Order Preserving Clustering by Finding Frequent Orders in Gene Expression Data

Li Teng and Laiwan Chan

Department of Computer Science and Engineering,
The Chinese University of Hong Kong, Hong Kong

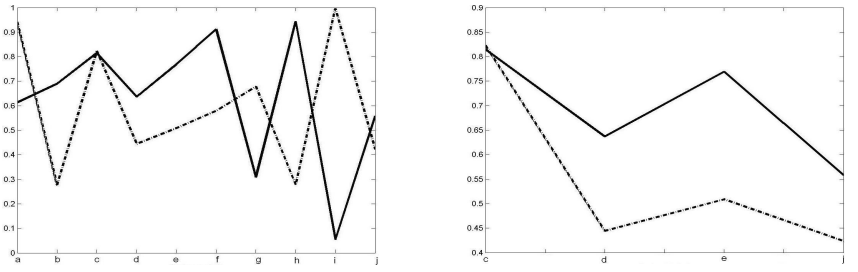
Abstract. This paper concerns the discovery of Order Preserving Clusters (OP-Clusters) in gene expression data, in each of which a subset of genes induce a similar linear ordering along a subset of conditions. After converting each gene vector into an ordered label sequence. The problem is transferred into finding frequent orders appearing in the sequence set. We propose an algorithm of finding the frequent orders by iteratively Combining the most Frequent Prefixes and Suffixes (CFPS) in a statistical way. We also define the significance of an OP-Cluster. Our method has good scale-up property with dimension of the dataset and size of the cluster. Experimental study on both synthetic datasets and real gene expression dataset shows our approach is very effective and efficient.

1 Introduction

In gene expression dataset, each row stands for one gene and each column stands for one condition. Traditional methods for pattern discovery in gene expression matrices are based on clustering genes (conditions) by comparing their expression levels in all conditions (genes). However, general understanding of cellular processes leads us to expect subsets of genes to be coregulated and coexpressed only under certain experimental conditions. Recent research works [1-8],focus on discovering such local patterns embedded in high dimensional gene expression data.

Order preserving clustering (OP-Clustering)[9] is one discipline of looking for submatrices in which value the rows induce the same linear ordering in the columns. In former study of gene expression profiles, researchers regard there are certain stages for genes. They use on or off to stand for the state of gene. There could be more than two stages. The idea of stages encourages us to measure the similarities by comparing the condition orders between two genes. Therefore we expect the data to contain a set of genes and a set of conditions such that the genes are identically ordered on the conditions. By finding the hidden order and genes that support it, we can potentially find the different stages shared by the genes. Figure 1 shows an example of the order preserving subsequences in two data sequences. In Figure 1(a) there is no obvious trend between the two sequences. However, if we only consider columns [c d e j] as shown in Figure 1(b), the two subsequences show strong coherence on these four columns. The

J.C. Rajapakse, B. Schmidt, and G. Volkert (Eds.): PRIB 2007, LNBI 4774, pp. 218–229, 2007.



(a) (b)

Fig. 1. An example of the Order Preserving Subsequences in two sequences.(a) Original data, (b) the OP subsequences.

two subsequences show the same ascending pattern if we rearrange the columns as [j d e c].

Problem Statement. Given an n×m gene expression dataset A, where $G = \{g_1, g_2, \ldots, g_n\}$ is the set of genes (also the rows), $C = \{c_1, c_2, \ldots, c_m\}$ is the set of conditions (columns). Order Preserving Cluster (OP-Cluster) $OPC = (P, Q)$ is a submatrix of A, where $P \subseteq G$, $Q \subseteq C$, that all the genes in P share the same linear order on the conditions in Q.

That means there is a permutation of the conditions in Q, after which all the genes in P show the ascending patterns. An embedded OP-Cluster can be identified by the order of conditions which involve in. Here we call the permutation of conditions the order of the OP-Cluster and the subset of genes the supports for that order. From this point of view the work of finding OP-Clusters is related to the so called sequential pattern mining in some other literature [10][11][12].

