Mining Subspace Clusters from DNA Microarray Data Using Large Itemset Techniques

Ye-In Chang[1], Jiun-Rung Chen[2], and Yueh-Chi Tsai[3]

Dept. of Computer Science and Engineering, National Sun Yat-Sen University

No. 70, Lienhai Rd., Kaohsiung 80424, Taiwan, R.O.C.

Abstract

Mining subspace clusters from the DNA microarrays could help researchers identify those genes which commonly contribute to a disease, where a subspace cluster indicates a subset of genes whose expression levels are similar under a subset of conditions. Since in a DNA microarray, the number of genes is far larger than the number of conditions, those previous proposed algorithms which compute the maximum dimension sets (MDSs) for any two genes will take a long time to mine subspace clusters. In this paper, we propose the *Large Itemset-Based Clustering* (LISC) algorithm for mining subspace clusters. Instead of constructing MDSs for any two genes, we construct only MDSs for any two conditions. Then, we transform the task of finding the maximal possible gene sets into the problem of mining large itemsets from the condition-pair MDSs. Since we are only interested in those subspace clusters with gene sets as large as possible, it is desirable to pay attention to those gene sets which have reasonable large support values in the condition-pair MDSs. From our simulation results, we show that the proposed algorithm needs shorter processing time than those previous proposed algorithms which need to construct gene-pair MDSs.

[1]Corresponding author. E-mail: changyi@cse.nsysu.edu.tw; Tel.: +886-7-5252000 (ext. 4334); Fax: +886-7-5254301.

[2]E-mail: jiunrung@gmail.com

[3]E-mail: tsaiyc@db.cse.nsysu.edu.tw

(**Key words:** FP-tree, large itemset, microarray, pCluster, subspace clustering)

1 Introduction

DNA microarrays are one of the latest breakthroughs in experimental molecular biology and have opened the possibility of creating datasets of molecular information to represent many systems of biological or clinical interest (Lazzeroni and Owen, 2002; West et al., 2001). Gene expression profiles can be used as inputs to large-scale data analysis, *e.g.*, to discover hidden taxonomies, or to increase our understanding of normal and disease states (Tefferi et al., 2002). However, the large number of genes and the complexity of biological networks greatly increase the challenges of comprehending and interpreting the resulting cluster of data, which often consists of millions of measurements. Analysis of such data is becoming one of the major bottlenecks in the utilization of the technology (Hakamada et al., 2006; Jiang et al., 2004; Liu and Wang, 2007; Yang et al., 2002, 2003).

The gene expression data from a microarray experiment can be represented by a real-value expression matrix, where rows represent genes, columns represent various samples, and each element of this matrix represents the expression level of the particular gene in the particular sample. Currently, a typical microarray experiment contains 10^3 to 10^4 genes, and this number is expected to reach to the order of 10^6. However, the number of samples involved in a microarray experiment is generally less than 100. Figure 1 shows an example of the DNA microarray dataset, where $M \gg N$, where $M > 10^3$ and $N < 10^2$ (Yang et al., 2002; Koo et al., 2006; Madeira and Oliveira, 2004; Wang et al., 2004; Yang et al., 2007).

Since the dataset which comes from the biological experiments is large, many data mining techniques are used to study the data efficiently. Mining techniques, such as classification and clustering, are proposed to analyze and predict the functions of newly found genes or proteins (Agrawal et al., 1998; Chang et al., 2007; Golub et al., 1999). Clustering is an important data mining problem (Aggarwal et al., 1999; Aggarwal and Yu, 2000; Cheng et al., 1999; Ester et al., 1996; Pei et al., 2003). For a set of objects, clustering is the process of grouping the objects into a set of disjoint classes, called *clusters*, such that objects within a cluster have high similarity to each other, while objects in different clusters are dissimilar (Jiang et al., 2005). Recent efforts in data mining have focused on methods for efficient and effective cluster analysis (Zhang et al., 1996) in large databases, *e.g.*, microarray datasets.

Clustering techniques have been proven to be helpful to understand gene function, gene

	c 1	c 2	. . .	c N
Gene 1	0.13	2.55	. . .	0.59
Gene 2	0.66	1.31	. . .	1.01
Gene 3	1.00	0.45	. . .	10.23
.	.	.	. . .	.
.	.	.	. . .	.
.	.	.	. . .	.
.	.	.	. . .	.
.	.	.	. . .	.
.	.	.	. . .	.
.	.	.	. . .	.
.	.	.	. . .	.
.	.	.	. . .	.
Gene M	0.38	12.34	. . .	5.12

Figure 1: A DNA microarray dataset

regulation, cellular processes, and subtypes of cells. Genes with similar expression patterns can be clustered together with similar cellular functions (Jiang et al., 2004; Sultan et al., 2002). Moreover, investigations show that more often than not, several genes contribute to a disease, which motivates researchers to identify a subset of genes whose expression levels are similar under a subset of conditions; that is, they exhibit fluctuation of a similar shape when conditions change (Brown and Botstein, 1999).

Most clustering models, including those used in subspace clustering, define similarity among different objects by distances over either all or only a subset of the dimensions. Some well-known distance functions include Euclidean distance, Manhattan distance, and cosine distance. However, distance functions are not always adequate in capturing correlations among the objects. In fact, strong correlations may still exist among a set of objects, even if they are far apart from each other as measured by the distance functions (Wang et al., 2002). This is demonstrated with an example in Figure 2 and Figure 3.

Figure 2 shows a data set with 3 genes and 10 conditions . No patterns among the 3 genes are visibly explicit. However, if we pick a subset of the conditions, $\{b, c, h, j, e\}$, and plot the values of the 3 genes on these conditions in Figure 3-(a), it is easy to see that they manifest similar patterns. However, these genes may not be considered in a cluster by any traditional clustering model, because the distance between any two of them is not

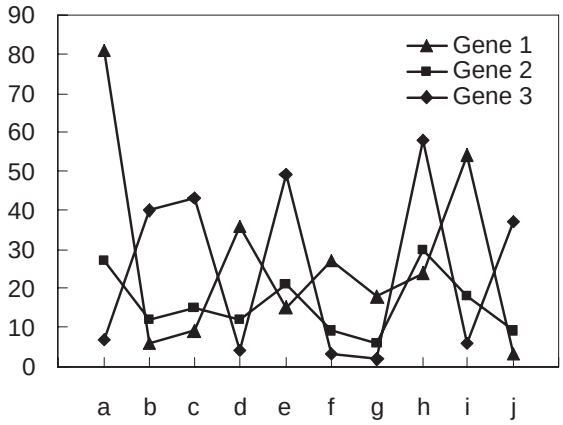


Figure 2: Raw data: 3 genes and 10 conditions

close. The same set of genes can form different patterns on different sets of conditions. In Figure 3-(b), we show another pattern in subspace $\{c, e, g, b\}$. If we think of conditions $\{c, e, g, b\}$ as different environmental stimuli, the pattern shows that the 2 genes respond to these conditions coherently. Our goal of this paper is to discover such subspace clusters from raw data sets such as Figure 2.

Many techniques have been proposed to find subspace clusters with the coherence expression of a subset of genes on a subset of conditions. Some well-known subspace clustering algorithms based on the main categories of approximate answers and complete answers are shown in Figure 4. Cheng and Church (Cheng and Church, 2000) proposed the bicluster model which captures the coherence of the genes and conditions in a submatrix of a DNA microarray. Next, based on the same bicluster model, Yang *et al.* (Yang et al., 2002) proposed a move-based algorithm, δ-cluster, to improve the performance of the biclustering algorithm to find subspace clusters. Unlike the biclustering algorithm (Cheng and Church, 2000) and the δ-clusters algorithm (Yang et al., 2002), the pCluster algorithm (Wang et al., 2002) simultaneously detects multiple clusters that satisfy the user-specified δ threshold. Moreover, since the pCluster algorithm provides the complete answer, they will not miss any qualified subspace clusters, while random algorithms, *e.g.*, the biclustering algorithm (Cheng and Church, 2000) and the δ-clusters algorithm, (Yang et al., 2002) provide only an approximate answer.

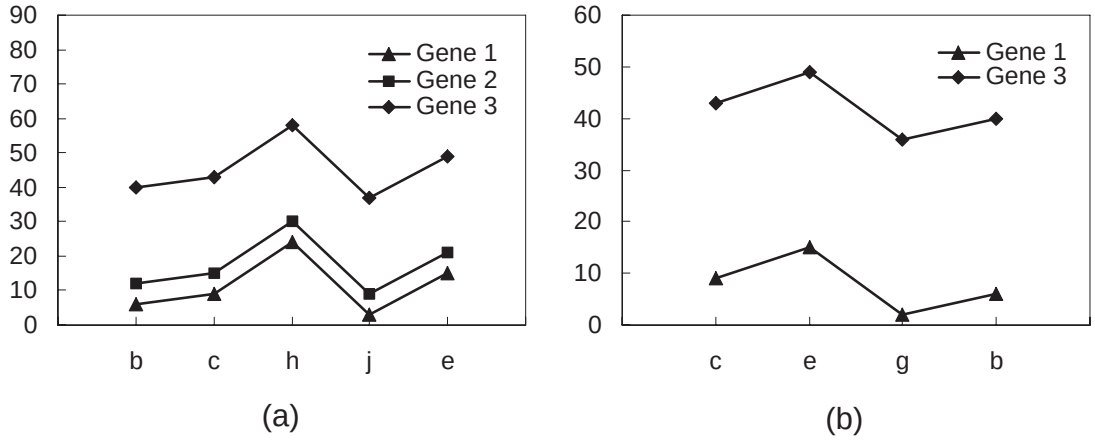


Figure 3: Genes from pattern on a set of conditions: (a) subspace cluster: (gene1, gene2, gene3)$\times(b, c, h, j, e)$; (b) subspace cluster: (gene1, gene3)$\times(c, e, g, b)$.

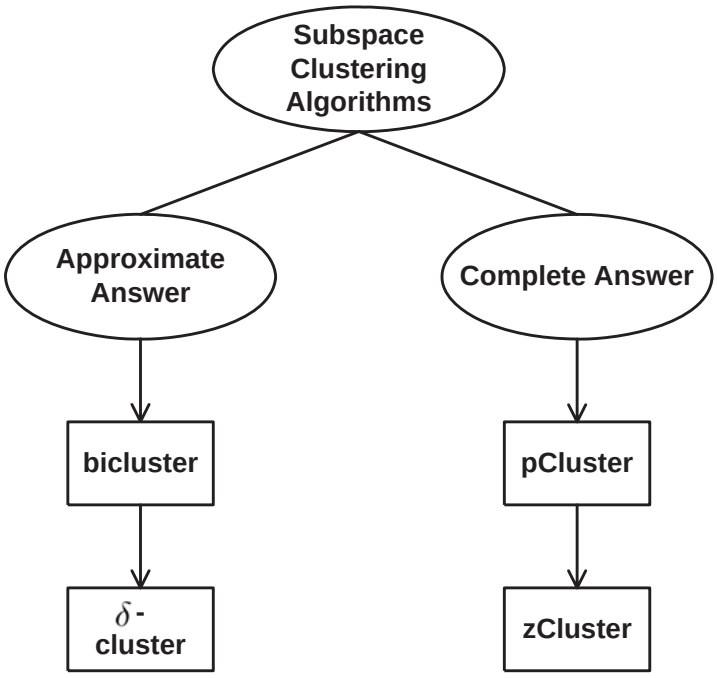


Figure 4: Subspace clustering algorithms

The cluster search problem is in general NP-hard (Sultan et al., 2002), and the subspace clustering problem is no exception (Yang et al., 2002; Wang et al., 2002; Cheng and Church, 2000). To cope with this computational challenge, the zCluster algorithm (Yoon et al., 2005) exploits a compact data structure called *zero-suppressed binary decision diagrams* (ZBDDs) (Minato, 1993) to implicitly represent and manipulate massive data. The ZBDDs have been used widely in other domains, namely, the computer-aided design of *very large-scale integration* (VLSI) digital circuits, and can be useful in solving many practical instances of intractable problems. The zCluster algorithm exploits this property of ZBDDs, and can find all the subspace clusters that satisfy specific input conditions without exhaustive enumeration.

