Efficient Mining of Distance Based Subspace Clusters

Guimei Liu[1], Kelvin Sim[2], Jinyan Li[3], Limsoon Wong[1]

[1] School of Computing, National University of Singapore, Singapore

[2] Institute for Infocomm Research, Singapore

[3] School of Computer Engineering, Nanyang Technological University, Singapore

Abstract

Traditional similarity measurements often become meaningless when dimensions of datasets increase. Subspace clustering has been proposed to find clusters embedded in subspaces of high dimensional datasets. Many existing algorithms use a grid based approach to partition the data space into non-overlapping rectangle cells, and then identify connected dense cells as clusters. The rigid boundaries of the grid based approach may cause a real cluster to be shattered in different cells. In this paper, we propose to use a sliding window approach to partition the dimensions to preserve significant clusters. We call this model nCluster model. The sliding window approach generates more bins than the grid-based approach, thus it incurs higher mining cost. We develop a deterministic algorithm, called MaxnCluster, to mine nClusters efficiently. MaxnCluster uses several techniques to speed up the mining, and it produces only maximal nClusters to reduce result size. Non-maximal nClusters are pruned without the need of storing the discovered nClusters in the memory, which is key to the efficiency of MaxnCluster. Our experiment results show that MaxnCluster can produce maximal nClusters efficiently and accurately.

1 Introduction

Clustering seeks to find groups of similar objects based on the values of their attributes. Traditional clustering algorithms use distance on the whole data space to measure similarity between objects. As the number of dimensions in a dataset increases, distance measures become increasingly meaningless [10, 17, 3]. In very high dimensional datasets, the objects are almost equidistant from each other. This is known as *the curse of high dimensionality* [9].

The concept of subspace clustering has been proposed to cope with this problem by discovering clusters embedded in the subspaces of high dimensional datasets. Many subspace clustering algorithms use a grid based approach to find dense regions [6, 13, 25, 12, 20]. They partition the data space into non-overlapping rectangular cells by discretizing each dimension into a number of bins. A cell is dense if the fraction of total objects contained in the cell is greater than a threshold. Dense cells in all subspaces are identified using a bottom-up strategy, and connected dense cells are merged together to form clusters.

In the grid based approach, objects around the boundaries of the bins have similar values, but they are put into different bins. As a result, a cluster may be divided into several small clusters as illustrated in the following example. Table 1 shows a dataset containing six objects and three attributes. The value range of the three attributes is [0, 10]. Objects 2, 3 and 4 have similar values on both attributes a and b, so object set {2, 3, 4} and attribute set {a, b} should form a subspace cluster. If we use the grid based approach and partition each attribute to two bins of equal length, then for each attribute, we have two bins [0, 5] and (5, 10]. Object 4 is in different bins with objects 2 and 3 on attribute a, and object 3 is in different bins with objects 2 and 4 on attribute b. We get two smaller clusters ({2, 3}, a) and ({2, 4}, $\{b\}$). Algorithms have been proposed to find the cutting points adaptively based on data distribution [25, 12, 20]. However, these algorithms do not allow overlap between different bins either, so it is still possible that objects with similar values on an attribute are placed into different bins of the attribute, which may cause a cluster to be shattered in different cells.

	a	b	c
1	0	10	1
2	**4**	**5**	4
3	**5**	**6**	0
4	**6**	**5**	7
5	9	0	10
6	10	1	6

Table 1: An example missing cluster

In this paper, we propose a distance based subspace clustering model called nCluster to overcome the problem discussed above. The nCluster model uses a sliding window approach to partition the dimensions, which allows overlap between different bins of an attribute. This may result in more bins than the grid based algorithms, which increases the complexity of the problem. To make the problem solvable, we consider only those clusters containing a non-trivial number of objects and attributes. Furthermore, we mine only maximal nClusters to avoid generating too many clusters. We develop an efficient algorithm to find the complete set of maximal nClusters, which uses dedicated data structures to produce object sets and attribute sets simultaneously and uses the size constraints on both object sets and attribute sets to prune the search space. MaxnCluster can also effectively and efficiently prune non-maximal nClusters, and it performs the pruning without the need of storing maximal nClusters in the memory. Instead, it utilizes the dataset itself to prune non-maximal nClusters, and several techniques are employed to minimize the overhead incurred.

The rest of the paper is organized as follows. Section 2 gives the formal definition of the nCluster model. We present the MaxnCluster algorithm in Section 3. The experiment results are reported in Section 4. Related work is discussed in Section 5. Finally, Section 6 concludes the paper.

2 Problem Definition

In this section, we give the formal definition of the nCluster model. The following notations are used in the paper. Let $\mathcal{O}$ be a set of objects. Each object has a set of attributes $\mathcal{A}$. Without loss of generality, we assume the value range of all continuous attributes is [0,1]. We use $x, y, \cdots$ to denote an object in $\mathcal{O}$, $a, b, \cdots$ to denote an attribute in $\mathcal{A}$, and v_{xa} to denote the value of an object x on an attribute a.

The distance of two objects x and y on an attribute a is defined as $|v_{xa} - v_{ya}|$. If the distance of x and y on an attribute a is smaller than a predefined threshold, then x and y are called neighbors on attribute a. Similarly, we can define neighbors of an object on a subset of attributes in $\mathcal{A}$, and they are called subspace neighbors.

Definition 1 (Subspace δ-neighbors) *Let x, y be two objects and $D \subseteq \mathcal{A}$ be a subset of attributes. If for every nominal attribute $a \in D$, we have v_{xa}=v_{ya}, and for every continuous attribute $a \in D$, we have $|v_{xa} - v_{ya}| \leq \delta$, where δ is a predefined threshold, then we say that x and y are δ-neighbors of each other in subspace D.*

If a set of objects T are δ-neighbors of one another on a set of attributes D, then these objects form a cluster on subspace D and we call it a δ-nCluster.

Definition 2 (Subspace δ-nCluster) *Let $T \subseteq \mathcal{O}$ be a set of objects and $D \subseteq \mathcal{A}$ be a set of attributes. If for every two objects x, $y \in T$ and every attribute $a \in D$, objects x and y are δ-neighbors on attribute a, then we say that (T, D) is a subspace δ-nCluster, or simply δ-nCluster.*

Example 1 *Table 2 shows a dataset with 4 attributes and 8 objects. If we set δ to 0.1, then $\{1, 2, 4, 6, 8\}$ and $\{a\}$ form a δ-nCluster, and ($\{1, 6\}$, $\{a, b, c\}$) is a δ-nCluster.*

Given two nClusters (T_1, D_1) and (T_2, D_2), if $T_1 \subseteq T_2$ and $D_1 \subseteq D_2$, then we say that (T_1, D_1) is a *sub-nCluster* of (T_2, D_2), and (T_2, D_2) is a *super-nCluster* of (T_1, D_1). If either $T_1 \subset T_2$ or $D_1 \subset D_2$ is true,

	a	b	c	d
1	**0.50**	**0.15**	**0.84**	0.00
2	**0.55**	0.80	0.00	0.85
3	0.40	0.32	0.70	0.30
4	**0.50**	**0.11**	0.35	0.72
5	0.00	1.00	1.00	0.60
6	**0.55**	**0.20**	**0.86**	1.00
7	1.00	0.00	0.50	0.45
8	**0.60**	0.67	0.20	0.15

Table 2: An example dataset

then we say (T_1, D_1) is a proper sub-nCluster of (T_2, D_2). The δ-nClusters have the following properties based on their definition.

Property 1 (anti-monotone property) *Let $T \subseteq \mathcal{O}$ be a set of objects and $D \subseteq \mathcal{A}$ be a set of attributes. If T and D form a δ-nCluster, then T forms a δ-nCluster with every subset of D, and D forms a δ-nCluster with every subset of T.*

Property 2 *If a set of objects are δ-neighbors of one another on two sets of attributes D_1 and D_2, then these objects are also δ-neighbors of one another on $D_1 \bigcup D_2$.*

Given a set of attributes $\mathcal{A}$, the number of subspaces of $\mathcal{A}$ is exponential to the number of attributes in $\mathcal{A}$. If $\mathcal{A}$ contains many attributes, it is impractical to exhaustively enumerate all the subspaces and find all the δ-nClusters in each individual subspace. For a cluster to be meaningful and useful, the cluster has to contain a non-trivial number of objects and attributes. We use two thresholds mr and mc to constrain the minimum number of objects and attributes contained in a δ-nCluster, and we are interested in mining only δ-nClusters containing at least mr objects and mc attributes.

Although restricting the minimum number of objects and attributes filters out insignificant δ-nClusters, there still can be a large number of δ-nClusters, and many of them are redundant in the sense that they can be subsumed by some larger δ-nClusters. Based on Property 1, if a set of objects T and a set of attributes D can form a δ-nCluster, then every sub-nCluster of (T, D) can form a δ-nCluster. These sub-nClusters of (T, D) provide no more information than (T, D). To avoid generating too many δ-nClusters, we mine only maximal δ-nClusters.

