Flexible Clustering by Tendency in High Dimensional Space

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

Clustering is the process of grouping a set of objects into classes of similar objects. Until recently, the concept of similarity is based on distances, e.g Euclidean distance and cosine distance. Our previous work on δ-cluster and δ-pCluster designed new similarity models to capture subspace coherency exhibited in data and focused on shifting patterns or scaling patterns. Along the same general direction, we propose a more flexible yet powerful clustering model, namely u-Cluster (Up-pattern Cluster). Under this model, two objects are similar in a subset of dimensions if there exist a permutation of these dimensions, along which both objects exhibit a consistent 'up' pattern. For instance, in DNA microarray analysis, the expression levels of two genes can rise synchronously in response to a sequence of environment stimuli. Although the magnitude of their expression levels might not be close and the amount by which they rise might not be equivalent, the 'up' patterns that they exhibit can be consistent. Discovery of such clusters of genes is essential in revealing significant connections in gene regulatory networks. In addition, E-Commerce applications such as collaborative filtering and stock analysis can also benefit from this model for identifying customer groups that have consistent trends in interests or activities (purchasing, browsing, etc). We also devise an efficient algorithm that takes advantage of fast sequential pattern mining to detect such clusters. Its efficiency and effectiveness have been demonstrated through experiments on several real data sets.

1. INTRODUCTION

Modern technology provides efficient methods for data collection. However, raw data is rarely of direct benefit for higher level analysis. As a method to aim the analysis of large databases, clustering has been studied extensively in many areas including statistics, machine learning and pattern recognition.

Recent efforts in data mining have focused on methods for efficient and effective cluster analysis in large databases. Much active research has been devoted to improve the scalability of clustering methods and especially to attack the curse of dimensionality. Clustering in high dimensional spaces is often problematic as theoretical results questioned the meaning of closest matching in high dimensional spaces. Recent work has focused on discovering clusters embedded in some subspace of the high-dimensional space. This problem is known as *subspace clustering*.

Most clustering models, including those proposed for subspace clustering, define similarities among objects via some distance functions. Some well-known distance functions include Euclidean distance, Manhattan distance, and cosine distance. However, distance functions are not always adequate in capturing correlations among objects. In fact, strong correlations may still exist among a set of objects even if they are far apart from each other if measured by some distance function. In light of this observation, the δ-pCluster model [22] was introduced to discover clusters by pattern similarity (rather than distance) from raw data sets. A major limitation of the δ-pCluster model is that it only considers either strict shifting patterns or strict scaling patterns[1], which is still insufficient in many cases. In this paper, we investigate a more flexible clustering model, u-Cluster, which is able to capture the general tendency of objects across a subset of dimensions in a high dimensional space.

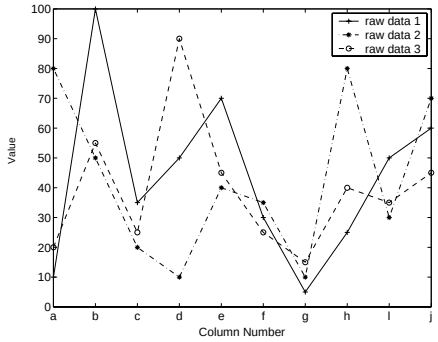


Figure 1: Raw data: 3 objects and 10 columns

Figure 1 shows a set of 3 objects with 10 columns (attributes). No obvious pattern is visible. If we pick the set of columns $\{b, c, e, g, l\}$ as in Figure 2, we can observe the following tendency: *the values of these columns obey the same relative order*. This tendency can be observed clearly if we rearrange the columns in ascending order of their values: g, c, l, e, b. All three objects exhibit consistent 'up' pattern as showed in Figure 3. However, this type of patterns may be revealed neither by any traditional distance-based clustering model nor by the δ-pCluster model, because of the restricted definition of similarity employed by this model. In the δ-pCluster model, object 3 might be identified as an outlier to the cluster containing objects 1 and 2 because it cannot be represented via shifting patterns (or scaling patterns).

1.1 Applications

[1]The scaling patterns can be transformed into shifting patterns by applying a logarithmic function on the raw data.

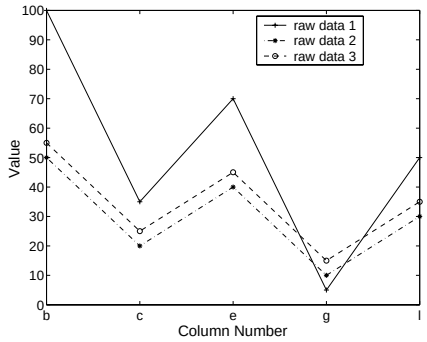


Figure 2: A shifting pattern and an outlier by δ-pCluster

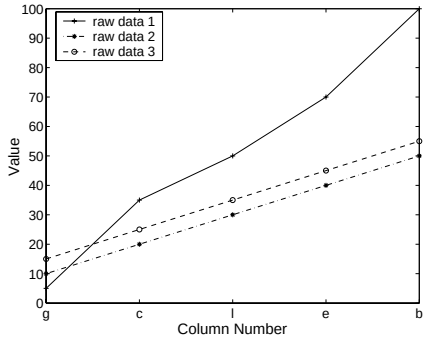


Figure 3: The "outlier" shows consistent "up" pattern

Discovery of clusters in data sets based on tendencies is of great importance because of its potential for actionable insights.

