Iterative Projected Clustering by Subspace Mining

Man Lung Yiu and Nikos Mamoulis

Abstract—Irrelevant attributes add noise to high-dimensional clusters and render traditional clustering techniques inappropriate. Recently, several algorithms that discover projected clusters and their associated subspaces have been proposed. In this paper, we realize the analogy between mining frequent itemsets and discovering dense projected clusters around random points. Based on this, we propose a technique that improves the efficiency of a projected clustering algorithm (DOC). Our method is an optimized adaptation of the frequent pattern tree growth method used for mining frequent itemsets. We propose several techniques that employ the branch and bound paradigm to efficiently discover the projected clusters. An experimental study with synthetic and real data demonstrates that our technique significantly improves on the accuracy and speed of previous techniques.

Index Terms—Database management, database applications, clustering, classification, and association rules.

1 Introduction

Clustering partitions a collection of objects S into a set of groups (i.e., clusters), such that the similarity between objects of the same group is high and objects from different groups are dissimilar. Clustering finds many applications in marketing (e.g., customer segmentation), image analysis, bioinformatics, document classification, indexing, etc. In many such applications, the objects to be clustered are represented by points in a high-dimensional space, where each dimension corresponds to an attribute/feature and the feature value of each object determines its coefficient in the corresponding dimension. A distance measure (e.g., Euclidean distance) between two points is used to measure the dissimilarity between the corresponding objects.

Beyer et al. [6] have shown that the distance of any two points in a high-dimensional space is almost the same for a large class of common distributions. On the other hand, the widely used distance measures are more meaningful in subsets (i.e., *projections*) of the high-dimensional space, where the object values are dense [10]. In other words, it is more likely for the data to form dense, meaningful clusters in a dimensional subspace [3].

Fig. 1 shows how irrelevant attributes can affect clustering. In this example, four records T_1–T_4 are to be clustered. If we consider all attributes and Manhattan distance (i.e., L_1) as the similarity metric, then T_2 and T_3 are likely to be placed in the same cluster, since their distance (i.e., 90) is the smallest compared to the distances between any other pair. However, we can see that there are two natural clusters: $C_1 = \{T_1, T_2\}$, where $\{a_1, a_2, a_3\}$ are relevant attributes and $\{a_4, a_5\}$ are noise attributes, and $C_2 = \{T_3, T_4\}$, where $\{a_3, a_4, a_5\}$ are relevant and $\{a_1, a_2\}$ are noise.

Therefore, a new class of *projected clustering* methods (also called *subspace clustering* methods) [1], [2], [3], [12] have emerged, whose task is to find 1) a set of clusters $\mathcal{C}$, and 2) for each cluster $C_i \in \mathcal{C}$, the set of dimensions D_i that are relevant to C_i. For instance, the projected clusters in the data set of Fig. 1 are $(C_1, D_1) = (\{T_1, T_2\}, \{a_1, a_2, a_3\})$ and $(C_2, D_2) = (\{T_3, T_4\}, \{a_3, a_4, a_5\})$. Not only do projected clustering methods disregard the noise induced by irrelevant dimensions, but they also provide more sound descriptions for the clusters. For example, the interpretation of a cluster involving only a few dimensions is much easier than that of a cluster with hundreds of dimensions.

CLIQUE [3] is one of the first known projected clustering algorithms. It works in a level-wise manner, exploring k-dimensional projected clusters after clusters of dimensionality $k-1$ have been discovered. PROCLUS [1] is a medoid-based projected clustering algorithm that improves the scalability of CLIQUE by selecting a number of good candidate medoids and exploring the clusters around them. It takes two parameters: the number k of clusters and the average dimensionality l of each cluster. Initially, a number ($> k$) of medoids, such that every pair of medoids are far from each other in the full-dimensional space, are greedily selected. Then, a random subset of k medoids is picked. Points near the medoids are used to determine the subspaces of the clusters. After the subspaces have been determined, each point is assigned to the cluster of the nearest medoid. A medoid is *bad* if the corresponding cluster has a smaller size than a predefined density threshold. Bad medoids are iteratively replaced by other candidates until the algorithm converges to a set of good medoids and clusters. PROCLUS can be fast, but it is not effective when the sizes of the clusters have a large variance, since it is likely that many medoids are chosen from large clusters and few or no medoids are chosen from small ones. In this case, large clusters may be split and small clusters may be missed (i.e., regarded as "outliers").

- *M.L. Yiu is with the Department of Computer Science and Information Systems at the University of Hong Kong, Room 504, Haking Wong Building, Pokfulam Road, Hong Kong. E-mail: mlyiu2@csis.hku.hk.*
- *N. Mamoulis is with the Department of Computer Science and Information Systems at the University of Hong Kong, Room 403, Chow Yei Ching Building, Pokfulam Road, Hong Kong. E-mail: nikos@csis.hku.hk.*

Manuscript received 29 Sept. 03; revised 22 Jan. 04; accepted 18 June 04. For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0191-0903.

	a_1	a_2	a_3	a_4	a_5
T_1	0	0	0	10	100
T_2	0	0	0	70	30
T_3	20	10	20	50	50
T_4	80	80	20	50	50

Fig. 1. Example of a projected clustering problem.

Another problem with PROCLUS is that it requires the projected clusters to have similar dimensionality (l on the average). Even in this case, setting an appropriate value for l is not trivial. ORCLUS [2] is an extension of PROCLUS that can select relevant attributes from the set of arbitrarily directed orthogonal vectors (in a transformed space). ORCLUS can discover arbitrarily oriented clusters; however, these clusters may be difficult to interpret.

DOC [12] is an algorithm that performs iterative (greedy) projected clustering. With this approach, a random point p is selected from the data set S. Then, the best projected cluster C that contains p is discovered. After this process has been repeated for a number of times, the best medoid p and its associated dimensional subspace D are chosen. All points near p in the subspace D are removed from S and added into a cluster C. The process is iteratively repeated for the remaining points $S - C$ until all projected clusters have been found. DOC overcomes some deficiencies of PROCLUS, since 1) it can automatically discover the number k of clusters, and 2) it can discover a set of clusters with large size differences.