Definition 3 (Maximal δ-nCluster) *Let $T \subseteq \mathcal{O}$ be a set of objects and $D \subseteq \mathcal{A}$ be a set of attributes, and T and D form a δ-nCluster. If there does not exist a δ-nCluster (T', D') such that (T, D) is a proper sub-nCluster of (T', D'), then (T, D) is called a maximal δ-nCluster.*

Example 2 *Let δ=0.1. In the example dataset shown in Table 2, δ-nCluster ({1, 6}, {a, b}) is not maximal because its attribute set can be extended by attribute c and its object set can be extended by object 4. δ-nClusters ({1, 4, 6}, {a, b}) and ({1, 6}, {a, b, c}) are maximal δ-nClusters because neither their object sets can be extended without reducing their attribute sets, nor their attribute sets can be extended without reducing their object sets.*

3 Mining Maximal nClusters

In this section, we present an algorithm called MaxnCluster for mining maximal δ-nClusters containing at least mr objects and at least mc attributes. The main challenge of mining δ-nClusters is in subspace enumeration. We use Property 1 to prune the subspaces.We start from δ-nClusters containing only one attribute, and extend them to find δ-nClusters containing more attributes. A δ-nCluster is extended if and only if it contains at least mr objects.

3.1 Finding nClusters with single attribute

We are interested in maximal δ-nClusters, so for every attribute a, we find the maximal object sets that can form δ-nClusters with $\{a\}$. An attribute can form δ-nClusters with multiple maximal object sets. We identify them based on the following observation.

Lemma 1 *Given an attribute a and a set of objects T, (T, $\{a\}$) is a δ-nCluster if and only if $max\{v_{xa}|x \in T\} - min\{v_{xa}|x \in T\} \leq \delta$.*

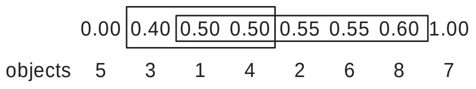


Figure 1: Finding maximal object sets

Based on the above lemma, we identify the maximal object sets of an attribute using a sliding window, which is similar to the method used in [34] for finding maximal dimension sets (MDS). We sort the objects in $\mathcal{O}$ in ascending order of their values on attribute a. We maintain two pointers, a left-end pointer and a right-end pointer, on the sorted sequence. Initially, the two pointers are placed at the first element of the sorted sequence. We move the right-end point rightward one position at a time until the difference between the two pointers is greater than δ, and the objects between the two pointers form a maximal object set of a. To find the next maximal object set, we move the left-end pointer rightward until the value pointed by the left-end pointer is different, and we then move the right-end pointer as described above. The process is repeated until the right-end pointer reaches the last element of the sorted sequence. Figure 1 shows how the two maximal object sets of attribute a are discovered.

Using the above method, if the number of distinct values of an attribute is very large, then the number of maximal object sets generated can be very large. This may pose a difficulty on the mining algorithm. To avoid generating too many highly overlapped maximal object sets on the same attribute, we use a threshold ω to control the overlap. Two adjacent windows can have at most $\omega \cdot \delta$ overlap on their value ranges. When ω= 0, we divide attributes into non-overlapping bins as in the grid based approach.

Table 3 shows the maximal object sets of all the attributes. Every attribute and its maximal object set forms a δ-nCluster containing only one attribute. We use these δ-nClusters as starting points to find δ-nClusters containing more attributes.

attr	maximal object sets
a_1	{1, 3, 4}
a_2	{1, 2, 4, 6, 8}
b_1	{1, 4, 5, 6}
c_1	{1, 6, 7}
c_2	{3, 4}
d_1	{1, 7, 8}

Table 3: Maximal object sets of attributes

obj	attribute lists	sorted lists
1	a_1, a_2, b_1, c_1, d_1	a_2, b_1, a_1, c_1, d_1
2	a_2	a_2
3	a_1, c_2	a_1, c_2
4	a_1, a_2, b_1, c_2	a_2, b_1, a_1, c_2
5	b_1	b_1
6	a_2, b_1, c_1	a_2, b_1, c_1
7	c_1, d_1	c_1, d_1
8	a_2, d_1	a_2, d_1

Table 4: Attribute lists of objects

3.2 Finding maximal nClusters containing more than one attribute

Given a δ-nCluster (T, D) and an attribute $a \notin D$, if there are at least mr objects in T that are δ-neighbors of one another on attribute a, then attribute a can be added to D to form a δ-nCluster with one more attribute. To find all such attribute a, we create an attribute list for every object, which contains all the attributes on which x has at least $(mr-1)$ δ-neighbors. To distinguish the different maximal object sets of the same attribute, we map each maximal object set to an item.

In the above example, attribute a has two maximal object sets, so we map them to two items a_1 and a_2. We also call a_1 and a_2 items of attribute a. The attribute lists of objects 1, 3 and 4 contain a_1, and the attribute lists of objects 1, 2, 4, 6, and 8 contain a_2. The attribute lists of all the objects in Table 2 are shown in the second column of Table 4. The above transformation is lossless, that is, we can reconstruct Table 3 from Table 4.

Since the attribute lists contain the complete information, so we use attribute lists to discover maximal δ-nClusters in the remaining mining. Our mining algorithm is based on the following observation.

Lemma 2 *A set of attributes D forms a δ-nCluster with a set of objects T if and only if the attribute lists of the objects in T all contain the same item of every attribute in D.*

If we regard an attribute list as a transaction and the set of attribute lists of all the objects in $\mathcal{O}$ as a transaction database, then mining maximal δ-nClusters can be transformed to mining frequent itemsets from a transaction database [7]. The concept of maximal δ-nClusters is used in the paper to remove redundant δ-nClusters, and it is similar to the frequent closed itemset concept [28], which is used to remove redundant frequent itemsets. An itemset is closed if it is maximal with respect to the set of transactions containing it. If a δ-nCluster is maximal, then its corresponding attribute item set is a closed itemset in the attribute lists.

In our previous work [21], we have used LCM [33], one of the most efficient frequent closed itemset mining algorithms, to mine maximal δ-nClusters. However, using frequent closed itemset mining algorithms to mine maximal δ-nClusters has several drawbacks: (1) Frequent itemset mining produces only itemsets (attribute sets), the corresponding object sets have to be generated in a post-processing step, which can be time-consuming when the number of objects and the number of generated attribute sets are very large. (2) Frequent closed itemset mining algorithms use only the size constraint on object sets to prune the search space, and the size constraint on attribute sets is not utilized. As a result, using frequent closed itemset mining algorithms to mine maximal δ-nClusters may produce many small uninteresting nClusters. (3) A closed itemset may not always yield a maximal δ-nCluster. For example, $\{a_1, a_2, b_1\}$ is a closed itemset in Table 4, and its corresponding attribute set is $\{a, b\}$ and object set is $\{1, 4\}$. However, ($\{1, 4\}$, $\{a, b\}$) is not a maximal nCluster because one of its super-nCluster ($\{1, 4, 6\}$, $\{a, b\}$) is also a δ-nCluster. Hence using frequent closed itemset mining algorithms to mine maximal nClusters may produce many non-maximal nClusters.

In this section, we present an algorithm called MaxnCluster that uses the size constraints on both object sets and attribute sets to prune the search space, and generates the attribute sets and object sets of nClusters simultaneously and efficiently. MaxnCluster can also effectively and efficiently prune non-maximal nClusters, and it performs the pruning without the need of storing maximal nClusters, which is key to its efficiency. MaxnCluster does need to store some additional information about the data to do the pruning. We use some techniques to minimize the overhead incured.

In the rest of this section, we first give the framework of the MaxnCluster algorithm, and then describe how to produce the object lists of δ-nClusters efficiently. At the end of this section, we describe how to identify and prune non-maximal δ-nClusters. To ease the presentation, we uses terminologies from frequent pattern mining. That is, we refer to attribute item sets as itemsets, and attribute lists of objects as transactions.

3.3 The mining framework

The MaxnCluster algorithm is modified from the pattern growth algorithm FP-growth [16]. As in frequent itemset mining, we define the *support* of an itemset l as the number of transactions containing it, denoted as *support*(l). We say an itemset is *frequent* if its support is no less than mr. An itemset is extended if and only if it is frequent.

The MaxnCluster algorithm first finds all the frequent items. The power set of the set of frequent items forms the search space of the maximal nCluster mining problem, which can be represented as a set-enumeration tree [31]. MaxnCluster uses the depth-first order to explore the search space. The items are sorted into descending frequency order. For each frequent item a_i, MaxnCluster uses the set of frequent items that are before a_i in the descending frequency order except those items that are of the same attribute as a_i to extend a_i, and these items are called the *candidate extensions* of a_i. MaxnCluster does not use those items that are of the same attribute as a_i to extend a_i because such extension does not introduce any new attribute. For example, with mr=2, the set of frequent items in Table 4 are {a_2:5, b_1:4, a_1:3, c_1:3, d_1:3, c_2:2}. Item c_2 is the last item in the descending frequency order, so its candidate extensions include all the other items except c_1 because c_1 is of the same attribute as c_2. Item d_1's candidate extensions include c_1, a_1, b_1 and a_2, and the first item a_2 does not have any candidate extensions.