- *DNA microarray analysis.* Micro-array is one of the latest breakthroughs in experimental molecular biology. It provides a powerful tool by which the expression patterns of thousands of genes can be monitored simultaneously and is already producing huge amount of valuable data. Analysis of such data is becoming one of the major bottlenecks in the utilization of the technology. The gene expression data are organized as matrixes–tables where rows represent genes, columns represent various samples such as tissues or experimental conditions, and numbers in each cell characterize the expression level of the particular gene in the particular sample. Investigators show that more often than not, if several genes contribute to a disease, it is possible to identify a subset of conditions, under which these genes show a coherent tendency. Since a gene's expression level may vary substantially due to its sensitivity to systematic settings, the direction of movement (up or down) in response to condition change is often considered more important than its actual value. Discovering clusters of genes sharing coherent tendency is essential in revealing the significant connections in gene regulatory networks[8]

- *E-commerce*. Recommendation systems and target marketing are important applications in the E-commerce area. In these applications, sets of customers/clients with similar behavior need to be identified so that we can predict customers' interest and make proper recommendation for future marketing. The following is an example. Three viewers rate four movies ("DareDevil", "The hours", "Chicago", "Lord of rings, the two towers") as (1,4,7,10), (5,6,7,8) and (3,4,9,10), where 1 is the lowest and 10 is the highest. Although the rates given by each viewer differ in both their values and their scales it is clear that they have coherent affinities. Therefore, the relative order of the ratings play a more important role than the absolute value of each rating. In the future, if the first and second viewers rate two movies as (2,6) and (4,5), respectively, we may have certain confidence that the third viewer may also favor the second movie over the first one.

1.2 Challenges

This observation motivated us to propose a more flexible model to characterize the general tendency and to discover clusters of objects that preserve coherent tendencies. To achieve this goal, we need to tackle two major challenges.

- *Huge number of potential tendencies.* If we have N attributes, there are $N!$ different permutations of (subsets of) attributes, each of which may exhibit some tendency. Data sets used in DNA array analysis or collaborative filtering can have hundreds of attributes. This results in a huge number of candidates of various lengths, which poses a significant challenge to the pattern discovery.

- *Massive Cluster Size.* Compared with δ-pCluster, which has a more restrictive similarity function, the clusters under our model tend to have a much larger size along the directions of both attributes and objects. As a result, scalability with respect to the size of cluster (rather than the volume of the data) becomes more important than ever.

1.3 Our Contributions

Our objective is to cluster objects that exhibit same tendency along a subset of dimensions, which includes the δ-pCluster model as a special case. Our contributions include that:

- We propose a new clustering model, namely u-Cluster, to capture general tendencies exhibited by the objects.

- The u-Cluster model is a generalization of existing subspace clustering models. It has a wide variety of applications, including DNA array analysis and collaborative filtering, where tendency along a set of attributes carries significant meaning.

- In addition to utilizing the techniques of sequential pattern mining, we also design an efficient and effective tree structure UPC-Tree to mine u-Cluster. Compared with one of fastest sequential pattern mining algorithms, prefixSpan, the UPC-Tree based algorithm delivers a shorter response time in most cases, especially when the data is pattern-rich.

1.4 Paper Layout

The remainder of the paper is organized as follows. Section 2 discusses some related work. Section 3 defines the model proposed in the paper. Section 4 presents the two algorithms in detail. An extensive performance study is reported in Section 5. Section 6 concludes the paper.

2. RELATED WORK

2.1 Subspace Clustering

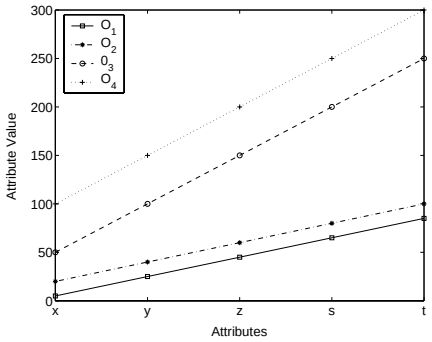
Clustering in high dimensional space is often problematic as theoretical results [7] questioned the meaning of closest matching in high dimensional spaces. Recent research work [23, 24, 3, 4, 6, 8, 14] has focused on discovering clusters embedded in the subspaces of a high dimensional data set. This problem is known as subspace clustering. Based on the measure of similarity, there are two categories of clustering model.

- *Distance based clustering*. In this category, one of the well known subspace clustering algorithms is CLIQUE [6]. CLIQUE is a density and grid based clustering method. It discretizes the data space into non-overlapping rectangular cells by partitioning each dimension to a fixed number of bins of equal length. A bin is dense if the fraction of total data points contained in the bin is greater than a threshold. The algorithm finds dense cells in lower dimensional spaces and merge them to form clusters in higher dimensional spaces. The PROCLUS [3] and the ORCLUS [4] algorithms find projected clusters based on representative cluster centers in a set of cluster dimensions. Another interesting approach, Fascicles [14], finds subsets of data that share similar values in a subset of dimensions.

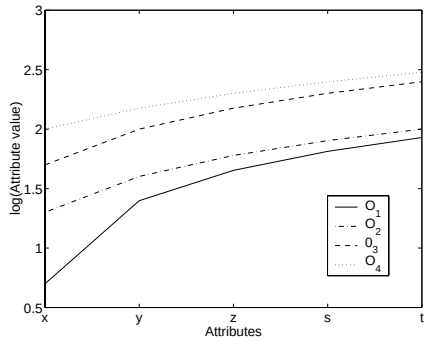
- *Pattern similarity based clustering*. The first algorithm proposed in this category is bicluster model [9] by Cheng et al. The algorithm tries to measure the coherence of the genes and the conditions in a sub-matrix of a DNA array. Yang et al [23] proposed a move-based algorithm to find biclusters in a more efficient way. However, as pointed out in [22], the bicluster model has two drawbacks: (1) A bicluster may contain some true outliers. (2) It requires the number of clusters as an input parameter. Recently, δ-pCluster is introduced by Wang et al [22] to cluster objects exhibiting shifting patterns in a data set in a very efficient way. Let $\mathcal{O}$ be a subset of objects in the database ($\mathcal{O} \subseteq \mathcal{D}$), and let $\mathcal{T}$ be a subset of attributes ($\mathcal{T} \subseteq \mathcal{A}$). The pair $(\mathcal{O}, \mathcal{T})$ specifies a sub-matrix. Given $x, y \in \mathcal{O}$, and $a, b \in \mathcal{T}$, $pScore$ of the 2×2 matrix is defined as:

$$pScore\left(\begin{bmatrix} d_{xa} & d_{xb} \\ d_{ya} & d_{yb} \end{bmatrix}\right) = |(d_{xa} - d_{xb}) - (d_{ya} - d_{yb})| \quad (1)$$

$(\mathcal{O}, \mathcal{T})$ forms a pCluster if, for any 2×2 submatrix X in $(\mathcal{O}, \mathcal{T})$, $pScore$ is less than some threshold δ^p.