In this paper we propose a model of OP-Clustering on numerical dataset. After converting the numerical dataset into a sorted label sequence set. We present a heuristic method of finding the frequent orders by combining the most frequent prefix and suffix iteratively in a statistical way. The structure of the paper is as follow. In section 2 some related work is discussed. The OP-Clustering algorithm (CFPS) is presented in section 3. The experimental results on both synthetic dataset and real gene expression dataset are presented in section 4 and we draw the conclusion in section 5.

2 Related Work

The concept of Order-Preserving SubMatrix (OPSM) was first introduced by Ben-Dor et al.[9] and they also proved the OPSM problem was NP-hard in the worst case. By discovering a subset of genes identically ordered among a subset of conditions they focused on the coherence of the relative order of the conditions rather than the coherence of actual expression levels. A stochastic model was developed by them to discover the best row supported submatrix given a fixed

size of conditions. For OPSM the quality of their resulted cluster is very sensitive to some given parameters and the initial selection of partial models.

Liu et al.[13] proposed an exhaustive bicluster enumeration algorithm to discover all significant OP-Clusters. Their pattern discovery algorithm is heavily inspired in sequential pattern min-ing algorithms [14]. Mining sequential pattern was first introduced in the work of Agrawal et al.[10], and most methods in this area are based on breadth-first or depth-first search. Liu et al. used a compact tree structure to store the identities of the rows and the sequences sharing the same prefixes. A depth-first algorithm was devised to discover the OP-Clusters with a user-specified threshold. The drawback is that the construction of the OPC-Tree is very time consuming and it needs excessive memory resources. For large dataset pruning techniques have to be taken before it can be effective.

Bleuler S. and Zitzler Z.[15] used a scoring function which combines the mean squared residue score with the OPSM concept. It allows the arbitrarily scale and degree of orderedness required for a cluster. They proposed an evolutionary algorithm framework that allows simultaneous clustering over multiple time course experiments. A local search procedure based on a greedy heuristic was applied to each individual before evaluation. Their method is still time consuming and they can only find non-overlapping clusters in one run.

Most existing methods based on exhaustive searching and need excessive computation and memory cost. In our work, a heuristic method based on statistical information is proposed to find the OP-Clusters by finding the frequent orders in the sequence set which comes from the original numerical dataset.

3 Algorithm

In this section we present the algorithm. Firstly the main procedures of the algorithm is introduced in section 3.1. Then some details of the algorithm are given in section 3.2-3.4. Examples are used to to illustrate the algorithm.

3.1 A Top Down Algorithm to Find Frequent Orders

We divide the model into the following three phases. In Step 1 numerical dataset is converted into sequence set. This processing takes only once in the first run. In Step 2 and 3 the frequent orders is constructed iteratively. These two steps would be repeated when necessary.

Step 1. Sorting Phase. Each condition is identified with a label. Then each gene expression vector is sorted in non-decreasing order. According to the sorting result each gene expression vector is converted into an ordered label sequence. The order of each label in a sequence stands for the ranking of the corresponding condition in the particular gene expression vector. Now the whole dataset is converted to a sequences set.

The conditions with very close values are grouped to form an order equivalent group in which we make no no difference on their relative order. That means

ID	a	b	c	d	e	f
1	4	5	5	3	8	1
2	3	4	5	2	7	8
3	5	3	7	4	9	1
4	8	1	5	4	3	7

(a)

ID	Sequence
1	f d a (c b) e
2	d a b c e f
3	f b d a c e
4	b e d c f a

(b)

Fig. 2. (a) Original numerical data matrix, (b) sequences of column labels of the data matrix

conditions in order equivalent group could have exchangeable orders. The strict order requirement is too sharp in OP-Clustering scheme and orders between close values would be meaningless. Since order between some conditions might be disturbed by sampling processing especially when noises exist. We define a threshold θ. When difference between two conditions is smaller than θ, we group them together to form an order equivalent group. θ is relative to the magnitude of dataset. Figure 2 shows an example the sorting phase. "()" means order equivalent group. Before going on we give some notations.