Algorithm 1 MaxnCluster

```
Input:
    $l$ is a frequent item set
    $CandExt(l)$ is the candidate extensions of $l$
    $mr$ is the minimum number of objects
    $mc$ is the minimum number of columns
Description:
 1: $FreqExt(l) = \{a_i | a_i \in CandExt(l) \wedge (l \cup \{a_i\}\ is\ frequent) \wedge (a_i\ does\ not\ have\ the\ same\ attribute\ as\ any\ item\ in\ l)\}$;
 2: if $|l| + |FreqExt(l)| < mc$ then
 3:     return ;
 4: Sort items in $FreqExt(l)$ into descending frequency order;
 5: for all item $a_i \in FreqExt(l)$ do
 6:     $l' = l \bigcup \{a_i\}$;
 7:     $CandExt(l') = \{b_j | b_j \in FreqExt(l) \wedge (b_j\ is\ before\ a_i) \wedge (b_j\ is\ not\ of\ the\ same\ attribute\ as\ a_i)\}$;
 8:     if $|l'| + |CandExt(l')| >= mc$ then
 9:         $l'' = l' \cup \{b_j | support(l') = support(l' \cup \{b_j\})\}$;
10:         if $l'' \subseteq (l' \cup CandExt(l'))$ then
11:             if $|l''| \geq mc$ then
12:                 Output the object set and attribute set of $l''$ as a maximal nCluster;
13:             if $CandExt(l') - l'' \neq \{\}$ then
14:                 MaxnCluster($l''$, $CandExt(l') - l''$, $mr$, $mc$);
```

Algorithm 1 shows the pseudo-code of the MaxnCluster algorithm. When Algorithm 1 is first called, l is set to the empty set and $CandExt(l)$ is set to the set of items in the attribute lists. For every frequent itemset l, Algorithm 1 first finds the set of frequent extensions of l (line 1) and sorts them into descending frequency order (line 4), and then extends l by one more attribute using these frequent extensions (line 6). We use l' to denote the frequent itemset obtained by extending l using one item in $FreqExt(l)$. Next, Algorithm 1 computes the candidate extensions of l' (line 7) and extend l' recursively (line 13-14). The codes at line 9-10 are for checking whether the attribute sets are maximal, which is discussed in Section 3.5.

MaxnCluster not only uses the mr threshold to prune the search space as frequent closed itemset mining algorithms, but also uses the mc threshold to prune the search space. Every itemset is extended by only its frequent extensions, so any itemset extended from itemset l must be a subset of $l \cup CandExt(l)$ and $l \cup FreqExt(l)$. If $l \cup CandExt(l)$ or $l \cup FreqExt(l)$ contains less than mc attributes, then there is no need to extend l further (line 2-3, line 8).

3.3.1 The FPO-tree structure for support counting

The MaxnCluster algorithm uses a compact prefix-tree structure, called FPO-tree, to store transactions to facilitate support counting. The FPO-tree structure is modified from the FP-tree structure [16], and it contains additional information for object list generation. The construction of an FPO-tree is similar to that of an FP-tree.

We use the transactions in Table 4 to illustrate the FPO-tree structure and its construction. The set of frequent items in Table 4 are {a_2:5, b_1:4, a_1:3, c_1:3, d_1:3, c_2:2}, and they are sorted into descending frequency order. The transactions in Table 4 are also sorted according to this order. The sorted transactions are shown in the third column of Table 4, and the FPO-tree storing these sorted transactions is shown in Figure 2.

An FPO-tree node contains an item, a support counter, a parent pointer, a child pointer, a right-sibling pointer and a node-link pointer like an FP-tree node. The support counter of an FPO-tree node records the frequency of the branch ended at that node. The node-links link the FPO-tree nodes containing the same item together, and they are denoted by dotted lines in Figure 2. The numbers in rectangle boxes are object ids. An additional pair of pointers are maintained at an FPO-tree node, which point to the set of objects whose attribute lists containing the branch ended at the node. The two pointers are used for generating the object sets of δ-nClusters. We describe them in subsection 3.4.

In an FPO-tree, the branches containing item a_i store the transactions containing a_i. To obtain all the items that co-occur at least mr times with a_i, MaxnCluster traverses all the branches containing a_i

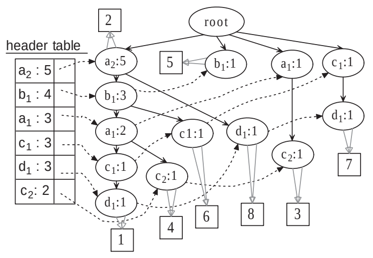
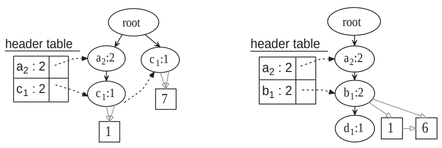


Figure 2: An example FPO-tree



(a) d_1 (b) c_1

Figure 3: New FPO-trees constructed for d_1 and c_1

via node-links. The first FPO-tree node containing a_i is maintained in a header table as shown in Figure 2. The items in FPO-trees are sorted according to descending frequency order, and the candidate extensions of an item include all the items that are before it in the descending frequency order. Therefore, in an FPO-tree, the candidate extensions of an item always appear above the item itself. For example, item b_1 is a candidate extension of c_2, so b_1 always appear in some ancestor nodes of the nodes containing c_2. To find the frequent extensions of a_i, MaxnCluster traverses the branches containing a_i upwards via parent pointers.

3.4 Generating object sets

To find δ-nClusters, we not only need to generate their attribute sets, but also need to generate their object sets. Therefore, MaxnCluster maintains an object list for each FPO-tree node, and the object list contains the ids of all the objects whose transactions fall into that FPO-tree node. Here we say a transaction falls into an FPO-tree node if the sorted transaction matches the branch ended at the FPO-tree node.

If the transaction of an object falls into an FPO-tree node, then the transaction must also fall into the parent node of the FPO-tree node. Therefore, the object list of an FPO-tree node is a subset of its parent's object list. To save space, the object lists of the FPO-tree nodes on the same path are shared in an FPO-tree. Initially, every object is placed at the lowest FPO-tree node its transaction falls into as shown in Figure 2. A pair of pointers is maintained at each FPO-tree node, which point to the first and the last element of the object list.

During the mining process, the object lists are propagated to the nodes at higher levels. In an FPO-tree, the items are sorted into descending frequency order while the δ-nClusters are discovered in the reverse order, which means that the FPO-tree nodes at lower levels are processed first. In the above example, the δ-nClusters containing c_2 are first discovered, and then the δ-nClusters containing d_1, c_1 and so on are discovered. After the δ-nClusters containing an item a_i are discovered, the two pointers pointing to the first and the last elements of the object lists of the FPO-tree nodes containing a_i are passed to their parent nodes so that their parent nodes can include the object lists of a_i into their own object lists. For example, after all the δ-nClusters containing c_2 are discovered, the object lists of the two FPO-tree nodes containing c_2 are passed to the two FPO-tree nodes containing item a_1. This process involves only two pointer adjustments for every FPO-tree node. It ensures that when MaxnCluster starts to mine the δ-nClusters containing an item a_i, the object lists of the FPO-tree nodes containing a_i include all the objects that are δ-neighbors on a_i. To get the maximal object set of an item a_i, we simply visit all the FPO-tree nodes containing a_i via node-links, and collect their object lists.

If a new FPO-tree needs to be constructed for an item, the object lists are also reused.

Lemma 3 *A branch in an FPO-tree Tr is still a single branch in the new FPO-trees constructed from Tr.*

The above lemma implies that the object list of an FPO-tree node is passed as a whole to the new FPO-tree, so we can simply pass the two pointers pointing to the first and the last elements of the object list. It is possible that multiple branches in the original FPO-tree are merged into one single branch in the new FPO-tree. In this case, the object lists are merged by pointer adjustment. For example, a new FPO-tree needs to be constructed for item c_1. The first two branches containing c_1 in Figure 2 are represented by a single branch in the new FPO-tree as shown in Figure 3(b), so the two object lists are connected together in the new FPO-tree. Note that this does not affect the object lists in the original FPO-tree. The nodes in the original FPO-tree still know where its object list begins and where its object list ends because an FPO-tree node maintains two pointers pointing to both the first and the last elements of its object list.