(a) Original Data



(b) Applying a logarithmic transformation to the original data

Figure 4: Shifting and scaling patterns

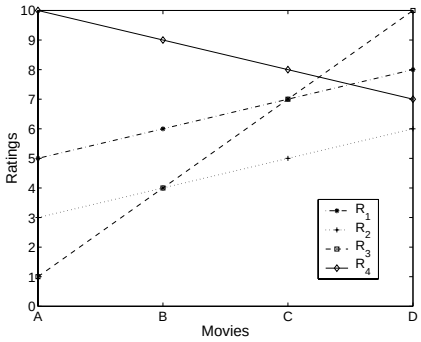


Figure 5: Four movie ratings for movies A, B, C, D

There are several limitations of the δ-pCluster model. In many applications, only allowing shifting or scaling patterns is too restrictive. Figure 4 (a) shows four objects O_1, O_2, O_3 and O_4. Clustering by shifting patterns, O_1 and O_2 can form a δ-pCluster and O_3 and O_4 forms another. However, if we consider the scaling pattern, Figure 4 (b) shows that O_2 and O_3 will be in a cluster without O_1 and O_3. Obviously, by restricting itself to simple shifting or scaling patterns, δ-pCluster may miss many more valuable clusters. This limitation is inherent to the model of δ-pCluster, which cannot be solved through relaxing the threshold δ^p. For example, in Figure 5, three dotted lines represent ratings given by three individuals for four different movies$\{A, B, C, D\}$. They are R_1:$[5, 6, 7, 8]$, R_2:$[3, 4, 5, 6]$ and R_3:$[1, 4, 5, 10]$. According to the definition of $pScore$, when $\delta^p = 1$, R_1 and R_2 form a shifting pattern with $pScore = 0$. But R_3 is not part of the pattern ($pScore > 1$) even though R_3 shares the same tendency (with a sharper slope). Most interestingly, they share consistent opinions by ranking the movies in the order $[D, B, C, A]$. This suggests that, even though people may provide different rates to a movie, the order or tendency of the rates plays a more important role than the actual rates. In this example, if we want to group R_1, R_2 and R_3 as a cluster, we need to increase the threshold to 6. However, this may include $R4$:$[10, 9, 8, 7]$ ($pScore = 6$) in the δ-pCluster, even if $R4$ shows a tendency completely opposite to the other three.

2.2 Sequential Pattern Mining

Since it was first introduced in [5], sequential pattern mining has been studied extensively. Conventional sequential pattern mining finds frequent subsequences in the database based on exact match. There are two classes of algorithms. On one hand, the breadth-first search methods (e.g., GSP [20] and SPADE [26]) are based on the Apriori principle [5] and conduct level-by-level candidate-generation-and-tests. On the other hand, the depth-first search methods (e.g., PrefixSpan [19] and SPAM [1]) grow long patterns from short ones by constructing projected databases. Some variations of the depth-first search methods mine sequential patterns with vertical format [27]. However,there is some non-trivial cost of projecting databases. Every subsequence in the projected database has be scanned once again even when they share the same prefix.

In this paper, a compact tree structure UPC-Tree for sequential pattern mining will be constructed. The UPC-Tree is a prefix-tree structure storing crucial information about frequent subsequences. It corresponds to "UP Pattern Clusters" under the u-Cluster model. In addition, the structure can become more light-weighted by collapsing adjacent nodes under certain circumstances A depth-first traversal of the tree is sufficient to generate frequent subsequences by recursively concatenating legible suffixes with the existing frequent prefixes and by grouping sequences sharing the same prefix.

3. MODEL

In this section, we define the u-Cluster model for mining objects that exhibit tendency on a set of attributes.

3.1 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 $\mathcal{O} \subseteq \mathcal{D}$, $\mathcal{T} \subseteq \mathcal{A}$
$x, y, ...$	Objects in $\mathcal{D}$
$a, b, ...$	Attributes in $\mathcal{A}$
d_{xa}	Value of object x on attribute a
δ	User-specified similarity group threshold
δ^p	User-specified shifting threshold
nc, nr	User-specified minimum # of column and minimum # of rows of a model

3.2 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 3.1. *Let o be an object in the database, $\langle d_{o1}, d_{o2}, ..., d_{on}\rangle$ be the attribute values in a non-decreasing order, n be the number of attributes and δ be the user specified threshold. We say that o is* **similar** *on attributes $i, i+1, ..., i+j$, ($n \geq i > 0, n \geq j > 0$), if*

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

we call the set of attributes $\langle i, i+1, ..., i+j\rangle$ 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 the average difference between every pair of attributes whose values are closest.

$$\mathcal{G}(\delta) = \delta \times \frac{(d_{o2} - d_{o1}) + (d_{o3} - d_{o2}) + \cdots + (d_{on} - d_{o(n-1)})}{n-1} \tag{3}$$

This definition is independent of d_{oi} and is usually used when each attribute has a finite domain and its value is evenly distributed within its domain. The previous example on movie rating belongs to this case. When the value of each attribute may follow a skew distribution, Equation 4 is a better choice. The gene expression data often belongs to this scenario. For the sake of simplicity in explanation, we use Equation 4 in the remainder of this paper, unless otherwise specified.

$$\mathcal{G}(\delta, d_{oi}) = \delta \times d_{oi}. \tag{4}$$

For example, suppose we have the rating of five movies (A, B, C, D, E) from someone, which is (1, 4, 4.5, 8, 10). If $\delta = 0.2$, 4 and 4.5 are considered equivalent to each other. The rating are divided into four groups $\{\{1\}, \{4, 4.5\}, \{8\}, \{10\}\}$.