Although the pCluster algorithm (Wang et al., 2002) and the zCluster algorithm (Yoon et al., 2005) provide the complete answer, they contain some time-consuming steps. First, the pCluster algorithm and the zCluster algorithm equally use the clusters containing only two genes or two conditions to construct larger clusters having more genes and conditions, which are called gene-pair and condition-pair MDSs. However, this step of measuring the difference of each gene-pair on the conditions of a DNA microarray is really time-consuming, since the number of genes in the real life microarray is usually very large. A typical microarray experiment contains 10^3 to 10^4 genes, and this number is expected to reach to the order of 10^6, while the number of samples involved in a microarray experiment is generally less than 100. Thus, the time complexity of constructing the gene-pair MDSs is much higher than the time complexity of constructing the condition-pair MDSs in those previous proposed clustering algorithms.

In addition, the pCluster algorithm (Wang et al., 2002) proposes a prefix tree structure using the depth-first algorithm to mine the final subspace clusters. The zCluster algorithm (Yoon et al., 2005) contains the similar step of mining. However, this step is the bottleneck of the mining. For each node, the pCluster algorithm has to examine the possible combinations of genes on the conditions registered in the path. The algorithm distributes the gene information in each node to other nodes which represent subsets of the condition set along the path of this node. This distributing operation is the major cause that the pCluster algorithm may not be efficient or scalable for large databases.

TID	Items
100	ACD
200	BCE
300	ABCE
400	BE

Figure 5: A transaction database for mining association rules

Therefore, to avoid the above disadvantages, in this paper, we propose the *Large Itemset-Based Clustering* (LISC) algorithm to mine the subspace clusters in the microarray. In our proposed algorithm, first, we only consider the condition-pair MDSs, instead of constructing the gene-pair MDSs. Second, we transform the task of mining the possible maximal gene sets into the mining problem of the *large itemsets* (*i.e.* frequent patterns) from the condition-pair MDSs. We make use of the concept of the large itemset which is used in mining association rules, where a large itemset is represented as a set of items appearing in a sufficient number of transactions. That is, given a database of sales transactions, it is desirable to discover the important associations among items such that the presence of some items in a transaction will imply the presence of other items in the same transaction. For example, a transaction database is given in Figure 5. Assuming that the minimum transaction support required is 2, the set of large itemsets, $\{A\colon\ 2, B\colon\ 3, C\colon\ 3, E\colon\ 3, AC\colon\ 2, BC\colon\ 2, BE\colon\ 3, CE\colon\ 2, BCE\colon\ 2\}$, composed with the minimum support required, can then be determined. Since we are only interested in the subspace cluster with gene set as large as possible, it is desirable to pay attention to those gene sets which have reasonably large support from the condition-pair MDSs. In other words, we want to find the large itemsets from the condition-pair MDSs; therefore, we obtain the gene set with respect to enough condition-pairs. In this step, we efficiently use the revised version of the FP-tree structure to find the large itemsets of gene sets from the condition-pair MDSs. The FP-tree structure has been shown to be one of the most efficient data structures for mining large itemsets, and is a extended prefix tree structure for storing compressed, crucial information about large itemsets. Thus, we can avoid the complex distributing operation and reduce the search space dramatically by using the FP-tree structure. Finally, we develop an algo-

rithm to construct the final clusters from the gene set and the condition-pair after searching the FP-tree. From our simulation results, we show that our proposed algorithm is more efficient than those previous proposed algorithms.

The rest of this paper is organized as follows. Section 2 gives a survey of two subspace cluster algorithms. Section 3 presents the proposed large itemset-based clustering (LISC) algorithm. In Section 4, we study the performance and make a comparison of the proposed algorithm with other previous proposed algorithm. Finally, we give a conclusion in Section 5.

2 Related Work

In this section, we will describe two well-known subspace clustering algorithm, pCluster (Wang et al., 2002) and zCluster (Yoon et al., 2005). In particular, pCluster algorithm (Wang et al., 2002) will be examined in detail.

2.1 pCluster

Wang *et al.* (Wang et al., 2002) proposed a clustering model, namely the *pCluster*, to capture not only the closeness of objects, but also the similarity of the patterns exhibited by the objects. Let D be a set of objects, where each object is defined by a set of attributes A. It is interested in objects that exhibit a coherent pattern on a subset of attributes of A. Let O be a subset of objects in the database ($O \subseteq D$), and let T be a subset of attributes ($T \subseteq A$). Pair (O, T) specifies a submatrix. Given x, $y \in O$, and $a, b \in T$, the *pScore* of the 2×2 matrix is defined as:

$$pScore(\begin{bmatrix} d_{xa} & d_{xb} \\ d_{ya} & d_{yb} \end{bmatrix}) = |(d_{xa} - d_{xb}) - (d_{ya} - d_{yb})|$$

Pair (O, T) forms a δ-pCluster if for any 2×2 submatrix X in (O, T), we have $pScore(X) \leq \delta$ for some $\delta \geq 0$.

Figure 6-(a) shows a microarray matrix with ten genes (one for each rows) under five experiment conditions (one for each column). This example is a portion of microarray data that can be found in (Tavazoie et al., 2000). Figure 6-(b) shows that a pCluster (VPS8, EFB1, CYS3, CH1I, CH1D, CH2B) is embedded in the microarray.

The algorithm is described in the following three steps.

Step 1: Finding attribute-pair and object-pair MDSs.

Clearly, a pCluster must have at least two objects and two attributes. Intuitively, we can use those pClusters containing only two objects or two attributes to construct larger pClusters having more objects and attributes.

conditions

genes	CH1I	CH1B	CH1D	CH2I	CH2B
CTFC3	4392	284	4108	280	228
VPS8	401	281	120	275	298
EFB1	318	280	37	277	215
SSA1	401	292	109	580	238
FUN14	2857	285	2576	271	226
SP07	228	290	48	285	224
MDM10	538	272	266	277	236
CYS3	322	288	41	278	219
DEP1	312	272	40	273	232
NTG1	329	296	33	274	228

(a)

conditions

genes	CH1I	CH1B	CH1D	CH2I	CH2B
CTFC3					
VPS8	401		120		298
EFB1	318		37		215
SSA1					
FUN14					
SP07					
MDM10					
CYS3	322		41		219
DEP1					
NTG1					

(b)

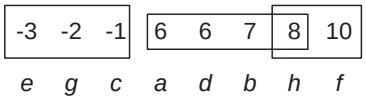
Figure 6: A pCluster of yeast genes: (a) gene expression data; (b) a pCluster.

Given a pair of objects, we compute the object-pair MDS (Maximum Dimension Set) efficiently. For example, Figure 7-(a) shows the attribute values of two objects. The last row shows the differences of the attribute values.

To compute the object-pair MDS, the pCluster algorithm sorts the attributes in the difference ascending order, as shown in Figure 7-(b). Suppose $\delta = 2$. The pCluster algorithm runs through the sorted list using a sliding window of variable width. Clearly, the attributes in the sliding window form a δ-pCluster provided the difference between the rightmost element and the leftmost one is no more than δ. For example, we firstly set the left edge

Object	Attributes							
	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
o_1	13	11	9	7	9	13	2	15
o_1	7	4	10	1	12	3	4	7
o_1 - o_2	6	7	-1	6	-3	10	-2	8

(a)



(b)

Figure 7: Finding MDS for object-pair: (a) the attribute values of two objects; (b) finding MDS: (e, g, c), (a, d, b, h), (h, f).

of the sliding window at the left end of the sorted list, and moves the right edge of the window until it sees the first 6. The attributes in between, $\{e, g, c\}$, is the set of attributes of an object-pair MDS. Then, we move the left edge of the sliding window to attribute g, and repeats the process until the left end of the window runs through all elements in the list. In total, three MDSs can be found, *i.e.*, $(\{o_1, o_2\}, \{e, g, c\})$, $(\{o_1, o_2\}, \{a, d, b, h\})$ and $(\{o_1, o_2\}, \{h, f\})$.

A similar method can be used to find the attribute-pair MDSs.

Step 2: Pruning Unpromising MDS.

In an object-pair MDS $(\{o_1, o_2, D\})$, if the number of attributes in D is less than min_a, the user-specified minimum number of attributes, then o_1 and o_2 can not appear together in any significant pCluster. Similarly, in an attribute-pair MDS $(R, \{a_1, a_2\})$, if the number of objects in R is less than min_o, the user-specified minimum number of objects, then a_1 and a_2 can not appear together in any significant pCluster. Based on this idea, pCluster algorithm conducts the dual pruning between the object-pair MDSs and the attribute-pair MDSs.

Step 3: Generating significant pClusters.

After pruning in Step 2, the pCluster algorithm inserts the surviving object-pair MDSs into a prefix tree. For each object-pair MDS, all attributes are sorted according to a global order $\Re$ and then inserted into the tree. The two objects are registered in the last node of the path corresponding to the sorted attribute list. If two object-pair MDSs share the same prefix with respect to $\Re$, they share the corresponding path from the root in the tree.

Clearly, since every object-pair MDS surviving from the pruning step must have at least min_a attributes, no object will be registered in any node whose depth is less than min_a. After all object-pair MDSs are inserted into the tree, the pCluster algorithm treats each node in the tree whose depth is at least min_a as a candidate pCluster, and verifies whether the objects registered at the node really form a pCluster. Moreover, if all objects at a node in the tree form a pCluster, any ancestor of the node in the tree registering the same set of objects also form a pCluster. Therefore, a post-order traversal of the prefix tree is conducted to examine the nodes whose depths are no less than min_a, and generate the pClusters.

2.2 zCluster

Yoon *et al.* (Yoon et al., 2005) proposed the *zCluster* algorithm based on the pCluster model that exploits the *zero-suppressed binary decision diagrams* (ZBDDs) data structure to cope with the computational challenges. The ZBDDs have been used widely in other domains, namely, the computer-aided design of *very large-scale integration* (VLSI) digital circuits, and can be useful in solving many practical instances of intractable problems. The zCluster algorithm exploits this property of ZBDDs, and can find all the subspace clusters that satisfy specific input conditions without exhaustive enumeration.

Let $U_G = \{g_0, g_1, ..., g_{n-1}\}$ and $U_E = \{e_0, e_1, ..., e_{m-1}\}$ represent a set of genes and a set of experimental conditions involved in gene expression measurement, respectively. The first step of zCluster is refering to $2 \times |E|$ or $|G| \times 2$ maximal clusters as *maximum dimension sets* (MDSs). In order to generate MDSs, zCluster uses an approach similar to that used in the pCluster algorithm. The zCluster algorithm differs in the remaining steps after constructing the prefix tree used in pCluster.

The zCluster algorithm efficiently utilizes ZBDDs (Minato, 1993) in the remaining steps. This ZBDD-based representation is crucial to keeping the entire algorithm computationally manageable. The key observation is that a set of condition-pair MDSs can be regarded as a set of combinations and represented compactly by the ZBDDs. Therefore, the symbolic representation using ZBDDs is more compact than the traditional data structures for sets. Moreover, the manipulation of condition-pair MDSs, such as union and intersection, is implicitly performed on ZBDDs, thus resulting in high efficiency.

	c_0	c_1	c_2	c_3	c_4	c_5
g_0	2	2	9	2	3	4
g_1	3	7	3	1	9	3
g_2	2	2	7	2	6	3
g_3	3	2	3	2	1	3
g_4	2	1	5	1	0	4
g_5	3	5	5	8	2	3
g_6	2	9	7	4	0	0

Figure 8: Example 1: A gene expression data matrix

3 The LISC Algorithm

In this section, we introduce the Large Itemset-Based Clustering (LISC) algorithm to mining subspace clusters from DNA microarray data.

3.1 Definitions and Problem Statement

Let $G = \{g_0, g_1, ..., g_{m-1}\}$ and $C = \{c_0, c_1, ..., c_{n-1}\}$ represent a set of genes and a set of experimental conditions involved in gene expression measurement, respectively. The result can be represented by the matrix $GE \in \Re^{|G|\times|C|}$ with the set of rows G and the set of columns C. Each element $GE[x][a] \in GE$ corresponds to the expression information of gene x in condition a (Yoon et al., 2005).