A core module of DOC is the identification of the best projected cluster around a given random point p. For this, DOC selects a random small sample X from S and finds all dimensions D where the distance of p from all points in X is bounded by a small number w. A candidate cluster C for p is then defined by 1) the set of dimensions D, and 2) all points in S within distance w from p in all dimensions D. A good projected cluster for p is one that has many points around p and as many as possible relevant dimensions. By using many samples X, the best candidate cluster C_p is chosen for p. Although with sampling we can avoid exhaustively searching for the best projected cluster that contains p, we may need to try a large number of samples until we identify the cluster.

In this paper, we propose a technique that replaces this randomized module DOC with systematic search for the best cluster defined by a random point p. We model each point q of S as an itemset that includes the dimensions in which q is close to p. Intuitively, a large, frequent itemset under this model corresponds to a projected cluster of high-dimensionality and many points. To discover the best cluster systematically, we adapt a mining frequent itemsets technique [9]. A frequent itemset in our setting corresponds to a subspace and its support corresponds to the size of the cluster in this subspace. Since we need not discover all itemsets, but only the one with the highest support and dimensionality, our technique employs branch-and-bound to prune early parts of the search space that cannot include a better cluster to the one we have discovered so far.

Several optimization techniques are proposed for our approach. We show the benefits of finding clusters concurrently for many medoids. We also show how we can combine our method for sampling in order to cope with large data sets. Finally, we discuss several heuristics that can refine and improve the quality of the discovered clusters. A comparison of our method with DOC and PROCLUS under several problem settings on synthetic and real data sets demonstrates its effectiveness and efficiency.

The rest of the paper is organized as follows: Section 2 describes in detail projected algorithms highly relevant to the proposed algorithm and reviews efficient techniques for mining frequent itemsets. Our methodology is presented in Section 3. Section 4 presents a comprehensive experimental comparison between projected clustering techniques. Finally, Section 5 concludes the paper and discusses issues for future work.

2 Background and Related Work

In this section, we formally define the projected clusters to be discovered from a large collection of high-dimensional points. We also review in detail previous work that is highly related to this research; we describe DOC [12], an algorithm that discovers projected clusters iteratively using randomized techniques and an efficient method for mining frequent itemsets [9].

2.1 Problem Definition

Let S be a collection of $|S|$ d-dimensional points $p = (p_1, p_2, \ldots, p_d)$ in $\mathbb{R}^d$. A *projected* cluster in S is a pair (C, D) where C is a subset of points and D is a subset of dimensions. A projected cluster must be *dense*. Specifically, the distance between every two points p and q in C in every dimension $i \in D$ must be at most w, where w is a problem parameter. Moreover, the size $|C|$ of cluster C (i.e, number of points in C) should be at least $\alpha|S|$, where $0 \leq \alpha \leq 1$ is a parameter. Formally:

Definition 1 (adopted from [12]). *Let S be a set of points in $\mathbb{R}^d$. Given a set of points $C \subseteq S$, a set of dimensions D, and parameters $0 < \alpha \leq 1$ and $w > 0$, (C, D) is a projected cluster if*

- $|C| \geq \alpha|S|$,
- $\forall i \in D, \max_{p \in C} p_i - \min_{q \in C} q_i \leq w$, *and*
- $\forall i \notin D, \max_{p \in C} p_i - \min_{q \in C} q_i > w$.

2.2 A Monte-Carlo Algorithm for Projected Clustering

DOC [12] is a density-based algorithm that iteratively discovers projected clusters in a data set S that conform to Definition 1. DOC discovers one cluster at a time. At each step, it tries to guess a good medoid for the next cluster to be discovered. It repeatedly picks a random point p from the database S and attempts to discover the cluster centered at p. For this, it runs an inner loop that selects a *discriminating set* $X \subset S$ to determine the bounded and unbounded dimensions for the projected cluster. A set of dimensions D, where *all* points in X are within distance w from p is selected. Then, a cluster C for X is *approximated* by a bounding box $\mathcal{B}_{p,D} = [l_1, h_1] \times [l_2, h_2] \times \cdots [l_d, h_d]$, where $[l_i, h_i] = [p_i - w, p_i + w]$ for $i \in D$ or $[l_i, h_i] = [-\infty, \infty]$ otherwise. C is defined by the set of points from S in $\mathcal{B}_{p,D}$. The process is repeated for a number of random points p and discriminating sets X for each p. Among all discovered C,

```
Algorithm DOC(S, α, β)
1    r := log(2d)/log(1/2β);
2    m := (2/α)^r ln 4;
3    for i:=1 to 2/α
4        Choose p ∈ S at random;
5        for j:=1 to m
6            Choose X ⊆ S of size r at random;
7            D:={k||q_k − p_k| ≤ w, ∀q ∈ X};
8            C:=S ⋂ B_{p,D};
9            if (|C| < α · |S|) then
10               (C, D):=(∅, ∅);
11   return cluster (C, D) that maximizes
     the μ value over all computed clusters;
```

Fig. 2. The DOC algorithm.

```
Algorithm FASTDOC(S, α, β)
1    r := log(2d)/log(1/2β);
2    m := min(MAXITER, (2/α)^r ln 4);
3    for i:=1 to 2/α
4        Choose p ∈ S at random;
5        for j:=1 to m
6            Choose X ⊆ S of size r at random;
7            D:={k||q_k − p_k| ≤ w, ∀q ∈ X};
8            if (|D| ≥ |D_M|) then
9                D_M:=D;
10           if (|D_M| ≥ d_0) then
11               go to 12;
12   C:=S ⋂ B_{p,D_M};
13   return cluster (C, D_M);
```

Fig. 3. The FASTDOC algorithm.

the cluster with the highest *quality* is finally selected. Thus, DOC does not discover all possible clusters according to Definition 1, but attempts to find the best ones conforming to the definition using randomized techniques.