The above method for generating object lists has two advantages: (1) The object lists are maintained by pointer adjustment. Merging two object lists involves only two pointer adjustment, and its cost is independent of the size of the object list. (2) The object lists of different attribute sets are shared. Every object id is stored only once. When the dataset is very large and dense, this method can save the time and space for maintaining the object lists.

3.5 Pruning non-maximal δ-nClusters

A δ-nCluster is not maximal either because its attribute set is not maximal or because its object set is not maximal. In this subsection, we describe how to prune these two types of non-maximal δ-nClusters during the mining process.

3.5.1 Pruning δ-nClusters with non-maximal attribute sets

The δ-nClusters with non-maximal attribute sets are pruned based on the following lemmas.

Lemma 4 *Let T be the set of objects whose transactions contain itemset l. Itemset l is maximal with respect to T if and only if $l = \bigcap_{x \in T} L_x$, where L_x is the transaction of object x. We call $\bigcap_{x \in T} L_x$ the closure of l, denoted as closure(l).*

Lemma 5 *If an item a_i is in the closure of an itemset l, then a_i must be in the closure of every superset l' of l.*

Lemma 6 *If the closure of an itemset l contains some item a_i such that a_i is not a candidate extension of l, then none of the itemsets extended from l can be maximal with respect to its object set.*

The above lemmas have been used in frequent closed itemset mining for pruning non-closed itemsets [33]. During the mining process, MaxnCluster compares every itemset with its closure (line 9-10 in Algorithm 1). If the itemset is not the same as its closure, then the itemset is discarded based on Lemma 4. If the closure of an itemset l contains some item a_i such that a_i is not a candidate extension of l, then there is no need to extend l further based on Lemma 6.

We now describe how to generate the closure of an itemset l from the FPO-tree structure. The following notations are used in the description. Let a_i be the last item of l and $l'{=}l-\{a_i\}$ be the prefix of l, that is, l is extended from l' by adding a_i. Let $FPO_{l'}$ be the FPO-tree constructed for l', that is, $FPO_{l'}$ contains the frequent extensions of l', including a_i.

The branches in $FPO_{l'}$ containing a_i store the transactions containing l. It seems that we can intersect these branches to obtain the closure of l. However, this is not enough because some items may be excluded from these branches because they are not candidate extensions of l'. To find all the items that are in the closure of l, we store some additional information in FPO-trees. If an item a_j is not a candidate extension of l', but $l' \cup \{a_j\}$ is frequent, then it is possible that a_j is in the closure of l''s supersets. In this case, a_j is stored in the FPO-tree constructed for l' for closure generation.

We use an example to illustrate how to generate the closure of an itemset. In the FPO-tree shown in Figure 2, there are three branches containing item c_1: $d_1c_1a_1b_1a_2$:1, $c_1b_1a_2$:1 and d_1c_1:1. To obtain the closure of c_1, we need to access all the nodes in these three branches. The FPO-tree nodes containing item c_1 split each of the three branches into two parts. The upper portions of the three branches contain the candidate extensions of c_1, and they are visited via parent pointers. The lower portions of the three branches are three subtrees, and the items in these three subtrees are not candidate extensions of c_1 but they may be in the closure of c_1. The three subtrees are traversed using the depth-first traversal strategy via child pointers. We find that the closure of c_1 is itself. We also find that item d_1 co-occurs 2 times with c_1. It is possible that d_1 is in the closure of c_1's supersets, so d_1 is included into the new FPO-tree constructed for c_1 as shown in Figure 3(b) even though d_1 is not a candidate extension of c_1. In the new FPO-tree, item d_1 is not included in the header table, and it is put after all the frequent extensions of c_1 so that d_1 is considered only for closure generation but never considered for extension.

An FPO-tree node needs to be visited for the closure generation of all its ancestors, which incurs high traversal cost. Furthermore, additional items are included into FPO-trees for closure generation, which makes the situation even worse. We use two techniques to avoid unnecessary traversal.

Technique 1: pruning based on the support of a node and the support of its child nodes. Let $node_{a_i}$ be a node in $FPO_{l'}$ containing item a_i, B be the branch ended at $node_{a_i}$ and FPO_{a_i} be the subtree rooted at $node_{a_i}$ as shown in Figure 4. For an item a_j in FPO_{a_i} to be in the closure of $l = l' \cup \{a_i\}$, the frequency of a_j in FPO_{a_i} has to be the same as the support of $node_{a_i}$ because otherwise, there is at least one transaction that contain l but does not contain a_j. Similarly, if the frequency of a_j in FPO_{a_i} is less than the support of $node_{a_i}$, then a_j cannot be in the closure of any superset of l that contains some items in B, so the a_js appearing in FPO_{a_i} can be ignored for closure generation.

The support of an FPO-tree node is always no larger than that of its ancestors. Therefore, if the support sum of the child nodes of an FPO-node p is less than the support of p, then the support of any item in the subtree rooted at p must be lower than that of p, and there is no need to access the subtree rooted at p for closure generation. For example, in Figure 2, the frequency sum of the child nodes of a_2 is less than the frequency of a_2, so there is no need to traverse this subtree for closure generation.

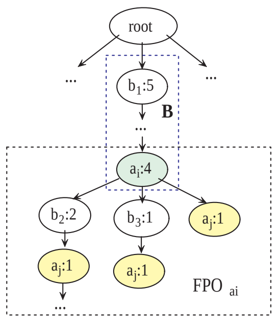


Figure 4: $FPO_{D'}$ (header tables and node links are omitted.)

Technque 2: pruning by maintaining the set of potential closure items at FPO-tree nodes. Let p be an FPO-tree node and FPO_p be the subtree rooted at p. We say an item is a potential closure item of p if its frequency in FPO_p is the same as the support of p. After the set of potential closure items of node p are discovered, they are stored at node p. When p is visited for generating the closure of its parent node, MaxnCluster collects the stored items directly instead of visiting the subtree rooted at p again. The closure of p's other ancestors are generated using the potential closure items maintained at p's parent node. Therefore, after the potential closure items of p's parent node are discovered, the potential closure items of p are disposed to save space. By using this technique, each node is visited at most once for its ancestors' closure generation.

The space overhead caused by technique 2 is very low because for every path in an FPO-tree, we maintain potential closure items at at most one node on the path at a time. The number of nodes at lower levels are large, but they have small number of potential closure items because their subtrees are small. The number of potential closure items of nodes at higher levels may be large, but the number of nodes at higher levels is small. During our performance study, we found that the space overhead incurred by this technique is indeed very small, and the running time saved is very significant.

3.5.2 Pruning δ-nClusters with non-maximal object sets

The pruning techniques described in the previous subsection prune all the δ-nClusters with non-maximal attribute sets, but they cannot avoid producing δ-nClusters with non-maximal object sets. The nClusters with non-maximal object sets are caused by the fact that the object set of an itemset is maximal with respect to only the itemset, but not necessarily maximal with respect to the corresponding attribute set. For example, in Table 3, object set {1, 4} is maximal with respect to itemset $\{a_1, b_1\}$, but it is not maximal with respect to the corresponding attribute set $\{a, b\}$ because $(\{1, 4, 6\}, \{a, b\})$ is a δ-nCluster, which is generated from itemset $\{a_2, b_1\}$.

We use the following lemma to prune δ-nClusters with non-maximal object sets.

Lemma 7 *Let l be an itemset, a_i, a_j be two candidate extensions of l and a_i, a_j have the same attribute, T' be the object set of $l \cup \{a_i\}$ and T'' be the object set of $l \cup \{a_j\}$. If $T' \subseteq T''$, then for every itemset l' extended from $l \cup \{a_i\}$, there must exist another itemset l'' extended from $l \cup \{a_j\}$ such that l'' and l' have the same attribute set and the object set of l'' is no smaller than the object set of l'. In this case, a_i can be discarded.*

To prune nClusters with non-maximal object sets, for every frequent itemset l, we count the co-occurrence of the candidate extensions of l that are of the same attribute. If for two candidate extensions a_i and a_j of l, we have $support(l \cup \{a_i\}) = support(l \cup \{a_i, a_j\}) \leq support(l \cup \{a_j\})$, which means that the object set of $l \cup \{a_i\}$ is a subset of the object set of $l \cup \{a_j\}$, then we exclude a_i from further extension based on the above lemma. The above pruning method cannot prune nClusters with non-maximal object sets completely. The remaining non-maximal nClusters are removed in a post-processing step using a hashing technique.

4 A Performance Study

In this section, we study the efficiency of the MaxnCluster algorithm and the quality of the δ-nCluster generated. We used both synthetic datasets and a real dataset in our performance study, and our experiments were conducted on a Windows machine with a 2.33Ghz Pentium IV CPU and 4GB memory.

