Debreceni Egyetem
Informatika Kar

Data Mining Algorithms in Bi-Clustering Applications

Témavezető:
Dr. Ispány Márton
Egyetemi Docens

Készítette:
Kocsis Annamária
Programtervező Informatika MSc

Debrecen

2009

Table of Contents

Table of Contents 1
1. Introduction 4
    1.1 The tool for knowledge discovery: clustering 5
    1.2 The role of clustering in genetic studies 5
2. Problem formulation 6
    2.1. Clustering in general 6
    2.2. Differences between clustering and biclustering 6
    2.3 Clustering in gene expression studies 7
    2.4 Formal definitions 8
    2.5 Problem Complexity 10
3. A Brief Introduction to Biclustering 11
    3.1 Bicluster Types 11
    3.2 Bicluster Structure 14
    3.3 Biclustering Algorithms – an overview 16
4. The RandomWalk Biclustering Algorithm 21
    4.1 The Satisfiability Problem and the WSAT algorithm 21
    4.2 Definitions 22
    4.3 The RandomWalk strategy and the local search algorithm 23
        4.3.1. The Local Search Algorithm 24
    4.4 Description of the algorithm 24
    4.5 The greedy move 28
    4.6 Computational complexity 30
    4.7 Practical implementation 31
        4.7.1. Input data options and definitions 31
        4.7.2. Program execution 32
    4.8 Experimental Results 33
    4.9 RandomWalk biclustering compared to Cheng and Church algorithm 34
        4.9.1. Methods in comparing two biclustering algorithms 34
        4.9.2 The Cheng and Church Algorithm 35
            4.9.2.1 Quantitative analysis 35
5. Software description 37
6. Data description 37
7. Conclusion 38
    7.1 Future improvement options of my implementation 39
Appendix 42
Acknowledgements 44
Bibliography 45

1. Introduction

Nowadays there is a very large amount of data stored in real-world databases, and this amount continues to grow fast. The size and complexity of these databases create both an opportunity and a need for (semi-) automatic methods that discover the knowledge "hidden" in such databases. If such knowledge discovery activity is successful, discovered knowledge can be used to improve the decision-making process of an organization.

The core step of knowledge discovery in databases is data mining, which refers to the step that actually extracts knowledge from data. In addition to the data mining step, the knowledge discovery process includes several pre-processing (or data preparation) and post processing (or knowledge refinement) steps. The goal of data preparation methods is to transform the data to facilitate the application of a given (or several) data mining algorithm(s), whereas the goal of knowledge refinement methods is to validate and refine discovered knowledge.

The knowledge discovery process is both iterative and interactive. It is iterative because the output of each step is often feedback to previous steps, and typically many iterations of this process are necessary to extract high-quality knowledge from data. It is interactive because the user, or more precisely an expert in the application domain, should be involved in this loop, to help in data preparation, discovered-knowledge validation and refinement. (*A. A. Freitas, 2002*)

With regard to practical applications, for instance, data about previous sales in a shop might contain interesting knowledge about which kind of product each kind of customer tends to buy. This knowledge can lead to an increase in the sales of a company. As another example, data about a hospital's patients might contain interesting knowledge about which kind of patient is more likely to develop a certain condition. This knowledge can lead to better care for future patients.

1.1 The tool for knowledge discovery: clustering

In essence, the clustering task consists of partitioning the data being mined into several groups (or clusters) of data instances, in such a way that: (a) each cluster has instances that are very similar (or "near") to each other; and (b) the instances in each cluster are very different (or "far away") from the instances in the other clusters. However, satisfying these two basic goals is not enough to obtain a good clustering solution, since these two goals can be trivially satisfied by simply assigning each data instance to a different singleton cluster. Therefore, it is also important to favour a relatively small number of clusters, increasing the number of data instances assigned to a cluster. A major challenge of the clustering task is to find a good trade-off between the above three goals. (*A. A. Freitas, 2002*)

1.2 The role of clustering in genetic studies

DNA chips and other techniques measure the expression level of a large number of genes, perhaps all genes of an organism, within a number of different experimental samples (conditions). The samples may correspond to different time points or different environmental conditions. In other cases, the samples may have come from different organs, from cancerous or healthy tissues, or even from different individuals. Simply visualizing this kind of data, which is widely called *gene expression data* or simply *expression data*, is challenging and extracting biologically relevant knowledge is harder still (*L. Lazzeroni and A. Owen, 2000*.).

In this thesis, I will give an introduction to the use of biclustering gene expression data, provide an overview about the most relevant existing approaches of biclustering, and illustrate the application of biclustering with an implementation of a specific approach in Java environment.

2. Problem formulation

2.1. Clustering in general

Genes are said to be similar if their expression patterns correlate and non-similar otherwise. The goal of gene clustering process is to partition the genes into distinct sets such that genes that are assigned to the same cluster should be similar, while genes assigned to different clusters should be non-similar. Usually there is no single solution that is the "true"/correct mathematical solution for this problem. A good clustering solution should have two merits:

1. High homogeneity: homogeneity measures the similarity between genes assigned to the same cluster.

2. High separation: separation measures the dissonance/ dissimilarity between the clusters.

Each cluster should represent a unique expression pattern. If two clusters have similar expression patterns, then probably they should be merged into one cluster. Note that these two measures are in a way opposite - if you wish to increase the homogeneity of the clusters you would increase the number of clusters, but the price would be a reduction of the separation. There are many formulations for the clustering problem, and most of them are NP-hard. For that reason, heuristics and approximations are used. *(Ron Shamir, 2007)*

2.2. Differences between clustering and biclustering

Clustering algorithms are used to transform a very large matrix of expression values to a more informative collection of gene and condition sets. The members of each cluster are assumed to share function or form some biological module. Clustering is a global technique and as such has several limitations. First, clustering partitions the

elements into groups, so each element may appear in at most one group. Second, when clustering gene expression data, we group the genes according to their behaviour over all experiments (similarly for conditions). This may be problematic when working with large databases that may include many different conditions, only few of which trigger some common gene behaviour.

To try and address these shortcomings, the concept of biclustering was introduced to gene expression analysis. Biclustering was first defined in the seventies (*J. A. Hartigan, 1975)* and was applied to several domains before Cheng and Church (*Y. Cheng and G. M. Church, 2000*) coined its usage in computational biology. Given a gene expression matrix, we search for submatrices that are tightly co-regulated according to some scoring criterion. We do not require the identified submatrices to be disjoint or to cover the entire matrix, instead we wish to build a diverse collection of submatrices that will capture all the significant signals in our data.

The key differences between biclustering and standard clustering Is that biclustering is a local technique by nature, i.e., we try to find local, significant signals in data. Clustering, on the other hand, tries to model the whole dataset by reducing it to a collection of subsets. The basic reasoning of using biclustering is that a successful collection of biclusters will provide a more detailed model of data and can uncover more biological implications of it. However, biclustering results will be harder to interpret. (*Ron Shamir et al., 2005*)

2.3 Clustering in gene expression studies

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

Gene expression matrices have been extensively analyzed in two dimensions: the gene dimension and the condition dimension.

However, applying clustering algorithms to gene expression data runs into a significant difficulty. Many activation patterns are common to a group of genes only

under specific experimental conditions. In fact, our general understanding of cellular processes leads us to expect subsets of genes to be co-regulated and co-expressed only under certain experimental conditions, but to behave almost independently under other conditions. Discovering such local expression patterns may be the key to uncovering many genetic pathways that are not apparent otherwise. It is therefore highly desirable to move beyond the clustering paradigm, and to develop algorithmic approaches capable of discovering local patterns in micro array data (*Amir Ben-Dor at al., 2002).*

Clustering methods can be applied to either the rows or the columns of the data matrix, separately. Biclustering methods, on the other hand, perform clustering in the two dimensions simultaneously. This means that clustering methods derive a *global model* while biclustering algorithms produce a *local model*. When clustering algorithms are used, each gene in a given gene cluster is defined using all the conditions. Similarly, each condition in a condition cluster is characterized by the activity of all the genes. However, each gene in a bicluster is selected using only a subset of the conditions and each condition in a bicluster is selected using only a subset of the genes. The goal of biclustering techniques is thus to identify subgroups of genes and subgroups of conditions, by performing simultaneous clustering of both rows and columns of the gene expression matrix, instead of clustering these two dimensions separately.

We can then conclude that, unlike clustering algorithms, biclustering algorithms identify groups of genes that show similar activity patterns under a specific subset of the experimental conditions. (*Sara C. Madeira, Arlindo L. Oliveira, 2004)*

2.4 Formal definitions

We will be working with an n by m matrix, where element a_{ij} will be, in general, a given real value. In the case of gene expression matrices, a_{ij} represents the expression level of gene i under condition j. Table I illustrates the arrangement of gene expression matrix.

We denote the input matrix of expression data as A, with set of rows X and set of columns Y, where the elements a_{ij} corresponds to a value representing the relation between row i and column j.

Such a matrix A, with n rows and m columns, is defined by its set of rows, $X = \{x_1, ..., x_n\}$, and its set of columns, $Y = \{y_1, ... ,y_m\}$. We will use (X, Y) to denote the matrix A. If $I \subseteq X$ and $J \subseteq Y$ are subsets of the rows and columns, respectively, $A_{IJ} = (I, J)$ denotes the submatrix A_{IJ} of A that contains only the elements a_{ij} belonging to the submatrix with set of rows I and set of columns J.