Definition 2 (adopted from [12]). *Let a be the number of points in a projected cluster C. Let b be the dimensionality of C. The* quality *of cluster C is defined by*

$$\mu(a, b) = a \cdot (1/\beta)^b, \tag{1}$$

where $\beta \in (0, 1]$ reflects the importance of the size of the subspace over the size of the cluster.

We can realize the importance of β through the equality $\mu(a, b) = \mu(\beta \cdot a, b + 1)$. Large β favors large clusters with small subspaces over small ones of high-dimensionality and vice-versa. This function is monotonic to both a and b. Fig. 2 (duplicated from [12]) describes the DOC algorithm. After a cluster C is discovered, DOC is called for $S - C$ to discover the next cluster, and the process continues until no further good clusters can be discovered.

In Fig. 2, the number $2/\alpha$ of outer loops and the number $m = (\frac{2}{\alpha})^r \ln 4$ of inner loops are tuned by the analysis in [12]. Observe that DOC can be quite expensive if the data set is large and/or the data dimensionality is high. FASTDOC [12] is a variant of DOC with reduced time complexity, described in Fig. 3 (duplicated from [12]), which uses three heuristics to reduce the search time. First, the number of inner iterations is upper bounded by *MAXITER*. In [12], *MAXITER* is tuned to $\min\{d^2, 10^6\}$. This value is much smaller than the original number of iterations $(2/\alpha)^r$ in DOC; however, it is still quite large. The second heuristic is more significant: The size of each candidate cluster is not verified in the inner loop, thus the database needs not be scanned in every inner iteration. Instead, from each sample, only the cluster subspace is computed and the subspace D_M with the largest number of dimensions is determined from all samples. This subspace is used later (lines 12–13) to determine the candidate cluster C by scanning the database only once. The third optimization (lines 10–11) quits the loops once a sufficiently large subspace (i.e., of at least d_0 dimensions, where d_0 is a user threshold) has been discovered.

Even with these optimizations, FASTDOC may still be quite expensive, since it relies on choosing a good discriminating set X with all its points being within distance w in a sufficiently large number of dimensions. The set of all possible discriminating sets is very large and FASTDOC only examines a very small portion of it. Due to its randomized nature, FASTDOC cannot guarantee the subspace with highest μ value can be found. Our motivation is to develop a deterministic method which can always find the subspace with the highest μ value for the projected cluster around a medoid p. As we will discuss in Section 3, this can be achieved by extending algorithms for mining frequent itemsets. In the next paragraph, we review in detail a data mining algorithm which we will adapt for our purpose.

2.3 Mining Frequent Itemsets

Discovery of frequent itemsets in transactional databases is a well-studied data mining topic. Given a database of transactions $\mathcal{D}$ and a collection of items $\mathcal{L}$, an itemset $\mathcal{I} \subseteq \mathcal{L}$ is *frequent* if the number of transactions in $\mathcal{D}$ that contain $\mathcal{I}$ (i.e., the *support* of $\mathcal{I}$) is at least min_sup. Apriori [4] is a classic algorithm for mining frequent itemsets. First, it finds all frequent 1-itemsets.[1] At each step, it generates candidate k-itemsets by joining $(k-1)$-itemsets and counts the supports of them in the database. Apriori is expensive in mining large (frequent) itemsets, where the supports of a huge number of candidates are counted at each level.

FP–growth [9] is a mining algorithm that avoids candidate generation and counting (i.e., the bottleneck of Apriori) and it is especially appropriate when there are large frequent itemsets in the database. First, the data set is scanned once to find all frequent 1-itemsets. These are sorted in descending order of their frequencies. Then, the data set is scanned again to construct an *FP–tree*, as shown in Fig. 4a. The FP–tree is a compact data structure that stores all patterns which appear in the database. For each transaction, the frequent items (in descending frequency order) are inserted as a path in this tree, where each node corresponds to an item and paths with

1. k-itemset denotes a set with k items.

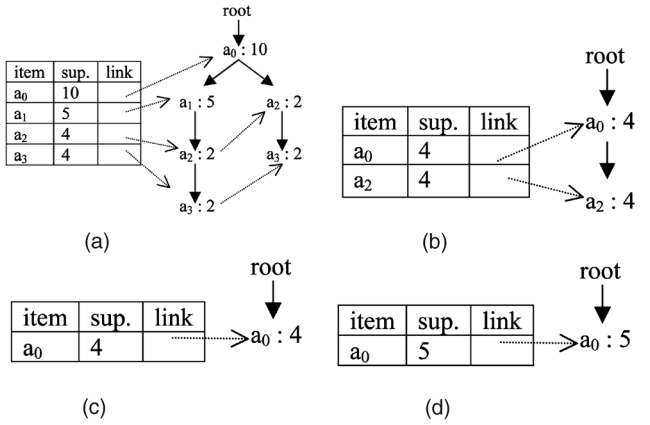

Fig. 4. Mining itemsets from FP-tree. (a) Global FP–tree. (b) Conditional a_3 tree. (c) Conditional a_2 tree. (d) Conditional a_1 tree.

common prefixes are compressed. An auxiliary structure, called *header table*, is used to link nodes with the same label. An algorithm, called FP–growth, is used to mine frequent patterns from the FP–tree as shown in Fig. 4. At each step, an item is picked from the table and all frequent patterns containing it are discovered. Therefore, we first find all patterns containing a_3, then the ones that contain a_2 (but not a_3, since we have already found these), then those that contain a_1, but not a_2 or a_3, etc.