Definition 1. For any two labels, no matter they are adjacent or not in a sequence with no repeating labels, the one that comes before is called the prefix and the other one is called the suffix. E.g. in sequence "fdecab", "e" is prefix of "c", "a" and "b". Also it is suffix of "f" and "d".

Definition 2. For two label sequences x and y, if all labels of x are contained in y and any two labels of x have the same order as they have in y then we say x is a subsequence of y or x appears in y. E.g. "dca" is a subsequence of "fdecab".

Definition 3. Sequence x is different from sequence y when any of the following two cases occurs, 1. There are labels in x which do not appear in y, 2. The common labels of the two sequences have different orders. E.g. sequence "adcdf" and "acbed" are different from sequence "abcde", while "abe" is not.

An embedded OP-Cluster can be identified by first finding the order of its columns, which is the frequent order appearing in the sequence set. The idea of frequent order is similar to the frequent pattern in the sequential pattern mining problem [16][11][12]. However, in our work it is more complicated than conventional sequential pattern mining problem. The original sequences we are handling have the same length. Different labels appear once and only once in each sequence. The identities of the genes associated with each frequent order have to be recorded in order to determine the genes involved in an OP-Cluster. While conventional sequential mining algorithms only keep the number of appearance of frequent subsequences but not their identities.

Step 2. Counting Phase. We scan the label sequences and construct an order matrix O which counts the occurrence of any label being prefix or suffix of any other labels. In the example there are 6 labels. Then order matrix O is a 6×6 matrix as Figure 3 shows. Each row/column stands for one label. $O(a, b)$ stands for the occurrence frequency of "a" being prefix of "b" (or "b" being suffix of "a") in all the rows. Suppose the original numerical data matrix has n rows an m columns then O is a $m \times m$ matrix and $O(a, b) + O(b, a) = n + z$ (z is the number of order equivalent groups which contain "a" and "b").

O	a	b	c	d	e	f	
a	0	2	3	0	3	1	9
b	2	0	4	2	4	2	14
c	1	1	0	0	3	2	7
d	4	2	4	0	3	2	15
e	1	0	1	1	0	2	5
f	3	2	2	2	2	0	11
	11	7	14	5	15	9	

Fig. 3. Finding the frequent prefix and suffix combination of [d e] from order occurrence matrix by picking out the row (column) with the maximum accumulated occurrence frequencies

Order matrix shows occurrence frequencies of any length-2 subsequences. Suppose original dataset is randomly constructed, orders between any two labels would be evenly distributed in the rows. That means the possibility of occurrence of "a" before "b" would be the same as that of "b" before "a". However if an OP-Cluster exists in data matrix some labels would have much higher frequencies of being prefix/suffix then the other labels in the global way. This statistical information is useful to find the frequent orders shared by a significant number of gene vectors.

Step 3. Finding the most frequent prefix and suffix. We accumulate the total number of occurrences of each label being prefix (suffix) of another labels in all rows. It is the summation of the corresponding row (column) of order matrix O. The most frequent prefix (suffix) is the label with the maximum number of accumulated occurrences of being prefix (suffix). In the example of Figure 3 "d" and "e" is the most frequent prefix and suffix respectively. We combine the most frequent prefix and suffix to form an initial frequent order candidate which has higher occurrence frequency in the sequences set. So [d e] is the seed for growing the frequent order in this example.