TABLE I

Gene Expression Data Matrix

	Condition 1	**...**	**Condition *j***	**...**	**Condition *m***
Gene 1	a_{11}	...	a_{1j}	...	a_{1m}
Gene ...	...	...	...	...	...
Gene *i*	a_{i1}		a_{ij}		a_{im}
Gene ...	...	...	...	...	...
Gene *n*	a_{n1}	...	a_{nj}	...	a_{nm}

Given the data matrix A, a cluster of rows is a subset of rows that exhibit similar behaviour across the set of all columns. This means that a row cluster $A_{IY} = (I,Y)$ is a subset of rows defined over the set of all columns Y, where $I = \{i_1, ..., i_k\}$ is a subset of rows $(I \subseteq X \text{ and } k <= n)$. A cluster of rows (I, Y) can thus be defined as a k by m submatrix of the data matrix A. Similarly, a cluster of columns is a subset of columns that exhibit similar behaviour across the set of all rows. A cluster $A_{XJ} = (X, J)$ is a subset of columns defined over the set of all rows X, where $J = \{j_1, ..., j_s\}$ is a subset of columns $(J \subseteq Y \text{ and } s <= m)$. A cluster of columns (X, J) can then be defined as an n by s submatrix of the data matrix A.

A *bicluster* is a subset of rows that exhibit similar behaviour across a subset of columns, and vice-versa. The bicluster $A_{IJ} = (I, J)$ is a subset of rows that exhibit similar behaviour across a subset of columns where $I = \{i_1, ..., i_k\}$ is a subset of rows $(I \subseteq X \text{ and } k <= n)$, and $J = \{j_1, ..., j_s\}$ is a subset of columns $(J \subseteq Y \text{ and } s <= m)$. A bicluster (I, J) can then be defined as a k by s submatrix of the data matrix A.

The specific problem is to identify a set of biclusters $B_k = (I_k, J_k)$ such that each bicluster B_k satisfies some specific characteristics of homogenety under the given data matrix *A*. (*Sara C. Madeira, Arlindo L. Oliveira, 2004)*

2.5 Problem Complexity

An interesting connection between data matrices and graph theory can be established. A data matrix can be viewed as a weighted bipartite graph. A graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, is said to be bipartite if its vertices can be partitioned into two sets L and R such that every edge in E has exactly one en in L and the other in $R: V = L \cup R$. The data matrix $A = (X, Y)$ can be viewed as a weighted bipartite graph where each node $n_i \in L$ corresponds to a row and each node $n_j \in R$ corresponds to a column. The edge between node n_i and n_j has weight a_{ij}, denoting the element of the matrix in the intersection between row i and column j (and the strength of the activation level, in the case of gene expression matrices). (*Sara C. Madeira, Arlindo L. Oliveira, 2004)*

Although the complexity of the biclusterig problem may depend on the exact problem formulation, and, specifically, on the merit function used to evaluate the quality of a given bicluster, almost all interesting variants of this problem are NP-complete.

In its simplest form the data matrix *A* is a binary matrix, where every element a_{ij} is either 0 or 1. When this is the case, a bicluster corresponds to a biclique in the corresponding bipartite graph. Finding a maximum size bicluster is therefore equivalent to finding the maximum edge biclique in a bipartite graph, a problem known to be NP-complete. *(Ren Peeters, 2003)*

More complex cases, where the actual numeric values in the matrix *A* are taken into account to compute the quality of a bicluster, have a complexity that is necessarily no lower than this one, since, in general, they could also be used to solve the more restricted version of the problem, known to be NP-complete.

Given this, the large majority of the algorithms use heuristic approaches to identify biclusters, in many cases preceded by normalization step that is applied to the data matrix in order to make more evident the patterns of interest. Some of them avoid

heuristics but exhibit an exponential worst case runtime. (*Sara C. Madeira, Arlindo L. Oliveira, 2004*)

3. A Brief Introduction to Biclustering

3.1 Bicluster Types

An interesting criterion to evaluate a bicluster algorithm concerns the identification of the type of biclusters the algorithm is able to find. The following four major classes of biclusters can be identified:

1. Biclusterw with constant values
2. Biclusters with constant values on rows or columns
3. Biclusters with coherent values
4. Biclusters with coherent evolutions

1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0

(a) Constant Bicluster

1.0	1.0	1.0	1.0
2.0	2.0	2.0	2.0
3.0	3.0	3.0	3.0
4.0	4.0	4.0	4.0

(b) Constant rows

1.0	2.0	3.0	4.0
1.0	2.0	3.0	4.0
1.0	2.0	3.0	4.0
1.0	2.0	3.0	4.0

(c) Constant Columns

1.0	2.0	5.0	0.0
2.0	3.0	6.0	1.0
4.0	5.0	8.0	3.0
5.0	6.0	9.0	4.0

(d) Coherent Values – Additive Model

1.0	2.0	0.5	1.5
2.0	4.0	1.0	3.0
4.0	8.0	2.0	6.0
3.0	6.0	1.5	4.5

(e) Coherent Values – Multi-plicative Model

S1	S1	S1	S1
S1	S1	S1	S1
S1	S1	S1	S1
S1	S1	S1	S1

(f) Overall Coherent Evolution

S1	S1	S1	S1
S2	S2	S2	S2
S3	S3	S3	S3
S4	S4	S4	S4

(g) Coherent Evolution on the Rows

S1	S2	S3	S4
S1	S2	S3	S4
S1	S2	S3	S4
S1	S2	S3	S4

(h) Coherent Evolution on the Columns

70	13	19	10
29	40	49	35
40	20	27	15
90	15	20	12

(I) Coherent Evolution on the Columns

Fig. 1. Examples of Different Types of Biclusters

The simplest biclustering algorithms identify subsets of rows and subsets of columns with constant values. An example of a constant bicluster is presented in Fig. 1(a). In this case it is natural to consider ways of reordering the rows and columns of the data matrix in order to group together similar rows and similar columns, and discover subsets of rows and subsets of columns (biclusters) with similar values. This approach, however, produces good results only when it is performed on non-noisy data, which does not correspond to the great majority of available data.

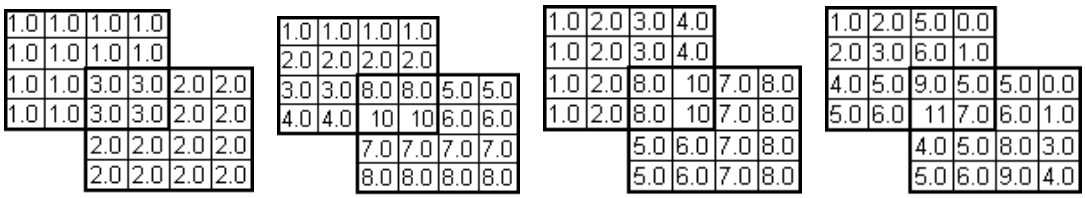
Hartigan (*J. A. Hartigan, 1972*) introduced a partition based algorithm called direct clustering that became known as *Block Clustering*. This algorithm splits the original data matrix into a set of sub-matrices (biclusters).

Other biclustering approaches look for biclusters with coherent values on the rows or on the columns of the data matrix. The bicluster presented in Fig. 1(b) is an example of a bicluster with constant rows, while the bicluster depicted in Fig. 1(c) is an example of a bicluster with constant columns.

In the case of gene expression data, a bicluster with constant values in the rows identifies a subset of genes with similar expression values across a subset of conditions, allowing the expression levels to differ from gene to gene. The row and column normalization allows the identification of biclusters with constant values on the rows or on the columns of the data matrix, respectively, by transforming these biclusters into constant biclusters before the biclustering algorithm is applied.

More sophisticated biclustering approaches look for biclusters with coherent values on both rows and columns. The biclusters in Fig. 1(d) and Fig. 1(e) are examples of this type of bicluster, where each row and column can be obtained by adding a constant to each of the others or by multiplying each of the others by a constant value.

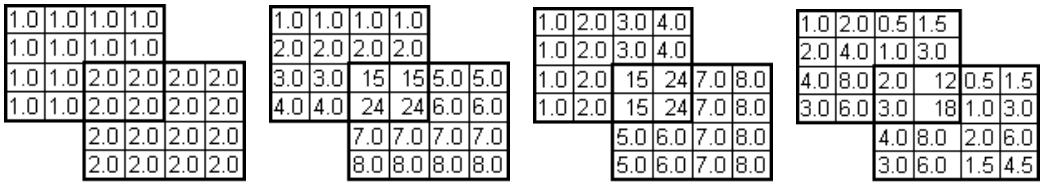
In the case of gene expression data, in the object of interest is to identify more complex biclusters where a subset of genes and a subset of conditions have coherent values on both rows and columns.



(a) Constant Biclusters (b) Constant Rows (c) Constant Columns (d) Coherent Values

Fig. 2. Overlapping Biclusters with General Additive Model

Many of the approaches are based either on additive or multiplicative models, which evaluate separately the contribution of each bicluster without taking into consideration the enteractions between biclusters. In particular, they do not explicitly take into account that the value of a given element, a_{ij}, in the data matrix can be seen as a sum of the contributions of the different biclusters to whom the row *i* and the column *j* belong.



