variance. To show the quality of the biclusters found by *RandomWalkBiclustering*, Figure 3 depicts some of the biclusters discovered in the experiments of Figure 2 by using the REMOVE-MAX strategy for $(w_{res}, w_{var}, w_{vol}) = (0.5, 0.3, 0.2)$. The x axis corresponds to conditions, while the y axis gives the gene expression level. The figures point out the good quality of the biclusters obtained. In fact, their expression levels vary homogenously under a subset of conditions, thus they present a high degree of coherence.

6 Conclusions

The paper presented a greedy search algorithm to find overlapped biclusters enriched with a local search strategy to escape poor local minima. The proposed algorithm is guided by a gain function that combines the mean squared residue, the row variance, and the size of the bicluster through user-provided weights. Different strategies to escape local minima have been introduced and compared. Experimental results showed that the algorithm is able to obtain groups of genes co-regulated and co-expressed under specific conditions. Future work will investigate the behavior of the algorithm for many different combinations of the input parameters, in particular for the weights w_{res}, w_{var}, and w_{vol} employed in the gain function, to study how the trade-off among residue, variance, and volume affects the quality of the solution. We are also planning an extensive comparison with other approaches, and an analysis of the biological significance of the biclusters obtained.

References

1. A. Ben.Dor, R. Shamir, and Z Yakhini. Clustering gene expression patterns. *Journal of Computational Biology*, 6(3-4):281–297, 1999.
2. Y. Cheng and G. M. Church. Biclustering of expression data. In *Proceedings of the 8th International Conference On Intelligent Systems for Molecular Biology (ISMB'00)*, pages 93–103, 2000.
3. M. B. Eisen, P. Spellman, P. O. Brown, and P. Botstein. Cluster analysis and display of genome-wide expression pattern. In *Proceedings of the National Academy of Sciences, USA 8*, pages 14863–14868, 1998.
4. M. R. Garey and D.S. Johnson. *Computers and intractability: A guide to the Theory of NP-Completeness.* San Francisco: Freeman, 1979.
5. G. Getz, E. Levine, and E. Domany. Coupled two-way cluster analysis of gene microarray data. In *Proceedings of the National Academy of Sciences, USA*, pages 12079–12084, 2000.
6. J. A. Hartigan. Direct clustering of a data matrix. *Journal of the American Statistical Association*, 67(337):123–129, 1972.
7. L. Lazzeroni and A. Owen. Plaid models for gene expression data. Technical report, Stanford University, 2000.
8. S. C. Madeira and A. L. Oliveira. Biclustering algorithms for biological data analysis: A survey. *IEEE Transactions on Computational Biology and Bioinformatics*, 1(1):24–45, 2004.
9. B. Selman, H.A. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the 12th Nation Conference on Artificial Intelligence (AAAI'94)*, pages 337–343, 1994.
10. S. Tavazoie, J.D. Hughes, M. Campbell, R.J. Cho, and G.M. Church. Systematic determination of genetic network architecture. *Natural genetics*, 22:281–285, 1999.
11. J. Yang, W. Wang, H. Wang, and P. Yu. Enhanced biclustering on expression data. In *Proceedings of the 3rd IEEE Conference On Bioinformatics and Bioengineering (BIBE'03)*, pages 321–327, 2003.

