

# Tendency based Subspace Clustering on Gene Expression Data

Jinze Liu and Wei Wang  
Computer Science Department  
University of North Carolina  
Chapel Hill, NC 27599  
U.S.A.  
{liuj, weiwang}@cs.unc.edu

## Abstract

Microarrays are one of the latest breakthroughs in experimental molecular biology. By monitoring expressions of different genes under different experiments, a large matrix representing the gene expression levels of varying experiments will be produced. To reveal patterns in such matrices, Ben-Dor et al. introduced a probabilistic model to discover the strictly order-preserving submatrix (OPSM) embedded in the gene expression matrix. The proposed algorithm to discover one hidden OPSM with designated column size  $s$ , starts from building the smallest partial model with the best qualities, and then iteratively grow the partial model(s) in a number of best directions by including extra elements. This terminates when the column size of the partial model is no smaller than  $s$ . Due to the probabilistic nature of this model, it suffers from several drawbacks. The OPSM algorithms favor large row support based on the intuitions that the OPSMs with large row support will be more significant and potentially, can be developed into OPSMs with more columns. However, when there are submatrices with both large and small row supports, the larger submatrices might prevent the smaller ones from discovering since the smaller ones could be eliminated in the early stage of the development. In addition, the probabilistic model and the given algorithm can only determine row support for a certain order of conditions for a specific OPSM. To tackle those problems, we propose a more general model (u-Cluster) and an efficient deterministic algorithm to discover all the submatrices exhibiting tendencies in one run. Experimental study on a yeast gene expression dataset and a drug activity dataset demonstrate that our algorithm is much more robust, more effective and more efficient than the OPSM algorithm.

**Keywords:** Gene expression Analysis, Clustering, Tendency

## 1 Introduction

Modern technology provides efficient methods for data collection. The advent of DNA microarray technologies has revolutionized the experimental study of gene expression. Thousands of genes are routinely probed in a parallel fashion. And the expression levels of their transcribed mRNA are reported. By repeating such experiments under different conditions (e.g different patients, different tissues or varying cells' environment), data from tens to hundreds of experiments can be gathered. The analysis of the resulting large datasets poses numerous algorithmic challenges.

So far, clustering is still the main approach to analyze gene expression data. A lot of efforts have been devoted to devise new models [4, 21], to extend existing clustering approaches [7, 3, 15], and to apply and evaluate clustering techniques to gene expression data [22, 16, 14, 24]. However, clustering in high dimensional spaces is often problematic as theoretical results questioned the meaning of closest matching in high dimensional spaces. In gene expression analysis, it is often more interesting to identify patterns that are common to only part of expression data matrix than to only find global clusters [4]. Based on general understanding of cellular processes, we expect subsets of genes to be co-regulated and co-expressed under subsets of experimental conditions, but to behave independently under other conditions. Discovering of patterns embedded in submatrices in gene expression data can be significant to reveal multiple genetic pathways.

To uncover patterns in this category, Cheng and Church [6] introduced the concept of *bicluster* to capture the coherence of a subset of genes and a subset of conditions. Let  $X$  be the set of genes and  $Y$  the set of conditions. Let  $I \subset X$  and  $J \subset Y$  be subsets of genes and conditions. The pair  $(I, J)$  specifies a sub matrix  $A_{IJ}$  with the following *mean squared residue score*:

$$H(I, J) = \frac{1}{|I||J|} \sum_{i \in I, j \in J} (d_{ij} - d_{iJ} - d_{Ij} + d_{IJ})^2 \quad (1)$$

where

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

are the row and column means and the means in the submatrix  $A_{IJ}$ . A submatrix  $A_{IJ}$  is called a  $\delta$ -bicluster if  $H(I, J) \leq \delta$  for some  $\delta > 0$ . A random algorithm is designed to find such clusters in gene expression matrix. An enhanced bicluster algorithm was proposed by Wang et al. [20], which addressed interference issue caused by random data replacement in the original algorithm and proposed an algorithm FLOC to accelerate the performance. The mean squared residue used in [6, 20] is an average measurement of the coherence for a set of objects. Even with a small mean squared residue, the outliers might still present in the clusters. On the other hand, when there is a large row variance along a set of objects, coherence fluctuation might still present in the set of objects although it might result in a large mean square residue. To attack those problem, the  $\delta$ -pCluster model [18] was introduced to discover clusters by pattern similarity including strict shifting and scaling patterns from raw data sets.

Ben-Dor et al. introduced a model, namely OPSM(order preserving submatrix) [4], to discover a subset of genes identically ordered among a subset of the conditions. Unlike the bicluster and pCluster model, it focused on the coherence of the relative order of the conditions rather than the coherence of the actual expression levels. These types of patterns can be expected when considering data from nominally identical exposure to environmental effects, such as drug treatment, temporal progression, etc. For example, in expression data that comes from a population of patients, it is reasonable to expect that each individual is in a particular stage of the disease. There is a set of genes that are co-expressed with this progression and we therefore expect the data to contain a set of genes and a set of patients such that the genes are identically ordered on this set of patients. The OPSM problem is proved to be NP-hard. The algorithm designed in the paper tries to grow partial model iteratively. The partial model is scored by measuring the expected number of planted(valid) rows associated with it. The larger the row support, the better the score. Given any  $n \times m$  matrix and the number of columns  $s$  included in the resulting OPSM, the algorithm starts from building the partial model with  $s = 2$ , then chooses  $l$  of them with the best scores. For each of these  $l$  partial models, the algorithm explores all  $m - 2$  extensions and chooses the best  $l$  of them. This iteration continues until the partial models have size  $s$ . Among them, the model with the best score will be selected as the OPSM.