DEFINITION 3.2. *Let o be an object in the database, and $(g_{o1})(g_{o2})...(g_{ok})$ 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 $a_1, a_2, ..., a_j$ if $a_1, a_2, ..., a_j$ is a subsequence of $(g_{o1})(g_{o2})...(g_{ok})$*

In the above example, $(1, 4, 4.5, 8, 10)$ is the rating for movies (A, B, C, D, E). After we apply the group similarity, we are able to transform the original rating to the sequence $A(BC)DE$. $ABDE$, AE, and $(BC)E$ show 'UP' patterns.

DEFINITION 3.3. *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 movie ratings o_1 and o_2 for movies (A, B, C, D, E). The ratings are (1, 4, 4.5, 8, 10) and (2, 5, 7, 4.5, 9), respectively. According to Definition 3.3, the corresponding sequence of groups for o_1 is $A(BC)DE$, and for o_2 is $A(DB)CE$. Since $ABCE$ is a common subsequence of them, we say that o_1 and o_2 form a u-Cluster on the attributes sets of $ABCE$.

LEMMA 3.1. *Let $(\mathcal{O}^p, \mathcal{T}^p)$ be a δ-pCluster, where δ^p is the user defined threshold for δ-pCluster. $(\mathcal{O}^p, \mathcal{T}^p)$ is also identified as a u-Cluster if the value difference between any two attributes of an object is at least $\frac{\delta^p}{2}$.*

PROOF. Given any two objects x and y ($x, y \subseteq \mathcal{O}$), we first sort attribute values of x for all attributes in $\mathcal{T}$ in non-descending order. Without loss of generality, assume that we have two attributes a and b ($a, b \subseteq \mathcal{T}$) where $d_{xa} < d_{xb}$. If $d_{ya} > d_{yb}$ and any two attributes of an object differ in value by at least $\frac{\delta^p}{2}$, we have

$$(d_{xa} - d_{xb}) < -\frac{\delta^p}{2}; (d_{ya} - d_{yb}) \geq \frac{\delta^p}{2}; \tag{5}$$

The absolute difference between them, which is $pScore$ defined in δ-pCluster, is

$$|(d_{xa} - d_{xb}) - (d_{ya} - d_{yb})| > \delta^p. \tag{6}$$

However, because $x, y \subseteq \mathcal{O}$ and $a, b \subseteq \mathcal{T}$, $(\mathcal{O}^p, \mathcal{T}^p)$ forms a δ-pCluster. We should have

$$|(d_{xa} - d_{xb}) - (d_{ya} - d_{yb})| < \delta^p. \tag{7}$$

Now, we get a contradiction between Equations 6 and 7. Therefore, our hypothesis of $d_{ya} > d_{yb}$ is incorrect. So, we have $d_{ya} \leq d_{yb}$ when $d_{xa} < d_{xb}$, and hence ($\{x, y\}$, $\{a, b\}$) forms a u-Cluster.

Similarly, we can prove that any two objects $\subseteq \mathcal{O}$ and any two attributes $\subseteq \mathcal{T}$ can form a u-Cluster. Thus, for any two objects, they form a u-Cluster on all attributes $\subseteq \mathcal{T}$. Since there is only one unique order of attributes $\subseteq \mathcal{T}$ such that all objects $\subseteq \mathcal{O}$ show the "UP" pattern. We conclude that $(\mathcal{O}, \mathcal{T})$ is also a u-Cluster. □

In above discussion, we assume that the threshold of group similarity is set to 0. In this case, if the difference between any two attributes of an object is smaller than the threshold $\frac{\delta^p}{2}$, it will be included in δ-pCluster, but it will not be able to present in u-Cluster. To accommodate this in u-Cluster, we can set $\delta = \frac{\delta^p}{2}$. Then, any δ-pCluster will be included in a u-Cluster.

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 δ, 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 3.3, and $|\mathcal{O}| \geq nr$, $|\mathcal{T}| \geq nc$.

4. 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. Since the second step is the most time consuming portion of the algorithm, we propose and evaluate two alternative methods. The first method is to utilize the prefixSpan algorithm and introduce several optimization strategies; whereas the second (and a more efficient) method is to use a novel compact structure UPC-Tree to organize the sequences and to guide the pattern generation. Compared with prefixSpan, UPC-Tree is more compact and sequences can be projected simultaneously instead of always scanned

sequentially as in prefixSpan. In addition, single path subtrees in UPC-Tree will not be examined further to detect embedded subsequences. However,the prefixSpan algorithm is not able to look forward and takes advantage of this special case.

4.1 Preprocessing

The first step in our algorithm is preprocessing. First, for each row, we sort all the entry values in the non-decreasing 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.

	a	b	c	d
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 δ 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 4).

	sequence
1	$db(ac)$
2	$c(bd)a$
3	$cdba$
4	$cdab$
5	$dbca$
6	$a(cd)b$

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.

4.2 Optimization of PrefixSpan

PrefixSpan [19] and SPAM [1] are among the fastest sequential pattern mining algorithms proposed recently. SPAM uses a vertical bitmap representation of the database. Because of the size limitation of the sequence that a bitmap(64 bits) can store, we cannot use this approach for our purpose.

The general idea of prefixSpan is to project the sequence database based on the frequent prefixes because any frequent subsequence can always be found by growing a frequent prefix. The following is an example.

EXAMPLE 4.1. *For the sequences in Table 2, with $nc = 3, nr = 3$, the prefixSpan algorithms works in the following steps.*

Step 1: *Find length-1 sequential patterns.* Since all the attributes appear in all the rows. The result of this step is that each of $\{a, b, c, d\}$ has a support count 6.

Step 2: *Partition the search space by prefix.* The set of sequential patterns is partitioned into four subsets, one for each distinct length-1 pattern, {a},{b},{c},{d}.

Step 3: *Find sequential patterns for each subset.* The sequential patterns can be mined by constructing projected databases and mining each recursively.

prefix	projected	frequent
$a-$	$(c), b, (cd)b$	
$b-$	$(ac), (d)a, (a), ca$	
$c-$	$(bd)a, dab, dba, a, (d)b$	$cda : 4$
$d-$	$b(ac), a, ab, ba, bca, b$	$dba : 4$

Table 3: Projected database and sequential patterns

The mining process starting from projecting each sequence into subsets with prefix $a-$, $b-$, $c-$ and $d-$. The projected databases are listed in the second column of Table 3. The above procedure is applied recursively until there is no more frequent pattern.