A subspace cluster is defined to be a subset of genes that exhibit similar behavior under a subset of experimental conditions, and vice versa. Thus, in the gene expression data matrix $GE = (G, C)$, a subspace cluster will appear as a submatrix of GE. We denote this submatrix by pair (R, D) where $R \subseteq G$ and $D \subseteq C$. We specify the size of the cluster by $|R| \times |D|$.

Example 1 *Let* $minG = 3$, $minC = 3$, $\delta = 1$, *and an example of data matrix* $GE = (G, C)$ *as shown in Figure 8. The final subspace clusters are* $(g_0, g_2, g_3, g_4) \times (c_0, c_1, c_3)$ *and* $(g_0, g_2, g_3) \times (c_1, c_3, c_5)$.

3.2 The Proposed Algorithm

In this subsection, we will present our proposed LISC algorithm. Basically, the algorithm contains three steps: (1) finding condition-pair MDSs, (2) mining conditional pattern bases, and (3) constructing subspace clusters. First, we will describe a similar algorithm with pCluster (Wang et al., 2002) to generate the *Maximum Dimension Sets* (MDSs) for each condition-pair. Next, we will find the large item set of the gene-condition pair using the revised version of an efficient data structure called *Frequent Pattern Tree* (FP-tree) (Han et al., 2000). Finally, we develop an algorithm to construct the final clusters from the gene set and the condition-pair after searching the FP-tree.

3.2.1 Step 1: Finding Condition-Pair MDSs

Wang *et al.* (Wang et al., 2002) proposed a subspace clustering algorithm called pCluster. The first step of their algorithm is to find all the *Maximum Dimension Set* (MDS) for gene-pair and condition-pair in the polynomial time. In our algorithm, first, we use a similar way to generate MDSs, but only for condition-pair. Then, our algorithm completely differs in the remaining steps.

A subspace cluster (R, D) is a subset of genes R that exhibits a coherent pattern on a subset of conditions D. To formulate the problem, it is essential to describe, given a subset of genes R and a subset of conditions D, how coherent the genes are on the conditions. The measure *pScore* (Wang et al., 2002) serves this purpose.

Definition 1 (H. Wang *et al.* (Wang et al., 2002)) *Let R be a subset of genes in the database ($R \subseteq G$), and let D be a subset of conditions ($D \subseteq C$). Pair (R, D) specifies a submatrix. We assume that each condition is in the domain of real numbers. The value on condition a of gene x is denoted as v_{xa}. For any genes x, $y \in G$ and any conditions a, $b \in C$, the pScore of the 2×2 matrix is defined as:*

$$pScore(\begin{bmatrix} v_{xa} & v_{xb} \\ v_{ya} & v_{yb} \end{bmatrix}) = |(v_{xa} - v_{xb}) - (v_{ya} - v_{yb})|$$

Pair (R, D) forms a subspace cluster, if for any 2×2 *submatrix* X *in (R, D), we have* $pScore(X) \leq \delta$ *for some* $\delta \geq 0$.

Given a set of genes G and a set of conditions C, it is not trivial to find all the MDSs for C, since C can be clustered on any subset of G. Below, we study a special case where C contains only two conditions. Given conditions a and b, and a gene set T , we define $S(a, b, T)$ as:

$$S(a, b, T) = \{v_{xa} - v_{xb} | x \in T\}$$

Based on Definition 1, we can make the following observation: Given conditions a and b, and a gene set T, a and y form a subspace cluster on T *iff* the difference between the largest and smallest value in $S(a, b, T)$ is below δ.

We use $\overrightarrow{S}(a, b, T)$ to denote a sorted sequence of values in $S(a, b, T)$:

$$\overrightarrow{S}(a, b, T) = s_1, ..., s_k$$

$$s_i \in S(a, b, T) \text{ and } s_i \leq s_j \text{ where } i < j$$

Thus, a and b forms a subspace cluster on T if $(s_k - s_1) \leq \delta$. Given a set of genes, G, it is also not difficult to find the MDSs for conditions a and b.

Given a set of dimensions C, $T_s \subseteq C$ is a MDS of a and b *iff*:

1. $\overrightarrow{S}(a, b, T_s) = s_i...s_j$ is a (contiguous) subsequence of $\overrightarrow{S}(a, b, T) = s_1...s_i...s_j...s_k$.

2. $(s_j - s_i) \leq \delta$, whereas $(s_{j+1} - s_i) > \delta$ and $(s_j - s_{i-1}) > \delta$.

Therefore, we can find the MDSs for conditions a and b in the following manner: we start with both the upper-end and the lower-end placed on the first element of the sorted sequence, and we move the lower-end downward one position at a time. For every movement, we compute the difference of the values at the two ends, until the difference is greater than δ. At that time, the elements between the two ends form an MDS. To find the next MDS, we move the upper-end downward one position, and repeat the above process. It stops when the lower-end reaches the last element of the sorted sequence.

Figure 9 shows an example of the above process. We want to find MDSs for two conditions c_3 and c_5 in Example 1. The values on the set of genes, G, are shown in Figure

Condition		c_3	c_5	c_3 - c_5
Gene	g_0	2	4	-2
	g_1	1	3	-2
	g_2	2	3	-1
	g_3	2	3	-1
	g_4	1	4	-3
	g_5	8	3	5
	g_6	4	0	4

(a)

-3	g_4
-2	g_0
-2	g_1
-1	g_2
-1	g_3
4	g_6
5	g_5

(b)

Figure 9: Finding MDSs for one condition-pair: (a) the gene expression data of two conditions; (b) finding MDSs: (g_0, g_1, g_4), (g_0, g_1, g_2, g_3).

9-(a). The patterns are hidden until we sort the values by the difference of c_3 and c_5 on each gene. The sorted sequence $\overrightarrow{S} = -3, -2, -2, -1, -1, 4, 5$ is shown in Figure 9-(b). Assuming $\delta = 1$, we start at the upper-end of S. We move downward until we stop at the first (-1), since $(-1) - (-3) > 1$. The columns between the upper-end and (-1), (g_4, g_0, g_1), is an MDS. We move the upper-end to (-2) and repeat the process until we find all two MDSs for c_3 and c_5: (g_0, g_1, g_4) and (g_0, g_1, g_2, g_3). Note that maximum dimension sets might overlap.

A formal description of the above process is given in Figure 10. We use the *condition_pairMDS* procedure to find MDSs for conditions a and b, where $minG$ is the user-specified minimal number of genes in a subspace cluster, and δ is the user-specified clustering threshold. We can find all condition-pair MDSs for Example 1 shown in Figure 10 based on the *condition_pairMDS* procedure.

3.2.2 Step 2: Mining Conditional Pattern Bases

We transform the task of mining the possible maximal gene sets into the mining problem of the *large itemsets* (*i.e.* frequent patterns) from the condition-pair MDSs. We efficiently

```
1: procedure condition_pairMDS(a, b, G, minG, δ);
2: begin
3:   for each (x ∈ G) do
4:     DGE[x] := GE[x][a] - GE[x][b];
5:   Sort array DGE, and let the sorted array be sortedDGE;
6:   startP := 0; endP := 1;
7:   cutflag := TRUE;
8:   while (endP < |G|)
9:   begin
10:    v := sortedDGE[endP] - sortedDGE[startP];
11:    if (|v| ≤ δ) then
12:    begin
13:      endP := endP + 1;
14:      cutflag := TRUE;
15:    end;
16:    else
17:    begin
18:      if (endP - startP ≥ minG and cutflag := TRUE) then
19:        output condition-pair MDS;
20:      startP := startP + 1;
21:      cutflag := FALSE;
22:    end;
23:  end;
24:  if (endP - startP ≥ minG and cutflag := TRUE) then
25:    output condition-pair MDS;
26: end;
```

Figure 10: Procedure *condition_pairMDS*

Condition-pair	**MDSs**
(c_0, c_1)	(g_0, g_2, g_3, g_4)
(c_0, c_3)	$(g_0, g_2, g_3, g_4), (g_1, g_3, g_4)$
(c_0, c_4)	(g_3, g_4, g_5, g_6)
(c_0, c_5)	$(g_0, g_2, g_4), (g_1, g_2, g_3, g_5)$
(c_1, c_3)	(g_0, g_2, g_3, g_4)
(c_1, c_5)	(g_0, g_2, g_3)
(c_2, c_5)	(g_1, g_3, g_4)
(c_3, c_5)	$(g_0, g_1, g_4), (g_0, g_1, g_2, g_3)$
(c_4, c_5)	$(g_0, g_3, g_5), (g_0, g_5, g_6)$

Figure 11: Condition-pair MDSs

use the revised version of the *Frequent Pattern Tree* (FP-tree) structure (Han et al., 2000) to find the large itemsets of the gene sets from the condition-pair MDSs. The FP-tree structure has been shown to be one of the most efficient data structures for mining large itemsets, and is a extended prefix tree structure for storing compressed, crucial information about large itemsets.

Since only the frequent genes will play a role in the large itemsets mining, it is necessary to perform one scan of the condition-pair MDSs table to identify the large 1-itemsets of genes with frequency support obtained as a byproduct. If we store the large 1-itemsets of genes of the condition-pair MDSs in some compact structure, it may avoid repeatedly scanning the database. If multiple MDSs share an identical large 1-itemset, they can be merged into one with the number of occurrences registered as the support. It is easy to check whether two sets are identical, if the large 1-itemsets in all of the MDSs are sorted according to a fixed order. With the above observation, we scan the condition-pair MDSs table shown in Figure 11 once, and derive a list of large 1-itemsets of genes, $\langle(g_3\colon\ 10), (g_0\colon\ 8), (g_4\colon\ 8), (g_2\colon\ 7), (g_1\colon\ 5), (g_5\colon\ 3), (g_6\colon\ 2)\rangle$, (the number after ":" indicates the support), in which genes ordered in frequency descending order. This result is shown in Figure 12, called large 1-itemsets, with genes denoted as frequent items listed according to the order of the descending support. Note that the support of the large 1-itemsets must be larger than or equal to $C_2^{minC} = C_2^3 = 3$, since the support of genes means the occurrence of genes in the condition-pair MDSs, and we are only interested in the subspace clusters with the size of conditions $\geq minC$. The gene with the occurrence less than C_2^{minC} can not construct the subspace cluster with the size of conditions $\geq minC$. Therefore, we prune $(g_6\colon\ 2)$ from large 1-itemsets and condition-pair MDSs.

According to the large 1-itemset table, we transform the condition-pair MDSs table into the transaction condition-pair MDSs table shown in Figure 13. We separate the MDSs of each condition-pair into distinct *Transaction ID* (TID), and sort the genes according to the descending order of supports, denoted as *transMDSs*. For example, we separte the MDSs $\{(g_0, g_2, g_3, g_4), (g_1, g_3, g_4)\}$ of condition-pair (c_0, c_3), and sort the genes. As shown in Figure 13, the *transMDSs* of (c_0, c_3) are $T_2\colon\ (g_3, g_0, g_4, g_2)$ and $T_3\colon\ (g_3, g_4, g_1)$.