(a) Constant Biclusters (b) Constant Rows (c) Constant Columns (d) Coherent Values

Fig. 3. Overlapping Biclusters with General Multiplicative Model

The last type of biclustering approaches addresses the problem of finding biclusters with coherent evolutions. These approaches view the elements of the matrix as symbolic values, and try to discover subsets of rows and subsets of columns with coherent behaviours regardless of the exact numeric values in the data matrix. The co-evolution property can be observed on the entire bicluster, that is on both rows and columns of the sub-matrix (see Fig. 1(f)), on the rows of the bicluster (see Fig. 1(g)), or on the columns of the bicluster (see Fig. 1(h) and Fig. 1(i)).

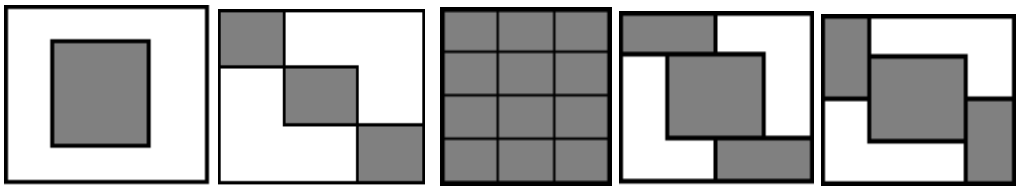
In the case of gene expression data, we may be interested in looking for evidence that a subset of genes is up-regulated or down-regulated across a subset of conditions without taking into account their actual expression values in the data matrix. (*Sara C. Madeira, Arlindo L. Oliveira, 2004)*

3.2 Bicluster Structure

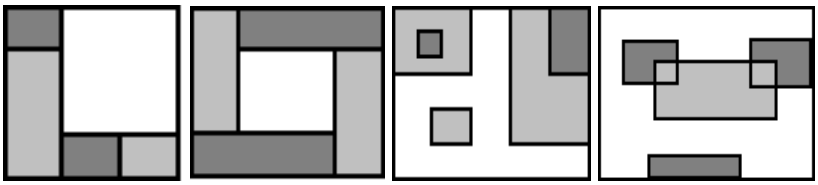
Biclustering algorithms assume one of the following situations: either there is one bicluster in the data matrix (see Fig. 4(a)), or the data matrix contains K biclusters, where K is the number of biclusters we expect to identify and is usually defined a priori.

When the biclustering algorithm assumes the existence of several biclusters in the data matrix, the following bicluster structures can be obtained (see Fig. 4(b) to Fig. 4(i)):

1. Exclusive row and column biclusters (rectangular, diagonal blocks after row and column reorder)
2. Non-Overlapping biclusters with checkerboard structure
3. Exclusive-rows biclusters
4. Exclusive-columns biclusters
5. Non-Overlapping biclusters with tree structures
6. Non-Overlapping non-exclusive biclusters
7. Overlapping biclusters with hierarchical structure
8. Arbitrarily positioned overlapping biclusters



(a) Single Bicluster | (b) Exclusive row and column biclusters | (c) Checkerboard Structure | (d) Exclusive-rows biclusters | (e) Exclusive-columns biclusters



(f) Non-Overlapping biclusters with tree structure

(g) Non-Overlapping non-exclusive biclusters

(h) Overlapping biclusters with hierarchical structure

(i) Arbitrarily positioned overlapping biclusters

Fig. 4. Bicluster Structure

A natural starting point to achieve the goal of identifying several biclusters in a data matrix A is to form a colour image of it with each element coloured according to the value of a_{ij}. It is natural then to consider ways of reordering the rows and columns in order to group together similar rows and similar columns, thus forming an image with blocks of similar colours. These blocks are subsets of rows and subsets of columns with similar expression values, hence, biclusters. An ideal reordering of the data matrix would produce an image with some number *K* of rectangular blocks on the diagonal (see Fig. 4(b)).

Facing this step, the next natural step is to consider that rows and columns may belong to more than one bicluster, and assume a checkerboard structure in the data matrix (see Fig. 4(c)). By doing this we allow the existence of *K* non-overlapping and non-exclusive biclusters where each row in the data matrix belongs to exactly *K* biclusters. The same applies to columns.

Other biclustering approaches assume that rows can only belong to one bicluster, while columns, which correspond to conditions in the case of gene expression data, can belong to several biclusters. This structure is presented in Fig. 4(d). However, these approaches can also produce exclusive-columns biclusters when the algorithm is used the opposite orientation of the data matrix. This means that the columns of the data matrix can only belong to one bicluster while the rows can belong to one or more biclusters (see Fig. 4(e)).

The structures presented in Fig. 4(b) to Fig. 4(e) assume that the biclusters are exhaustive, that is, every row and every column in the data matrix belongs at least to one bicluster. However, we can consider non-exhaustive variations of these structures that make it possible that some rows and columns do not belong to any bicluster. (*Sara C. Madeira, Arlindo L. Oliveira, 2004)*

3.3 Biclustering Algorithms – an overview

Biclustering algorithms may have two different objectives: to identify one or to identify a given number of biclusters. Some approaches attempt to identify one bicluster at a time. Some of them mask the identified bicluster with random numbers, and repeat the procedure in order to eventually find other biclusters.

Other biclustering approaches discover one set of biclusters at a time. Hartigan identifies two biclusters at the time by splicing each existing bicluster into two pieces at each iteration. There are approaches, which perform two-way clustering on the row and column dimensions of the data matrix separately.

Other approaches perform simultaneous bicluster identification, which means that the biclusters are discovered all at the same time. It can happen, e. g. with generating a set of initial biclusters by adding each row/column to each one of them with independent probability and then iteratively improve the quality of the biclusters.

Given the complexity of the problem, a number of different heuristic approaches have been used to address this problem. They can be divided into five classes:

1. Iterative Row and Column Clustering Combination
2. Divide and Conquer
3. Greedy Iterative Search
4. Exhaustive Bicluster Enumeration
5. Distribution Parameter Identification

(*Sara C. Madeira, Arlindo L. Oliveira, 2004)*

3.3.1 Iterative Row and Column Clustering Combination

The conceptually simpler way to perform biclustering using existing techniques is to apply standard clustering methods on the column and row dimensions of the data matrix, and then combine the results to obtain biclusters. A number of authors have proposed methods based on this idea, like CTWC, ITWC or DCC (Double Conjugated Clustering). (*Sara C. Madeira, Arlindo L. Oliveira, 2004.)*

The CTWC (Coupled Two-Way Clustering) uses a hierarchical clustering algorithm applied on each set generating stable clusters of rows and columns, and consequently a set of biclusters at a time. *(Getz et al., 2000)*

The Interrelated Two-Way Clustering (ITWC) is an iterative biclustering algorithm based on a combination of the results obtained by clustering performed on each of the two dimensions of the data matrix separately. For clustering the rows and the columns of the data matrix, Tang used the K-means algorithm. *(Chun Tang et al., 2001)*

3.3.2 Divide-and-Conquer

Divide and conquer algorithms have the significant advantage of being potentially very fast. However, they have the very significant drawback of being likely to miss good biclusters that may be split before they can be identified. (*Sara C. Madeira, Arlindo L. Oliveira, 2004)*

Block clustering was the first divide-and-conquer approach to perform biclustering. The block clustering algorithm begins with the entire data in one block (bicluster). At each iteration it finds the row or column that produces the largest reduction in the total "within block" variance by splicing a given block into two pieces. In order to find the best split into two groups the rows and columns of the data matrix are sorted by row and column mean, respectively. The splitting continues until a given number K of blocks is obtained or the overall variance within the blocks reaches a certain threshold. *(J. A. Hartigan, 1972)*

3.3.3 Greedy Iterative Search

Greedy iterative search methods are based on the idea of creating biclusters by adding or removing rows/columns from them, using a criterion that maximizes the *local* gain. It can happen that they make wrong decisions and loose good biclusters, but they have the potential to be very fast. (*Sara C. Madeira, Arlindo L. Oliveira, 2004)*

Cheng and Church were the first to apply biclustering to gene expression data. Given a data matrix A and a maximum acceptable mean squared residue (*H)* score $\delta > 0$, the goal is to find δ-biclusters, that is, subsets of rows and subsets of columns, with a score no larger than δ (*Y., Cheng, G. M., Church., 2000)*

The FLOC (Flexible Overlapped biClustering) algorithm is also based on the bicluster definition used by Chang and Church, however it performs simultaneous bicluster identification. (*J. Yang et al. 2002, 2003)*

Ben-Dor et al. addressed the identification of large order-preserving submatrices (OPMs) (*A. Ben-Dor et al., 2002*), while Murali and Kasif inroduced a biclustering algorithm that aims at finding xMOTIFs. An xMOTIF is a bicluster with coherent evolutions on the rows. *(T. M. Murali, Simon Kasif, 2003*)

Califano et al. introduced an algorithm that addresses the problem of finding δ – valid ks – patterns. Their goal is to find groups of rows that exhibit coherent values in a subset of the columns but do not have any coherence of values in any of the remaining columns. *(Califano et al., 2000)*

3.3.4 Exhaustive Bicluster Enumeration

Exhaustive bicluster enumeration methods are based on the idea that the best biclusters can only be identified using an exhaustive enumeration of all possible biclusters existent in the data matrix. These algorithms certainly find the best biclusters, if they exist, but have a very serious drawback. Due to their high complexity, they can only be executed by assuming restrictions on the size of the biclusters. (*Sara C. Madeira, Arlindo L. Oliveira, 2004)*

The SAMBA (Statistical-Algorithmic Method for Bicluster Analysis) algorithm performs simultaneous bicluster identification by using exhaustive

enumeration. The algorithm avoids an exponential runtime by restricting the number of rows the biclusters may exhibit. The data is structured into a bipartite graph and the objective is to identify a maximum weight sub-graph, assuming that the weight of a sub-graph will correspond to its statistical significance.