4.1 Datasets

We generate synthetic datasets in matrix forms, where rows represent objects and columns represent attributes. The value ranges of all the attributes are set to [0, 1]. We embed a number of δ-nClusters in the data. Our data generator takes several parameters: the number of rows T, the number of columns A, the number of δ-nClusters embedded in the data N, the minimum and maximum number of attributes min_a and max_a, the minimum and maximum number of objects min_o and max_o, and the minimum and maximum δ threshold min_δ and max_δ of the embedded δ-nClusters. Regions in the matrix not covered by any embedded δ-nClusters are filled with random values drawn from a uniform distribution.

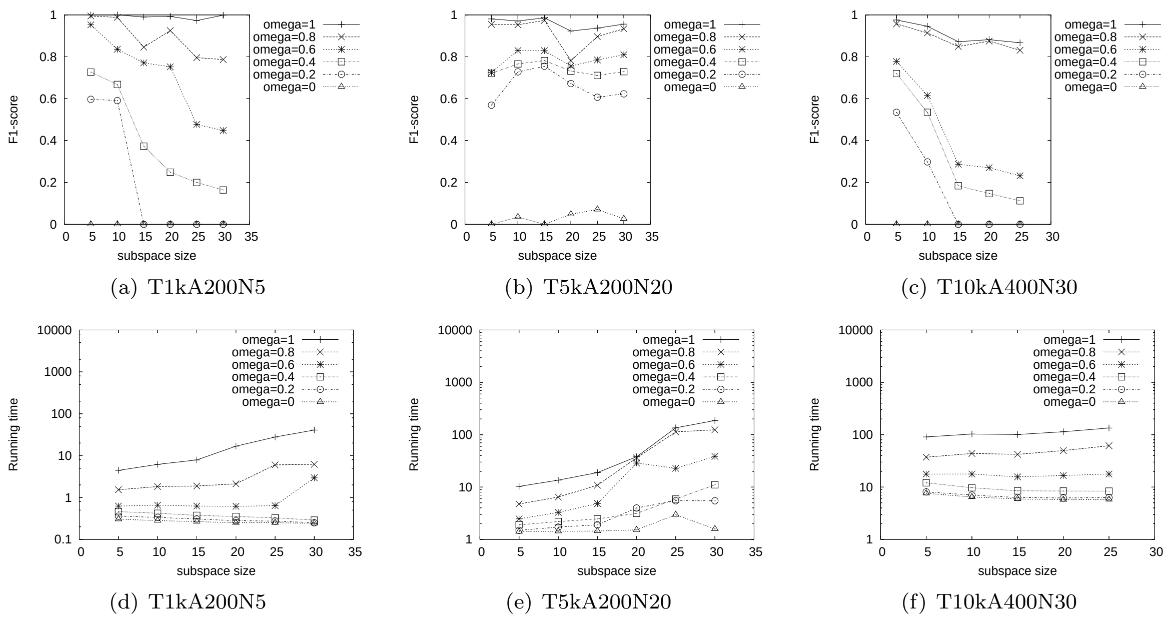
We use one real dataset, the yeast gene expression data used in [34]. It contains 2884 rows and 17 columns, and is available at `http://arep.med.harvard.edu/biclustering/`.

4.2 Performance measures

We use precision and recall to assess the accuracy of different subspace clustering algorithms. In subspace clustering, a cluster is defined by its object set and subspace together. Therefore, we take both object sets and attribute sets of clusters into consideration when defining recall and precision. We call the clusters generated by a subspace clustering algorithm predicted clusters. Given a true subspace cluster $C = (T_C, D_C)$ and a predicted subspace cluster $S = (T_S, D_S)$, where T_C and T_S are the object sets of C and S respectively, and D_C and D_S are the attribute set of C and S respectively, the match between C and S is defined as follows:

$$match(S, C) = |T_C \cap T_S| \times |D_C \cap D_S|$$

Given a set of true clusters $\mathcal{C} = \{C_1, C_2, \cdots, C_n\}$ and a set of predicted clusters $\mathcal{S} = \{S_1, S_2, \cdots, S_m\}$, recall and precision are defined as follows:



(a) T1kA200N5 (b) T5kA200N20 (c) T10kA400N30

(d) T1kA200N5 (e) T5kA200N20 (f) T10kA400N30

Figure 5: Running time and F1-score of MaxnCluster under different ω values

$$Recall(\mathcal{C},\mathcal{S}) = \frac{\sum_{i=1}^{n} \max\{match(C_i,S_j)|j=1,\cdots,m\}}{\sum_{i=1}^{n}|T_{Ci}|\times|D_{Ci}|}$$

$$Precision(\mathcal{C},\mathcal{S}) = \frac{\sum_{j=1}^{m} \max\{match(C_i,S_j)|i=1,\cdots,n\}}{\sum_{j=1}^{m}|T_{Sj}|\times|D_{Sj}|}$$

The accuracy of a clustering algorithm is measured using F1-score, which is defined as

$$F1\text{-}score(\mathcal{C},\mathcal{S}) = \frac{2\cdot Recall(\mathcal{C},\mathcal{S})\cdot Precision(\mathcal{C},\mathcal{S})}{Recall(\mathcal{C},\mathcal{S})+Precision(\mathcal{C},\mathcal{S})}$$

4.3 The effect of the sliding window approach

MaxnCluster uses a sliding window approach to partition every dimensions into overlapping bins. As described in Section 3.1, a parameter ω is used to control the overlap between adjacent bins. If ω=1, there is no constraint on the overlap. Two adjacent bins can overlap as much as they can. If ω=0, the sliding window approach reduces to a grid based approach that partitions every attribute into equal-length bins. In this experiment, we study the effect of the values of parameter ω.

We use three sets of synthetic datasets in this experiment. The first set of datasets, denoted as T1kA200D5, are generated with T=1000, A=200, N=5, min_o=100, max_o=200, min_δ=0.05 and max_δ=0.09, and the minimum and maximum size of subspaces min_a and max_a are varied from 5 to 30. The second set of datasets, denoted as T5kA200N20, are generated with T=5000, A=200, N=20, min_o=300, max_o=500, max_δ=0.03, max_δ=0.05, min_a=5, and the maximum size of subspaces max_a are varied from 5 to 30. The third set of datasets, denoted as T10kA400D30, are generated with T=10000, A=400, N=30, min_o=600, max_o=1000, min_δ=0.03, max_δ=0.05, min_a=5, and the maximum size of subspaces max_a is varied from 5 to 30. We do not allow overlap among clusters on T1kA200D5, but allow clusters to overlap on T5kA200D20 and T10kA400D30. For every parameter setting, we generate 10 datasets, and take the average of the running time and F1-score as the final results.

Figure 5 shows the running time and F1-score of MaxnCluster under different ω values. The other parameters of MaxnClusters are set as follows: ms=min_a, mr=min_o and δ=max_δ. When ω=0, MaxnCluster partitions every attributes to equal-length non-overlapping bins. The F1-score is close to 0. When ω gets larger, the F1-score of MaxnCluster improves. When ω=1, F1-score is close to 1. We inspected the recall and precision of MaxnCluster under different ω values. The precision of MaxnCluster is always close to 1 under different ω values, while recall shows similar trend as F1-score. This indicates the grid-based approach may shatter a cluster into small pieces, and the sliding window approach can preserve clusters. When ω=1, almost all clusters can be preserved. However, this advantage does not come free. Figure 5(d) 5(e) and 5(f) show that the running time of MaxnCluster increases with the increase of ω.

4.4 Comparison with other algorithms

We compare MaxnCluster with MAFIA [25], CFPC [40] and STATPC [23]. MAFIA uses an adaptive approach to partition each attributes into non-overlapping bins. CFPC uses the same cluster definition as MaxnCluster, but it uses a randomized algorithm to find clusters, and it does not allow overlap among clusters. STATPC [23] is a recently proposed algorithm, and it uses statistical significance thresholds to define and find clusters. STATPC is very slow, but it has been shown to have higher accuracy than other clustering algorithms. We obtained STATPC and CFPC from their respective authors, and MAFIA was kindly provided by Gabriela Moise.

We use the same datasets as in the previous experiment. The parameters of the algorithms are set as follows. For STATPC, we set α_0=1.0E-10, α_K=α_H=0.001 as suggested in [23]. For MIFIA, we set β=0.35, no_tiny_bins=50, $no\_intervals\_unif\_distrb$=5. We tried three values for α: 1.2, 1.5 and 2, and picked the setting with the highest accuracy to report the results. For CFPC, we set w=max_δ, α=min_o/T, β=0.25, $maxout$=50, where T is the number of objects in the dataset. For MaxnCluster, we set ms=min_a, mr=min_o and δ=max_δ and ω=1.