Afterward, we use a similar algorithm with respect to the FP-tree construction algorithm

Large 1-itemset	**Support**
g_3	10
g_0	9
g_4	8
g_2	7
g_1	5
g_5	4
g_6	2

Figure 12: Large 1-itemsets

TID	**Condition-pair**	**transMDSs**
T_1	(c_0, c_1)	(g_3, g_0, g_4, g_2)
T_2	(c_0, c_3)	(g_3, g_0, g_4, g_2)
T_3	(c_0, c_3)	(g_3, g_4, g_1)
T_4	(c_0, c_4)	(g_3, g_4, g_5)
T_5	(c_0, c_5)	(g_0, g_4, g_2)
T_6	(c_0, c_5)	(g_3, g_2, g_1, g_5)
T_7	(c_1, c_3)	(g_3, g_0, g_4, g_2)
T_8	(c_1, c_5)	(g_3, g_0, g_2)
T_9	(c_2, c_5)	(g_3, g_4, g_1)
T_{10}	(c_3, c_5)	(g_0, g_4, g_1)
T_{11}	(c_3, c_5)	(g_3, g_0, g_2, g_1)
T_{12}	(c_4, c_5)	(g_3, g_0, g_5)

Figure 13: TDB for Example 1

(Han et al., 2000) to construct a revised version of FP-tree from TDB and mine conditional pattern bases, as shown in Figure 14 and Figure 15, respectively. In the revised version of the FP-tree structure, every branch of the FP-tree represents an MDS, and the nodes along the branches are stored according to the decreasing order of the corresponding genes frequency, with leaves representing the condition-pairs. First, we create the root of a tree, labeled with "null". Scan the TDB once. The scan of T_1 leads to the construction of the first branch of the tree: $\langle(g_3\colon 1), (g_0\colon 1), (g_4\colon 1), (g_2\colon 1)\rangle$ with a leaf node labeled $T_1\colon (c_0, c_1)$. Note that the gene in the transMDSs is ordered according to the order in the list of large 1-itemsets shown in Figure 12. Figure 16 shows the result of constructing the FP-tree of T_1. For T_2, since its gene set (g_3, g_0, g_4, g_2) is identical to the gene set of T_1, the path is shared with the count of each node along the path increased by 1. The tree is denoted as $\langle(g_3\colon 2), (g_0\colon 2), (g_4\colon 2), (g_2\colon 2)\rangle$ with a leaf node labeled $T_1\colon (c_0, c_1)$ and $T_2\colon (c_0, c_3)$ shown in Figure 17. For T_3, since its gene set (g_3, g_4, g_1) shares a common prefix node g_3 with the existing path $\langle(g_3\colon 2), (g_0\colon 2), (g_4\colon 2), (g_2\colon 2)\rangle$, the count of node g_3 is increased by 1, and one new node $(g_4\colon 1)$ is created and linked as a child of $(g_3\colon 3)$ and another new node $(g_1\colon 1)$ is created and linked as the child of $(g_4\colon 1)$. The leaf node of this branch is denoted as $T_1\colon (c_0, c_1)$. Figure 18 shows the result after constructing the FP-tree of T_1, T_2 and T_3. Repeat the similar way above, and we can construct the FP-tree shown in Figure 14.

To facilitate tree traversal, an item header table is built in which each item points to its occurrence in the tree via a head of node-link, where the items are listed according to the large 1-itemsets shown in Figure 12. Nodes with the same item-name are linked in sequence via such node-links. After scanning the TDB, the tree with the associated node-links is constructed. Figure 19 shows the sub-tree of the FP-tree shown in Figure 14, where the sub-tree contains g_4, and partially presents the node-links of items, especially for g_4.

Next, we use a similar algorithm with respect to FP-tree (Han et al., 2000) to mine the large itemsets denoted as *conditional pattern bases* (CPB) for each large 1-itemset denoted as *gItem* shown in Figure 15. For any *gItem* g_i, all the possible CPBs that contain g_i can be obtained by following g_i's node-links, starting from g_i's head in the FP-tree header. This is based directly on the construction process of FP-tree. It facilitates the access of all the pattern information related to g_i by traversing the FP-tree once following g_i's node-links.

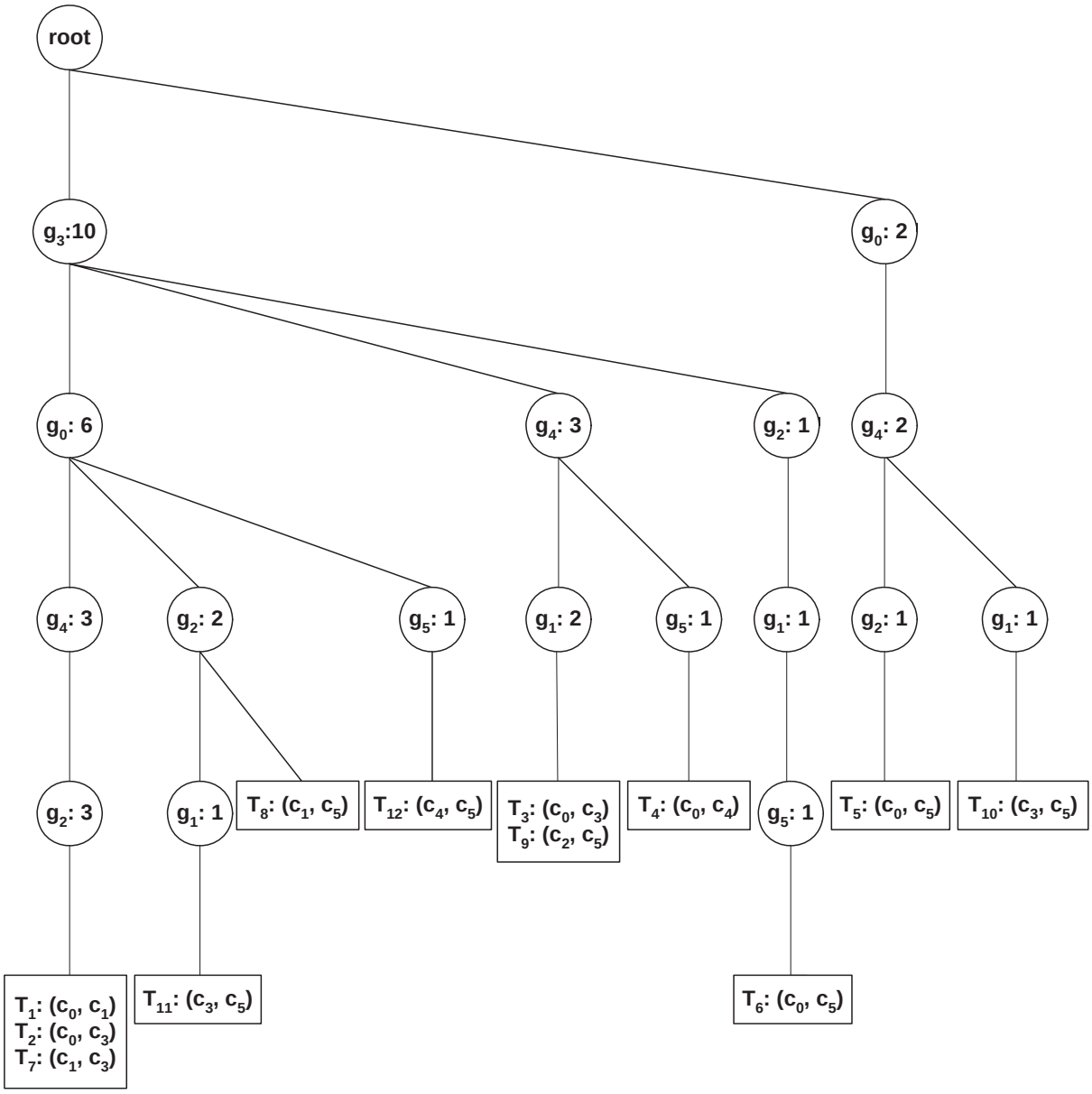


Figure 14: The FP-tree of TDB

gItem	Conditional Pattern Base(CPB)
g_5	$(g_3, g_4) \times (c_0, c_4)$ $(g_3, g_2, g_1) \times (c_0, c_5)$ $(g_3, g_0) \times (c_4, c_5)$
g_1	$(g_3, g_4) \times (c_0, c_3)\&(c_2, c_5)$ $(g_3, g_2) \times (c_0, c_5)$ $(g_0, g_4) \times (c_3, c_5)$ $(g_3, g_0, g_2) \times (c_3, c_5)$
g_2	$(g_3, g_0, g_4) \times (c_0, c_1)\&(c_0, c_3)\&(c_1, c_3)$ $(g_0, g_4) \times (c_0, c_5)$ $(g_3, g_0) \times (c_3, c_5)\&(c_1, c_5)$
g_4	$(g_3, g_0) \times (c_0, c_1)\&(c_0, c_3)\&(c_1, c_3)$

Figure 15: Conditional Pattern Base Table for each $gItem$ (gCPBT)

We collect all the CPBs that a node g_i participates by starting from g_i's head (in the header table) and following g_i's node-links. We examine the mining process by starting from the bottom of the header table. For example, we derive the possible CPBs that contain g_4 by following g_4's node-links, starting from g_4's head in the FP-tree header. The CPBs of g_4 are the collocation of the ancestral nodes of g_4 and the condition-pair of the leaf node in the same path of the FP-tree: $(g_3,g_0)\times(c_0,c_1)\&(c_0,c_3)\&(c_1,c_3)$, $(g_3)\times(c_0,c_3)\&(c_2,c_5)\&(c_0,c_4)$ and $(g_0)\times(c_0,c_5)\&(c_3,c_5)$. Therefore, we only need the CPB whose gene set has the size larger than or equal to $(minG-1)$, since a valid subspace cluster is constructed by the gene set which is large enough. Accordingly, the CPBs of g_4 are $(g_3, g_0) \times (c_0, c_1)\&(c_0, c_3)\&(c_1, c_3)$ shown in Figure 15. We will mine the final subspace clusters based upon the CPBs in the following subsection.

3.2.3 Step 3: Constructing Subspace Clusters

The final step of our algorithm is to construct the subspace clusters by Procedure $FindCluster$ shown in Figure 20, where $C[k]$ means $CandidateSet[k]$ and $L[k]$ means $LargeItemSet[k]$. The flowchart of Procedure $FindCluster$ is shown in Figure 21. According to the condition pattern base table, the CPBs of each $gItem$ are independent with those of the other $gItem$. The process of constructing the eventual subspace clusters from the condition pattern base of each $gItem$ will be described in the following steps:

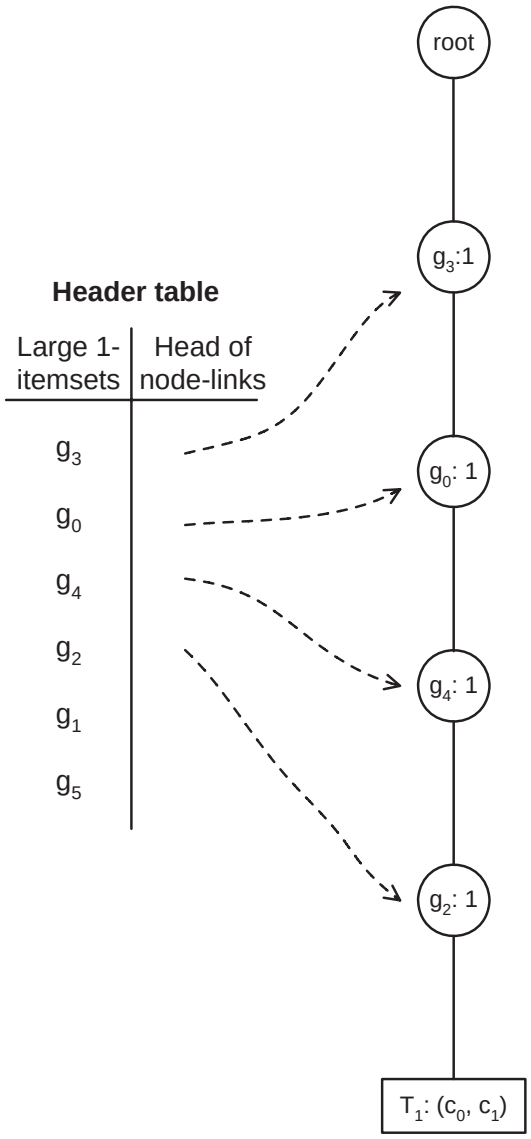


Figure 16: Constructing the FP-tree of T_1: $(g_3, g_0, g_4, g_2) \times (c_0, c_1)$