Discovering the most significant biclusters in the data matrix under these weighting schemes is equivalent to the selection of the heaviest sub-graphs in the model bipartite graph. *(A. Tanay et al., 2002)*

Another biclustering algorithm, addressed to perform exhaustive bicluster enumeration, is pClusters. The algorithm starts by deriving a set of candidate *Maximum Dimension Sets* (MDS) for each pair of rows and for each pair of columns. An (x, y) row-pair MDS is a set of columns that defines a maximum width bicluster that includes rows x and y. A similar definition holds for a column-pair MDS. The set of candidate MDS is computed using an efficient method that generates all possible MDS for each row pair and for each column pair. This is done in linear time by ordering the columns in increasing order of the differences between row elements (in the case of the row-pair MDS), and performing a left to right scanning of these ordered array of columns. The set of candidate MDSs is then pruned using properties that relate row-pair MDSs with column-pair MDSs. *(H. Wang et al., 2002)*

3.3.5 Distribution Parameter Identification

Distribution parameter identification biclustering approaches assume a given statistical model and try to identify the distribution parameters used to generate the data by minimizing a certain criterion through an interactive approach. (*Sara C. Madeira, Arlindo L. Oliveira, 2004)*

3.3.6 Overall comparison of the biclustering algorithms

Overall, biclustering algorithms may grouped along the dimensions of analysis considered above. Figure 5 provides an overview of various algorithms, DESCRIBE WHAT TYPE MEANS HERE, provides an example for the structure of the bicluster the algorithm may find, and the way they discover biclusters, approach used to achieve the goal. (*Sara C. Madeira, Arlindo L. Oliveira, 2004)*

	Type	**Structure**	**Discovery**	**Approach**
Block Clustering	Constant	4. f	One set at a time	Divide and Conquer
δ – biclusters	Coherent Values	4. i	One at a time	Greedy
FLOC	Coherent Values	4. i	Simultaneous	Greedy
pClusters	Coherent Values	4. g	Simultaneous	Exhaustive Enumeration
Plaid Models	Coherent Values	4. i	One at a time	Distribution Parameter Identification
PRMs	Coherent Values	4. i	Simultaneous	Distribution Parameter Identification
CTWC	Constant Columns	4. i	One set at a time	Clustering Combination
ITWC	Coherent Values	4. d/4. e	One set at a time	Clustering Combination
DCC	Constant	4. b/4. c	Simultaneous	Clustering Combination
δ – Patterns	Constant Rows	4. i	Simultaneous	Greedy
Spectral	Coherent Values	4. c	Simultaneous	Greedy
Gibbs	Constant Columns	4. d/4. e	One at a time	Distribution Parameter Identification
OPSMs	Coherent Evolution	4. a/4. i	One at a time	Greedy
SAMBA	Coherent Evolution	4. i	Simultaneous	Exhaustive Enumeration
xMOTIFs	Coherent Evolution	4. a/4. i	Simultaneous	Greedy
OP – Clusters	Coherent Evolution	4. i	Simultaneous	Exhaustive Enumeration

Fig. 5 shows the different biclustering approaches in an overall comparison

4. The RandomWalk Biclustering Algorithm

In this section I would like to present the *RandomWalk Biclustering* (RWB), a biclustering algorithm based on a greedy technique enriched with a local search strategy to escape poor local minima. The basic schema of my method derives from the *WSAT* algorithm for the *Satisfiability problem* (*B. Selman et al. 1994*), opportunely modified to deal with the biclustering problem. (*F. Angiulli, C. Pizzuti. 2005*)

4.1 The Satisfiability Problem and the WSAT algorithm

The satisfiability problem (SAT) consists in finding truth assignment that makes a boolean expression true. Many practical NP-complete problems (*M. R. Garey, D. S, Johnson, 1979*) can be represented by a boolean formula in a conjunctive normal form (C N F) and formulated as SAT problems, thus the development of efficient (exact and heuristics) methods for SAT can be exploited for solving combinatorial optimization problems. Among the proposed methods those based on local search (*J., Gu, 1994; B., Selman et al., 1992; B., Selman et al., 1994*) have received a lot of attention because they have been successfully applied to certain hard classes of large satisfiability problems (*S. A. Cook, D. Mitchell, 19xx*). Local search is a very efficient technique devised to solve NP-hard combinatorial optimization problems. Given an initial point, a local minimum is found by searching for a local neighbourhood which improves the value of the object function. Satisfiability can be formulated as an optimization problem in which the goal is to minimize the number of unsatisfied clauses. Thus the optimum is obtained when the value of the object function equals zero, which means that all clauses are satisfied. The main problem in applying local search methods to combinatorial problems is that the search space presents a lot of local optima and, consequently, the algorithm can get trapped at local minima. One of the most popular local search methods for solving *SAT* is *GSAT* (*B., Selman et al., 1992*). This algorithm starts with a randomly generated truth assignment. It then changes (flips) the assignment of the variable that leads to the largest decrease in the total number of unsatisfied clauses. An extension of *GSAT*,

referred to as *random walk SAT (WSAT)*, has been realized with the purpose of escaping from local optima by making upwards moves that could increase the number of unsatisfied clauses (*G. Folino et al., 1998*).

4.2 Definitions

Definition: A bicluster is a sub-matrix *B = (I, J)* of A, where I ε X is a subset of the rows of A, and J ε X is a subset of the columns of A.

Definition: Given a bicluster *B = (I, J)* let

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

denote the mean of the *ith* row of *B*,

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

the mean of the *jth* column of *B*, and

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

the mean of all the elements in the bicluster, where | I | and | J | denote the number of rows and columns of B respectively.

Definition: The residue r_{ij} of an element a_{ij} of the matrix *A* 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 thus be evaluated by computing the mean squared residue r_{IJ} , i.e. the sum of all the squared residues of its elements.

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

$$v_{IJ} = |I| \times |J|.$$

Definition: he residue of a bicluster *B = (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 an element a_{ij} of B and v_{IJ} is its volume.

The mean squared residue of a bicluster, as outlined by Cheng and Church in provides the similarity score of a bicluster. The lower the residue, the larger the coherence of the bicluster, and thus better its quality.

Definition: Given a threshold $\delta < 0$, a sub-matrix $B = (I, J)$ 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 a relatively high variance could be preferred to discard biclusters with almost constant values.

Definition: The row variance var_{IJ} of a bicluster $B = (I, J)$ is defined as

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

The problem of biclustering can then be re-formulated as: given a data matrix A, find a set k of biclusters $B = (I_i, J_i)$, $i = 1; \ldots ; k$ having large volume, a relatively high variance, and a mean squared residue lower than a given threshold δ.

A quality measure of a bicluster based on volume, variance, and residue allows to detect maximal sub matrices having rows, thus genes, coherent though different. In the next section the Random Walk Biclustering (RWB) algorithm is presented. The method adopts the concept of gain that combines mean squared residue, row variance, and volume to improve the quality of a bicluster (*F. Angiulli, C. Pizzuti. 2005*).

4.3 The RandomWalk strategy and the local search algorithm

Local search is a well known optimization technique devised to solve NP-hard combinatorial optimization problems. Given an initial point, a local minimum is found by searching for a local neighbour which improves the value of the object function. The main problem in applying local search methods to combinatorial problems is that

the search space presents many local optima and, consequently, the algorithm can get trapped at local minima. Some heuristics have been implemented to overcome this problem. Some of them are based on allowing random moves to a new neighbouring point of the local search space even if the value of the evaluation function increases or to randomly moving to a point furthest in the search space.

4.3.1. The Local Search Algorithm

The local search algorithm is frequently used in computationally hard optimization problems, as a metaheuristic algorithm. Its aim is to choose a solution from the candidate solutions with maximizing a criterion. Local search algorithm moves from one solution to another which in the search space have already defined. It maximizes the criterion after every move until it finds the solution deemed optimal, however the algorithm is also terminated if the time bound is elapsed.

A good example for the usage of the algorithm is the travelling salesman problem, or the Boolean satisfiability problem.