To speed up this procedure, two strategies, namely bi-level projection and pseudo projection, are used to minimize the number of projected databases and to record only the sequence ID and position number instead of physically keeping all the projected database.

Due to the unique characteristics of the sequences in our problem, we propose two additional optimization strategies.

4.2.1 Position Matrix and pseudo Projection

There is an unique characteristic of our input sequences: *every length-1 pattern appears once and exactly once in each sequence.* Based on this observation, we can decide whether one element precedes another element by simply checking their positions in a *position matrix* instead of scanning the sequence. Table 4 shows the position matrix of sequences in Table 2.

	a	b	c	d
1	3	2	4	1
2	4	2	1	3
3	4	3	1	2
4	3	4	1	2
5	4	2	3	1
6	1	4	2	3

Table 4: Position Matrix

Therefore, if we want to further project $c-$'s projected database on $cd-$, we need to check whether d's position is after c's in every sequence in $c-$'s projected database. For example, in sequence 2, d's position is 3, which is after c's position 2. So, sequence 2 will be in $cd-$'s projected database.

By maintaining the position matrix, we only need to record the sequence IDs in the projected databases. The performance of is improved since the projection of a sequence can now be done in constant time.

4.2.2 Pruning With Parameter nc

An additional difference between the u-Cluster model and the sequential pattern mining problem is that we have a parameter nc, which is the minimum number of columns of a u-Cluster. No subsequence with length less than nc will be useful to generate u-Cluster. Therefore, we can also use the position of the element and the length of the original sequence to eliminate short subsequences as early as possible. Let the length of a prefix pre be l_{pre}, the length of the original sequence be l, and the current projection element x's position be p_x. $(pre) + x$ is not a candidate subsequence if $l_{pre} + 1 + (l - p_x) < nc$. When the prefix is not long and x's position is close to the end of the sequence, no projected subsequence will be long enough to meet nc. For example, for sequence 4 in Table 2, when we first project it on element a, we found out that the length of the projected subsequence

is going to be below $nc = 3$. Hence, this projection of sequence 3 will not be performed After using this pruning, the projected database in Table 3 shrinks. The pruned database is showed in Table 5.

	projected
$a-$	$(cd)b$
$b-$	$(ac), (d)a, ca$
$c-$	$(bd)a, dab, dba, (d)b$
$d-$	$b(ac), ab, ba, bca,$

Table 5: Pruned database

However, in the above example, we notice that to get cluster $(\{1, 4, 5\}, \{d, b, a\})$. All three rows are first projected on d. And then, all of them will project through another node b, although d and b are adjacent. The main cost of depth-first approach is on computing projection. If we can save the number of projecting operations by grouping the rows sharing same projection nodes together, we can save a lot of projection time. Let's consider an extreme example. Suppose we have 10 entirely identical sequences with 10 elements each, if we apply the prefixSpan algorithm, we have the following computation. For length-1 patterns, we need to project each sequence onto each element, which is 10×10 projections. For length-2 patterns, we need $10 \times 9 + 10 \times 8 + 10 \times 7 + 10 \times 6 + ... + 10$ projections. The projection time grows exponentially as we proceed, even though there is just one simple big cluster! To attack this inefficiency, in the following subsection, we introduce a novel UPC-Tree structure which is much more compact and effective for mining u-Cluster.

4.3 UPC-Tree

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

EXAMPLE 4.2. *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 6 (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 6 (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 6. 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 6 (2) is deleted in Figure 6 (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 6(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.

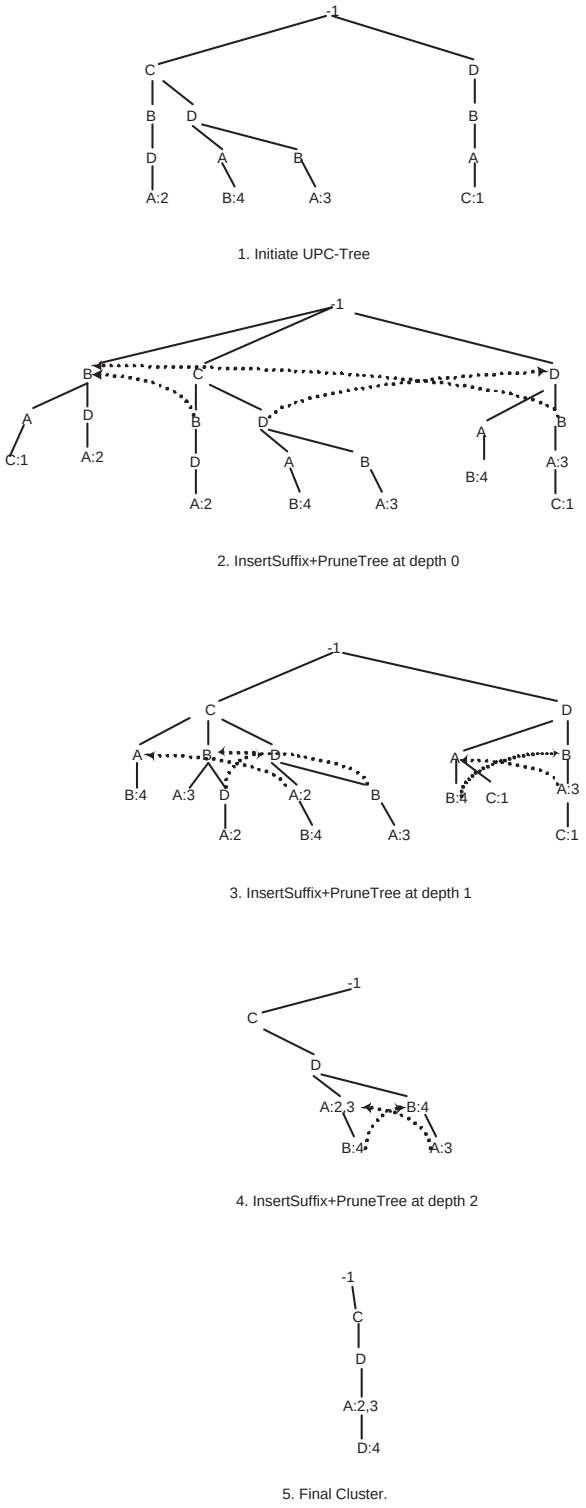


Figure 6: UPC-Tree for Table 2

DEFINITION 4.1. *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 $\mathcal{S}$
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 4.1. *Given a matrix M, a similarity grouping threshold, 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.