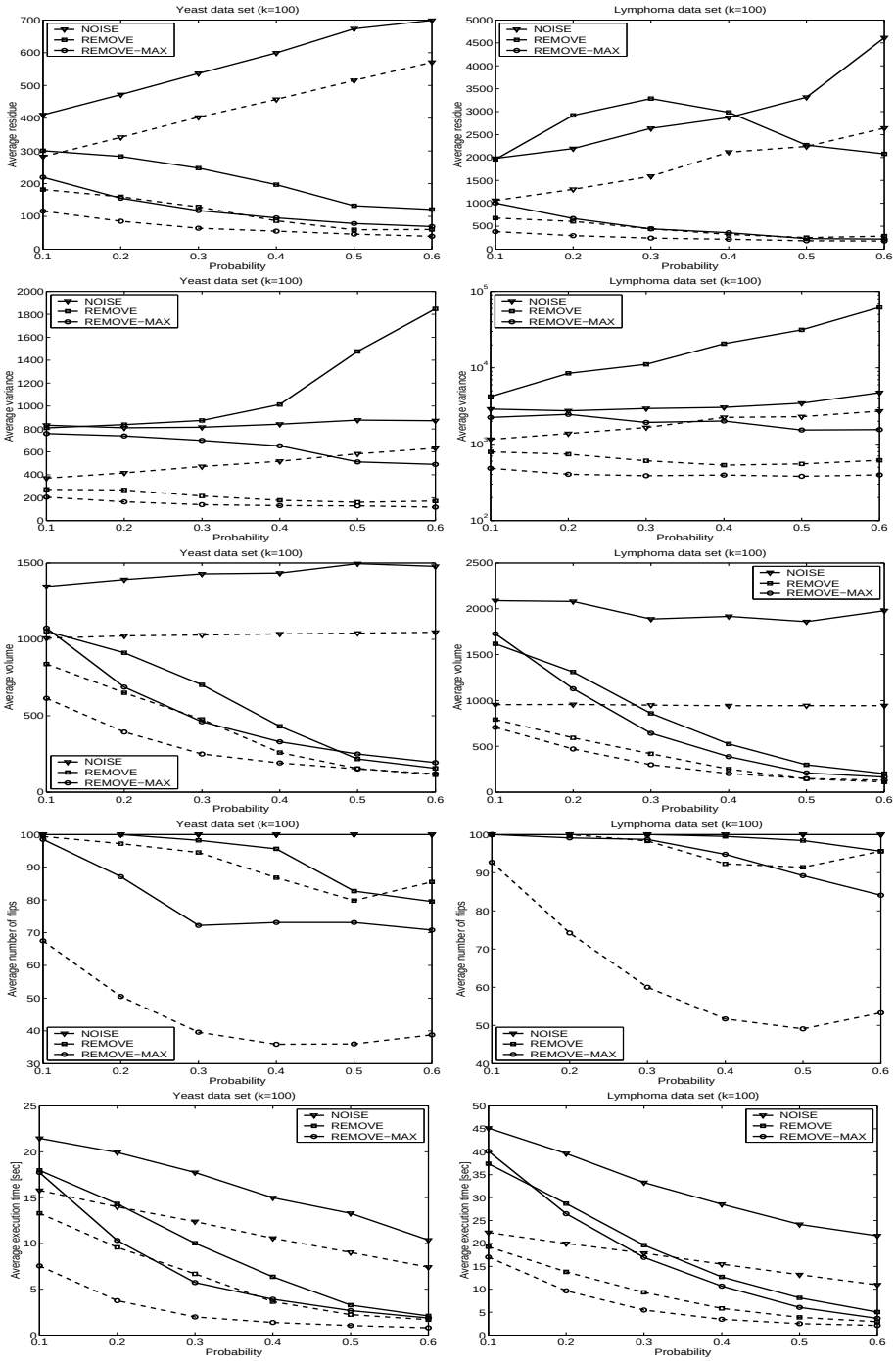

Fig. 2. Average residue, variance, volume, number of flips and execution time for Yeast (on the left) and Lymphoma (on the right) data sets.