The local search algorithm starts from a candidate solution and then iteratively moves on to a neighbour solution, defined in the search space. The algorithm accepts the neighbour solution which maximizes the criterion. The search can be also terminated by a time bound, or by a solution, which has not been improved in a given number of steps. Local search algorithms are typically incomplete algorithms since the search may stop even if the best solution found is not optimal. This can happen in case the termination is due to the impossibility of improving the solution, as the optimal solution can lie far from the neighbouring solutions in the search space. (Hoos, H. H., Stutzle, T., 2005)

4.4 Description of the algorithm

A bicluster $B = (I, J)$ can be encoded as a binary string b of length $N + M$, where N and M are the number of rows (genes) and columns (conditions) of the expression matrix, respectively. The first N bits of b are related to the rows, and the remaining M bits are related to the columns. If the value of the *ith* bit is set to 1 it

means that the corresponding *ith* gene, if $1 <= i <= N$, or column, if $N < i <= N+M$, belongs to the bicluster.

The algorithm starts with an initial random bicluster $B = (I, J)$ and searches for a δ – bicluster by successice transformation of B, until a gain function is improved. The transformations consist in the change of membership (called *flip* or *move*) of the row/coloumn that leads to the largest increase of the gain function. If a bit is set from 0 to 1 it means that the corresponding gene or condition, which was not included in the bicluster B, is added to B. Vice versa, if a bit is set from 1 to 0 it means that the corresponding gene or condition, which is removed from the bicluster.

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} (respectively, res_{new}, var_{new}, and vol_{new}) are respectively the values of the residue, row variance and volume of B before (after) the move. 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 $[-\infty, +1]$, consequently the terms $2^{\Delta res}$, $2^{\Delta var}$, and $2^{\Delta vol}$ range in the interval [0, 2], and the whole function assumes values between -1 and +1. The weights w_{res}, w_{var}, and w_{vol} proved 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 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, *RWB* 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 either an argumentation of the residue or a decrease of the row variance or volume.

Algorithm RWB

Input:

- *matrix*: a gene-expression matrix
- δ : stop when this value of residue is reached (0 = stop at local minimum)
- *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 respectively
- 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* = $+\infty$, *local_minimum* = false

while *flips* < *max_flips* and δ < *res* and not *local_minimum*

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_minimum=true*

return the bicluster computed

Fig. 6. The Random Walk Biclustering algorithm

During its execution, in order to avoid get trapped into poor local minima (i. e. low quality biclusters with negative gain in their neighbourhood), instead of performing the flip maximizing the gain, with a user-provided probability *p* the algorithm is allowed to execute a random move. F. Angiulli et al. 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 / from the bicluster if it belongs to it 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.

Fig. 6 shows the algorithm *RWB*. 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 constraints (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_flips*) is reached, or a δ – bicluster is found, or the solution cannot ulteriorly be improved (get trapped into a local minimum). Until the stop condition is not reached, it executes a random move with probability *p*, and a greedy move with probability $(1 - p)$.

When a greedy move is executed, the current value *res* of the mean squared residue is updated. It is used to check if a bicluster having residue below δ has been found. On the contrary, if the value of res_{min} is zero, then the algorithm continues to be executed until the gain can be improved. Since RWB finds one cluster at a time, in order to compute *k* biclusters, the algorithm must be executed *k* times. By fixing two frequency thresholds, f_{row} and f_{col}, the degree of overlapping may be controlled among the biclusters. The former binds a generic row to participate to at most $k \times f_{row}$ biclusters among the *k* to be found. Analogously, f_{col} limits the presence of a column in at most $k \times f_{col}$ 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. (*F. Angiulli et al., 2008*)

4.5 The greedy move

The RWB algorithm executes a pre-defined random move with probability *p*, or a greedy move with probability *(1-p).*

According to the laws of the local search, the greedy move must examine the neighbouring solutions, and include the one, which maximizes a criterion. This case the algorithm must choose that solution, which maximizes the *gain* function. To this end the residue, the variance and the volume must be updated to serve as inputs for the

calculation of the *gain*. In my implementation the random move uses the REMOVE strategy and the greedy move is build upon the followings. Fig. 7 illustrates an example of an initial bicluster generated throughout the former steps of the algorithm, like the first selection from the whole data set or the random moves. When a random generated number reaches or exceeds the value of the probability *p*, the greedy move will be executed.

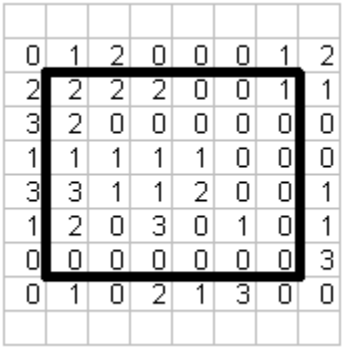


Fig. 7 An initial bicluster

This move is based on the local search algorithm so neighbouring solutions must be defined. A neighbouring solution can be achieved by adding one of the neighbouring columns to the bicluster. Figure 8 displays this solution.

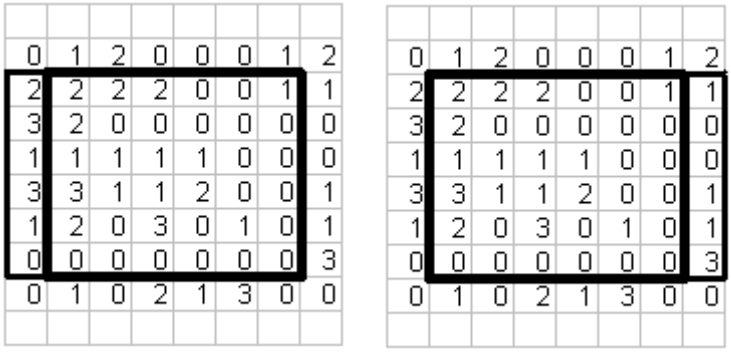


Fig. 8 Adding one of the neighbouring columns

Analogously, the gain may be improved by adding one of the neighbouring rows. Figure 9 shows this possibility. As I earlier mentioned, after each of the different solution the residue, variance and volume must be updated. For each of them the gain function is calculated and compared to each other. The algorithm chooses the solution,

which improves this function with the highest value. If gain exceeds zero, the program executes the chosen solution and creates a new bicluster.

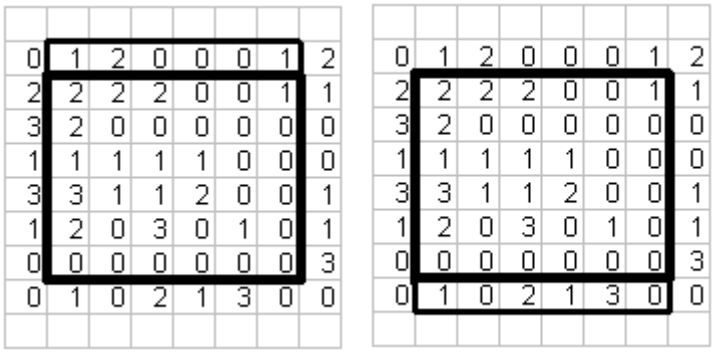


Fig. 9 Adding one of the neighbouring rows

4.6 Computational complexity

In this section the time and space complexity of the method is discussed. As far as the temporal cost of the algorithm is concerned, it 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 $\Sigma_{j \in J} a_{ij}^2$, 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} $(1 <= i <= | I |)$ and a_{Ij} $(1 <= j <= | J |)$ 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, $| I | | J | << NM$, and in general it is dominated by the number N of genes of the overall matrix.

Computing the new row variances can be done in a fast way by exploiting the summations $\Sigma_{j \in J} a_{ij}^2$ already stored. Indeed, if a column is added or removed, the new row variances can be obtained quickly by evaluating the $|I|$ expression:

$$\frac{1}{|J|} \sum_{ij} (a_{ij}^2) - a_{iJ}^2 \; (1 \leqslant i \leqslant |I|)$$

For example, if the *qth* column is added, in order to compute the new variance of the 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 or removed, the corresponding row variance must be computed and added to or subtracted from, respectively, the overall sum of row variances.

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. (*F. Angiulli et al., 2008*)

4.7 Practical implementation

4.7.1. Input data options and definitions

The program was written to cluster the data set given by the Central Institute of Mental Health; however it can be used with other data sets. To this end, a file-chooser interface was developed, which allows the selection of input files – the user should provide the program with a text file, which contains the normalized data, as the algorithm can only handle short type variables (2 bytes, signed, -32,768 to 32,767); databases containing character variables must be converted to numerical format before processing.

Furthermore, user defined input shall be provided for the maximum number of iterations (*max_flips*) or the threshold to define a termination criterion of the algorithm (*threshold*). The user must also specify the probability of the random move and the weights for residue, variance and volume.These are stored in float type variables, which allows a precise definition of these. As default , in line with the practice established in work of F. Anguilli et al., the program performs a maximum of 100 flips, with 0 as a threshold, which means, that the algorithm is allowed to reach the local minimum, the probability of a random move is 0.6, and (0.5, 0.2, 0.3) are used for the weights of residue, variance and volume.

4.7.2. Program execution

The algorithm generates a 20 × 20 initial bicluster in the first step. This bicluster is modified by the random moves or the greedy moves throughout the running of the algorithm. The program terminates when the number of flips reaches the number of maximum flips, when a bicluster with lower residue than the given threshold is found or when the number of rows and columns reaches its minimum (10). The number of rows and columns can be decreased until 10 × 10, (random move), but can be increased (greedy move) until a good bicluster is found or the program terminates. In case of a random move, the REMOVE strategy is applied, which in practice means that a row or a column is randomly removed from the bicluster. The implemented greedy move selects from four neighbouring solutions: the algorithm examines the two neighbouring rows and columns and adds the column/row to the bicluster which maximizes the gain function. Once the algorithm terminates (by 100 flips or finding a local minimum or reaching the minimum number of rows or columns) the computed bicluster is returned, displaying the bicluster and the value of residue, as well as the genes and conditions belonging to the bicluster found. The program computes one bicluster at a time, and the results can be saved into a file after which the next bicluster can be computed analogously as described before.