4.3.1 Mining u-Cluster Using UPC-Tree

LEMMA 4.2. *The developed UPC-Tree on a set of sequences contains all subsequences hidden in the initial UPC-Tree.*

Rationale: Given any sequence $S = x_1x_2x_3x_4 \dots x_n$, we want to show that all of the subsequences of S will be found in a path starting from root. Through the initiation of UPC-Tree, we know that S will exist in the UPC-Tree. Then given any subsequence $SS = x_ix_j \dots x_s, (i \geq 1, s \leq n)$, we can obtain SS by the following steps. First, at node x_i, if $i = 1$, then insert suffix $x_ix_{i+1} \dots x_n$. Now in the subtree of x_i, we can find node x_j because it will be along the path $x_ix_{i+1} \dots x_n$ that is inserted in the first step. Similarly, we insert the suffix $x_j \dots x_n$ starting from x_j. Now we get the path $x_ix_jx_{j+1} \dots x_n$. By repeating the same procedure until we insert the suffix starting with x_s, we get the path $x_ix_j \dots x_s$. Since we insert all the suffixes in the UPC-Tree, the UPC-Tree contains all the subsequences presented in the original UPC-Tree.

Rows in a u-Cluster share the same set of columns, which share the same path in the UPC-Tree. We can conclude that the UPC-Tree contains all the clusters. This leads to the following lemma.

LEMMA 4.3. *The developed UPC-Tree on a set of sequences contains all potential u-Clusters. The columns in these clusters are on the paths leading from the root to any tree node with depth no less than nc and row support count in its subtree no less than nr .*

4.3.2 Pruning UPC-Tree

Without any pruning, the whole UPC-Tree fits well into memory when we have a small matrix (15 columns by 3000 rows). However, for large matrix, some pruning strategies have to be employed to minimize the size of the UPC-Tree. There are two pruning techniques used in our implementation. One strategy is to prune the suffixes with the to-be subsequence shorter than nc; the other is to prune the subtrees where the row support count is below nr.

LEMMA 4.4. *For a node N in UPC-Tree with depth d, and for a suffix S with length l in its sub-tree, if $d + l < nc$ (the minimum columns required for a cluster), this suffix S will not be useful in forming any u-Cluster cluster.*

Rationale: The length of the path L we can get by combining the path from root to N and S is $d + l$. Based on Lemma 4.3, L will not form any cluster. Therefore, suffix S needs not to be inserted. In our implementation, we check depth of the node at which the end of the suffix is inserted. If the depth is smaller than nc, the row IDs recorded in this node will be deleted.

4.4 Improvement with Collapsing Node

The major cost of UPC-Tree development is to concatenate suffix trees. To minimize the cost incurred by dynamically allocating memory, we introduced a more compact u-Cluster structure, in which single-path tree can collapse into one node.

Figure 7 shows the procedure to construct collapsed u-Cluster structure for the same problem as in Example 4.2. All collapsed nodes are denoted by rectangles. There are two scenarios when collapsed node needs to be split.

- *The collapsed node will split if a new branch has to be inserted in the middle of path.* For example, in Figure 7, 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 7, when the subsequence DBA in sequence 3 is inserted into $DBAC$ of sequence 1, DBA is a portion of

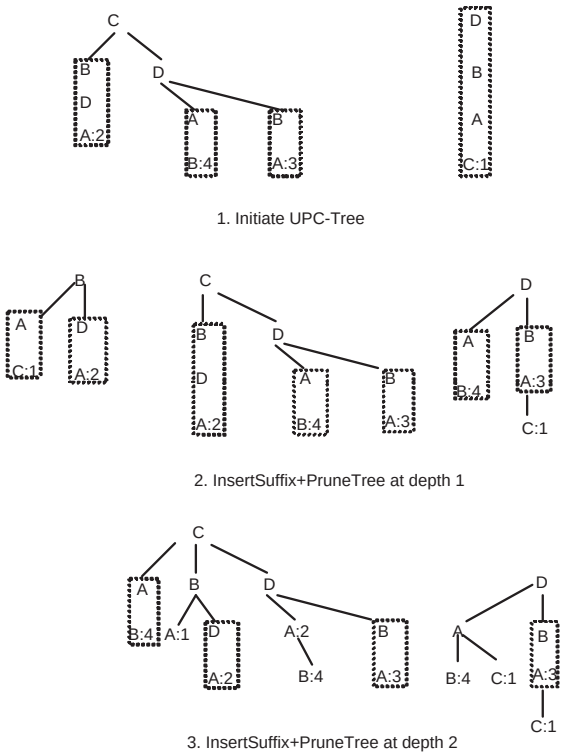


Figure 7: Collapsing UPC-Tree for Table 2

$DBAC$, $DBAC$ 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.

4.5 Addition Feature: δ-pCluster

According to Lemma 3.1, δ-pCluster can be a special case of u-Cluster if $\delta \geq \frac{\delta^p}{2}$. Therefore, our algorithm for mining u-Cluster can also be used to find δ-pCluster. Some experiments along this direction is presented in Section 5.

4.6 Additional Feature: Extension of Grouping Technique

Based on Definition 3.1, 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 $(BCD)A$. If 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). And 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.

5. 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. We also implemented the optimized prefixSpan algorithm for comparison. The following tests are organized into three categories. First, we study the sensitivity of u-Cluster to various parameters. Secondly, we evaluate the performance of UPC-Tree and compare it with the prefixSpan algorithm. At last, we show two promising patterns found in the drug activity data set.