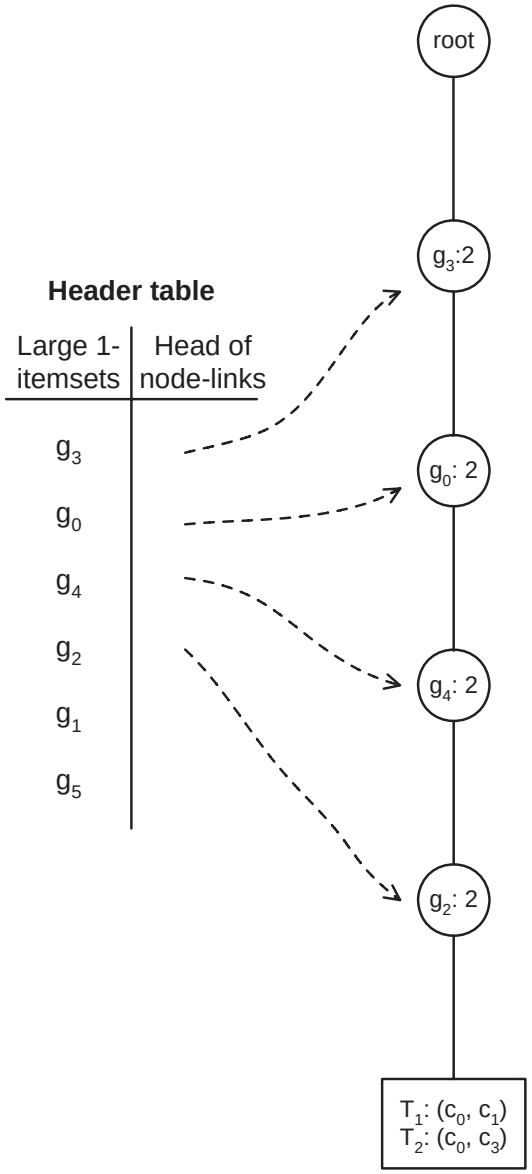


Figure 17: Constructing the FP-tree of T_2: $(g_3, g_0, g_4, g_2) \times (c_0, c_3)$

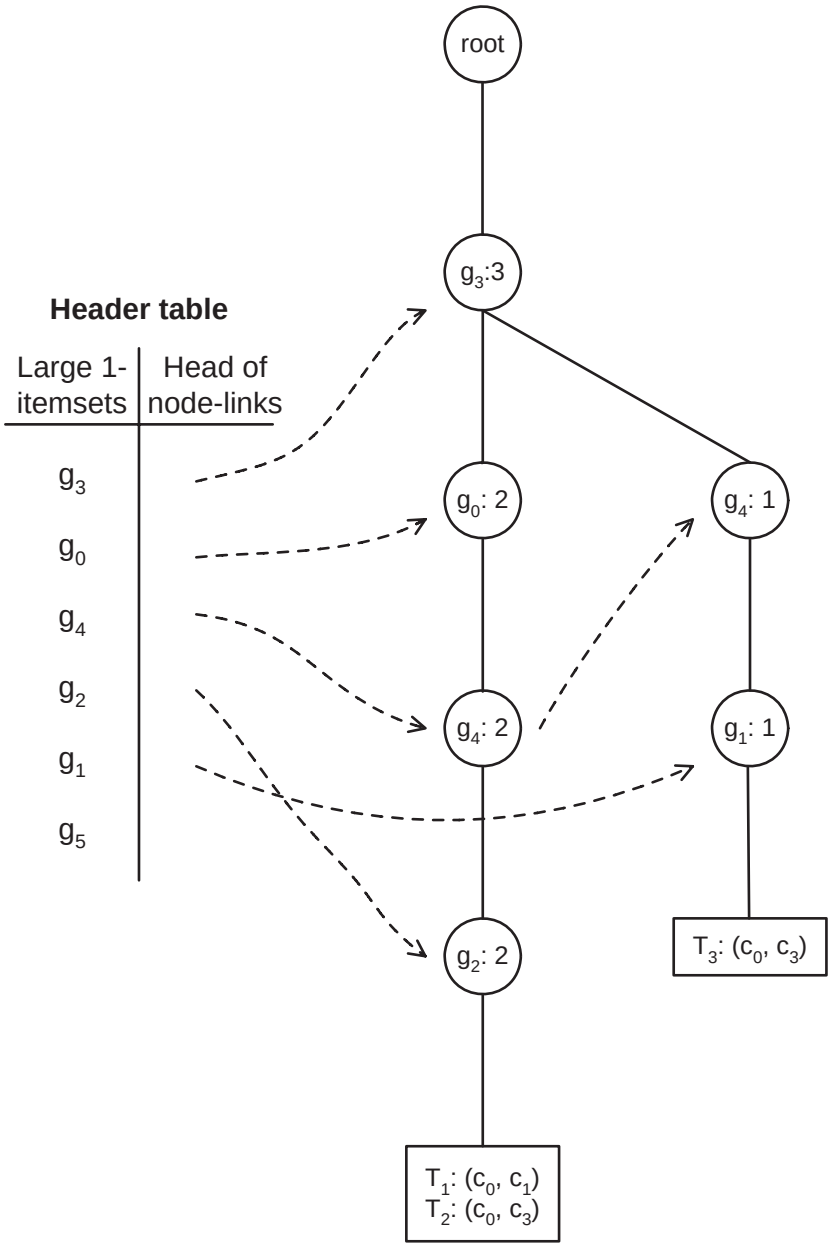


Figure 18: Constructing the FP-tree of T_3: $(g_3, g_4, g_1) \times (c_0, c_3)$

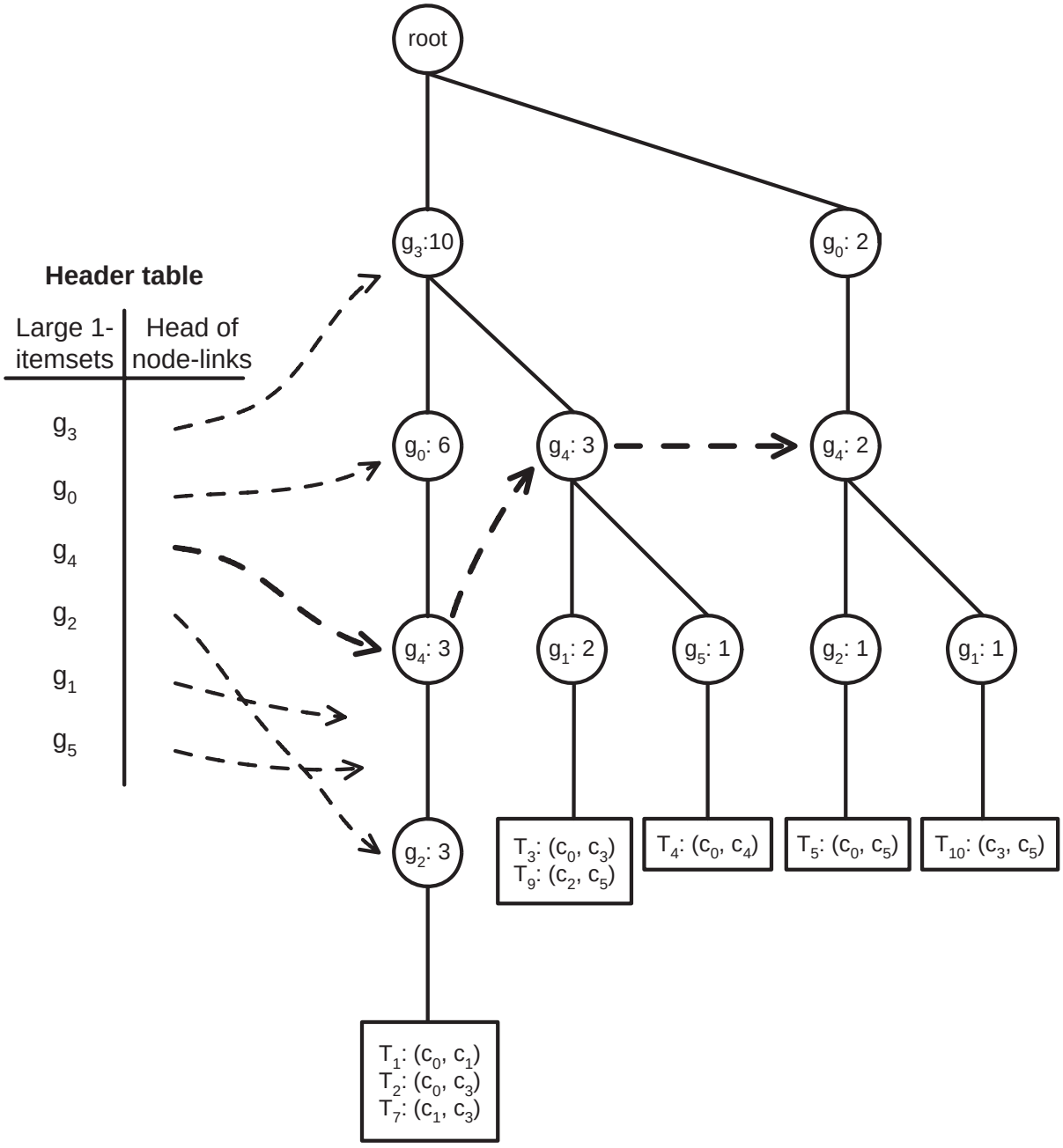


Figure 19: Node-links of the FP-tree

```
1: procedure FindCluster(gCPBT, minG);
2: begin
3:   for each (gItem ∈ gCPBT) do
4:   begin
5:     k := 2;
6:     finalC := ∅;
7:       C[k] := find_all_pairgene_cond_set(gCPBT[gItem]);
8:     if (C[k] = ∅) then break;
9:     else
10:    begin
11:      while (k ≥ 2)
12:      begin
13:        L[k] := CombineCondition(C[k]);
14:        if (L[k] = ∅) then break;
15:        else
16:        begin
17:          if (k ≥ minG − 1) then
18:          begin
19:            if (any subset of L[k] ∈ finalC) then
20:              finalC :=finalC - (the subset of L[k]);
21:            finalC :=finalC ∪ ((gItem ∪ L[k].g) × L[k].c);
22:          end;
23:          k++;
24:          C[k] := enlarge_geneset(L[k − 1]);
25:          if (C[k] = ∅) then break;
26:        end;
27:      end;
28:  end;
29: end;
```

Figure 20: Procedure *FindCluster*

Step (a): Finding all the combination of ($minG$-1)-length gene sets.

Since we are only interested in the subspace clusters with the size of genes $\geq minG$, we take the $(minG-1)$-length gene sets from the CPBs except the $gItem$ to construct the clusters. In the light of the CPBs of each $gItem$, we take the $(minG-1)$-length gene sets as a unit to separate the gene set of each CPB. Therefore, the CPB will be represented as the collocation of the gene-pair and the condition-pair called the $CandidateSet[minG-1]$ on account of the genes with "$(minG-1)$"-length of the combination. Figure 22 shows the process of transforming the CPB to $CandidateSet[2]$ with respect to $g_2 \in gItem$ with $minG = 3$. The dashed line with an arrow means the action of "separation", and the solid line means the "collocation" of the gene-pair and the condition-pair. For example, the gene set (g_3, g_0, g_4) is separated to (g_3, g_0), (g_3, g_4) and (g_0, g_4) collocated with condition sets (c_0, c_1), (c_0, c_3) and (c_1, c_3), respectively. Figure 23 shows $CandidateSet[2]$ of all $gItems$.

Step (b): Constructing LargeItemSet[k] from CandidateSet[k].

We combine the condition set of $CandidateSet[k]$ for each gene set. If all $(i-1)$-subsets of an i-length condition set X exist in $CandidateSet[k]$ for a gene set Y, we combine all the $(i-1)$-subset to form the i-length condition set X, and let the collocation of the gene set Y and the condition set X belong to $CandidateSet[k]$. We repeat the combination until no larger condition set could be created. For example, Figure 24 shows all the condition set of gene set (g_3, g_0) belong to $CandidateSet[2]$ of $g_2 \in gItem$. The heavy solid line with an arrow means the "combination". We can combine the condition sets (c_0, c_1), (c_0, c_3) and (c_1, c_3) to form the condition set (c_0, c_1, c_3), and further combine the condition sets (c_1, c_3), (c_3, c_5) and (c_1, c_5) to form the condition set (c_1, c_3, c_5). However, condition sets (c_0, c_1, c_3) and (c_1, c_3, c_5) can not be combined to a larger set, so the action of combination stops. The gene-condition pairs $(g_3, g_0) \times (c_0, c_1, c_3)$ and $(g_3, g_0) \times (c_1, c_3, c_5)$ are denoted as $LargeItemSet[2]$ of $g_2 \in gItem$. Then let the gene-condition pairs $(g_2, g_3, g_0) \times (c_0, c_1, c_3)$ and $(g_2, g_3, g_0) \times (c_1, c_3, c_5)$ be the subspace clusters. Figure 25 shows the other example of this step. In this example, we can combine the condition sets (c_0, c_1, c_3), (c_0, c_1, c_5), (c_0, c_3, c_5) and (c_1, c_3, c_5) to form the condition set (c_0, c_1, c_3, c_5). The condition set (c_0, c_1, c_4) can not be combined the other condition sets to a larger one, so the action of combination stops. Figure 26 shows $LargeItemSet[2]$ of all $gItems$.