4.8 Experimental Results

F. Angiulli et al. 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. $w_1 = (w_{res}; w_{var}; w_{vol}) = (1; 0; 0)$ and $w_2 = (w_{res}; w_{var}; w_{vol}) = (0.5; 0.3; 0.2)$. Notice that w_{res} = 1 and $w_{var} = w_{vol} = 0$ means that the gain function is completely determined by the residue value. The author 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.

According to the observations, 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.

As regards the variance, the variance of REMOVE is greater than that of NOISE, and that of NOISE is greater than that of REMOVE-MAX for both w_1 and 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 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, 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 w_1 w.r.t. 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. The NOISE strategy, which works in a purely random way, gives biclusters with higher residue 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 lower values of residue, while the REMOVE strategy extracts biclusters with higher variance. (*F. Angiulli et al., 2008*)

4.9 RandomWalk biclustering compared to Cheng and Church algorithm

4.9.1. Methods in comparing two biclustering algorithms

With the help of a heat map the graphical representation of data can be shown. To this end, the values of the variables are represented as colours in a two-dimensional map. To create the map the authors used a Heat map Builder software. The BIVOC is a powerful software to generate heat maps because it searches for an optimal reordering of rows and columns before creating the picture.

There exist several studies to compare and validate clustering approaches. Generally they make use of validity indices to assess their quality. Validity indices are classified in internal, external, and relative.

Internal indices rely on the input data and are mainly based on the concepts of homogeneity and separation between the groups, external indices use additional data to validate the clustering outcomes, and relative indices measure the influence of the input parameters on the results. In the context of gene expression data there is no general agreement on which validity indices to adopt because of the different types of

biclusterings the algorithms detect: constant values, coherent values, coherent evolutions, checkerboard structure, overlapping, etc.

However, internal indices can be meaningful for comparing the RandomWalk Biclustering approach and the Cheng and Church algorithm, because the basis of the comparison are clearly the residue, volume, and variance values. Furthermore, external criteria correspond to prior biological knowledge on the data set being studied. (*F. Angiulli et al., 2008*)

4.9.2 The Cheng and Church Algorithm

Cheng and Church introduced as first the new paradigm of biclustering to gene expression data analysis. They used greedy search heuristics which generate suboptimal biclusters satisfying the condition of having the mean squared residue below a given threshold δ. Cheng and Church proved that the problem of finding biclusters with low mean squared residue, in particular maximal biclusters with scores under a fixed threshold, is NP-hard because it includes the problem of finding a maximum biclique in a bipartite graph as a special case (*Garey, M. R., Johnson D. S, 1979).*

The algorithm starts with the original data matrix and removes the rows and columns with the highest score as long as the mean squared residue is above the threshold δ. Then rows and columns are added provided that they do not increase the residue above the threshold. To be able to avoid re-obtaining the same biclusters, the algorithm is repeated on a modified data matrix where the values of those elements a_{ij} already inserted in a bicluster are substituted with random numbers. This choice, however, has the negative effect of precluding the discovery of new biclusters, in particular those having significant overlaps with those already found. In addition, the Cheng and Church algorithm assumes that the data matrix does not contain missing values. (*F. Angiulli et al., 2008*)

4.9.2.1 Quantitative analysis

The experiments analyze the biclusters obtained by the two algorithms when the number of biclusters k, and the maximum residue value δ are varied. In practice,

the two algorithms search a number of biclusters from 5 to 50, that is for k = 5; 10; 20; 30; 50, when the residue threshold δ is fixed to 0, 500, 800 and 1000. For all of experiments the RWB algorithm used the weight configuration w = (0.6; 0.1; 0.3), a maximum number of flips of 3000, the frequency threshold f_{row} = 100% and f_{col} = 100%. This means, that each row and column can potentially participate in all the k biclusters. Even though a constrain, like that could lead to heavy overlapping between biclusters, F. Anguilly et al. verified that the rate of overlapping is marginal, below the 2%.

Finally, the probability of random move was fixed to p = 0.5 for all the experiments, and REMOVE-MAX strategy was used as a random move. Both RWB and Cheng and Church algorithms have been executed 10 times and the values obtained by averaging these 10 executions are reported.

F. Anguilly at al. used two data sets, one of them was the Yeast Saccharomyces cerevisiae cell cycle expression data set, the other was the human B-cell Lymphoma data set. The former contained 2884 genes and 17 conditions, while the latter 4026 genes and 96 conditions. Since the Central Institute of Mental Health Mannheim presented a data set with 4496 genes and 72 conditions which can be compared with the Lymphoma data set, therefore the results of the latter will be emphasized.

According to the observations the RWB algorithm, on the Leukemia data set, always obtains biclusters with lower residue, higher volume, and lower variance than the Cheng and Church algorithm. This means that the former algorithm is able to find highly coherent genes and conditions which lead to low residue while the latter tends to find smaller biclusters with less coherence, i.e. larger residue. Lower variance results for RWB are due to the fixed weight value of 0.1. Another important observation to emphasize is that RWB maintains a residue value very near to the threshold requested as long as the number of biclusters to find increases. Analogously, the volume of biclusters do not sensibly varies along the executions. On the contrary, Cheng and Church is not able to satisfy the threshold requirement because, though when we ask for few biclusters the average residue does not significantly move away from the fixed value, when the number of biclusters augments, the residue sensibly

increases while the volume decreases. This behaviour probably is due to the introduction of random numbers adopted by Cheng and Church to bias the algorithm to find non-overlapping biclusters. As regards RWB, the random strategy applied allows to explore different parts of the search space thus assuring low levels of overlapping, although the frequency thresholds of 100% enable each row and column to take part to all the biclusters. (*F. Angiulli et al., 2008*)

5. Software description

Angiulli et al. implemented the RandomWalk Biclustering algorithm in C programming language. For my implementation I used the Java language. The program was written under two operation systems, OpenSUSE Linux and Windows XP, with the help of Eclipse. It uses the version of Java 2.

The Java implementation of the Cheng and Church algorithm is publicly available on the web site of the authors[1]. (*F. Angiulli et al., 2008*)

6. Data description

The data set was collected by the Department of Genetic Epidemiology in Psychiatry, Central Institute of Mental Health, in Mannheim, Germany. It contains 3211 patients' data in the rows, and 72 conditions as columns.

The patients are sorted into four groups according to the disorders, from which they suffer. These are the bipolar disorder, depression, schizoaffective disorder, and schizophrenia. The data set mainly consist of 0/1 codes however there are exceptions. Ethnicity, for example, is defined in a 1-6 range of numbers, while sex is described with letters, F, for females, and M, for males. Where no data were available, a special code: -999 was used. There are variables, which contain continuous values, like age, age at the time of morbidity, sociability and withdrawal, peer relationship, scholastic performance, adoption to school and socio-sexual aspects of life[2].

[1] Downloadable on http://cheng.ececs.uc.edu/biclustering/Biclustering.java
[2] The other conditions are enumerated in Appendix.

The whole data set were normalized to be able to gain information from that faster. The new format consists of 4496 rows and 72 columns.

7. Conclusion

In this thesis I wanted to give a brief summary about the role of data mining, and especially the role of clustering, in biological applications. Data mining algorithms are frequently used to cluster and to discover knowledge in huge databases; however they have limitations in applications which use e. g. micro array data. Clustering techniques can be used to group either genes or conditions separately, while e. g. gene expression data sets need the selection of groups of genes only under specific experimental conditions. Biclustering approaches, however, offer the possibility to form clusters in the two dimensions simultaneously.

To be able to examine biclustering methods more detailed, I selected an approach, the RandomWalk Biclustering, and tried to describe the theory on which it is based e. g. Satisfiabilty problem or local search optimization problem. I also summarized the functioning of this approach, as well as the experimental results compared to another approach.

The RandomWalk method uses greedy technique, and it executes either a pre-defined random move, or a greedy move based on the local search algorithm. The author, F. Angiuilli, made offers for three different random move strategies, from that I implemented one, the REMOVE strategy. Therefore my implementation can be complemented in the future.

In addition, I would like to mention a powerful biclustering application written by the Reverse Engineering Group at Swiss Federal Institute of Technology, Zurich in the Java programming language. The Biclustering Analysis Toolbox (BicAT) is a software platform for clustering-based data analysis that integrates various biclustering and clustering techniques in terms of a common graphical user interface. Furthermore, BicAT provides different facilities for data preparation, inspection, and post-processing such as discretization, filtering of biclusters according to specific criteria, or gene pair analysis for constructing gene interconnection graphs. To help

visualization, the toolbox implements heat maps. The possibility to use different biclustering algorithms inside a single graphical tool allows the user to compare clustering results and choose the algorithm that best fits a specific biological scenario (*Simon Barkow et al., 2005*). Figure 10 displays the graphical interface of BicAT.