Assume, for example, that $min\_sup = 4$. First, frequent patterns containing item a_3 are extracted. For this, all paths that end at this item (called the *conditional pattern base* for a_3) are retrieved from the tree and inserted into a smaller, *conditional* FP–tree for a_3 (see Fig. 4b). These paths are $\{a_0a_1a_2 : 2, a_0a_2 : 2\}$; a_3 is included and the numbers after the patterns correspond to the support of a_3 in the corresponding paths. Observe that a_1 is pruned from the patterns, since its support in all conditional patterns is smaller than min_sup. Thus, this tree has just a single path. We find all combinations of patterns in the path and concatenate them with the *conditional itemset* (that is a_3). Thus, the frequent itemsets $\{a_3 : 4, a_0a_3 : 4, a_2a_3 : 4, a_0a_2a_3 : 4\}$ are generated. Fig. 4c and Fig. 4d show the conditional FP–trees for a_2 and a_1, respectively. The trivial patterns extracted from these trees are $\{a_2 : 4, a_0a_2 : 4\}$ and $\{a_1 : 5, a_0a_1 : 5\}$, respectively. Finally, the first item a_0 in the header table generates the frequent itemset $a_0 : 10$. During the whole process, four conditional FP–trees have been created and nine frequent itemsets are generated.

3 Projected Clustering by Mining Frequent Itemsets

In this section, we propose a method that improves the accuracy and speed of DOC/FASTDOC. First, we discuss how to identify the set D of relevant dimensions for a medoid p, using techniques for mining frequent itemsets. Next, we describe an adaptation of the FP–growth method [9] that efficiently discovers the relevant subspaces, by exploiting the properties of the μ function. The adapted mining technique is then combined with the Monte-Carlo method (FASTDOC) to discover clusters. We discuss how diffset mining algorithms can be applied for our problem

a_1	a_2	a_3	a_4
1	2	3	8
2	1	9	6
3	**2**	**6**	**3**
4	8	1	2
9	6	2	1
7	3	3	2

(a)

Itemset
$\{a_1, a_2\}$
$\{a_1, a_2\}$
$\{a_1, a_2, a_3, a_4\}$
$\{a_1, a_4\}$
$\{a_4\}$
$\{a_2, a_4\}$

(b)

Fig. 5. Transformation from data set to itemsets. (a) Original table. (b) Corresponding itemsets.

and explain how we can scale the mining methods for large problems. Finally, we suggest ways that improve the quality of discovered clusters.

3.1 From Projected Clustering to Mining Frequent Itemsets

Given a random medoid $p \in S$, we can transform the problem of finding the best projected cluster containing p, to the problem of mining frequent itemsets in transactional databases. Fig. 5 shows a motivating example. Data set S (Fig. 5a) contains four numerical attributes and six records. Assume that the record marked in bold is the medoid p. If the attribute of a record is bounded by p with respect to the width w (here, $w = 2$), an item for that attribute is added to the corresponding itemset. This process constructs the transactional table of Fig. 5b. Observe that all frequent itemsets (i.e., combinations of dimensions) are candidate clusters containing p. Later, we will see how the minimum support min_sup relates to the minimum density α (described in Section 2.1).

Therefore, the problem of finding the best projected cluster for a random medoid p can be transformed to the problem of finding the best itemset in a transformation of S, like the table of Fig. 5b, where goodness is defined by the μ function (described in Section 2.2). In other words, we want to solve an *optimization* problem by identifying an itemset which 1) is frequent and 2) maximizes the μ function. The FP–growth method was shown very efficient in [9] for frequent itemset counting, therefore we chose to extend it for our subspace mining problem. In the next paragraphs, we describe some optimizations of the original FP–growth method (described in Section 2.3) that use the μ function to greatly reduce the search space.

The idea of using frequent itemsets discovery in clustering has been applied before, but in a different context. Beil et al. [5] propose a document clustering algorithm by treating documents as itemsets of keyword terms. A data mining algorithm is then applied to discover the frequent itemsets and a cluster is defined as a set of documents containing the same frequent term set. In our clustering problem, the domains of dimensions are different (numerical as opposed to binary). In addition, we adapt itemset mining methods to be used as modules of an optimization algorithm that finds the dimension-set that maximizes the μ function (instead of all dimension-sets above a minimum support). Finally, our method is iterative and based on sampling.

Other related work includes *pattern-based clustering* algorithms [11], [14], [15] where the problem is to find

projected clusters of points that have the same *trend* in the dimensions of the subspace (rather than points close to each other in the subspace). An example of such a cluster would be a set of genes (points) whose expression levels have the same behavior (e.g., if the level of one gene in the cluster falls, then the levels of other genes also fall by the same rate) for a subset of tested environmental conditions (dimensions). A pattern-based cluster is formally defined by a set of points C and a set of dimensions D, such that for every pair of points $p, q \in C$ and pair of dimensions $i, j \in D$, $|(p_i - q_i) - (p_j - q_j)| \leq \delta$, where δ is a distance threshold. The pattern-based clustering problem finds all clusters for which $|C| > min_o$ and $|D| > min_a$, where min_o and min_a define the minimum number of points and dimensions that determine the the level of interest in a cluster. Essentially, our task and algorithms are different than those of pattern-based clustering, since we deal with classic clustering of high-dimensional points based on their distances in dimensional subspaces and not on the relative differences of their dimensional values. Also, we solve the optimization problem of finding the *best* cluster that contains a given point and then iteratively find *only the best* overall projected clusters that partition the space, instead of finding *all* (or the maximal [11]) pattern-based clusters according to δ, min_o, and min_a.

3.2 Using FP–Growth with Optimizations

A brute-force application of a mining method on the transformed space would 1) generate all frequent subspaces and 2) find the one with the maximum μ value. Instead, we update the globally best itemset dynamically during search. Each time a frequent itemset is generated, the best itemset is updated if the μ value of the current frequent itemset is higher than that of the best itemset. To find the best frequent itemset quickly, we can apply two simple optimizations in the original FP–growth procedure. Recall from Section 2.3 that a conditional itemset is an itemset shared by all patterns generated from a specific FP–tree. Assume that $\mathcal{I}_{cond}$ is the conditional itemset of the FP–tree and $\mathcal{I}$ is a frequent itemset in the FP–tree. Then, $\mathcal{I}_{cond} \cup \mathcal{I}$ is the frequent itemset generated.