Only the supporting rows for this order is kept. Other rows which do not support the order is removed. And we extract subsequence between the current prefix and suffix from the remaining rows. Figure 4(a) shows the 3 subsequences

ID	Subsequences					
1		d	a	(c	b)	e
2	d	a	b	c	e	
3			d	a	c	e
4						

(a)

O'	a	b	c	d	e	f	
a	0	2	3	0	0	0	5
b	0	0	2	0	0	0	2
c	0	1	0	0	0	0	1
d	0	0	0	0	0	0	0
e	0	0	0	0	0	0	0
f	0	0	0	0	0	0	0
	0	3	5	0	0	0	

(b)

Fig. 4. (a) Set of the extracted subsequences, (b) updated order matrix

"acb", "abc" and "ac". Then we go back to step 2 and update order matrix O based on the remain subsequences as shown in Figure 4(b). In the second run [a c] is found as the new prefix and suffix combination. Since order [d e] is supported by all remaining rows and "d" is the first prefix ("e" is the last suffix) of all labels in the subsequences, [a c] is put between the former prefix and suffix to form an enlarged frequent order of [d a c e]. Step 2 and 3 could be repeated when the minimum support (will be discussed later) is satisfied. By iteratively finding the most frequent prefix (suffix) and combining them we enlarge frequent order candidate step by step. Due to the noise in data we could miss some prefix or suffix of target frequent order during iterations. Enhancement can be made by repeating the whole procedures on the remaining rows and all columns.

3.2 Minimum Support for the Frequent Order

Some researchers choose an exact value as the minimum support While we use $p(t)$ as the minimum support criterion,

$$p(t) = \frac{1 - p_{ini}}{ln(t)\_max} ln(t) + p_{ini} \tag{1}$$

where t stands for the current iteration. In the first iteration $p(1) = p_{ini}$ and $p(t) \leq 1$. $p(t)$ increases with the iteration. t_max stands for the maximum value of t. If original dataset has m columns, the maximum value of the iteration would be $m/2$.

We compute the portion of number of supports for enlarged frequent order to number of current remaining rows. Iteration goes on only when the value is bigger than $p(t)$. Since we delete the non-supporting rows after each iteration, number of remaining rows keeps decreasing. If non-frequent sub-prefix or sub-suffix is added to the frequent order candidate number of remaining supports would decrease by a large number. In this way we reject non-frequent orders. To loosen the minimum support requirement one can use $p(t)^2$ instead.

ID	Sequences					
1	f	d	a	c	b	e
2	d	a	b	e	c	f
3	c	b	d	a	f	e
4	b	e	d	c	f	a

(a)

O'	a	b	c	d	e	f	
a	0	2	1	0	0	1	4
b	0	0	0	0	0	0	0
c	0	1	0	0	0	0	1
d	0	0	0	0	0	0	0
e	0	0	0	0	0	0	0
f	0	0	0	0	0	0	0
	0	3	1	0	0	1	

(b)

Fig. 5. (a) Updated sequence set (the gray elements construct an OP-Cluster), (b) updated occurrence matrix

3.3 Find Proper Single Label When No Frequent Combination of Prefix and Suffix Exists

When target frequent order has an odd number of labels, it can not be discovered by including both prefixes and suffixes to the frequent order iteratively. The hidden OP-Cluster in Figure 6(a)has frequent order of [d a e]. [d e] would be found in the first interation and [a b] in the second iteration. While for order [d a b e] there are only two supports. That could lead to a non-frequent order. So when no significant prefix and suffix combination could be found, we count the occurrence frequencies of all labels in remaining subsequences. Column with the maximum number of appearance would be checked to see whether it could be added into the candidate order. So "a" is added to form the frequent order [d a e] and result in a 3×3 OP-Cluster when satisfying the minimum support.

3.4 Find Multiple OP-Clusters

We repeat the whole algorithm and whenever we find a new frequent order candidate we compare it to all existing frequent orders. If the candidate is not different from the existing frequent orders (see Definition 3), we ignore it and update the occurrence matrix O then repeat until new candidate which is different from the existing ones is found. Theoretically as iteration goes on, all label combinations could be tested. But frequent orders would be processed first and non-frequent sequences would be rejected at very early stage without much processing. This scheme reduces a large proportion of computation cost. There is rarely multiply operation in our algorithm. The major cost is on computing matrix O by counting the accumulated occurrence. Cost for the whole algorithm is hard to estimate since number of iteration varies a lot for different datasets. In step 1 the sorting requires time in $O(nm)$ and only take place for once. Calculation of matrix O in the first run is an $O(m^2n)$ effort. This cost drops dramatically in following iterations since number of supporting rows and length of subsequences

decrease a lot. Space complexity of the whole algorithm is $O(nm)$ which is very small comparing to most of the existing methods.