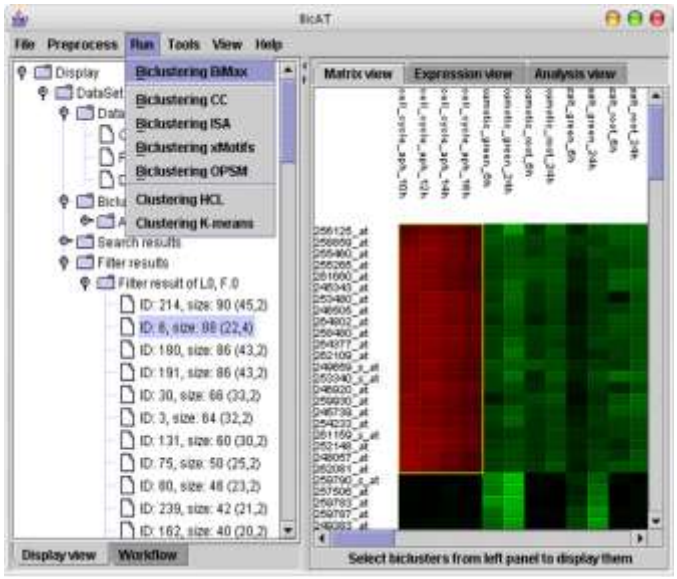


Figure 10. The BicAT application[3]

This toolbox implements five biclustering algorithms, like Cheng and Church, ISA, OPSM, xMotif, Bimax, and two clustering algorithm, such as K-Means and hierarchical clustering.

7.1 Future improvement options of my implementation

One of the options to improve my implementation is to create the two other offered random move strategies, NOISE and REMOVE-MAX.

NOISE is a purely random strategy, which generates a random number in the range of columns or rows, and then that column or row will be removed or added from/to the initial bicluster. This random strategy uses the whole matrix as its basis

[3] The BicAT toolbox is freely available at http://www.tik.ee.ethz.ch/sop/bicat

instead of the bicluster. So it is possible to add a row or column to the bicluster, which is out of it, but exists in the whole data matrix.

The other random strategy, REMOVE-MAX, chooses the rows and columns it removes, based on calculations. This strategy uses a method for scoring, which is analogue to the scoring algorithm of the Cheng and Church. The aim is to decrease the residue of the bicluster under a given threshold δ. The method finds the rows and the columns whose scores are the highest, and remove them from the bicluster. The scoring method is based on the calculation of the residue of an element and the residue of the bicluster, to define how the data fits together within that matrix.

In my implementation the RWB algorithm generates a 20×20 initial bicluster, which could be reduced to 10×10. This feature may be modified to allow the user to define the size of the generated bicluster, and the minimum number of rows and columns based on the appropriate data set. Analogously, the degree of overlapping may be also implemented, and defined by the user.

To make the program more powerful a visualization technique can be also implemented. To this end an open source Java tool, the JFreeChart[4] would be the most efficient. This tool enables developers to display professional quality charts.

[4] The JFreeChart is downloadable on: http://www.jfree.org/jfreechart/

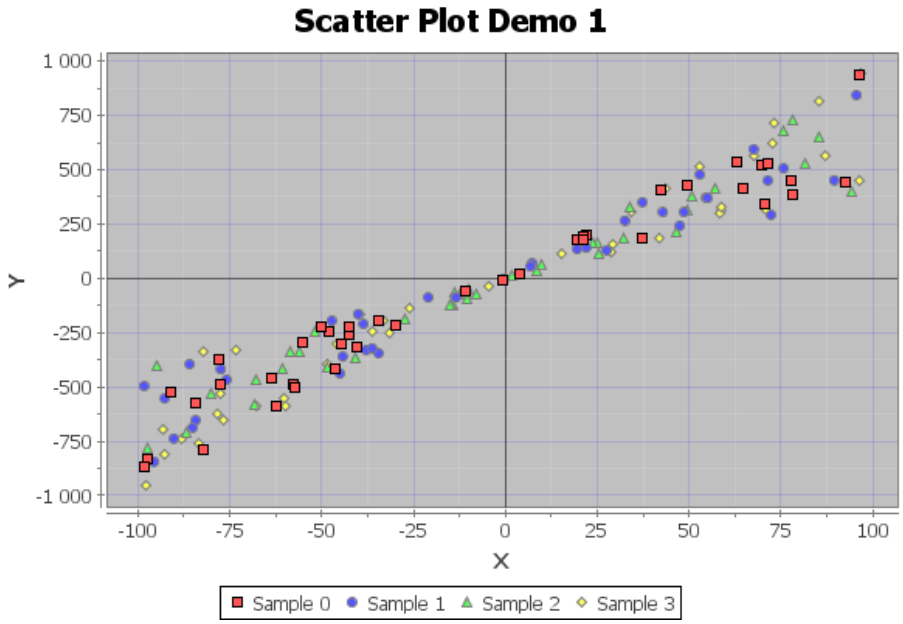


Figure 11 Scatter Plot technique with JFreeChart

To visualize the bicluster found, the Scatter Plot or the Line Chart technique could work effectively. With the implementation of JFreeChart the program would offer a useful help to analyze the results. Figure 11 illustrates the Scatter Plot and Figure 12 the Line Chart technique.

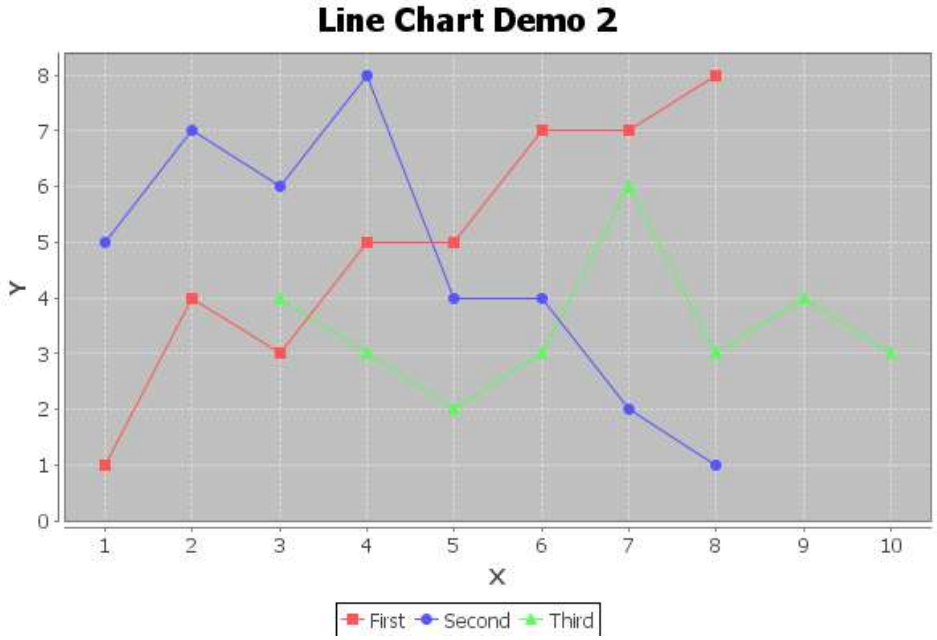


Figure 12 Line Chart technique with JFreeChart

Appendix

The conditions examined in the data sets, and their possible values.

Employment\"' {1,0}
Alcohol / Drug abuse within year of onset\"' {1,0}
Family history of schizophrenia\"' {1,0}
Family history of other psychiatric disorder\"' {1,0}
Coarse brain disease prior to onset\"' {1,0}
Definite psychosocial stressor prior to onset\"' {1,0}
Bizarre behavior\"' {1,0}
Catatonia\"' {1,0}
Excessive activity\"' {2,1,0}
Reckless activity\"' {2,1,0}
Distractibility\"' {2,1,0}
Reduced need for sleep\"' {2,1,0}
Agitated activity\"' {3,2,1,0}
Slowed activity\"' {3,2,1,0}
Loss of energy / tiredness\"' {3,2,1,0}
Speech difficult to understand\"' {1,0}
Incoherent\"' {1,0}
Positive formal thought disorder\"' {1,0}
Negative formal thought disorder\"' {1,0}
Pressured speech\"' {2,1,0}
Thoughts racing\"' {2,1,0}
Restricted affect\"' {1,0}
Blunted affect\"' {1,0}
Inappropriate affect\"' {1,0}
Elevated mood\"' {2,1,0}
Irritable mood\"' {2,1,0}
Dysphoria\"' {3,2,1,0}
Diurnal variation\"' {1,0}
Loss of pleasure\"' {3,2,1,0}
Diminished libido\"' {1,0}
Poor concentration\"' {3,2,1,0}
Excessive self reproach\"' {3,2,1,0}
Suicidal ideation\"' {3,2,1,0}
Initial insomnia\"' {3,2,1,0}
Middle insomnia\"' {1,0}
Early morning waking\"' {3,2,1,0}
Excessive sleep\"' {3,2,1,0}
Poor appetite\"' {3,2,1,0}
Weight loss\"' {3,2,1,0}
Increased appetite\"' {3,2,1,0}