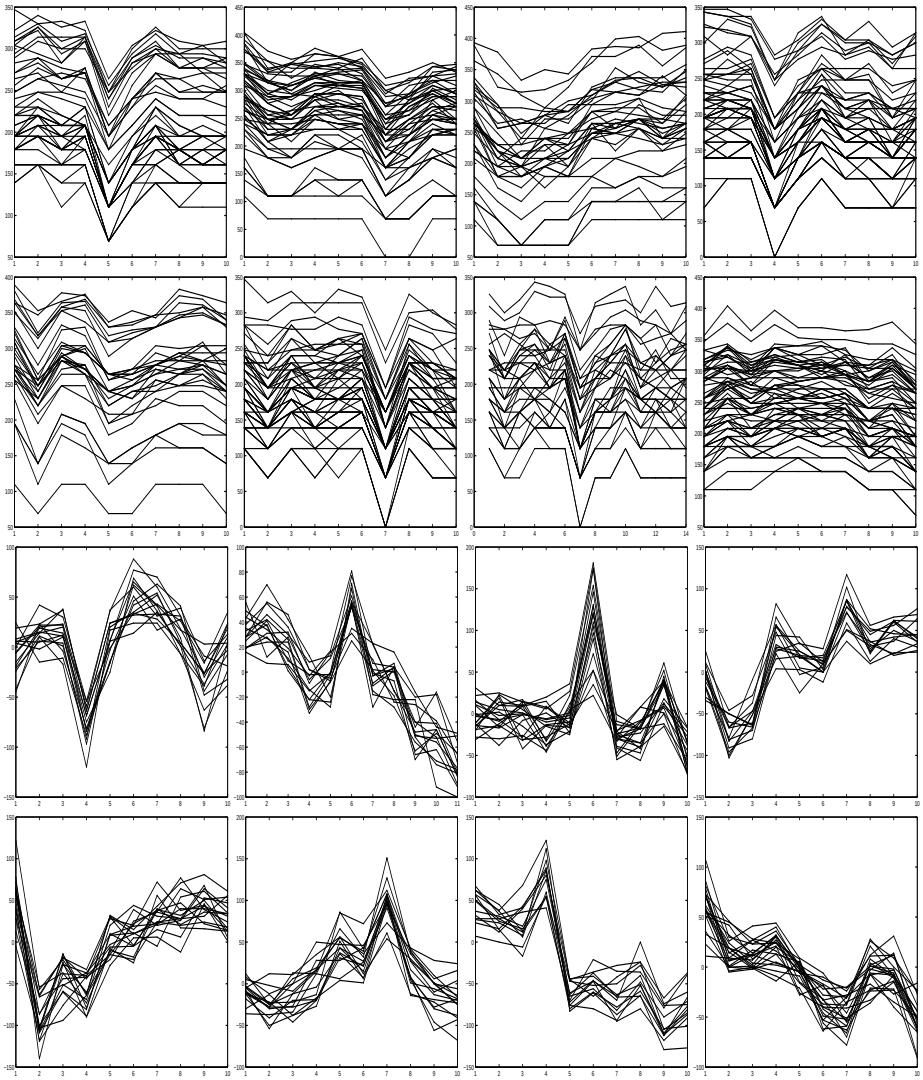

Fig. 3. Biclusters obtained by using the REMOVE-MAX strategy with $(w_{res}, w_{var}, w_{vol}) = (0.5, 0.3, 0.2)$ in the experiments of Figure 2. The first two rows show 8 biclusters of the Yeast data set ($p = 0.3$), while the subsequent two rows show 8 biclusters of the Lymphoma data set ($p = 0.5$). From left to right and from top to bottom the values of (residue, variance, volume) are the following: (70.14, 590.65, 460), (99.51, 705.58, 530), (160.89, 834.79, 360), (113.47, 674.25, 630), (83.04, 439.81, 310), (136.31, 788.27, 580), (180.03, 545.23, 518), and (111.01, 356.24, 640) for the Yeast data set, and (214.63, 1414.45, 150), (169.96, 1626.74, 165), (366.18, 2012.5, 190), (181.34, 2135.5, 140), (323.17, 2472.65, 170), (182.45, 1499.95, 160), (200.24, 3412.19, 130), and (172.94, 1197.03, 220) for the Lymphoma data set.