5.1 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 [21]. 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 [9].

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.

5.2 Model Sensitivity Analysis

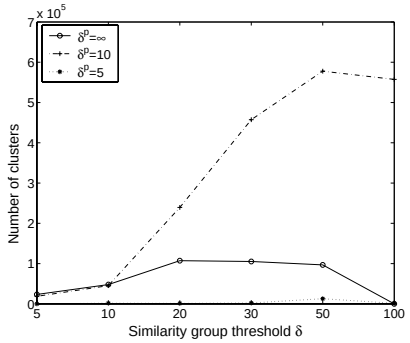
In this section, We evaluate how the similarity threshold δ and δ^p can influence the number of clusters and their sizes. We use the yeast data set in this set of experiments. The minimum number of rows is set to be 30 and the minimum number of columns is 10. We vary δ^p from ∞ to 5. Figure 8 (a) shows the number the clusters generated and (b) presents the maximum cluster size.

u-Clusters are generated when δ^p is infinity. As δ 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 decreases. This is because the overlapped clusters generated by small δ begin to merge into bigger clusters when $delta$ increases. Since long subsequence of columns has higher chance to fall on a single path, and consequently, smaller enclosed subsequences will not be counted, the total number of clusters decreases. Figure 8(b) shows that the maximum size of the u-Cluster increases dramatically in this case.

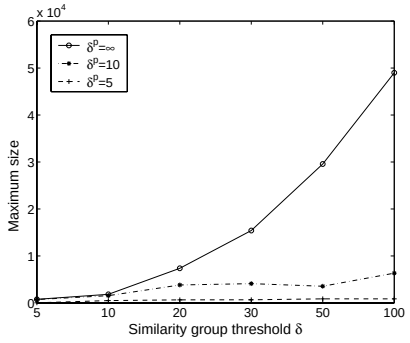
As δ^p decreases, large clusters tends to split into smaller ones and the total number of clusters increases. The size of each cluster becomes smaller. When more clusters with size smaller than $nc \times nr$ are eliminated, the total number of clusters restricted by δ^p drops below the number of u-Clusters.

5.3 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 9 shows the response time of the drug activity

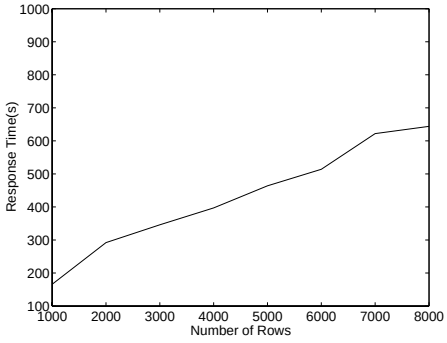


(a) Total number of clusters as a function of similarity threshold

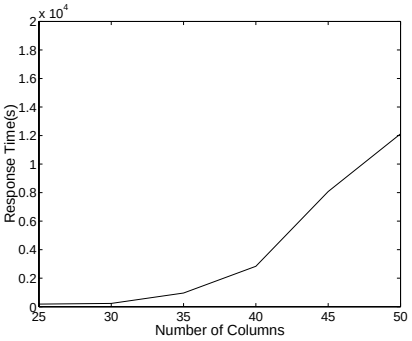


(b) Size(#column×#rows) as a function of similarity threshold

Figure 8: 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 9: Performance Study: Response time V.S. number of columns and number of rows

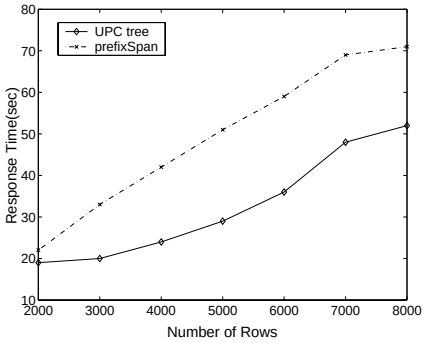


Figure 10: Performance comparison of prefixSpan and UPC-tree

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 9 demonstrate similar trends.

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

Figure 10 presents the performance comparison between the prefixSpan algorithm and the UPC-Tree algorithm. In this experiment, we use the drug activity data to see the performance with different number of rows. The parameter setting for this set of experiment is the following: $nc = 9$, $nr = 0.01N$, $\delta = 0.2$. The number of columns is set to be 20. We can observe that the UPC-Tree algorithm can constantly outperform the prefixSpan algorithm and the advantage becomes more substantial with larger data set.

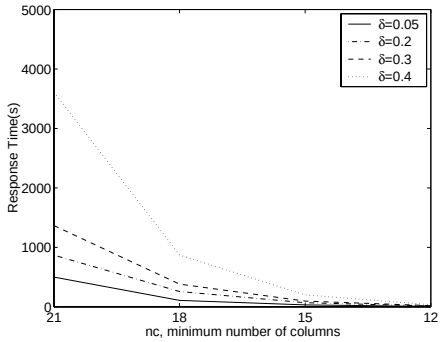
Next, we study the impact of the parameters (δ, nc, and nr) towards the response time. The results are shown in Figure 11. The size of matrix is 27×3776. When nc and nr are fixed, the response time prolongs when the similarity threshold increases. This is because the size of the clusters increase as 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 11. According to the pruning techniques we discuss in Lemma 4.4, 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 11(b).

5.4 Results from Real Data

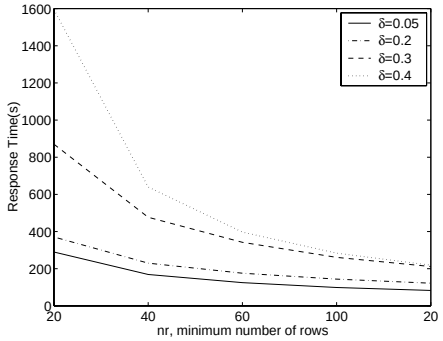
We apply the u-Cluster algorithm on the drug activity data set. With parameters $nc = 10$, $nr = 30$, $\delta = 20$, some interesting clusters are reported. Two of them are showed in Figure 12. They present a series of consistent patterns under a subset of features. From both figures, we can observe that the patterns cannot be captured by neither the traditional distance measure nor the strict shifting/ scaling factors.