4 Experiments

The algorithm is tested with both synthetically generated datasets and real gene expression datasets for effectiveness and efficiency. The experiments are implemented with MATLAB and executed on a computer with a 3.2 GHz and 0.99 GB main memory.

Size of OP-Cluster, which means number of columns and rows involved, is the measurement for its significance. Also length of potential frequent order and number of its supports are two critical factors that affect the performance of our algorithms. However, the best OP-Cluster is hard to define since shorter orders are supported by much more genes. For n rows with length m, the probability of finding at least k supports for any order with length m is,

$$P(n, m, k) = \sum_{i=k}^{n} (\frac{1}{m!})(\frac{m! - 1}{m!})^{n-i} C_n^i \tag{2}$$

P decreases with the increasing of m and k. P could be used to measure the significance of an OP-Cluster. The smaller P is, the more significant the OP-Cluster is.

4.1 Synthetic Data

We generate the synthetic data in this way: Firstly, random datasets were generated. Then OP-Clusters were inserted manually into the datasets. In the following experiment each case has been implemented for 10 times. We get the average value of all runs.

Sometimes instead of finding the exact manually inserted OP-Cluster we find OP-Clusters which overlap with the inserted one. These OP-Clusters could be formed by chance when other conditions were included in the frequent order. Suppose a $k \times l$ OP-Cluster was inserted and an OP-Cluster, which has p rows and q columns in common with the inserted one, is found. The found OP-Cluster could have more conditions than the inserted one. In that case it may have fewer supports than the inserted one. We define the accuracy to be,

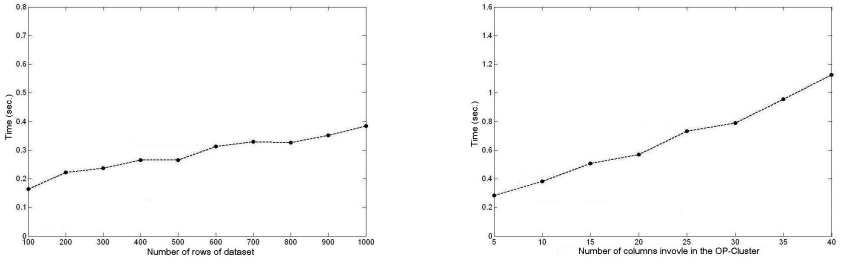
$$accuracy = (\frac{p}{k} + \frac{q}{l})/2 \tag{3}$$

When the exact OP-Cluster is found, the accuracy is 1. In other cases the accuracy is proportional to the volume of the overlapping part between the embedded OP-Cluster and the found one.

Test For Effectiveness. Number of supporting rows and length of frequent order relative to the size of dataset are the two important factors which effect the

Table 1. Result on varying the size of the dataset and the inserted OP-Cluster

	100×30	200×30	500×30
20×15	100%	100%	96.5%
40×10	100%	100%	100%
200×15	–	100%	100%



(a) (b)

Fig. 6. Scale-up experiments. Response time V.S. (a) number of rows of dataset, (b)number of columns involved in OP-Cluster.

performance. To test the effectiveness we run the algorithm with combinations of different sizes of datasets and OP-Clusters. For the OP-Clusters we also change the ratio of rows to columns.

Table 1 shows the result. Rows of the table show the size of inserted cluster and columns show the size of the original dataset. Average accuracy of each case of 10 runs was shown. Our method works very well in nearly all cases, especially when size of inserted OP-Cluster is significant to size of original dataset. 20×15 OP-Cluster was exactly found for 9 out of the 10 runs when it is inserted into a 500×30 data matrix. And in another case our algorithm finds a 6×16 OP-Cluster, which covers all the columns involved in the inserted OP-Cluster.