However, due to the probabilistic model and the selection of partial models based on scores, there are some inevitable drawbacks in this pioneering work. First, the resulting model can be easily affected by the given parameter  $l$ . The larger the value of  $l$ , the larger the candidate pool for selecting the best partial model at each iteration, and therefore the higher the chance that the best OPSMs will not be eliminated because of possible low score in the partial model. However, this is at the expense of much longer execution time. Secondly, the resulting OPSM can also be affected by the number of unplanted rows in the original matrix. For example, the submatrix  $M_1$  generated from the matrix  $M$  might be different from the submatrix  $M_2$  generated from a submatrix of  $M$  containing  $M_1$ , even though they are expected to be the same theoretically. Thirdly, the OPSM algorithm favors large row support based on the intuitions that the OPSMs with large row support will be more significant and potentially, can be developed into OPSMs with more columns. However, when there are submatrices with both large and small row supports, the larger submatrices might prevent the smaller ones from being discovered since the smaller ones could be eliminated in the early stage of the development. Therefore, this approach might not be suitable for the pattern-rich data sets where there are patterns across the subsets of columns with different sizes and overlapping with each other.

To tackle those problems in OPSM model, we propose a deterministic subspace clustering model, namely u-Cluster, to capture all the general tendencies exhibited by a subset of objects along a subset of dimensions in one run. A u-Cluster is more general than an OPSM by allowing mixed order among a subset of the conditions with similar expression levels.

The remainder of the paper is organized as follows. Section 2 defines the model proposed in the paper. Section 3 presents the two algorithms in detail. An extensive performance study is reported in Section 4. Section 5 concludes the paper and discuss some future work.

## 2 Model

In this section, we extend the OPSM by requiring that, the order among a subset of conditions which have similar expression levels is treated the same. We introduce the new model u-Cluster for this purpose.

### 2.1 Notations

$\mathcal{D}$	A set of objects
$\mathcal{A}$	Attributes of objects in $\mathcal{D}$
$(\mathcal{O}, \mathcal{T})$	A sub-matrix of the data set, where $\mathcal{O} \subseteq \mathcal{D}, \mathcal{T} \subseteq \mathcal{A}$
$x, y, \dots$	Objects in $\mathcal{D}$
$a, b, \dots$	Attributes in $\mathcal{A}$
$d_{xa}$	Value of object $x$ on attribute $a$
$\delta$	User-specified similarity group threshold
$\delta^p$	User-specified shifting threshold
$nc, nr$	User-specified minimum # of column and minimum # of rows of a model

### 2.2 Definitions and Problem Statement

Let  $\mathcal{D}$  be a set of objects, where each object is associated with a set of attributes  $\mathcal{A}$ . We are interested in subsets of objects that exhibit a coherent tendency on a subset of attributes of  $\mathcal{A}$ .

**Definition 2.1** Let  $o$  be an object in the database,  $\langle d_{o1}, d_{o2}, \dots, d_{on} \rangle$  be the attribute values in a non-decreasing order,  $n$  be the number of attributes and  $\delta$  be the user specified threshold. We say that  $o$  is **similar** on attributes  $i, i + 1, \dots, i + j$ , ( $n \geq i > 0, n \geq j > 0$ ), if

$$(d_{o(i+j)} - d_{oi}) < \mathcal{G}(\delta, d_{oi}) \quad (2)$$

we call the set of attributes  $\langle i, i + 1, \dots, i + j \rangle$  a **similar group**. Attribute  $d_{oi}$  is called a **pivot point**.

The intuition behind this definition is that, if the difference between the values of two attributes is not significant, we regard them to be “equivalent” in value and group them together. There are multiple ways to define the grouping function  $\mathcal{G}(\delta, d_{oi})$ . One way is to define it as

$$\mathcal{G}(\delta, d_{oi}) = \delta \times d_{oi}. \quad (3)$$

For example, suppose we have five expression levels for different samples ( $A, B, C, D, E$ ) from one gene, which is (1, 4, 4.5, 8, 10). If  $\delta = 0.2$ , 4 and 4.5 are considered equivalent to each other. The samples are divided into four groups  $\{\{1\}, \{4, 4.5\}, \{8\}, \{10\}\}$ .

**Definition 2.2** Let  $o$  be an object in the database, and  $(g_{o1}) (g_{o2}) \dots (g_{ok})$  be a sequence of similar groups of  $o$  by Equation 2 and in non-descending order of their values.  $o$  shows an “UP” pattern on an ordered list of attributes  $a_1, a_2, \dots, a_j$  if  $a_1, a_2, \dots, a_j$  is a subsequence of  $(g_{o1})(g_{o2}) \dots (g_{ok})$

After we apply the group similarity in the previous, we are able to transform the original grouping to the sequence  $A(BC)DE$ .  $ABDE$ ,  $AE$ , and  $(BC)E$  show “UP” patterns.

**Definition 2.3** let  $\mathcal{O}$  be a subset of objects in the database,  $\mathcal{O} \subseteq \mathcal{D}$ . Let  $\mathcal{T}$  be a subset of attributes  $\mathcal{A}$ .  $(\mathcal{O}, \mathcal{T})$  forms a u-Cluster if there exists a permutation of attributes in  $\mathcal{T}$ , on which every object in  $\mathcal{O}$  shows the “UP” pattern.