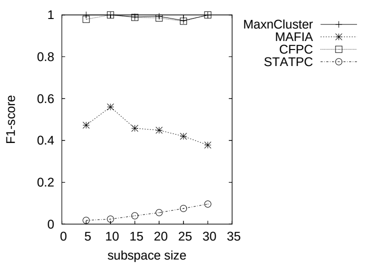
Figure 6 shows the running time and F1-score of the subspace clustering algorithms. STATPC has the lowest F1-score. The recall of STATPC is rather high, but its precision is very low. We inspected the clusters generated by STATPC, and we found that the subspaces of the clusters generated by STATPC is much larger than the actual subspaces of the true clusters. That is why when the subspaces of the cluster get larger, the F1-score of STATPC increases. If we consider only the object sets, the average F1-score of STATPC is around 0.4. We did not get the results of STATPC on the other two datasets because it could not finish mining after more than 10 hours.

MAFIA uses an adaptive approach to partition attributes into non-overlapping bins. Its F1-score is around 0.5, which is much higher than the F1-score of the equal-length partitioning method, but it is still much lower than the F1-score of MaxnCluster. This indicates that partitioning attributes into non-overlapping bins has the risk of shattering clusters into small pieces even when a smart adaptive strategy is used.

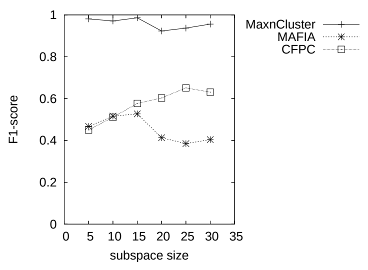
The F1-score of CFPC on dataset T1kA200N5 is close to 1, but CFPC has a much lower F1-score on dataset T5kA200N30. The reason being that CFPC does not allow overlap among clusters, and clusters in T1kA200N5 do not overlap with one another, while clusters on T5kA200N30 have overlaps. CFPC failed to run on dataset T10kA400N30. The F1-score of MaxnCluster is close to 1 on both datasets since it allows overlap among clusters.

Among the several algorithms, STATPC takes the longest time to find clusters. MAFIA is faster than MaxnCluster when the subspaces of the clusters are smaller than 20, but its running time increases rapidly when the size of the cluster subspaces gets larger than 20. CFPC and MaxnCluster show relative stable running time with resepct to the size of cluster subspaces.

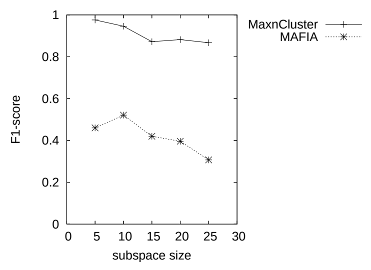
We also compared MaxnCluster with LCM-nCluster which we developed previously [21]. LCM-nCluster first uses frequent closed itemset mining algorithm LCM to mine attribute sets, and then produces the corresponding maximal nClusters in a post-processing step. The parameter settings of LCM-



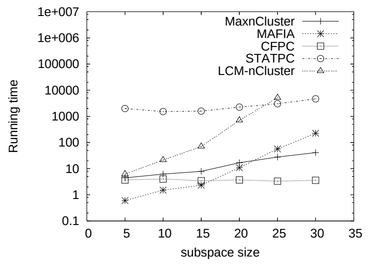
(a) T1kA200N5



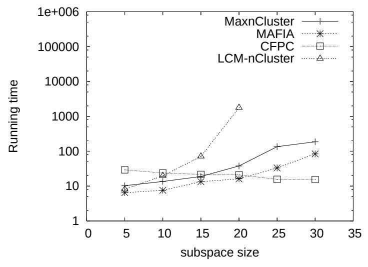
(b) T5kA200N20



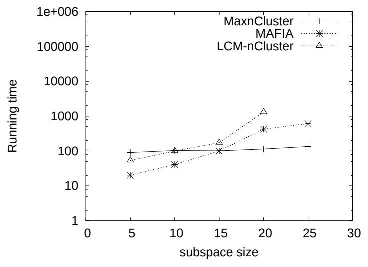
(c) T10kA400N30



(d) T1kA200N5



(e) T5kA200N20



(f) T10kA400N30

Figure 6: Running time and F1-score of different subspace clustering algorithms

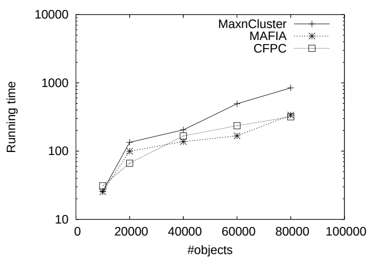
nCluster were set to the same as MaxnCluster. The two algorithms produce the same set of nClusters, so their F1-scores are the same. Both LCM-nCluster and MaxnCluster can prune the nClusters with non-maximal attribute sets completely. LCM-nCluster cannot prune the nClusters with non-maximal object sets during the mining process, while MaxnCluster can prune most of them using the techniques described in Section 3.5. Furthermore, MaxnCluster can generate attribute sets and object sets of clusters simultaneously, while LCM-nCluster have to use a post-processing step to generate object sets of clusters. Therefore, MaxnCluster is times of faster than LCM-nCluster. On some settings, LCM-nCluster could not finish mining because too many clusters were generated.

4.5 Scalability

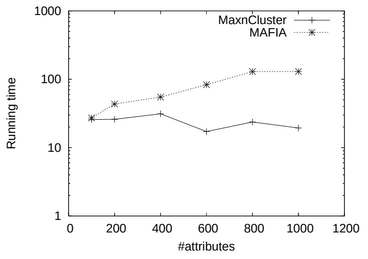
Figure 6 shows that the running time of MaxnCluster, CFPC, STATPC is relatively stable with respect to the size of the subspaces of the embedded clusters, while MAFIA is more sensitive to the dimensions of the clusters. In this experiment, we studied the scalability of the algorithms with respect to the total number of objects and the total number of attributes in datasets. We did not include STATPC in this experiment because it takes too long to finish one run.

The datasets were generated using the following parameters: N=15, min_a=5, max_a=20, min_o=0.06T, max_o=0.1T, min_δ=0.02 and max_δ=0.04. When studying the scalability of the algorithms with respect to the number of objects T, we fixed the number of attributes A at 200, and varied T from 10,000 to 80000. When studying the scalability of the algorithms with respect to the number of attributes A, we fixed T at 10,000, and varied A from 100 to 1000.

Figure 7 shows the running time of MaxnCluster, MAFIA and CFPC. CFPC fail to run when the number of attribute is larger than 200, so we did not study its scalability with respect to the number of attributes. The running time of all the three algorithms increases with the number of objects. MaxnCluster is more sensitive to the number of objects than MAFIA and CFPC, but is less sensitive to the



(a) Varying #objects T



(b) Varying #attributes A

Figure 7: Scalability

number of attributes.

4.6 Performance on the real dataset yeast

In this experiment, we study the accuracy of the several subspace clustering algorithms on real dataset yeast, which contains 2884 yeast genes (objects) and 17 conditions (attributes). Each yeast gene is annotated with one or more GO terms from gene ontology (GO) (`http://www.geneontology.org/`), which indicate the biological processes, cellular components or functions of the genes. The GO terms are organized hierarchically. GO terms at high levels may occur in many genes, and they are too general to be useful. GO terms appearing in very few genes are also not very useful. In our experiments, we select only informative GO terms. A GO term is informative if itself occurs in at least 30 genes, but none of its children appears in at least 30 genes [41]. Genes with same functions tend to have similar gene expression profiles, so here we select only functional GO terms in our study, and 42 informative functional GO terms are selected using the method described above.

We regard each group of genes that are annotated with the same informative GO term as a cluster. Since the subspaces of the clusters are unknown, we consider only the object sets of clusters when calculating precision and recall. Given a true subspace cluster $C = (T_C, D_C)$ and a predicted subspace cluster $S = (T_S, D_S)$, the match between C and S is defined as follows:

$$match_o(S, C) = |T_C \cap T_S|$$

Given a set of true clusters $\mathcal{C} = \{C_1, C_2, \cdots, C_n\}$ and a set of predicted clusters $\mathcal{S} = \{S_1, S_2, \cdots, S_m\}$, recall and precision are defined as follows:

$$Recall_o(\mathcal{C}, \mathcal{S}) = \frac{\sum_{i=1}^{n} \max\{match_o(C_i, S_j)|j = 1, \cdots, m\}}{\sum_{i=1}^{n} |T_{Ci}|}$$

$$Precision_o(\mathcal{C}, \mathcal{S}) = \frac{\sum_{j=1}^{m} \max\{match_o(C_i, S_j)|i = 1, \cdots, n\}}{\sum_{j=1}^{m} |T_{Sj}|}$$

The parameters of the algorithms are set as follows. For STATPC, we set α_0=1.0E-10, α_K=α_H=0.001. For MIFIA, we set *no_tiny_bins*=50, *no_intervals_unif_distrb*=5, and we tried three values for α: 1.2, 1.5 and 2.0, and three values for β: 0.35, 0.6, 0.9. For CFPC, we set α=0.01, β=0.25, *maxout*=50, and tried five values for w: 0.02, 0.04, 0.06, 0.08, 0.1. For MaxnCluster, we set mr=29 (0.01), ms=4, ω=0.5 and tried five values for δ=0.02, 0.04, 0.06, 0.08, 0.1. For each algorithm, we pick its best results. MaxnCluster generates many overlapped clusters. Here we consider only object sets of clusters. The object set of one cluster may be a proper subset of the object set of another cluster, but it is discovered in a higher subspace. In such case, we remove the cluster.