The first optimization is to generate only the necessary itemsets from FP–trees with a single path. Suppose there are n items in the path. The original FP–growth procedure would generate all the $(2^n - 1)$ combinations of these items concatenated with $\mathcal{I}_{cond}$. As we only need to find the best combination for each dimensionality ($\mu(a, b) \geq \mu(a, b')$ $\forall b \geq b'$), we need only generate the itemsets corresponding to all prefixes of the path since these have the largest subspaces for the same support. The second optimization is to generate only the necessary itemsets of dimensionality $dim(\mathcal{I}_{cond}) + 1$ from the table header. The original FP–growth procedure would generate an itemset $\{x\} \cup \mathcal{I}_{cond}$ for each item x in the table header. Since $\mu(a, b) \geq \mu(a', b)$ $\forall a \geq a'$, it is only necessary to generate the itemset $\{x_0\} \cup \mathcal{I}_{cond}$ where item x_0 is the most frequent item in the table header. Fig. 6 illustrates an example.

Item	Support
a_4	80
a_5	60
a_6	40

(a)

Itemset	Support	μ value
$\{a_4\}$	**80**	$\mu(80, 1)$
$\{a_5\}$	60	$\mu(60, 1)$
$\{a_6\}$	40	$\mu(40, 1)$

(b)

Fig. 6. Generating only the necessary $(|\mathcal{I}_{cond}| + 1)$-itemsets from an FP–tree's table header. (a) FP–tree table header. (b) The itemsets.

3.3 The $\mu Growth$ Algorithm

We will now show how the branch-and-bound paradigm can be effectively applied for our mining problem. Assume that $\mathcal{I}_{best}$ is the itemset with the maximum μ value found so far and let $dim(\mathcal{I}_{best})$ and $sup(\mathcal{I}_{best})$ be its dimensionality and support, respectively. For the sake of readability, we overload the μ function and denote $\mu(sup(\mathcal{I}), dim(\mathcal{I}))$ by $\mu(\mathcal{I})$. We can take advantage of this bound to avoid generating patterns for specific entries of the header table. Let $\mathcal{I}_{cond}$ be the current conditional pattern. Its support $sup(\mathcal{I}_{cond})$ gives an upper bound for the supports of all generated patterns that contain it. Moreover, the dimensionality of the itemsets that contain $\mathcal{I}_{cond}$ and can be generated by the current stage of FP–growth is at most $dim(\mathcal{I}_{cond}) + l$, where l is the number of items above the items in $\mathcal{I}_{cond}$ in the header table. Therefore, if

$$\mu(sup(\mathcal{I}_{cond}), dim(\mathcal{I}_{cond}) + l) \leq \mu(\mathcal{I}_{best}), \tag{2}$$

we can avoid constructing the conditional FP–tree for $\mathcal{I}_{cond}$, since it is not possible that this tree will generate a better pattern than $\mathcal{I}_{best}$. This bound can help prune the search space of the original mining process effectively. Consider, for example, the problem of Fig. 4. Assume $min\_sup = 4$ and $\beta = 0.1$. The global FP–tree is shown in Fig. 4a. The conditional itemset $\mathcal{I}_{cond}$ of the global tree is $\emptyset$. First, we use the table header to derive an initial $\mathcal{I}_{best} = \{a_0\}$ with support 10, dimensionality 1, and $\mu(\mathcal{I}_{best}) = 100$.

Now, we begin with the last item $\mathcal{I}_{cond} = a_3$ in the header table. Its support is 4, so the maximum support for all patterns containing a_3 is 4. The maximum dimensionality of discovered patterns is $dim(\mathcal{I}_{cond}) + l = 4$ (i.e., the position of a_3 in the table). Given these numbers, the condition of (2) does not hold, therefore we have to construct the conditional FP–tree (single path) for a_3 and use the method described in the previous section to mine patterns $\{a_0, a_3\}$ and $\{a_0, a_2, a_3\}$ both with support 4. Now, the best pattern becomes $\mathcal{I}_{best} = \{a_0, a_2, a_3\}$, with $\mu(\mathcal{I}_{best}) = 4,000$. Next, item $\mathcal{I}_{cond} = a_2$ in the header table is examined. The size of the subspaces obtained from the conditional a_2 FP–tree is at most $dim(\mathcal{I}_{cond}) + l = 3$ and their support at most 4. Given these values, the condition of (2) is satisfied and we can *avoid* extracting the conditional pattern base for a_2 and growing the corresponding conditional FP–tree. Finally, item $\mathcal{I}_{cond} = a_1$ is examined. The size of the itemsets obtained from the conditional a_1 FP–tree is at most $dim(\mathcal{I}_{cond}) + l = 2$ and their support at most 5. Again, the application of (2) suggests that there is no need to grow the corresponding FP–tree. The algorithm now terminates by creating only two FP–trees (the ones in Fig. 4a and Fig. 4b) and generating only three patterns, instead of the four FP–trees and nine patterns required by the original FP–growth method.