Suppose we have two gene expression levels  $g_1$  and  $g_2$  for samples ( $A, B, C, D, E$ ). The expression levels are (1, 4, 4.5, 8, 10) and (2, 5, 7, 4.5, 9), respectively. According to Definition 2.3, the corresponding sequence of groups for  $g_1$  is  $A(BC)DE$ , and for  $g_2$  is  $A(DB)CE$ . Since  $ABCE$  is a common subsequence of them, we say that  $g_1$  and  $g_2$  form a u-Cluster on the attributes sets of  $ABCE$ . In the following sections, since the input data is a matrix, we refer to objects as rows and attributes as columns.

**Problem Statement** Given a cluster threshold  $\delta$ , a minimal number of columns  $nc$ , and a minimal number of rows  $nr$ , the goal is to find all (maximum) submatrices  $(\mathcal{O}, \mathcal{T})$  such that  $(\mathcal{O}, \mathcal{T})$  is a u-Cluster according to Definition 2.3, and  $|\mathcal{O}| \geq nr$ ,  $|\mathcal{T}| \geq nc$ .

With the introduction of group similarity, our u-Cluster model is definitely a generalization of both the OPSM model and the  $\delta$ -pCluster model. Interested readers please refer to [11] for a formal proof. Our model is capable of uncovering a large number of significant patterns. Especially when applied to the microarray data, which is usually not neat and error prone, our model is more robust than other existing models.

### 3 Algorithm

In this section, we present our algorithm to generate u-Clusters, which consists of three steps: (1) preprocess the data into sequences of similarity groups; (2) find sequential patterns; and (3) generate a u-Cluster for each sequential pattern. We propose a novel compact structure UPC-Tree to organize the sequences and to guide the pattern generation.

#### 3.1 Preprocessing

The first step in our algorithm is preprocessing. First, for each row, we sort all the entry values in ascending order. Secondly, we organize each sorted row into a sequence of similarity groups. The resulted sequences for all rows will be taken as the input of the second step — sequential pattern generation. Let’s look at the raw data in Table 1.

	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>
1	4392	284	4108	228
2	401	281	120	298
3	401	292	109	238
4	280	318	37	215
5	2857	285	2576	226
6	48	290	224	228

Table 1: Raw Data Set

If we set the threshold  $\delta$  for group similarity to be 0.1, the sequences are shown in Table 2. Attributes in each “()” are in the same similarity group. Since order does not matter for these attributes, we use the alphabetical order. For example, for the object 1, the sorted order of attributes is [228 : *D*, 284 : *B*, 4108 : *C*, 4392 : *A*]. *A* and *C* can be grouped together since  $4392 - 4108 < 4108 \times 0.1$  (Equation 3).

	sequence
1	<i>DB(AC)</i>
2	<i>C(BD)A</i>
3	<i>CDBA</i>
4	<i>CDAB</i>
5	<i>DBCA</i>
6	<i>A(CD)B</i>

Table 2: Sequences after Preprocessing

In this way, we turn a clustering problem into sequential pattern mining problem. Our task in the next step is to find all the frequent subsequences embedded in these sequences.

### 3.2 UPC-Tree

Before we define UPC-Tree formally, we first give the following example.

**Example 3.1** For the sequences in Table 2, with  $nc = 3, nr = 3$ , the UPC-Tree algorithm works in the following steps.

**Step 1:** Create root  $-1$  and insert all the sequences into the tree. This is showed in Figure 1 (1). Notice that same prefix falls on same branch of the tree. The sequence ID is stored in the leaves. The current root is  $-1$  and the current depth is 0.

**Step 2:** For each child of the root, insert suffixes in its subtree to the root's child that has a matching label. In Figure 1 (2),  $C$  is a child of the root  $-1$ . In this subtree, the suffix subtree starting at  $D$  (for sequence 3, 4) is inserted into the root  $-1$ 's child  $D$ . Each insertion is illustrated by a dotted line connecting the two involved nodes, with the arrow pointing to the destination node in Figure 1. The sequence IDs associated with the suffixes are combined with existing IDs in the destination node. In the case where a suffix is too short to satisfy  $current\ depth + length\ of\ the\ suffix > nc$ , the suffix will not be inserted. For example,  $BA$  in sequence 3 is also a suffix, it is not to be inserted because the  $depth\ 0 + length\ of\ BA < nc$ .

**Step3:** Prune current root's children. If the number of rows fall in a subtree is smaller than  $nr$ , the subtree will be deleted because no further development can generate a cluster with more than  $nr$  rows. For example, subtree leading from  $-1B$  in Figure 1 (2) is deleted in Figure 1 (3) since there are only two sequences falling in this subtree.

**Step4:** Repeat Step2-Step5 for the root's first child recursively until there is no child node left. For example,  $C$  is the first child of root  $-1$ . Therefore, the same procedure in Step2 is applied to  $C$  first. The suffixes of  $C$ 's subtree  $D$ , such as  $BA$  and  $AB$  are inserted into  $C$ 's subtree  $B$  and  $A$  respectively. Since there was less than three sequences fall on  $C$ 's subtrees  $A$  and  $B$ , the branches  $-1CA-$  and  $-1CB-$  are deleted. Following the same procedure, we develop  $C$ 's only subtree  $-1CD-$ , which is shown in Figure 1(4).

**Step5:** Repeat Step2-Step5 for the root's next siblings recursively. For example, after finishing  $-1C-$ 's subtree development, the next subtree to develop is  $-1C-$ 's sibling  $-1D-$ .  $-1DB$ 's suffix  $AC$  is inserted to subtree  $-1DA$ . However, both subtrees are deleted because they do not have enough support count.