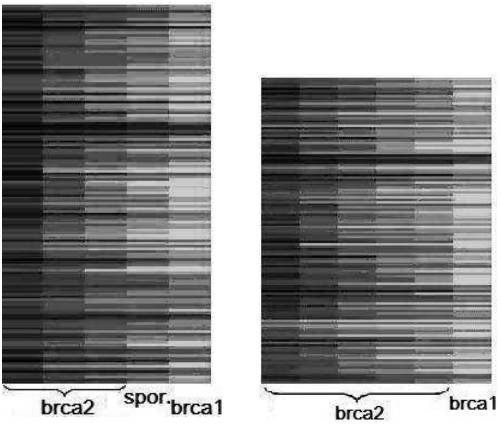
Test For Scalability. The scale-up properties of the algorithm were analyzed by varying size of dataset and embedded cluster. We report the time cost for the first run. Figure 6(a) shows the result when varying the number of rows of dataset. A 50×20 OP-Cluster was inserted into datasets which have 50 columns but increasing number of rows from 100 to 1000. Figure 6(b) shows the result when increasing the number of columns involve in the OP-Cluster. An OP-Cluster with 200 supports was inserted into 1000×50 datasets. The number of conditions involve was increased from 5 to 40. Our algorithms got high precision almost in all cases and scale linearly with size of dataset and size of OP-Cluster.

4.2 Microarray Data

In addition to simulated datasets, we run our algorithm on the breast tumor dataset from Chen et al. [17]. This dataset has 3,226 genes and 22 tissues. Among

Table 2. Comparison of result significance with OPSM algorithm and OPC-Tree algorithm. (N/A: not available)

Number of tissues	Number of Max supporting rows		
	CFPS	OPSM	OPC-Tree
4	771	347	690
5	142	N/A	N/A
6	124	42	126
7	32	8	47



(a) (b)

Fig. 7. (a) The largest 5-tissue OP-Cluster, (b) the largest 6-tissue OP-Cluster

the 22 tissues, there are 7 brca1 mutations, 8 brca2 mutations and 7 sporadic breast tumors. Our experiments demonstrate the power of finding biological OP-Clusters in the gene expression data. We compared our result with that from the OPSM algorithms of Ben-Dor et al. [9] and OPC-Tree of Liu et al. [13].

Firstly, we report the significance of our clusters in table 2 with comparisons with other two algorithms. Our clustering algorithm generates much more significant clusters than OPSM. It costs only 7.17 seconds for the finding the first 20 OP-Clusters. These OP-Clusters cover all the 22 tissues and 74.3% of the genes. Ben-Dor et al. reported only three clusters with size 4×347, 6×42 and 8×7 respectively. However, our OP-Clustering algorithm was able to find 4 tissues clusters supported by a maximum of 771 genes, which doubles the number of genes of the result of OPSM and also outperformed OPC-Tree. Our algorithm also found 5-tissue clusters with a maximum support of 142 genes. $P(5, 142, 3226) \approx 1.1659e-55$ that means the cluster has very high significance. 5-tissue clusters were not reported by OPSM and OPC-Tree. The order imposed on these five tissues are interesting, 3 brca2 mutations show lower expression, 1 brca2 mutation in the middle and 1 sporadic breast tumor shows the highest

expression. This result suggest these 142 genes has different expression levels in these tissues. Figure 7 shows the largest 5-tissue and 6-tissue OP-Clusters and type of the tissues involved. No OP-Clusters with more than 7 tissues were reported. Since our algorithm is good at finding the statistical majority. Larger clusters would be found first. And we only process the first 20 clusters, small cluster would lose the chance of being found.