The pseudocode of the $\mu Growth$ algorithm is presented in Fig. 7. Lines 1–9 apply the optimizations discussed in Section 3.2. Conditional FP–trees are built and mined recursively in the for loop. The order of the for loop does

```
/* Entries in T's table header (hl) is in descending order of support */
/* traversal of the single path is the same as traversal of the table header */
Algorithm μGrowth(T, I_cond, I_best)
1   if T is a single path then
2       I := I_cond;
3       for l:=1 to T.hl.length
4           I := I ∪ {T.hl[l].item}; sup(I) := T.hl[l].support;
5           Update I_best if μ(I) > μ(I_best);
6   else
7       I := I_cond ∪ {T.hl[1].item}; sup(I) := T.hl[1].support;
8       Update I_best if μ(I) > μ(I_best);
9       for l:=T.hl.length down to 2
10          I := I_cond ∪ {T.hl[l].item};
11          if μ(T.hl[l].support, dim(I_cond) + l) > μ(I_best) then
12              Construct T's conditional pattern base;
13              Create T's conditional FP–tree T_T;
14              if (T_T ≠ ∅) then
15                  μGrowth(T_T, I, I_best);
```

Fig. 7. The $\mu Growth$ algorithm.

not affect the final result but it affects the running time significantly. We mine from the least frequent item to the most frequent item in the table header of the current FP–tree. With this search order, potential itemsets of large sizes can be found earlier and a high $\mu(\mathcal{I}_{best})$ is likely to be found earlier, which prunes a large part of the search space. As $\mu(a/\beta^j, b) = \mu(a, b+j)$, the pruning power of large frequent subspaces is high for small β and vice versa. Projected clustering using FASTDOC is effective for $1/(4d) \leq \beta < 1/2$, as suggested in [12].

The application of $\mu Growth$ incurs significant performance savings compared to the simple optimizations discussed in the previous section. Especially, when the embedded cluster dimensionality is large (and the value of β is small) the speed-up can be orders of magnitude, as experienced with our implementation. For each p, $\mu Growth$ requires only two scans of the data set; one for constructing the header table of the FP–tree and one for constructing the tree itself. Then, the cluster and its associated subspace from the FP–tree are discovered efficiently.

The $\mu Growth$ mining method can be integrated into the Monte-Carlo algorithm described in Section 2.2. We replace the inner loop of FASTDOC, which finds a good cluster for each p, approximately, by $\mu Growth$ which finds the subspace with the maximum μ value, systematically. We call our algorithm FPC, from Frequent-Pattern-based Clustering (see Fig. 8). We denote the initial data set by S_0 and the data set for each for each invocation of FPC by S. Like FASTDOC, FPC computes one cluster at a time. Then, the cluster will be removed from the data set S and FPC will be invoked again.

```
Algorithm FPC(S, α, β)
1   min_sup := α · |S_0|; // |S_0| is the original dataset size
2   p_best := NULL; //the best medoid found so far
3   I_best := ∅; //features of the best cluster found so far
4   for j:=1 to 2 · |S|/min_sup
5       Choose p ∈ S uniformly at random;
6       TB := { {i||x_i − p_i| ≤ w} | ∀x ∈ S}; // a table of itemsets
7       Construct FP–tree Tree from TB with minimum support min_sup;
8       I := I_best;
9       μGrowth(Tree, ∅, I);
10      if μ(I) > μ(I_best) then //found a better p and D
11          p_best := p; I_best := I;
12  return (p_best, I_best);
```

Fig. 8. Algorithm for computing a cluster.

The process is repeated until no more clusters can be discovered. Notice that the minimum support for constructing the FP–trees ensures that the discovered subspaces correspond to α-dense projected clusters.

Observe also that FPC is a branch-and-bound method that utilizes the best p and its related subspace to prune FP–trees of the same or different ps that may not produce better subspaces according to the $\mu Growth$ principle. In other words, if a good p is found during the first iterations, then a lot of time can be saved for worse ps with smaller subspaces support in subsequent iterations. Therefore, FPC can converge to a good solution fast and many outer iterations are not required. On the other hand, FASTDOC fails to utilize results from previous iterations effectively so it performs a constant number of iterations of high cost.

The $\mu Growth$ mining method can return the subspace of the best projected cluster around point p. Note that during FPC, we invoke $\mu Growth$ for a number of medoids p, until we determine the best p and its associated projected cluster. In this way, the work for discovering all clusters around any other medoid $q \neq p$ is wasted. FPC has to be reinvoked in order to rediscover these clusters, since it returns only one cluster at a time. We could improve the performance of clustering if we did not waste some good clusters discovered during FPC. This is the idea behind an extension of FPC, which we call CFPC for Concurrent Frequent-Pattern based Clustering (see Fig. 9). Instead of returning a single cluster, CFPC concurrently discovers and eventually returns a set G of multiple nonoverlapping projected clusters. At each iteration, a medoid p is randomly chosen (as in FPC) and $\mu Growth$ is invoked to return the best subspace D of p. CFPC inserts the new discovered (p, D) into G if $\mu(D) > \mu(D')$ for all $(p', D') \in G$ overlapping with

```
Algorithm CFPC(S, α, β)
1   min_sup := α · |S_0|; // |S_0| is the original dataset size
2   G := ∅; // the best set of non-overlapping clusters found so far
3   for j:=1 to 2 · |S|/min_sup
4       Choose p ∈ S uniformly at random;
5       TB := { {i||x_i − p_i| ≤ w} | ∀x ∈ S}; // a table of itemsets
6       Construct FP–tree Tree from TB with minimum support α · |S|;
7       I := ∅;
8       for each (p', D') ∈ G
9           if ((B_{p,D*} ∩ B_{p',D'} ≠ ∅) ∧ (μ(p', D') > μ(I))) then // D* is the subspace containing all the dimensions
10              I := (sup(p', D'), D');
11      μGrowth(Tree, ∅, I);
12      if (∀(p', D') ∈ G, (B_{p,I} ∩ B_{p',D'} ≠ ∅) ⇒ (μ(I) > μ(D'))) then
13          Remove all (p', D') from G where B_{p,I} ∩ B_{p',D'} ≠ ∅;
14          G := G ∪ (p, I);
15  return G;
```

Fig. 9. Concurrent algorithm for clusters.