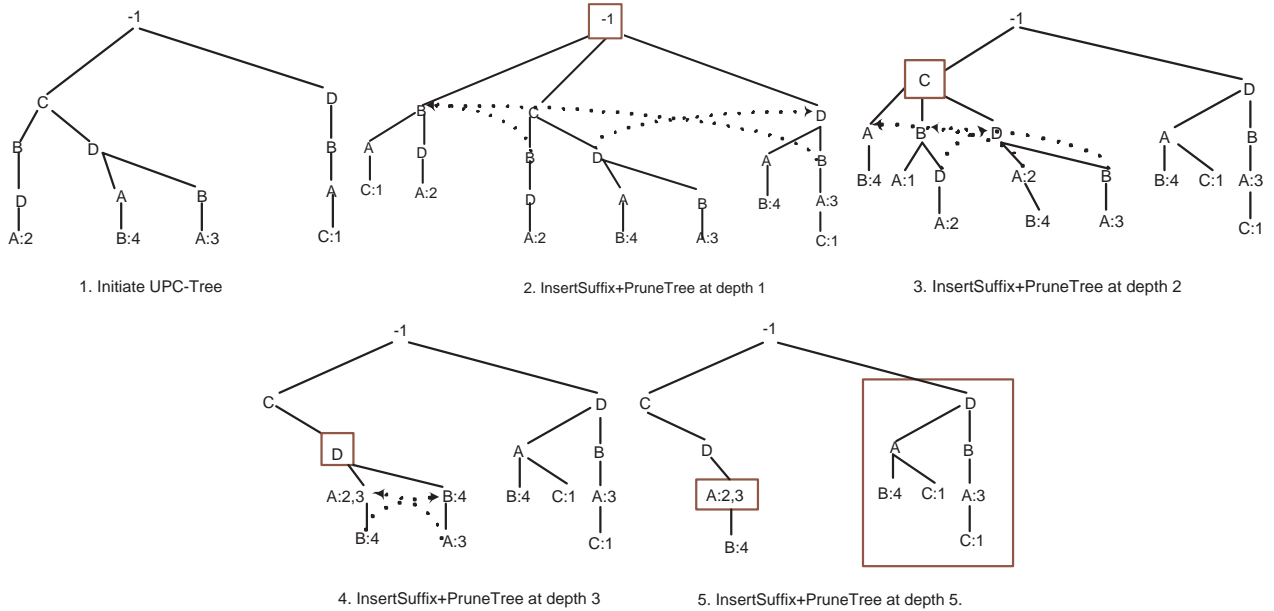


Figure 1: UPC-Tree for Table 2

**Definition 3.1** UPC-tree (Up Pattern Clustering tree). An UPC-Tree is a tree structure defined below.

1. It consists of one root labeled as  $-1$ , a set of subtrees as the children of the root;
2. Each node in the subtrees consists of four entries: entry value, a link to its first children node, a link to its nearest

sibling node, and a link list of all the rows that share the same path leading from root to this node but do not have longer subsequences passing this node. In another word, the sequence IDs are only recorded at the nodes which marked the end of a subsequence.

**Algorithm** *UPC-Tree*( $S, nr, nc$ )

**Input:**  $S$ : The sequence set from preprocessing of original Matrix,  $nr$ : minimal number of rows,  $nc$ : minimal number of columns

**Output:** All the subsequence with frequency count  $\geq nr$  and length  $\geq nc$

(\* Main program to develop the hello \*)

1. Create the root of an UPC-Tree,  $T$ , and label it as "1".
2. **for** each sequence  $s$  in  $S$
3.     **do** insertSequence( $s, T$ )
4.     growTree( $T$ ) recursively;
5. **return**.

**Algorithm** *insertSequence*( $s, T$ )

**Input:**  $s[i..n]$ : the sequence to be inserted,  $T$ : the root of UPC-Tree

**Output:**  $T$ : tree with the path corresponding to  $s$

(\* Insert a sequence into the root of a tree \*)

1. **if**  $i = n$ ,
2.     **then** insert the  $ID$  of  $s$  into  $N$ 's IDlist.
3.     **return**
4.     **else if**  $T$  has a child  $N$  such that  $N.value = s[i].value$ ,
5.         **then** insertSequence( $s[i + 1..n]$ );  $N$ );
6.         **else** create a new node  $N$ .
7.             **if**  $T$ 's first child  $\neq nil$
8.                 **then** the last sibling's next sibling  $\leftarrow N$ .
9.                 **else**  $T$ 's first child  $\leftarrow N$ .
10.             insertSequence( $s[i + 1..n]$ );  $N$ )
11. **return**

**Algorithm** *growTree*( $T, nc, nr, depth$ )

**Input:**  $T$ : the root of the initiated tree,  $nc$  and  $nr$

**Output:** u-Cluster existed in  $T$

(\* Grow patterns based on original  $T$  \*)

1. **if**  $T = nil$
2.     **return**;
3.      $T_{child} \leftarrow T$ 's first child;
4.     **for** any sub-tree  $subT$  of  $T$
5.         **do** insertSubTree( $subT, T$ );
6.     pruneTreeNode( $T$ );
7.     growTree( $T_{child}, nc, nr, depth + 1$ );
8.     growTree( $T$ 's next sibling,  $nc, nr, depth$ );
9. **return**.