We also include the Biclustering algorithm [14] in this experiment, which models biclusters as submatrices in gene expression data that have low mean squared residue scores, and uses a greedy algorithm to find biclusters. We obtained the 100 biclusters generated by the Biclustering algorithm from `http://arep.med.harvard.edu/biclustering/`. These 100 biclusters are generated using parameters estimated from some prior known clusters [14].

Algorithm	recall	precision	F1-score
Biclustering	0.388	0.114	0.176
STATPC	0.507	0.101	0.168
MAFIA	0.515	0.111	0.183
CFPC	0.288	0.130	0.179
MaxnCluster	0.392	0.126	0.191

Table 5: Accuracy of different algorithms on yeast dataset

Table 5 shows the recall, precision and F1-score of the several subspace clustering algorithms on the yeast dataset. The several algorithms show comparable F1-score. Most of the clusters generated by STATPC contain more than 1000 genes. That is why STATPC has higher recall and lower precision than other algorithms. CFPC has the highest precision among all the algorithms, while MaxnCluster has the highest F1-score. The best result of CFPC is achieved when w=0.1, and the best result of MaxnCluster is achieved when δ=0.1. The F1-scores of all the algorithms are not high, which indicates that it is still a challenging task to find subspace clusters on real gene expression datasets. The noisy nature of gene expression data also make the problem harder. It is also possible that some clusters generated by the algorithms do not represent functional groups, but they may have other biological implications.

5 Related Work

High dimensionality poses great challenges on several problems such as clustering, nearest neighbor search, and indexing [10, 17, 2, 3]. One solution to tackle high dimensionality is dimensionality reduction. The major drawback of dimensionality reduction is that the transformed attributes often have no intuitive meaning any more and thus the resulting clusters are often hard to interpret [27]. Recent research work has focus on finding clusters in subspaces of high dimensional datasets, and a number of surveys [22, 27] reviewed and compared these algorithms.

Different subspace clustering models have been proposed, including distance based model [4, 5, 35], density based model [6, 13, 25, 12, 20, 30, 18] and coherent pattern based model [37, 34, 29, 36]. No one clustering model is better than the others, but some are more appropriate for certain problems [27]. Domain specific knowledge is often very helpful in determining which type of cluster formation is the most appropriate.

5.1 Density and grid based approaches

Density based model aims to find regions of high density in subspaces that are separated by regions of lower density. CLIQUE [6], ENCLUS [13], MAFIA [25], CBF [12] and CLTree [20] are density and grid based subspace clustering algorithms. They discretize the data space into non-overlapping rectangular cells by partitioning each dimension to a number of bins, and then use an apriori-style search to find overlapping clusters. Both CLIQUE and ENCLUS use a static sized grid to divide each dimension into bins. The other algorithms use data driven strategies to determine the cut-points for each dimension. MAFIA and CBF use histograms to analyze the density of data in each dimension. CLTree uses a decision tree based strategy. Another density based clustering algorithm SUBCLU [18] does not use the grid-based approach. It calls the DBSCAN algorithm [15] to find clusters of arbitrary shapes in individual subspaces in a level-wise manner, which may be very costly.

The density based algorithms typically use a global density threshold to ensure anti-monotonic properties for efficient search. However, they ignore that density decreases with dimensionality. Large density threshold will result in only low-dimensional clusters, whereas small density threshold will result in a larger number of clusters, many of which are meaningless [23]. To overcome this problem, SCHISM [32] uses a non-linear monotonically decreasing density threshold, which does not guarantee mining all interesting subspaces. DUSC [8] uses an unbiased density measure by taking the expected density for subspaces into account. However, this unbiased density measure do not have the anti-monotone property for efficient search. As a solution, DUSC uses a relaxed threshold, which equals to a global density threshold.

FRIES [19] uses a filter-and-refinement approach to detect clusters of different densities, but it may not be able to detect all clusters. Moise et al. [23] propose an algorithm called STATPC which takes statistical significance thresholds instead of density as input parameters. A greedy algorithm is used to extract a non-redundant set of statistically significant clusters.

The main difference between the nCluster model proposed in this paper and the density and grid based approaches is on the definition of clusters. In density based model, two objects in the same cluster can be far apart from each other even on the subspace of the cluster, and they are in the same cluster because they are connected by surrounding objects. In the nCluster model, every pair of objects are close to each other on every dimension of the subspaces. In some sense, the density based model can be viewed as average-link clustering, and the nCluster model can be viewed as complete-link clustering. Furthermore, grid based approaches divide the domain of every dimension into non-overlapping bins, here we use a sliding-window approach to preserve signficant clusters.

5.2 Distance and partition based approaches

PROCLUS [4], ORCLUS [5], and FINDIT [35] are distance and partition based subspace clustering algorithms. They use L_p norm or variants of L_p norm as distance measure, and use a top-down strategy to find non-overlapping clusters. They start by finding an initial approximation of the clusters in the full dimension space with equally weighted dimensions, and then each dimension is assigned a weight for each cluster. The updated weights are then used in the next iteration to regenerate the clusters. The top-down algorithms require two parameters: the number of clusters and the average size of subspaces, which are often difficult to decide and are also critical to the performance of the algorithms [27].

Many improvements have been made on the basis of PROCLUS and ORCLUS. DiSH [1] can detect clusters in subspaces of significantly different dimensionality, and it also uncovers complex hierarchies of nested subspace clusters. HARP [38] does not take input parameters. It automatically selects relevant dimensions using a agglomerative hierarchical approach. SSPC [39] is a semi-supervised algorithm and it aims to find clusters in extremely low-dimensional subspaces. PreDeCon [11] first generates the preference dimensions for each data point, and then finds density connected subspace clusters. EPCH [26] uses histograms to identify dense regions in subspaces. To construct k-D histograms, EPCH needs to consider C_n^k combination of attributes, where n is the number of attributes. Therefore, EPCH is not scalable with respect to the dimensionality of histograms and number of attributes. P3C [24] first generates cluster cores, which are defined as regions of the data space containing an unexpectedly high number of points, in an Apriori-like fashion, and then refine them into projected clusters. DOC/FASTDOC [30] and FPC/CFPC [40] use the same cluster definition as the nCluster model, but both algorithms are randomized algorithms based on sampling, so they cannot guarantee to find all the clusters satisfying the definition.

The main difference between the nCluster model and the distance and partition based approaches such as PROCLUS and ORCLUS is that the nCluster model allows overlap among clusters, while the partition based approaches do not allow overlap. The distance measures used are also different. PROCLUS, ORCLUS and HARP use L_1, L_2 norms or their variants as distance measure, while nCluster uses L_∞ norm. DOC and CFPC use the same cluster definition as the nCluster model, but both algorithms use randomized algorithms based on sampling to find clusters, thus they cannot guarantee to find all the clusters satisfying the definition. Furthermore, the distance between objects in the clusters they generated can be as large as 2θ instead of θ. DOC randomly picks some points as seeds and then uses a randomized approach to find the best projected cluster around a random seed. CFPC [40] replaces the randomized module in DOC with systematic search for the best cluster of a random seed using frequent itemset mining techniques. The MaxnCluster algorithm proposed in this paper takes one step further by searching for the best cluster of all the objects simultaneously and systematically, and it can find all the clusters satisfying the definition.

5.3 Pattern based clustering and biclustering

Pattern-based clustering methods such as δ-Clusters [37] and pClusters [34, 29] find clusters of objects that exhibit coherent patterns in subspaces rather than objects close to each other in the subspaces. Yang et al. [37] use a randomized process to find δ-clusters. Wang et al. [34] enumerate all pClusters using a levelwise approach. Pei et al. [29] improved the work of Wang et al. by mining only maximal pClusters. Cheng et al. [14] model biclusters as submatrices in gene expression data that have low mean squared residue scores, and use a greedy algorithm to find biclusters.

The main difference between the nCluster model and the pattern-based clustering model is the definition of clusters. In pattern based clustering and biclustering, rows and columns are treated equally, and they aim to find groups of objects that exhibit coherent patterns on subsets of attributes, instead of groups of objects that are close to one another on subsets of attributes. There is some connections between the nCluster model and the pCluster model. Given a set of objects T and a set of attributes D, T and D form a δ-pCluster if for every two objects $x, y \in T$, and every two attributes $a, b \in D$, we have $|(v_{xa} - v_{xb}) - (v_{ya} - v_{yb})| \leq \delta$. Assume all the attributes have the same value range of 1. If (T, D) is a δ-nCluster, then it must be a 2δ-pCluster because $|(v_{xa}-v_{xb})-(v_{ya}-v_{yb})| \leq |v_{xa}-v_{xb}|+|v_{ya}-v_{yb}| \leq 2\delta$. We can use the algorithm for mining 2δ-pClusters to mine δ-nClusters. However, this approach is very inefficient because we have to use a larger threshold to mine pClusters and a 2δ-pCluster may contain many objects that are not δ-neighbors. In our previous work [21], we have shown that using MaPle [29] to mine nClusters is very slow.