6. CONCLUSIONS

In this paper, we proposed a new model called u-Cluster to capture the consistent tendency clusters exhibited by a subset of dimensions in high dimensional space. In many applications including collaborative filtering and DNA array analysis, although the distance (e.g., measured by Euclidean distance or cosine distance) among the objects may not

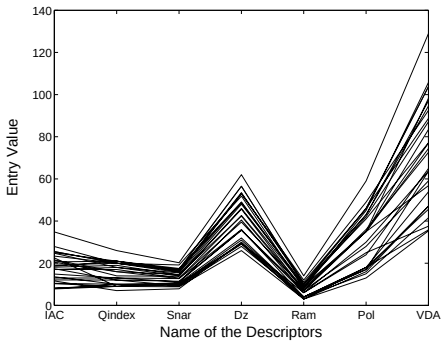


(a) Response time varying similarity threshold and nc

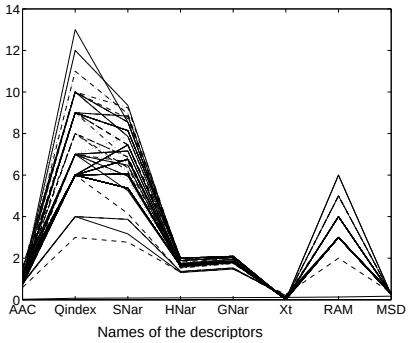


(b) Response time varying similarity threshold and nr

Figure 11: Performance Study: Response time V.S. similarity threshold , nc and nr



(a) Cluster #1



(b) Cluster #2

Figure 12: Cluster Analysis: Two example UPC-Tree in drug activity data

be close, they can still manifest consistent 'up' pattern over a permutation of a subset of dimensions. To address this issue, we introduce a new model called u-Cluster to model tendency among a set of objects. We proposed a compact tree structure, namely UPC-Tree, and devised a depth-first algorithm that can efficiently and effectively discover all u-Clusters with a user-specified minimum size.

7. REFERENCES

[1] J. Ayres, J. E. Gehrke, T. Yiu, and J. Flannick. Sequential PAttern Mining Using Bitmaps. In *SIGKDD*, July 2002.

[2] R. C. Agarwal, C. C. Aggarwal, and V. Parsad. Depth .rst generation of long patterns. In *SIGKDD*, 2000.

[3] C. C. Aggarwal, C. Procopiuc, J. Wolf, P. S. Yu, and J. S. Park. Fast algorithms for projected clustering. In *SIGMOD*, 1999.

[4] C. C. Aggarwal and P. S. Yu. Finding generalized projected clusters in high dimensional spaces. In *SIGMOD*, pages 70-81, 2000.

[5] R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE*, 3-14, Mar. 1995.

[6] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Authomatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD*, 1998.

[7] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is nearest neighbors meaningful. In *Proc. of the Int. Conf. Database Theories*, pages 217-235, 1999.

[8] C. H. Cheng, A. W. Fu, and Y. Zhang. Entropy-based subspace clustering for mining numerical data. In *SIGKDD*, pages 84-93, 1999.

[9] Y. Cheng and G. Church. Biclustering of expression data. In *Proc. of 8th International Conference on Intelligent System for Molecular Biology*, 2000.

[10] P. Dhaeseleer, S. Liang, and R. Somogyi. Gene expression analysis and genetic network modeling. In *Paci.c Symposium on Biocomputing*, 1999.

[11] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-bsed algorithm for discovering clusters in large spatial databases with noise. In *SIGKDD*, pages 226-231, 1996.

[12] D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. In *Machine Learning*, 1987.

[13] K. Fukunaga. Introduction to Statistical Pattern Recognition. Academic Press, 1990. 12] H. V. Jagadish, J. Madar, and R. Ng. Semantic compression and pattern extraction with fascicles. In *VLDB*, pages 186-196, 1999.

[14] H.V.Jagadish, J.Madar, and R. Ng. Semantic compression and pttern extraction with fasicicles. In *VLDB*, pages 186-196, 1999.

[15] R. S. Michalski and R. E. Stepp. Learning from observation: conceptual clustering. In *Machine Learning: An Arti.cial Intelligence Approach*, pages 331-363, 1983.

[16] F. Murtagh. A survey of recent hierarchical clustering algorithms. In *The Computer Journal*, 1983.

[17] H. Nagesh, S. Goil, and A. Choudhary. Ma.a: E.cient and scalable subspace clustering for very large data sets. Technical Report 9906-010, Northwestern University, 1999.

[18] R. T. Ng and J. Han. E.cient and e.ective clustering methods for spatial data mining. In *VLDB*, 1994. 1995.

[19] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen U. Dayal, and M.-C. Hsu. PrefixSpan mining sequential patterns e.ciently by prefix projected pattern growth. In *ICDE* 2001, pages 215-226, Apr. 2001.

[20] R.Srikant and R.Agrawal. Mining sequential patterns: Generalizations and performance improvements. In $EDBT'96$, pages 3-17, Mar. 1996.

[21] 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.

[22] H. Wang, W. Wang, J. Yang, and P. Yu. Clustering by pattern similarity in large data sets, in *SIGMOD*, pp. 394-405, 2002.

[23] J. Yang, W. Wang, H. Wang, and P. S. Yu. δ-clusters: Capturing subspace correlation in a large data set. In *ICDE*, pages 517-528, 2002.

[24] J. Yang, W. Wang, P. Yu, and J. Han. Mining long sequential patterns in a noisy environment. In *SIGMOD*,pp.406-417, 2002.

[25] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *SIGMOD*, pages 103114, 1996.

[26] M. J. Zaki, S. Parthasarathy, M.Orihara, and W. Li. Parallel algorithm for discovery of association rules. In *DMKD*, 343-374, 1997.

[27] M.J.Zaki.Efficient enumeration of frequent sequences. In *CIKM*, 68-75, Nov.1998.