**Analysis of UPC-Tree construction** Only one scan of the entire data matrix is needed during the construction of the UPC-Tree. For each row, we sort it into a sequence of similarity groups. Then we insert the sequences into the UPC-Tree. As a result, rows that have the same prefixes will share the same paths from root to the end of prefixes. To save memory, the row number associated with each path is only recorded at the node corresponding to the end of the sequence. To find the u-Cluster using the UPC-Tree, subsequences are developed by adding suffixes of each sub-tree as the tree's children, via a pre-order traversal of the UPC-Tree.

**Lemma 3.1** *Given a matrix  $M$ , a similarity grouping threshold  $\delta$ , the initiated UPC-Tree contains all the information of matrix  $M$ .*

**Rationale:** Based on the UPC-Tree construction process, each row in the matrix is mapped onto one path in the UPC-Tree. The row IDs and the order of the columns are completely stored in the initiated UPC-Tree.



- *The collapsed node will split if a new branch has to be inserted in the middle of path.* For example, in Figure 2, sequence 1 ( $DBAC$ ) is collapsed into one node when the tree is initiated. In the development of depth 2, since the subsequence  $DAB$  in sequence 4 will be inserted into path  $DBAC$ , and the only common prefix they have is  $D$ , a new branch  $AB$  has to be added in  $D$ 's sub-tree. The original node which contains  $DBAC$  will split into two nodes which contain  $D$  and  $BAC$  respectively.  $BAC$  will become a sub-tree of  $D$ .
- *The collapsed node will split if the inserted branch is a contiguous portion of the single path in the collapsed node.* For example, in Figure 2, when the subsequence  $DBA$  in sequence 3 is inserted into  $DBAC$  of sequence 1,  $DBA$  is a portion of  $DBAC$ ,  $DBAC$  is split into two parts  $DBA$  and  $C$ . The number 3 is stored at the end of  $DBA$  to record sequence ID correctly.

Compared with the original UPC-Tree, the collapsed UPC-Tree occupies much less space and takes much less time. For example, at depth 0, the original tree needs 15 nodes, while the collapsed tree only needs 5. At depth 1, the original tree needs 21 nodes, but collapsed one needs only 12. In addition, with collapsed UPC-Tree, inserting suffix of single-path tree is avoided. The single-path is compacted into one collapsed node already. The u-Cluster can be identified immediately.

### 3.4 Additional Feature: Extension of Grouping Technique

Based on Definition 2.1, we can generate different similarity groups if we starts from different pivot attributes. For example, If we have an object  $[0.5, 1, 1.5, 2]$  with the similarity threshold as  $\delta = 100\%$ . We can group them either as  $(AB)(CD)$  or as  $A(BCD)$ . Now have objects 2 and 3 with attribute values  $[1, 2, 4, 5]$  and  $[9, 4, 5, 6]$ , respectively. Their corresponding group sequences are  $(AB)(CD)$  and  $(BCD)A$ . When we set  $nc = 3$  and  $nr = 2$ , the cluster we can get is  $(AB)(CD)$  if we group object 1 as  $(AB)(CD)$ . However, if we use  $A(BCD)$ , the cluster will be  $(BCD)$ . Both are valid clusters. To find them, we propose an alternative grouping approach. We put all (possibly overlapping) similar groups in one sequence. For example, in the above example, object 1 becomes  $A(AB)(BCD)(CD)$ . Then, we can find all sequential patterns of these three objects, which are  $(AB)(CD)$  and  $(BCD)$ . Since we introduce some redundancy, when one attribute appears more than once in a final cluster, we only keep one by removing all duplicates.

## 4 Experiments

We experimented our u-Cluster algorithm with two real data sets. The algorithm was implemented in C and executed on a Linux machine with a 700 MHz CPU and 2G main memory. The following tests are organized into three categories. First, we show four promising patterns found in real data sets, which have been overlooked by previous models. Secondly, we study the sensitivity of u-Cluster to various parameters. Finally, we evaluate the performance of UPC-Tree.

### 4.1 Data Sets

We experiment our u-Cluster algorithm with two real data sets.

- **Gene Expression Data.** Gene expression data are generated by DNA chips and other microarray techniques. The yeast microarray contains expression levels of 2,884 genes under 17 conditions [17]. The data set is presented as a matrix. Each row corresponds to a gene and each column represents a condition under which the gene is developed. Each entry represents the relative abundance of the mRNA of a gene under a specific condition. The entry value, derived by scaling and logarithm from the original relative abundance, is in the range of 0 and 600. Biologists are interested in finding a subset of genes showing strikingly similar up-regulation or down-regulation under a subset of conditions [6].
- **Drug Activity Data.** Drug activity data is also a matrix with 10000 rows and 30 columns. Each row corresponds to a chemical compound and each column represents a descriptor/feature of the compound. The value of each entry varies from 0 to 1000.