5 Discussion

Order preserving clustering has been used in many applications to capture the consistent tendency exhibited by a subset of objects in a subset of dimensions in high dimensional space. We proposed a heuristic approach CFPS which discovers the OP-Clusters by finding the frequent orders using a top-down scheme in a statistical way. The method is easy to use with a low computation and space cost. Few parameter has to be initialized for the algorithm. We define the significance of an OP-Cluster and by it we can discriminate the meaningless OP-Clusters constructed by chance. The algorithm works very well in the experiment. It scale linearly to the size of the dataset and the size of the clusters. For the real gene expression profiles our method outperform OPSM and OPC-Tree in finding significant clusters. But the nature of NP-hardness of this problem implies that there may be sizeable OP-Clusters evading the search by any efficient algorithm.

There are several extensions we can make based on our algorithm. Although we permit exchangeable orders for conditions with very close values, the requirement on exactly the same order is still sharp in some applications, especially when noise or outlier exists. One extension of the current model is to explore similar but not exact the same order among conditions. There are many overlapping OP-Clusters in our result. Further steps could be taken to merge some of the overlapping clusters to form new clusters with more columns.

References

1. Cheng, Y., Churhc, G.: Biclustering of expression data. In: ISMB'00, pp. 93–103. ACM Press, New York (2000)
2. Tanay, A., Sharan, R., Shamir, R.: Discovering statistically significant biclusters in gene expression data. IEEE Transactions on Knowledge and Data Engineering 18, 136–144 (2002)
3. Wang, H., Wang, W., Yang, J., Yu, P.: Clustering by pattern similarity in large data sets. In: ACM SIGMOD Conference on Management of Data'02, pp. 394–405 (2002)
4. Yang, J., Wang, W., Wang, H., Yu, P.: δ-clustering: Capturing subspace correlation in a large data set. In: 18^{th} IEEE Int'l. Conf. Data Eng., pp. 517–528 (2002)
5. Bleuler, S., Prelic, A., Zitzler, E.: An ea framework for biclustering of gene expression data. In: Congress on Evolutionary Computation'04, pp. 166–173 (2004)
6. Cho, H., Dhillon, I.S., Guan, Y., Sra, S.: Minimum sum-squared residue cococlustering of gene expression data. In: Fourth SIAM Int'l. Conf. Data Mining (2004)

7. Teng, L., Chan, L.: Biclustering gene expression profiles by alternately sorting with weighted correlation coefficient. In: IEEE International Workshop on Machine Learning for Signal Processing'06 (2006)
8. Madeira, S., Oliveira, A.: Biclustering algorithms for biological data analysis: A survey. IEEE/ACM Transactions on Computational Biology and Bioinformatics 1, 24–45 (2004)
9. Ben-Dor, A., Chor, B., Karp, R., Yakhini, Z.: Discovering local structure in gene expression data: The order-preserving submatrix problem. In: RECOMB'02. ACM Press, New York (2002)
10. Agrawal, R., Srikant, R.: Mining sequential patterns. In: 11^{th} International Conference on Data Engineering, pp. 3–14 (1995)
11. Han, J., Pei, J., Yin, J.: Mining frequent frequent patterns without candidate generation. In: ISMB'00 ACM SIGMOD Conference on Management of Data'02, pp. 1–12 (2000)
12. Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., Hsu, M.: Mining sequential patterns by pattern-growth: The prefixspan approach. IEEE Transactions on Knowledge and Data Engineering 16, 1424–1440 (2004)
13. Liu, J., Yang, J., Wang, W.: Biclustering in gene expression data by tendency. In: IEEE Computational Systems Bioinformatics Conference, pp. 182–193. IEEE Computer Society Press, Los Alamitos (2004)
14. Hipp, J., Guntzer, U., Nakhaeizadeh, G.: Algorithms for association rule mining-a general survey and comparison. SIGKDD Explorations 2, 58–64 (2000)
15. Bleuler, S., Zitzler, E.: Order preserving clustering over multiple time course experiments. In: EvoBIO'05, pp. 33–43 (2005)
16. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: 20^{th} Int'l. Conf. Very Large Data Bases, pp. 487–499 (1994)
17. Chen Y., Radmacher, M., Bittner, M., Simon, R., Meltzer, P.: Gene expression profiles in hereditary breast cancer. NEJM 344, 539–548 (2001).