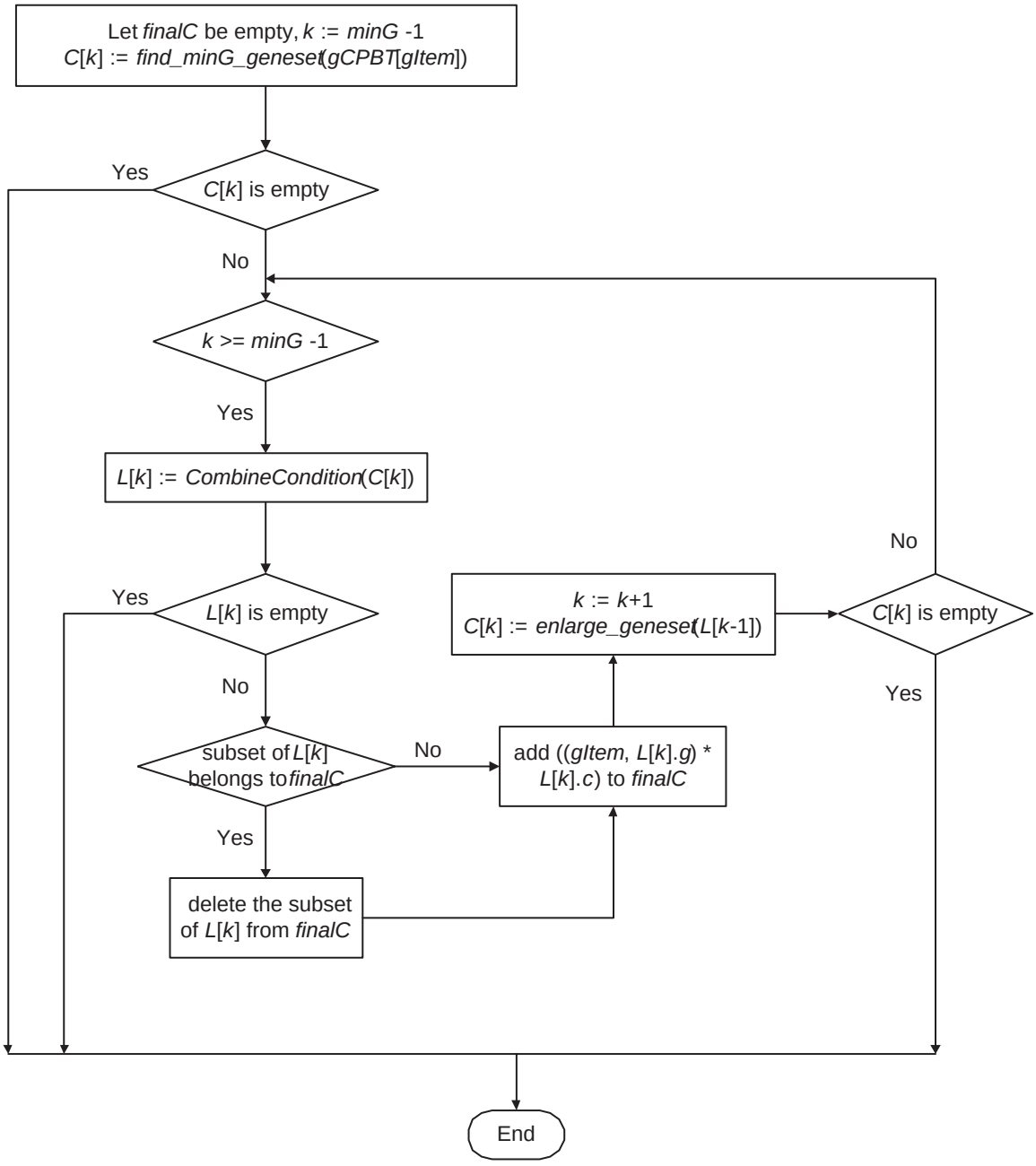


Figure 21: The flowchart of the proposed *FindCluster* algorithm

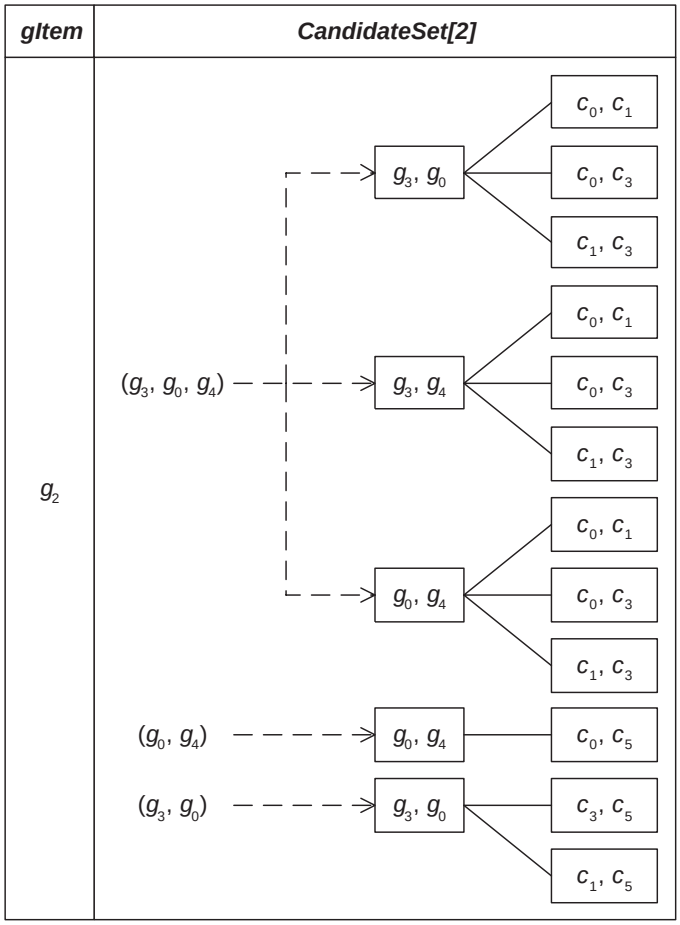


Figure 22: CandidateSet[2] of g_2

gItem	***CandidateSet*[2]**
g_5	$(g_3, g_4) \times (c_0, c_4)$
	$(g_3, g_2) \times (c_0, c_5)$
	$(g_3, g_1) \times (c_0, c_5)$
	$(g_2, g_1) \times (c_0, c_5)$
	$(g_3, g_0) \times (c_4, c_5)$
g_1	$(g_3, g_4) \times (c_0, c_3)$
	$(g_3, g_4) \times (c_2, c_5)$
	$(g_3, g_2) \times (c_0, c_5)$
	$(g_0, g_4) \times (c_3, c_5)$
	$(g_3, g_0) \times (c_3, c_5)$
	$(g_3, g_2) \times (c_3, c_5)$
	$(g_0, g_2) \times (c_3, c_5)$

gItem	***CandidateSet*[2]**
g_2	$(g_3, g_0) \times (c_0, c_1)$
	$(g_3, g_0) \times (c_0, c_3)$
	$(g_3, g_0) \times (c_1, c_3)$
	$(g_3, g_4) \times (c_0, c_1)$
	$(g_3, g_4) \times (c_0, c_3)$
	$(g_3, g_4) \times (c_1, c_3)$
	$(g_0, g_4) \times (c_0, c_1)$
	$(g_0, g_4) \times (c_0, c_3)$
	$(g_0, g_4) \times (c_1, c_3)$
	$(g_0, g_4) \times (c_0, c_5)$
	$(g_3, g_0) \times (c_3, c_5)$
	$(g_3, g_0) \times (c_1, c_5)$

gItem	***CandidateSet*[2]**
g_4	$(g_3, g_0) \times (c_0, c_1)$
	$(g_3, g_0) \times (c_0, c_3)$
	$(g_3, g_0) \times (c_1, c_3)$

Figure 23: $CandidateSet[2]$ of all $gItems$

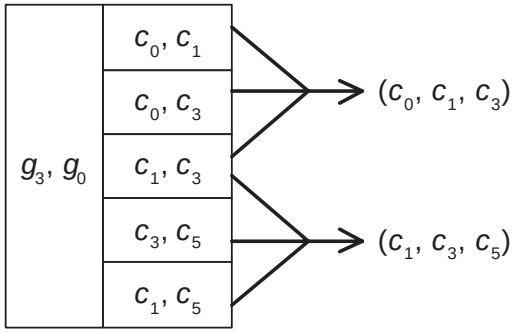


Figure 24: CandidateSet[2]→ LargeItemSet[2]

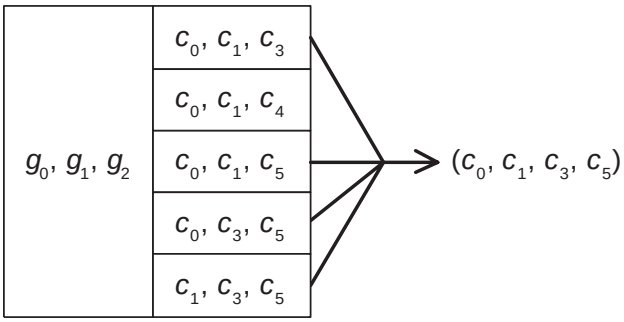


Figure 25: CandidateSet[3]→ LargeItemSet[3]

gItem	***LargeItemSet[2]***
g_2	$(g_3, g_0) \times (c_0, c_1, c_3)$ $(g_3, g_0) \times (c_1, c_3, c_5)$ $(g_3, g_4) \times (c_0, c_1, c_3)$ $(g_0, g_4) \times (c_0, c_1, c_3)$
g_4	$(g_3, g_0) \times (c_0, c_1, c_3)$

Figure 26: LargeItemSet[2] of all gItems

Step (c): Constructing CandidateSet[k+1] from LargeItemSet[k].

This step will be described in Function *enlarge_geneset* shown in Figure 27. We first check all the gene set of $LargeItemSet[k]$ for $x \in gItem$ if they can combine to form a $(k+1)$-length gene set. If we find a $(k+1)$-length gene set J which is denoted as the gene set of $CandidateSet[k+1]$ for $x \in gItem$, we check which conditions of all the k-length gene subsets of J in $LargeItemSet[k]$ occur simultaneously. Those conditions will be denoted as the condition set of J in $CandidateSet[k+1]$. Figure 28 shows how $LargeItemSet[2]$ of $g_2 \in gItem$ is transformed to $CandidateSet[3]$. The heavy solid line with an arrow means the "combination", and the dotted line with an arrow means the "intersection". In this case, we can combine the gene sets (g_3, g_0), (g_3, g_4) and (g_0, g_4) to form the gene set (g_3, g_0, g_4), and we find the condition set (c_0, c_1, c_3) of those gene sets occur simultaneously. Thus, $(g_3, g_0, g_4) \times (c_0, c_1, c_3)$ is denoted as $CandidateSet[3]$.