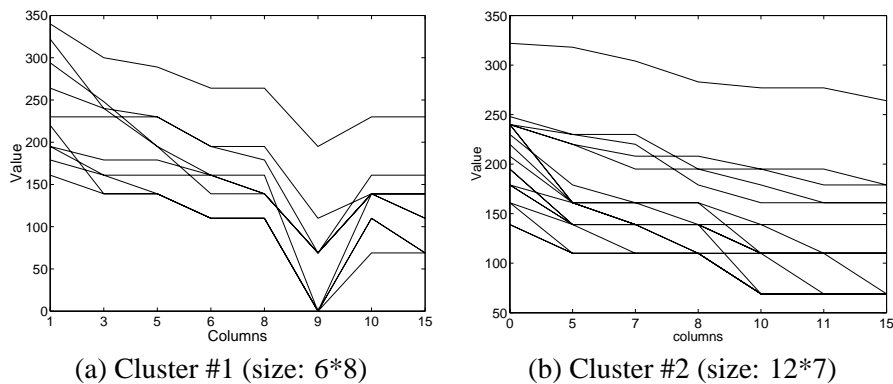


Figure 3: Cluster Analysis: Two example uClusters in yeast data

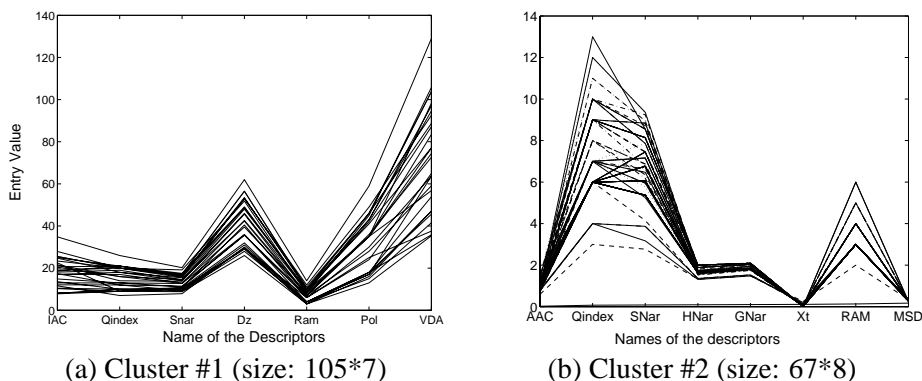


Figure 4: Cluster Analysis: Two example uClusters in drug activity data

## 4.2 Results from Real Data

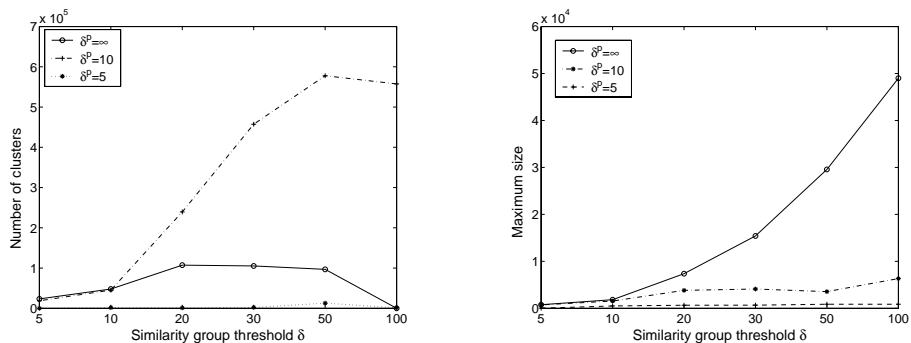
First, we apply the u-Cluster algorithm to two data sets. With parameter  $\delta = 10$ , Some interesting clusters are reported in both of the data sets. As showed in Figure 3, the two patterns generated from yeast dataset [6] present the coherent tendency along the columns on the Y axis. In Figure 3(a), if we rearrange the columns as 15, 10, 9, 8, 6, 5, 3, 1, we will see the 'up' pattern embedded in it. Figure 3 (b) shows another interesting cluster which present with a 'down' tendency itself. Another pair of patterns are showed in Figure 4. They present a series of consistent patterns under a subset of features. It is also interesting to notice that, the patterns includes the three descriptors: SNar, GNar and HNar.

Besides this, in both of the figures, we can observe that the curves with sharper slopes are discovered with 'up' pattern, while it can never be discovered by in the traditional distance measure nor other pattern-based models.

## 4.3 Model Sensitivity Analysis

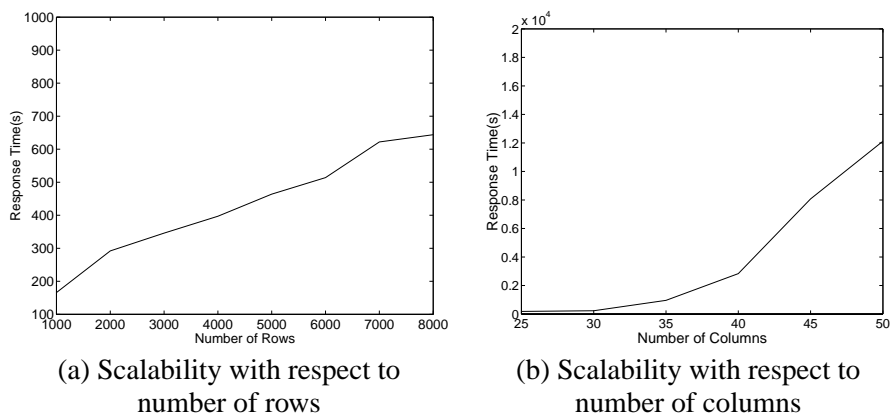
In this section, We evaluate how the similarity thresholds  $\delta$  and  $\delta^p$  can influence the number of clusters and their sizes. We use the yeast data set in this set of experiments. The minimum number of rows is set to be 30 and the minimum number of columns is 10. We vary  $\delta^p$  from  $\infty$  to 5. Figure 5 (a) shows the number the clusters generated and (b) presents the maximum cluster size. u-Clusters are generated when  $\delta^p$  is infinity. As  $\delta$  increases, the total number of clusters begins to increase, which implies that more columns are grouped together and that more rows are sharing the same subsequences. However, when the similarity threshold is larger than 100%, the total number of clusters drops. This is because the overlapped clusters generated by small  $\delta$  begin to merge into bigger clusters when  $\delta$  increases. Since long subsequence of columns has higher chance to fall on a single path, consequently, shorter enclosed subsequences will not be counted, and the total number of clusters decreases. Figure 5(b) shows that the maximum size of the u-Cluster increases dramatically in this case. As  $\delta^p$

decreases, large clusters tend to split into smaller ones and the total number of clusters increases. As more clusters with size smaller than  $nc \times nr$  are eliminated, the total number of clusters restricted by  $\delta^p$  drops below the number of u-Clusters.