Weight gain\"' {3,2,1,0}
Increased sociability\"' {2,1,0}
Persecutory delusions\"' {1,0}
Well organised delusions\"' {1,0}
Increased self esteem\"' {2,1,0}
Grandiose delusions\"' {2,1,0}
Delusions of influence\"' {1,0}
Bizarre delusions\"' {1,0}
Widespread delusions\"' {1,0}
Delusions of passivity\"' {1,0}
Primary delusional perception\"' {1,0}
Other primary delusions\"' {1,0}
Delusions & Hallucinations last for one week\"' {1,0}
Persecutory / jealous delusions & hallucinations\"' {1,0}
Thought insertion\"' {1,0}
Thought withdrawal\"' {1,0}
Thought broadcast\"' {1,0}
Delusions of guilt\"' {1,0}
Delusions of poverty\"' {1,0}
Nihilistic delusions\"' {1,0}
Thought echo\"' {1,0}
Third person auditory hallucinations\"' {1,0}
Running commentary voices\"' {1,0}
Abusive / accusatory / persecutory voices\"' {1,0}
Other auditory hallucinations\"' {1,0}
Non-affective hallucination in any modality\"' {1,0}
Life time diagnosis of alcohol abuse / depend\"' {1,0}
Life time diagnosis of cannabis abuse / depend\"' {1,0}
Life time diagnosis of other abuse / depend\"' {1,0}
Alcohol abuse / dependence with psychopathology\"' {1,0}
Cannabis abuse / dependence with psychopathology\"' {1,0}
Other abuse / dependence with psychopathology\"' {1,0}

Acknowledgements

I would like to express my gratitude to all those who gave me the possibility to complete this thesis.

I want to thank to my professors, Gábor Galambos and József Békési, who taught me at the University of Szeged, Gyula Juhász Teacher Training College, that they trusted me and wrote about me letters of recommendation, and helped me to be able to work at the University of Heidelberg.

I have furthermore to thank to my supervisor, Márton Ispány, from the University of Debrecen, Faculty of Computer Science for the recommendation and for all his support.

I am deeply indebted to the KAAD (the Catholic Academic Support Service for Foreign Students, Germany) that they made it possible for me to be enrolled in the University of Heidelberg, and for the scholarship they gave me, which enabled to be able to work without worries.

I am honestly thankful to the Department of Combinatorial Optimization at the University of Heidelberg, especially to Professor Gerhard Reinelt, who invited me to participate in this project, and to Markus Oswald for his explanations, suggestions and valuable help.

Especially, I would like to give my special thanks to Dr. Ádám Gondos for all his help, support, interest, for correcting the English style and grammar and offering suggestions. He helped this work every possible way.

Finally, a good friend of mine, Zsolt Rapcsák was a great help in writing the program in Java.

Bibliography

Anguilli, Fabrizio; Pizzuti, Clara. Gene Expression Biclustering Using Random Walk Strategies. In *A. Min Tjoa, J. Trujillo (Eds.): DaWaK 2005, LNCS 3589, Springer-Verlag Berlin Heidelberg*, pp. 509-519, 2005.

Angiulli, Fabrizio; Cesario, Eugenio; Pizzuti, Clara. RandomWalk biclustering for microarray data. *Elsevier Inc.* 2007

Barkow, Simon; Bleuler, Stefan; Preli´c, Amela; Zimmermann, Philip; Zitzler, Eckart. BicAT: A Biclustering Analysis Toolbox. In *Bioinformatics*, Vol. 00 no. 00 Pages 1–2, Oxford University Press, 2005

Ben-Dor, Amir; Chor, Benny; Karp, Richard; Yakhini, Zohar. Discovering local structure in gene expression data: The order–preserving submatrix problem. In *Proceedings of the 6th International Conference on Computacional Biology (RECOMB'02)*, pages 49–57, 2002.

Busygin, Stanislav; Jacobsen, Gerrit; Kramer, Ewald. Double conjugated clustering applied o leukemia microarray data. In *Proceedings of the 2nd SIAM International Conference on Data Mining*, pages 420-436, 2002

Califano, Andrea; Stolovitzky, Gustavo; Tu, Yunai. Analysis of gene expression microarrays for phenotype classification. In *Proceedings of the International Conference on Computational Molecular Biology*, pages 75-85, 2000

Cheng, Y.; Church G., M.. Biclustering of expression data. *In Proc. ISMB'00,* pages 93-103. AAAI Press, 2000

Cook, S. A.; Mitchell, D.. Finding hard instances of the satisfiability problem: a survey, *DIMACS series in Discrete Mathematics*, 19xx

Flanagan, David. Java Examples in a Nutshell - Second edition. A tutorial companion to Java in a Nutshell *O'Reilly & Associates*, 2000

Freitas, A., A. Data Mining and Knowledge Discovery with Evolutionary Algorithms. *Springer Verlag Berlin Heidelberg*, 2002

Folino, Gianluigi; Pizzuti, Clara; Spezzano, Giandmenico. Combining Cellular Genetic Algorithms and Local Search for Solving Satisfiability Problems. Tools with

Artificial Intelligence, 1998. Proceedings. (Tenth IEEE International Conference) pages: 192-198

Garey, M. R.; Johnson, D. S.. Computers and Intractability. A guide to the theory of NP-completeness. *San Fransisco, Feeman*, 1979

Geary, David M.. Graphic Java 2. Mastering the JFC – Third edition, Volume II, Swing. *The Sun Microsystems Press*, 1999

Getz, G.; Levine, E.; Domany, E. Coupled Two-Way Clustering Analysis of Gene Microarray Data. In *Proceedings of the Natural Academy Sciences USA*, pages 12079-12084, 2000

Hartigan, J. A.. Direct clustering of a data matrix. *Journal of the American Statistical Association (JASA)*, 67(337): 123-129, 1972

Hartigan, J. A.. Clustering Algorithms. *John Wiley and Sons*, 1975

Holzner, Steve. Eclipse – First edition. *O´Reilly Media Inc.* 2004

Hoos, H., H.; Stutzler, T.. Stochastic Local Search: Foundations and Applications. Morgan Kaufmann, 2005

Lazzeroni, L.; Owen, A.. Plaid Models for Gene Expression Data, technical report, *Stanford Univ.*, 2000.

Murali, T. M.; Kasif, Simon. Extracting conserved gene expression motifs from gene expression data. In *Proceedings of the Pacific Symposium on Biocomputing*, *volume 8*, pages 77-78, 2003

Naughton, Patrick; Schildt, Herbert. Java 2 The Complete Reference – Third edition. *Osborne/McGraw-Hill*, 1999

Niemeyer, P.; Knudsen, J. Learning Java. *O'Reilly Media Inc.* 2005

Peeters, Ren. The maximum edge biclique problem is NP-complete. *Discrete Applied Mathematics*, 131(3):651-654, 2003

Sara C., Madeira; Arlindo, L., Oliveira. Biclustering Algorithms for Biological Data Analysis: A Survey in *INESC-ID TEC. REP. 1/2004 JAN 2004*

Selman, B.; Levesque, H.; Mitchell, D.. *A New Method for Solving Hatd Satisfiability Problems. Proc. of AAAI, 1992*

Selman, B.; Kautz, H. A.; Cohen, B.. *Noise strategies for improving local search. In Proceedings of the 12th Nation Conference on Artificial Intelligence (AAAI` 94), pages 337-343, 1994*

Shamir, Ron; Hoffer, Gil; Peleg, **Tal.** Analysis of Gene Expression Data. *Lecture 4: March 22,* 2007. Tel Aviv University

Shamir, Ron; Felder, Yifat; Ourfali, Oved. Analysis of Gene Expression Data. *Lecture 3: March 15, 2007*. Tel Aviv University

Sheng, Qizheng; Moreau, Yves; De Moor, Bart. Biclustering microarray data by gibbs sampling. In *Bioinformatics, volume 19 (Suppl. 2)*, pages ii196-ii205, 2003

Tanay, Amos; Sharan, Roded; Shamir, Ron. Discovering statistically significant biclusters in gene expression data. In *Bioinformatics*, volume 18 (Suppl. 1), pages S136-S144, 2002

Tang, Chun; Zhang, Idon; Ramanathan, Murali. Interrelated Two-Way Clustering: an Unsupervised Approach for Gene Expression Data Analysis. In *Proceedings of the 2nd IEEE International Symposium on Bioinformatics and Bioengineering*, pages 41-48, 2001

Yang, Jiong; Wang, Wei; Haixum, Wang; Yu, Philip. δ – clusters: Capturing subspace correlation in a large data set. In *Proceedings of the 18th IEEE International Conference on Data Engineering*, pages 517-528, 2002

Yang, Jiong; Wang, Wei; Wang, Haixun; Yu, Philip. Enhanced biclustering on expression data. In *Proceedings of the 3rd IEEE Conference on Bioinformatics and Bioengineering*, pages 321-327, 2003

Zukowsky, J.; Vanhelsuwé, L.; Holzner, S.; Heller, P.; Roberts, S. Java 2 – Das Buch. *SYBEX-Verlag GmbH*, Düsseldorf, 2001

Wang, Haixun; Wang, Wei; Yang, Jiong; Yu, Philip. Clustering by pattern similarity in large data sets. In Proceeding of the 2002 ACM SIGMOD International Conference on Management of Data, pages 517-528, 2002