We repeat Steps (b) and (c) until no more $CandidateSet[j]$ or $LargeItemSet[j]$ could be created. In other words, if any $CandidateSet[j]$ or $LargeItemSet[j]$ does not exist or can not be formed by $LargeItemSet[j-1]$ or $CandidateSet[j]$, respectively, the algorithm will stop. The subspace cluster which we found within the process of Step (b) are the final solution. If any subset of $LargeItemSet[j]$ is already determined as the final cluster previously, we remove the subset from the result, and output $LargeItemSet[j]$ as the final cluster.

```
1: function enlarge_geneset(L[k]: gene-condition pair);
2: begin
3:    t := 0;
4:    for each (L[k][x] of L[k]) do
5:    begin
6:       for each (L[k][y] of L[k]) do
7:       begin
8:          i := 0;
9:          while (i ≥ 0)
10:         begin
11:            if (L[k][x].g[i] = L[k][y].g[i]) then i++;
12:            else break;
13:         end;
14:         if (i = k − 1) then C[k + 1][t].g := L[k][x].g ∪ L[k][y].g[k − 1]; t++;
15:         if (any k-subset of C[k + 1][t].g ∉ L[k].g) then t−−;
16:         else
17:            C[k + 1][t − 1].c := ⋂ condition-set of all k-subset of C[k + 1][t].g in L[k];
18:      end;
19:   end;
20:   return C[k + 1];
21: end;
```

Figure 27: Function *enlarge_geneset*

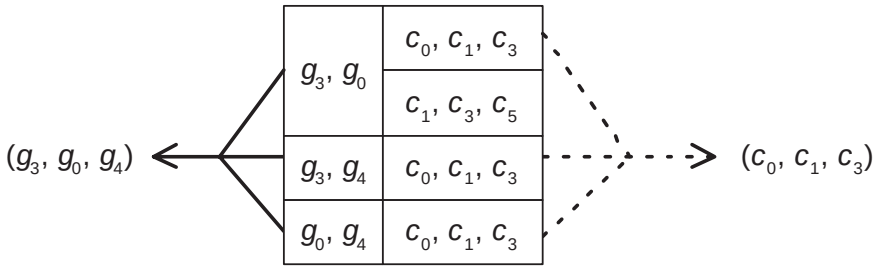


Figure 28: LargeItemSet[2]→ CandidateSet[3]

4 Performance

In this section, we study the performance of our LISC algorithm. The algorithm is implemented on an Intel Pentium 4 machine with a 1.60 GHz CPU, 768 MB of main memory, and running under Windows XP Professional Edition. All experiments are implemented in Java and compiled by JDK 1.5.0.

4.1 Data Sets

We experiment our LISC algorithm with the synthetic data and the real life data set. First, we start our experiments with the synthetic data sets to validate the correctness of our algorithm. In addition, the synthetic data sets can serve as convenient benchmarks to compare different algorithms. We generate these synthetic data sets in matrix forms by the algorithm introduced in (Wang et al., 2002). Initially, the matrix is filled with random values ranged from 0 to 500, and then we embed a fixed number of the subspace clusters in the raw data. Besides the size of the matrix, the data generator takes several other parameters: (1) $minG$, the average number of genes of the embedded subspace clusters, (2) $minC$, the average number of conditions, and (3) k, the number of the subspace clusters embedded in the matrix. To make the generator algorithm easy to implement, and without loss of generality, we embed the *perfect* subspace clusters in the matrix, *i.e.*, each embedded subspace cluster satisfies a cluster threshold $\delta = 0$. We investigate the performance of our LISC algorithm using the synthetic data.

Next, we investigate the performance of our algorithm with several combination of $\delta, minG$ and $minC$ from the yeast gene expression data (Tavazoie et al., 2000), leukemia dataset (Golub et al., 1999) and breast cancer data (West et al., 2001) by our algorithm and the other techniques. The gene expression data is generated by DNA chips and other microarray techniques. The detailed information about these datasets is shown in Figure 29, where G is the number of genes and C is the number of conditions.

The yeast microarray is obtained from the yeast *Saccharomyces cerevisiae* cell cycle expression levels. 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

Dataset	G	C
yeast gene expression data	2884	17
leukemia dataset	5000	38
breast cancer data	7129	49

Figure 29: Description of real datasets

entry represents the relative abundance of the mRNA of a gene under a specific condition. Biologists are interested in the finding of a subset of genes showing strikingly similar up-regulation and down-regulation under a subset of conditions (Cheng and Church, 2000). The leukemia dataset uses the bone marrow samples taken from 27 patients suffering from acute lymphoblastic leukemia (ALL) and 11 patients suffering from acute myeloid leukemia (AML) and analyzed using Affymetrix arrays. We wish to identify the genes that are up-regulated or down-regulated in ALL relative to AML (*i.e.*, to see if a gene is differentially expressed between the two groups). The samples used in the breast cancer data are taken from 49 breast cancer patients, before and after a course of doxorubicin chemotherapy, and analyzed using microarray. There are two measurements from each patient, one before treatment and one after treatment. These two measurements relate to one another. We are interested in the difference between the two measurements to determine whether a gene has been up-regulated or down-regulated in breast cancer following doxoruicin chemotherapy.

Our algorithm and the pCluster algorithm (Wang et al., 2002) do not take as input the exact number of the subspace clusters to generate. Thus, we run these algorithms multiple times with different values of δ, $minG$ and $minC$ to find the subspace clusters.

Both of the pCluster and the zCluster algorithms contain three steps to mine the subspace clusters: (1) generating the gene-pair MDSs, (2) generating the condition-pair MDSs, and (3) constructing the tree to find the final clusters. We will compare the total processing time of our proposed LISC algorithm with the processing time of the first step of the pCluster and the zCluster algorithms, which is generating gene-pair MDSs.

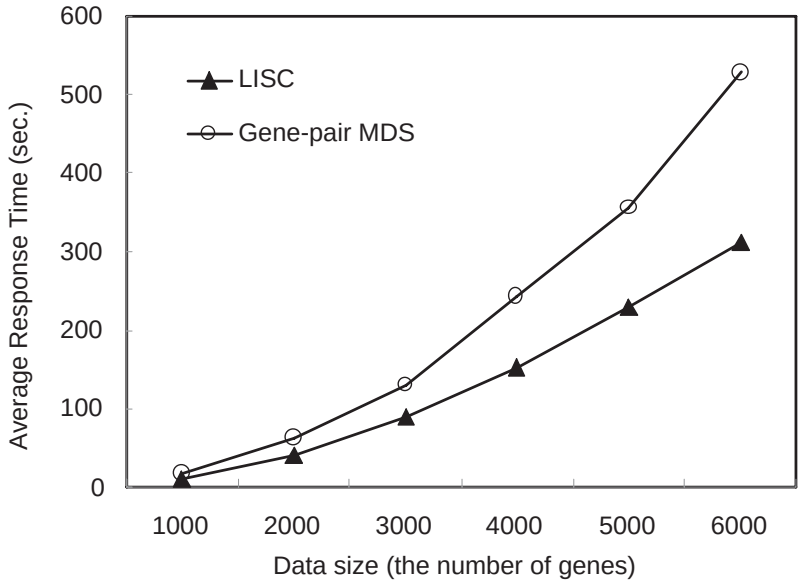


Figure 30: A comparison of the response time under different number of genes in data sets

4.2 Experiment Result

In this subsection, we show the experiment result of the synthetic data and the real life data set.

4.2.1 Synthetic Data

In this subsection, we show the experiment results using the synthetic data. We evaluate the performance of the LISC algorithm as we increase the number of genes and conditions in the dataset. The results presented in Figure 30 and Figure 31 are the average response time obtained from a set of 10 synthetic data.

Figure 30 shows the comparison of the average response time between the LISC algorithm and the step of generating gene-pair MDSs. The pCluster and the zCluster algorithms equally use the step of generating the gene-pair MDSs. Data sets used for the comparison are generated with number of conditions fixed at 30. There is a total of 30 embedded clusters in the data. The minimal number of conditions is 5, and the minimal number of genes is set to $0.01N$, where N is the number of genes of the synthetic data. The mining algorithm is invoked with $\delta = 0$. From this Figure, we show that the LISC algorithm always

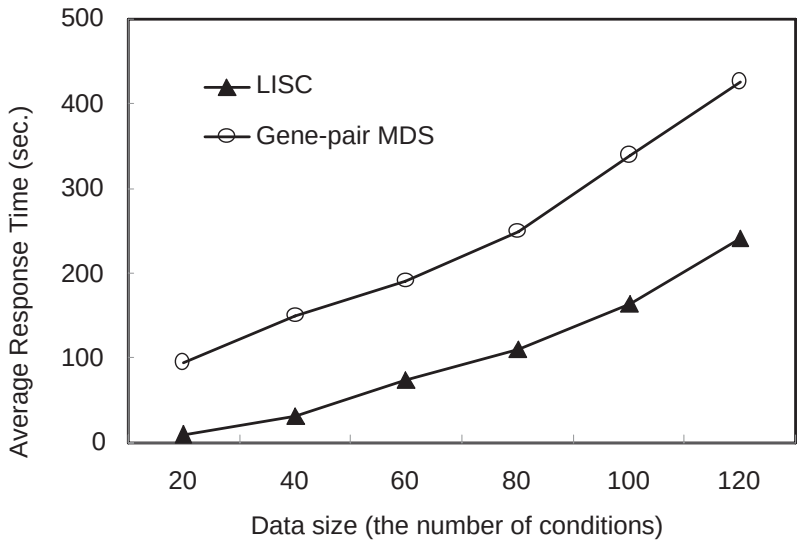


Figure 31: A comparison of the response time under different number of conditions in data sets

requires less time than the step of generating gene-pair MDSs no matter what value of the number of genes is. Moreover, as the number of the genes is increased, the difference of the average response time is increased. The reason is that this step of measuring the difference of each gene-pair on the conditions of a DNA microarray is really time-consuming, since the number of genes is usually very large, about 10^3 to 10^4, and it is expected to reach to the order of 10^6. Our algorithm avoids constructing the gene-pair MDSs. Therefore, our algorithm is more efficient than the algorithm.

Data sets used in Figure 31 are generated in the same manner, except that the number of genes is fixed at 3,000. The mining algorithm is invoked with $\delta = 0$, $ming = 30$, $minc = 0.2C$, where C is the number of conditions of the data set. From this Figure, we show that the LISC algorithm always requires less time than the step of generating gene-pair MDSs no matter what value of the number of conditions is. The number of conditions in a real microarray data is usually less than 100. Even though we let the number of conditions larger than 100, our algorithm is more efficient than the step of generating gene-pair MDSs. Moreover, as the number of the conditions is increased, the difference of the average response time between our algorithm and the step of generating gene-pair MDSs is increased.

	Step 1	Step 2	Step 3
Percentage	5	15	80

Figure 32: The percentage of time spent in each step of LISC

We now evaluate the percentage of the processing time of each step in our proposed LISC algorithm. In Step 1, we generate the MDSs for each condition-pair. Then, we find the large item set using the revised version of FP-tree in Step 2. Finally, we develop an algorithm to construct the final clusters from the gene set and the condition-pair after searching the FP-tree in Step 3. The size of data sets in use is $5,000 \times 30$. Figure 32 shows the result. The percentage of time spent in Step 3 is much longer than that in other steps.

4.2.2 Real Microarray Datasets

We apply the LISC algorithm on the yeast gene microarray (Tavazoie et al., 2000), the leukemia dataset (Golub et al., 1999) and the breast cancer data (West et al., 2001). We compare the total processing time of our proposed LISC algorithm with the processing time of the first step of the pCluster and the zCluster algorithms, which is generating gene-pair MDSs. We experiment our algorithm in several cases with different combination of the parameters δ, $minG$ and $minC$ in each real dataset shown in Figure 33. The results of experimenting our algorithm in these cases and generating gene-pair MDSs in the yeast gene microarray, leukemia dataset and breast cancer data are shown in Figure 34, Figure 35 and Figure 36, respectively. From our simulation results, our algorithm has performance advantage over the the pCluster and the zCluster algorithms, as the processing time spent in their first step is longer than our total processing time.