(a) Total number of clusters as a function of similarity threshold (b) Size(#column x #rows) as a function of similarity threshold

Figure 5: Performance Study: cluster number and cluster size V.S. similarity threshold



(a) Scalability with respect to number of rows (b) Scalability with respect to number of columns

Figure 6: Performance Study: Response time V.S. number of columns and number of rows

#### 4.4 Scalability

We evaluate the performance of the u-Cluster algorithm as we increase the number of objects and the number of columns in the data set. The response time of the UPC-Tree is mostly determined by the size of the tree. Figure 6 shows the response time of the drug activity data set. As we know, the columns and the rows of the matrix carry the same significance in the u-Cluster model, which is symmetrically defined in Formula 2. Although the algorithm is not entirely symmetric in the sense that it chooses to project column-pairs first, the curves in Figure 6 demonstrate similar trends.

For experiments in Figure 6(a), the number of columns is 30. The minimal number of columns of the u-Cluster is 9, and the minimal number of rows is set to  $0.01N$ , where  $N$  is the number of rows. The mining algorithm is invoked with  $\delta = 0.2$ ,  $nc = 9$ , and  $nr = 0.01N$ . Data sets used in Figure 6(b) are taken from the drug activity data with the number of rows fixed as 1000. The mining algorithm is invoked with  $\delta = 0.2$ ,  $nc = 0.66C$ , and  $nr = 30$ .

Next, we study the impact of the parameters ( $\delta$ ,  $nc$ , and  $nr$ ) towards the response time. The results on the yeast gene expression data are shown in Figure 7. When  $nc$  and  $nr$  are fixed, the response time prolongs when the similarity threshold increases. This is because the size of the clusters is getting larger as we relax the similarity threshold. Therefore, the UPC-Tree has to spend more time to construct a deeper tree. When similarity threshold is fixed, it takes longer time to construct the UPC-Tree as  $nc$  decreases. This is showed in Figure 7(a). According to the pruning techniques we discuss in Lemma

3.4, a fewer number of subsequences can be eliminated when using smaller  $nc$ . As a result, a larger tree is constructed, which consumes more time. A similar effect can be observed with respect to  $nr$  from Figure 7(b).

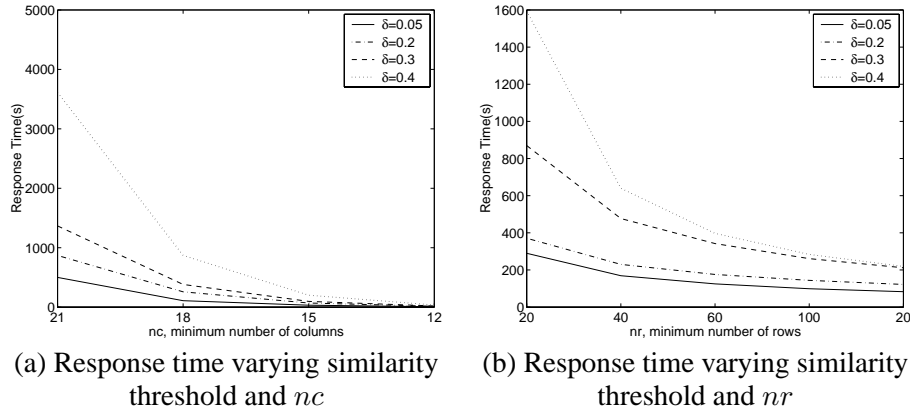


Figure 7: Performance Study: Response time V.S. similarity threshold ,  $nc$  and  $nr$

## 5 Conclusions

To capture the the consistent tendency clusters exhibited by a subset of conditions in gene expression data, in this paper, we introduce a new model namely u-Cluster and devised a depth-first algorithm that can efficiently and effectively discover all u-Clusters with a user-specified minimum threshold. u-Cluster extends the original OPSM by relaxing strict orders among conditions with similar expression levels measured by  $\delta$ . The most important difference, however, lies in capability to discover all the hidden submatrices of interests. The OPSM is only able to generate the largest OPSM for a given  $s$ , the column size of the OPSM. However, it may be unable to uncover the OPSMs with less row support in the matrix. In contrast, our algorithm can identify multiple u-Clusters satisfying the minimum size threshold simultaneously. Moreover, our algorithm is deterministic in that it can discover all the qualified submatrices, while OPSM can only generate approximate answer of the order of a subset of columns and the row support for this order. Furthermore, the algorithm of OPSM is sensitive to the number of unplanted rows.

Still, there are some extensions we can make based on our model. Although we've relaxed the strict order requirement among the conditions with similar expression level, it is still too optimistic to expect the order of among groups are meaningful, due to the complexity of biological patterns and the noises in microarray data. Therefore, to explore *similar* yet exact orders among a subset groups will be one relaxation of current model. The similarity measure of two sequences can base on the number of reverse pairs. For example, two sequences  $ABCD$  and  $DCBA$  have six reverse pairs,  $\{\{A, B\}, \{A, C\}, \{A, D\}, \{B, C\}, \{B, D\}, \{C, D\}\}$  while  $ABCD$  and  $ACBD$  only have one reverse pairs,  $\{\{B, C\}\}$ . If the similarity threshold is 3, we can take  $ABCD$  and  $ACBD$  as two similarly ordered genes. Our algorithm can also be adapted to accommodate this by adding similarity check at each branch during the depth-first development.