6 Conclusion

In this paper, we have proposed a new subspace clustering model called nCluster to find groups of similar objects embedded in subspaces of high dimensional datasets. The nCluster model uses a sliding window approach to partition the attributes into overlapping bins, which can preserve clusters that are shattered by the grid-based approach. An efficient algorithm MaxnCluster has been developed to generate maximal nClusters. Our performance study shows that MaxnCluster has high accuracy and is efficient in mining maximal nClusters.

MaxnCluster allows overlap among different bins of attributes. As a result, MaxnCluster may generate many highly-overlapping clusters with similar subspaces. In the future, we plan to remove or merge the highly-overlapped clusters and present a concise set of clusters to users.

Acknowledgements

We would like to thank Gabriela Moise for providing us with her STATPC algorithm and algorithm MAFIA, and Dr. Man Lung Yiu for providing us with his CFPC algorithm for our performance study.

References

[1] E. Achtert, C. Böhm, H.-P. Kriegel, P. Kröger, I. Müller-Gorman, and A. Zimek. Detection and visualization of subspace cluster hierarchies. In *Proc. of the 12th DASFAA conference*, pages 152–163, 2007.

[2] C. C. Aggarwal. Re-designing distance functions and distance-based applications for high dimensional data. *SIGMOD Record*, 30(1):13–18, 2001.

[3] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. On the surprising behavior of distance metrics in high dimensional spaces. In *Proc. of the 8th ICDT Conference*, pages 420–434, 2001.

[4] C. C. Aggarwal, C. M. Procopiuc, J. L. Wolf, P. S. Yu, and J. S. Park. Fast algorithms for

projected clustering. In *Proc. of the 1999 ACM SIGMOD Conference*, pages 61–72, 1999.

[5] C. C. Aggarwal and P. S. Yu. Finding generalized projected clusters in high dimensional spaces. In *Proc. of the 2000 ACM SIGMOD Conference*, pages 70–81, 2000.

[6] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proc. of the 1998 ACM SIGMOD Conference*, pages 94–105, 1998.

[7] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *Proc. of the 1993 ACM SIGMOD Conference*, pages 207–216, 1993.

[8] I. Assent, R. Krieger, E. Müller, and T. Seidl. Dusc: Dimensionality unbiased subspace clustering. In *ICDM*, pages 409–414, 2007.

[9] R. Bellman. *Adaptive Control Processes: A Guided Tour.* Princeton University Press, 1961.

[10] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *Proc. of the 7th ICDT Conference*, pages 217–235, 1999.

[11] C. Böhm, K. Kailing, H.-P. Kriegel, and P. Kröger. Density connected clustering with local subspace preferences. In *Proc. of the 4th IEEE International Conference on Data Mining*, pages 27–34, 2004.

[12] J.-W. Chang and D.-S. Jin. A new cell-based clustering method for large, high-dimensional data in data mining applications. In *Proc. of the 2002 ACM symposium on Applied computing*, pages 503–507, 2002.

[13] C. H. Cheng, A. W.-C. Fu, and Y. Zhang. Entropy-based subspace clustering for mining numerical data. In *Proc. of the 5th ACM SIGKDD Conference*, pages 84–93, 1999.

[14] Y. Cheng and G. M. Church. Biclustering of expression data. In *Proc. of the 8th International Conference on Intelligent Systems for Molecular Biology*, pages 93–103, 2000.

[15] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of the 2nd ACM SIGKDD Conference*, pages 226–231, 1996.

[16] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. of the 2000 ACM SIGMOD Conference*, pages 1–12, 2000.

[17] A. Hinneburg, C. C. Aggarwal, and D. A. Keim. What is the nearest neighbor in high dimensional spaces? In *Proc. of the 26th VLDB Conference*, pages 506–515, 2000.

[18] K. Kailing, H.-P. Kriegel, and P. Kröger. Density-connected subspace clustering for high-dimensional data. In *Proc. of the 4th SIAM International Conference on Data Mining*, 2004.

[19] H.-P. Kriegel, P. Kröger, M. Renz, and S. Wurst. A generic framework for efficient subspace clustering of high-dimensional data. In *Proc. of the 5th IEEE International Conference on Data Mining*, pages 250–257, 2005.

[20] B. Liu, Y. Xia, and P. S. Yu. Clustering through decision tree construction. In *Proc. of the 9th CIKM conference*, pages 20–29, 2000.

[21] G. Liu, J. Li, K. Sim, and L. Wong. Distance based subspace clustering with flexible dimension partitioning. In *Proc. of the 23rd ICDE Conference*, pages 1250–1254, 2007.

[22] S. C. Madeira and A. L. Oliveira. Biclustering algorithms for biological data analysis: A survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 01(1):24–45, 2004.

[23] G. Moise and J. Sander. Finding non-redundant, statistically significant regions in high dimensional data: a novel approach to projected and

subspace clustering. In *Proc. of the 14th ACM SIGKDD Conference*, pages 533–541, 2008.

[24] G. Moise, J. Sander, and M. Ester. P3c: A robust projected clustering algorithm. In *Proc. of the 6th IEEE International Conference on Data Mining*, pages 414–425, 2006.

[25] H. Nagesh, S. Goil, and A. Choudhar.

[26] E. K. K. Ng, A. W.-C. Fu, and R. C.-W. Wong. Projective clustering by histograms. *IEEE Transactions on Knowledge and Data Engineering*, 17(3):369–383, 2005.

[27] L. Parsons, E. Haque, and H. Liu. Subspace clustering for high dimensional data: a review. *SIGKDD Exploration Newsletter*, 6(1):90–105, 2004.

[28] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Proc. of the 7th ICDT Conference*, pages 398–416, 1999.

[29] J. Pei, X. Zhang, M. Cho, H. Wang, and P. S. Yu. Maple: A fast algorithm for maximal pattern-based clustering. In *Proc. of the 3rd ICDM Conference*, pages 259–266, 2003.

[30] C. M. Procopiuc, M. Jones, P. K. Agarwal, and T. M. Murali. A monte carlo algorithm for fast projective clustering. In *Proc. of the 2002 ACM SIGMOD Conference*, pages 418–427, 2002.

[31] R. Rymon. Search through systematic set enumeration. In *Proc. of the Internation Conference on Principles of Knowledge Representation and Reasoning*, 1992.

[32] K. Sequeira and M. J. Zaki. Schism: A new approach for interesting subspace mining. In *Proc. of the 4th IEEE International Conference on Data Mining*, pages 186–193, 2004.

[33] T. Uno, M. Kiyomi, and H. Arimura. Lcm ver. 3: Collaboration of array, bitmap and prefix tree for frequent itemset mining. In *Proc. of the ACM SIGKDD OSDM workshop*, 2005.

[34] H. Wang, W. Wang, J. Yang, and P. S. Yu. Clustering by pattern similarity in large data sets. In *Proc. of the 2002 ACM SIGMOD Conference*, pages 394–405, 2002.

[35] K.-G. Woo, J.-H. Lee, M.-H. Kim, and Y.-J. Lee. Findit: a fast and intelligent subspace clustering algorithm using dimension voting. *Information and Software Technology*, 46(4):255–271, 2004.

[36] X. Xu, Y. Lu, A. K. H. Tung, and W. Wang. Mining shifting-and-scaling co-regulation patterns on gene expression profiles. In *Proc. of the 22nd ICDE Conference*, 2006.

[37] J. Yang, W. Wang, H. Wang, and P. S. Yu. δ-clusters: Capturing subspace correlation in a large data set. In *Proc. of the 18th IEEE ICDE Conference*, pages 517–528, 2002.

[38] K. Y. Yip, D. W. Cheung, and M. K. Ng. Harp: A practical projected clustering algorithm. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1387–1397, 2004.

[39] K. Y. Yip, D. W. Cheung, and M. K. Ng. On discovery of extremely low-dimensional clusters using semi-supervised projected clustering. In *Proc. of the 21st ICDE conference*, pages 329–340, 2005.

[40] M. L. Yiu and N. Mamoulis. Iterative projected clustering by subspace mining. *IEEE Transactions on Knowledge and Data Engineering*, 17(2):176–189, 2005.

[41] X. Zhou, M. C. Kao, and W. H. Wong. Transitive functional annotation by shortest-path analysis of gene expression data. *PNAS*, 99(20):12783–8, 2002.