Case	Real Dataset	*delta*	*minG*	*minC*
1	yeast microarray	1	30	5
2		2	40	6
3		2	40	7
4	leukemia dataset	0	50	10
5		1	50	10
6		2	50	10
7	breast cancer data	0	70	15
8		1	70	15
9		2	70	15

Figure 33: Several cases in each real dataset

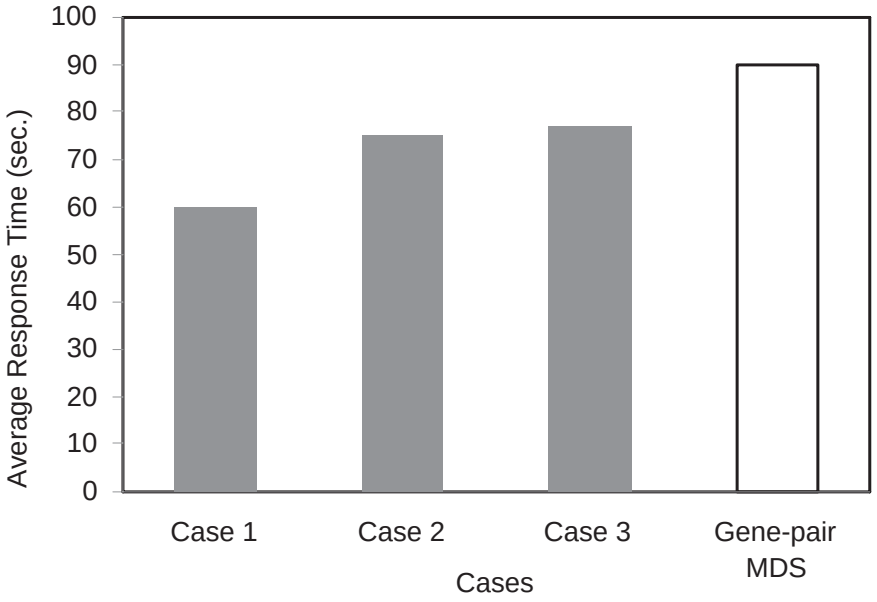


Figure 34: A comparison of the response time under different cases in the yeast gene microarray and the step of generating gene-pair MDSs

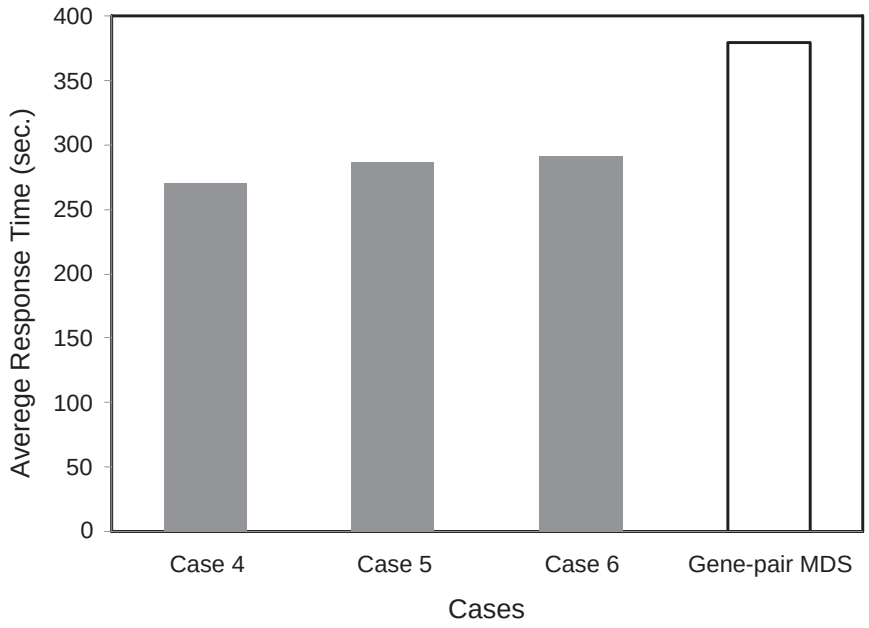


Figure 35: A comparison of the response time under different cases in the leukemia dataset and the step of generating gene-pair MDSs

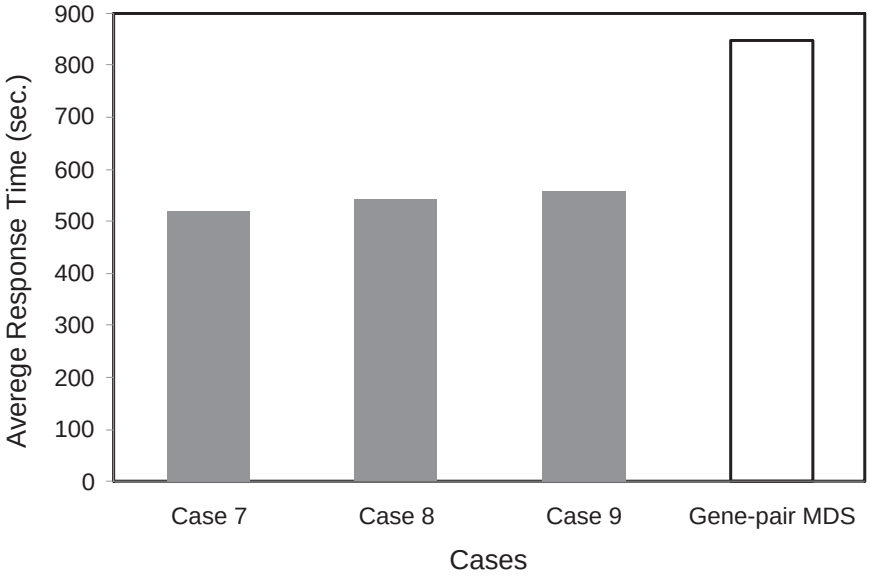


Figure 36: A comparison of the response time under different cases in the breast cancer data and the step of generating gene-pair MDSs

5 Conclusion

DNA Microarrays are one of the latest breakthroughs in experimental molecular biology and have opened the possibility of creating datasets of molecular information to represent many systems of biological or clinical interest. Clustering techniques have been proven to be helpful to understand gene function, gene regulation, cellular processes, and subtypes of cells. In this paper, we have proposed the LISC algorithm to mine the subspace clusters from the microarray data. In the proposed algorithm, we consider only the condition-pair MDSs, instead of constructing the gene-pair MDSs. Moreover, we transform the task of mining the possible maximal gene sets into the problem of mining large itemsets from the condition-pair MDSs. We efficiently utilize a revised version of the FP-tree structure to find the frequent gene set from the condition-pair MDSs. In our performance study, we have compared the total processing time of the proposed algorithm with the processing time of the first step of the pCluster (Wang et al., 2002) and the zCluster algorithms (Yoon et al., 2005), which generate gene-pair MDSs. We have experimented the algorithm with both synthetic and real life data sets. From our simulation results, we have shown that our proposed algorithm is more efficient than those previous proposed algorithms.

Acknowledgments

This research was supported in part by the National Science Council of Republic of China under Grant No. NSC-95-2221-E-110-079-MY2. The authors also like to thank "Aim for Top University Plan" project of NSYSU and Ministry of Education, Taiwan, for partially supporting the research.

References

Aggarwal, C. C., Procopiuc, C., Wolf, J. L., Yu, P. S., Park, J. S., 1999. Fast Algorithms for Projected Clustering. Proc. of ACM SIGMOD Conf. on Management of Data, 61–72.

Aggarwal, C. C., Yu, P. S., 2000. Finding Generalized Projected Clusters in High Dimensional Spaces. ACM SIGMOD Record 29 (2), 70–81.

Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P., Park, J. S., 1998. Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications. Proc. of ACM SIGMOD Conf. on Management of Data, 94–105.

Brown, P. O., Botstein, D., Jan. 1999. Exploring the New World of the Genome with DNA Microarrays. Nature Genetics 21 (1), 33–37.

Chang, Y. I., Chen, J. R., Lee, L. W., 2007. An Efficient Union Approach to Mining Closed Large Itemsets in DNA Microarray Datasets. Proc. of the Int. Medical Informatics Symp.

Cheng, C. H., Fu, A. W., Zhang, Y., 1999. Entropy-based Subspace Clustering for Mining Numerical Data. Proc. of ACM SIGMOD Conf. on Knowledge Discovery and Data Mining, 84–93.

Cheng, Y., Church, G. M., 2000. Biclustering of Expression Data. Proc. of the 8th Int. Conf. on Intelligent System for Molecular Biology, 93–103.

Ester, M., Kriegel, H., Sander, J., Xu, X., 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Proc. of the 2nd Int. Conf. on Knowledge Discovery and Data Mining, 226–231.

Golub, T. R., Slonim, D. K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J. P., Coller, H., Loh, M. L., Downing, J. R., Caligiuri, M. A., Bloomfield, C. D., Lander, E. S., 1999. Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring. Science 286 (5439), 531–537.

Hakamada, K., Okamoto, M., Hanai, T., Jan. 2006. Novel Technique for Preprocessing High Dimensional Time-course Data from DNA Microarray: Mathematical Model-based Clustering. Bioinformatics 22 (7), 843–848.

Han, J., Pei, J., Yin, Y., 2000. Mining Frequent Patterns without Candidate Generation. Proc. of ACM SIGMOD Int. Conf. on Management of Data, 1–12.

Jiang, D., Pei, J., Zhang, A., 2005. A General Approach to Mining Quality Pattern-based Clusters from Microarray Data. Proc. of the 10th Int. Conf. on Database Systems for Advanced Applications, 188–200.

Jiang, D., Tang, C., Zhang, A., Nov. 2004. Cluster Analysis for Gene Expression Data: A Survey. IEEE Trans. on Knowledge and Data Eng. 16 (11), 1370–1386.

Koo, J. Y., Sohn, I., Kim, S., Lee, J. W., Feb. 2006. Structured Polychotomous Machine Diagnosis of Multiple Cancer Types Using Gene Expression. Bioinformatics 22 (8), 950–958.

Lazzeroni, L., Owen, A., Jan. 2002. Plaid Models for Gene Expression Data. Statistica Sinica 12 (1), 61–86.

Liu, X., Wang, L., 2007. Computing the Maximum Similarity Bi-clusters of Gene Expression Data. Bioinformatics 23 (1), 50–56.

Madeira, S. C., Oliveira, A. L., Jan. 2004. Biclustering Algorithms for Biological Data Analysis: A Survey. IEEE/ACM Trans. on Computational Biology and Bioinformatics 1 (1), 24–45.

Minato, S., 1993. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. Proc. of IEEE/ACM Design Automation Conf., 272–277.

Pei, J., Zhang, X., Cho, M., Wang, H., Yu, P. S., 2003. Maple: A Fast Algorithm for Maximal Pattern-based Clustering. Proc. of the 3rd IEEE Int. Conf. on Data Mining, 259.

Sultan, M., Wigle, D. A., Cumbaa, C. A., Maziarz, M., Glasgow, J., Tsao, M. S., Jurisica, I., 2002. Binary Tree-Structured Vector Quantization Approach to Clustering and Visualizing Microarray Data. Bioinformatics 18, 111–119.

Tavazoie, S., Hughes, J., Campbell, M., Cho, R., Church, G., 2000. Yeast Micro Data Set. http://arep.med.harvard.edu/biclustering/yeast.matrix.

Tefferi, A., Bolander, M. E., Ansell, S. M., Wieben, E. D., Spelsberg, T. C., Sept. 2002. Primer on Medical Genomics Part III: Microarray Experiments and Data Analysis. Mayo Clinic Proc. 77 (9), 927–940.

Wang, H., Chu, F., Fan, W., Yu, P. S., Pei, J., 2004. A Fast Algorithm for Subspace Clustering by Pattern Similarity. Proc. of the 16th Int. Conf. on Scientific and Statistical Database Management, 51–60.

Wang, H., Wang, W., Yang, J., Yu, P. S., 2002. Clustering by Pattern Similarity in Large Data Sets. Proc. of ACM SIGMOD Int. Conf. on Management of Data, 394–405.

West, M., Blanchette, C., Dressman, H., Huang, E., Ishida, S., Spang, R., Zuzan, H., Marks, J. R., Nevins, J. R., 2001. Predicting the Clinical Status of Human Breast Cancer Using Gene Expression Profiles. Proc. of the National Academy of Science, 11462–11467.

Yang, E., Foteinou, P. T., King, K. R., Yarmush, M. L., Androulakis, I. P., Sept. 2007. A Novel Non-overlapping Bi-clustering Algorithm for Network Generation Using Living Cell Array Data. Bioinformatics 23 (17), 2306–2313.

Yang, J., Wang, H., Wang, W., Yu, P. S., 2003. Enhanced Biclustering on Expression Data. Proc. of the 3rd IEEE Int. Symposium on BioInformatics and BioEngineering, 321–327.

Yang, J., Wang, W., Wang, H., Yu, P. S., 2002. δ-Clusters: Capturing Subspace Correlation in a Large Data Set. Proc. of the 18th Int. Conf. on Data Eng. , 517–528.

Yoon, S., Nardini, C., Benini, L., Micheli, G. D., Oct. 2005. Discovering Coherent Biclusters from Gene Expression Data Using Zero-Suppressed Binary Decision Diagrams. IEEE/ACM Trans. on Computational Biology and Bioinformatic 2 (4), 339–354.

Zhang, T., Ramakrishnan, R., Livny, M., 1996. BIRCH: An Efficient Data Clustering Method for Very Large Databases. Proc. of ACM SIGMOD Int. Conf. on Management of Data, 103–114.