Current microarray technology permits the examination of gene expression patterns of tens of thousands of genes. One challenge facing the biologist is to interpret the results properly. The generated UPC-Tree of our algorithm naturally present a hierarchical structure of those patterns. For example, for any internal node  $N$  in UPC-Tree, the subset of genes recorded in the subtree of  $N$  sharing a longer path with each other. In other words, they are more closely related to each other than the subset of genes at  $N$ . Further study on the tree can potentially lead to the discovery of unknown gene ontology.

In addition, to accelerate the performance of our algorithm, we can combine the probability score used in OPSM in our algorithm to eliminate the paths with little probability to be included in a cluster earlier. One the other hand, by developing all the paths with high probability to be the potential clusters, we can still keep the quality and completeness of the desired u-Clusters.

Besides the above extensions and improvement, we expect our model can be used to uncover significant hidden submatrices showing consistent tendency in microarray data, and meanwhile, we expect to investigate functional, clinical or

biochemical interpretations of the generated patterns.

## References

- [1] C. C. Aggarwal and P. S. Yu. Finding generalized projected clusters in high dimensional spaces. In *SIGMOD*, pages 70-81, 2000.
- [2] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD*, 1998.
- [3] A. Ben-Dor, R. Shamir, and Z. Yakhini. Clustering gene expression patterns. in *J Comput Biol* 6(3-4):281-97.
- [4] A. Ben-Dor, B. Chor, R. Karp, and Z. Yakhini. Discovering Local Structure in Gene Expression Data: The Order-Preserving Submatrix Problem. In *RECOMB* 2002.
- [5] C. H. Cheng, A. W. Fu, and Y. Zhang. Entropy-based subspace clustering for mining numerical data. In *SIGKDD*, pages 84-93, 1999.
- [6] Y. Cheng and G. Church. Biclustering of expression data. In *Proc. of 8th International Conference on Intelligent System for Molecular Biology*, 2000.
- [7] M.B.Eisen, P.T.Spellman, P.O.Brown, and D.Botstein. Cluster analysis and display of genome-wide expression patterns. In *Proc Natl Acad Sci U S A*, 95(25):14863-8, 1998.
- [8] P. Dhaeseleer, S. Liang, and R. Somogyi. Gene expression analysis and genetic network modeling. In *Pacific Symposium on Biocomputing*, 1999.
- [9] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *SIGKDD*, pages 226-231, 1996.
- [10] K. Fukunaga. Introduction to Statistical Pattern Recognition. Academic Press, 1990.
- [11] J.Liu and W.Wang. Flexible clustering by tendency in high dimensional spaces. Technical Report TR03-009, Computer Science Department, UNC-CH, 2003.
- [12] R. S. Michalski and R. E. Stepp. Learning from observation: conceptual clustering. In *Machine Learning: An Artificial Intelligence Approach*, pages 331-363, 1983.
- [13] F. Murtagh. A survey of recent hierarchical clustering algorithms. In *The Computer Journal*, 1983.
- [14] V. Olman, D. Xu, and Ying Xu, CUBIC: Identification of Regulatory Binding Sites through Data Clustering, *Journal of Bioinformatics and Computational Biology*, to appear, 2003.
- [15] R. Sharan and R. Shamir. Click: A clustering algorithm with applications to gene expression analysis. In *Proc. ISMB*, pages 307-316, 2000
- [16] P.T. Spellman, G. Sherlock, M.Q.Zhang, V.R.Lyer, K.Anders, M.B.Eisen, P.O.Brown, D.Botstein, and Futcher. Comprehensive identification of cell cycle-regulated genes of the yeast *saccharomyces cerevisiae* by microarray hybridization. *Molecular Biology of the Cell*, 9:3273-3297, 1998.
- [17] S. Tavazoie, J. Hughes, M. Campbell, R. Cho, and G. Church. Yeast micro data set. In <http://arep.med.harvard.edu/biclustering/yeast.matrix>, 2000.
- [18] H. Wang, W. Wang, J. Yang, and P. Yu. Clustering by pattern similarity in large data sets, in *SIGMOD*, pp. 394-405, 2002.
- [19] J. Yang, W. Wang, H. Wang, and P. S. Yu.  $\delta$ -clusters: Capturing subspace correlation in a large data set. In *ICDE*, pages 517-528, 2002.
- [20] J. Yang, H. Wang, W. Wang, and P. Yu. Enhanced biclustering on gene expression data, *Proc. IEEE BIBE*, 2003.
- [21] Ka Yee Yeung, Chris Fraley, A. Murua, Adrian E. Raftery, Walter L. Ruzzo: Model-based clustering and data transformations for gene expression data. *Bioinformatics* 17(10): 977-987, 2001.
- [22] Ka Yee Yeung, David R. Haynor, Walter L. Ruzzo: Validating clustering for gene expression data. *Bioinformatics* 17(4): 309-318, 2001.
- [23] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *SIGMOD*, pages 1031-14, 1996.
- [24] Yan Zhou, Guyang Matthew Huang, Liping Wei: UniBLAST: a system to filter, cluster, and display BLAST results and assign unique gene annotation. *Bioinformatics* 18 (9): 1268, 2002.