(p, D). In this case, CFPC removes from G all (p', D') overlapping with (p, D). Similar to FPC, CFPC employs branch-and-bound heuristics that utilize the best subspaces previously discovered. Let D^* be the subspace containing all dimensions. Observe that $\forall D\ \mathcal{B}_{p,D^*} \subseteq \mathcal{B}_{p,D}$ holds. To discover a better subspace for a new p, its new μ value must be greater than the maximum μ value of all those hyperboxes in G overlapping with $\mathcal{B}_{p,D^*}$. In other words, if a good p is found during the first iterations, a lot of time can be saved for worse p in subsequent iterations. Like FPC, CFPC is invoked several times in order to discover all the clusters. However, CFPC can discover multiple clusters at a time, improving the efficiency of the clustering process.

3.4 Adapting Diffset Mining Algorithms

Depth first search (DFS) algorithms, like FP–growth, are preferred for our problem, which is essentially an optimization problem because large itemsets that facilitate pruning can be found early. We have already seen how FP–growth [9] can be adapted to a $\mu Growth$ subspace mining algorithm. In this paragraph, we discuss the adaptation of dEclat [17], another DFS mining algorithm for this problem.

In data mining problems, there are two representations for itemsets. The first format (used by Apriori [4] and FP–growth [9]) represents itemsets by rows of items. The second format (used by dEclat [17]) represents itemsets by columns of transaction IDs (i.e., TIDs). For the latter format, the support of an itemset can be computed by intersecting different TID sets. However, these TID sets may be too large to fit in memory. A new vertical representation called *diffset* [17] was proposed, which only maintains the set difference of the TID set of a candidate pattern from its parents. Thus, dEclat mines frequent itemsets from diffsets. It was shown to be scalable for large and high-dimensional data sets and much more efficient than Apriori and FP–growth.

For our problem, we can extend dEclat to $\mu dEclat$, which computes the itemset with the maximum μ value. First, we use support-based ordering to mine from more frequent prefixes to less frequent prefixes (the reverse order was found to be much slower). Second, we add pruning conditions (as in the $\mu Growth$ algorithm) to facilitate efficient pruning. Details are omitted for the sake of readability.

The major cost of dEclat is the construction of the diffsets. Apart from that, this algorithm has several advantages over Apriori and FP–growth. First, diffsets keep shrinking at lower search levels. Second, the size of a diffset is usually smaller than the corresponding TID set. On the other hand, the adapted $\mu dEclat$ may not be able to compute the *best* itemset as fast as the $\mu Growth$ algorithm. The search space of a DFS algorithm is a tree graph. To discover a frequent s-itemset (subspace), $\mu dEclat$ needs to visit a search path of depth s. For this, it needs to create at least s diffsets at the root level, $s-1$ diffsets at the level under the root, and so on until the node representing the s-itemset. As a result, at least $O(s^2)$ diffsets need to be created. Note that the paths for similar itemsets in FP–trees can be shared but different diffsets cannot be shared. Thus, for the case of FP–tree with a single path, the $\mu Growth$ algorithm computes all the necessary μ values efficiently. In Section 4.2, we experimentally compare $\mu Growth$ with $\mu dEclat$ and show that the former method is actually more efficient.

3.5 Making Mining Algorithms Scalable

Mining algorithms like FP–growth and dEclat require a significant amount of data to be stored in main memory structures (FP–trees or diffsets) in order for the mining process to be efficient. Han et al. [9] and Zaki and Gouda [17] discuss disk-based structures that make these mining methods applicable for large problems; however, the proposed solutions would require many I/O transfers. In order to make our methods scalable for large problems, we can apply mining on a sample of the database. As suggested in [13], [18], a sample size independent of the data set size can give accuracy guarantees. These studies also provide experimental results, which show that in practice even a small sample size can achieve good accuracy. It turns out that applying our methods on a fixed-sized sample is enough to guarantee high quality in the results.

In the worst case, the cost of the mining algorithms is linear to the size of the database but exponential to the number of items (i.e., the dimensionality in our case). However, since we use branch-and-bound algorithms to prune away the search space fast, in practice the time complexity of our methods is much lower. Moreover, when using a fixed sample size, the time complexity of mining is independent to the data set size. In Section 4, we study how the sample size affects the accuracy and the running time of our algorithms.

3.6 Heuristics for Refining Projected Clusters

In this section, we discuss several heuristics that can improve the quality of clusters discovered by our mining-based clustering algorithm. We first define some measures that can quantitatively capture the characteristics of clusters and assist the definition of the refinement heuristics.

The *spread* measure is used to determine the compactness of a cluster. To define the spread of a cluster C, we first compute its centroid $\overline{X}(C)$ [2], defined by $\overline{X}(C)_i = \sum_{x \in C} x_i/|C|$ for each numerical dimension i. Assuming that D is the subspace of C, the spread $R(C, D)$ of C is defined by $R(C, D) = \sum_{x \in C} dist_d(x, \overline{X}(C))/|C|$. A small spread means that subspace D is good for cluster C and vice-versa. As a distance measure, we use the *Manhattan segmental distance* [1]. Given two records p and q, their distance with respect to the subspace D is defined as $dist_D(p, q) = (\sum_{i \in D} |p_i - q_i|)/|D|$, where $|D|$ is the number of dimensions in D. In order for all dimensions to have the same effect in the distance measure, we first normalize all values of a numerical attribute to be in the interval $[0, 1]$.

The *skewness* measure reflects the relevance of each attribute to the cluster. The relevance of each attribute is defined by comparing its local distribution in the cluster with the global one; an attribute with dense regions in the cluster may not be relevant if its global distribution is also dense. We use the *skew ratio* measure to describe the relevance of the attributes. For a numerical attribute i, the skew ratio is defined as $SkewRatio(i, C) = \frac{\text{variance of dimension } i \text{ in } S_0}{\text{variance of dimension } i \text{ in } C}$, where S_0 is the initial data set.

Our first refinement heuristic attempts to recover some marginally missed dimensions in the projected clusters discovered by FPC, CFPC, or DOC. In other words, a more relaxed projected cluster model than in Definition 1 is used. Fig. 10 illustrates such a case. Assume that the relevant

a_3	a_5	a_7	Corresponding itemsets
2.0	**5.0**	**8.0**	$\{a_3, a_5, a_7\}$
2.1	4.9	8.3	$\{a_3, a_5, a_7\}$
2.0	5.2	7.7	$\{a_3, a_5, a_7\}$
4.3	5.0	8.0	$\{a_5, a_7\}$
1.8	5.5	8.0	$\{a_3, a_5, a_7\}$
1.5	4.5	7.5	$\{a_3, a_5, a_7\}$
2.0	2.8	8.0	$\{a_3, a_7\}$
2.2	5.0	8.5	$\{a_3, a_5, a_7\}$
2.5	6.5	9.5	$\{a_3, a_5, a_7\}$

Fig. 10. Losing relevant dimensions, $w = 2$.

attributes for this cluster are a_3, a_5, and a_7, and the medoid is the first record. FPC would return the subspace $\{a_7\}$, if the minimum support is set to 9. The supports of a_3 and a_5 are slightly smaller than 9 so they cannot be in a frequent itemset. The missing relevant dimensions can be discovered as follows: Given a cluster C and its original subspace D, the *refined* subspace D' is defined as

$$D' = \{z | SkewRatio(z, C) \geq \min_{j \in D}(SkewRatio(j, C))\}.$$

A dimension is relevant if its skew ratio is at least the minimum skew ratio of the original subspace. In Fig. 10, the refined subspace would include those attributes provided a_3 and a_5 that their skew ratios are no smaller than that of a_7. This is possible when a_3 and a_5 have lower global skewness than a_7. After refining the subspace, the medoid p may not be at the center of the cluster, but close to its boundary (see Fig. 11a). Thus, the centroid is used as the representative of the cluster instead. Another potential problem is that the resulting cluster may be part of a large cluster that spreads in higher ranges than the bounding box. In Fig. 11a, the bounding box indicates the boundaries of a discovered cluster. According to the definition, only the points inside the bounding box belong to the discovered cluster. However, some points near the boundary are missed by the definition, although they are part of the natural cluster. The boundary constraints are then relaxed by including records having distance at most max_dist from the cluster centroid $\overline{X}(C_i)$ where max_dist is the distance of the farthest point in the discovered C_i from the centroid. For simplicity, $\overline{X}(C_i)$ and max_dist does not change throughout the assignment process. The set of these assigned points is denoted by ΔC. Then, we update C_i to C_i' that includes ΔC. Formally,

$$C_i' = C_i \cup \{q | q \in S \wedge dist_{D'}(p, q) \leq max\_dist\},$$

where p is the medoid and $max\_dist = \max_{q \in C_i} dist_{D'}(p, q)$. The discovered cluster is removed from S and the iterative phase continues until no α-*dense* clusters can be discovered.

As a second refinement, we can prune clusters having μ values significantly lower than the rest. First, we sort the clusters according to their μ values in descending order. After sorting, we have $\mu(|C_i|, |D_i|) > \mu(|C_{i+1}|, |D_{i+1}|)$. Then, we find the position pos such that $\mu_{pos}/\mu_{pos+1} \geq \mu_i/\mu_{i+1}$ $\forall i$. This position divides the clusters into the set *strong* clusters C_i $(i \leq pos)$ and the set of *weak* clusters C_i $(i > pos)$. Since the target number of clusters is k, no pruning is performed when $pos < k - 1$. Otherwise, the *weak* clusters are pruned and their records are added back to S. The target number k of clusters is optional. Since k is only used in the pruning and merging phases, the user can set k to a huge value in order to skip these phases.

At a final refinement, clusters close together with similar subspaces can be merged. Fig. 11b shows a case where cluster merging is needed. In this case, the natural cluster is a projected cluster spanned in the subspace $\{a_1, a_2\}$ where a_1 is the horizontal axis and a_2 is the vertical axis. FASTDOC may split the cluster into two projected clusters spanned also in the subspace $\{a_1, a_2\}$. Procopiuc et al. [12] attempted to solve this problem by using a larger β, which tends to discover large clusters with smaller subspaces. However, with this method it is possible that some relevant dimensions (e.g., a_1 in this case) will remain undiscovered. Like FASTDOC, the FPC algorithm may also split large clusters. Therefore, we hope to merge these two clusters without losing too much subspace information. We need to merge small clusters, which are close to each other and have similar subspaces, in order to recover the original, larger ones. Clusters are merged following the agglomerative hierarchical clustering paradigm until k clusters remain. Given clusters C_x and C_y, the merged cluster is $C_x \cup C_y$, its subspace is $D_x \cap D_y$, its spread is $R(C_x \cup C_y, D_x \cap D_y)$, and its μ value is $\mu(|C_x \cup C_y|, |D_x \cap D_y|)$. A good merged cluster should have small spread and large μ value (large subspace) and we use both measures to determine the next pair to merge. We consider two rankings of the cluster pairs; one with respect to spread and one with respect to μ value. Then, the pair with the highest sum of ranks in both orderings is merged first.

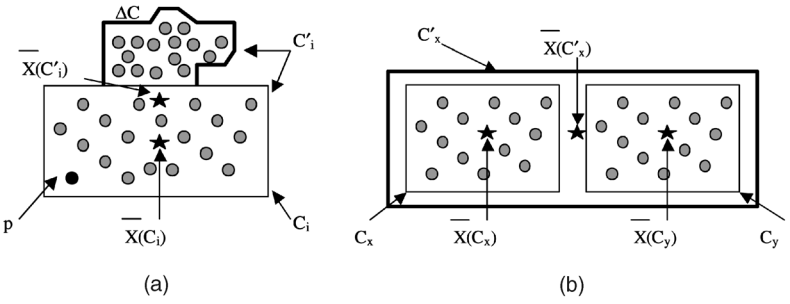

Fig. 11. Refining the clusters. (a) Assigning nearby points. (b) Merging clusters.

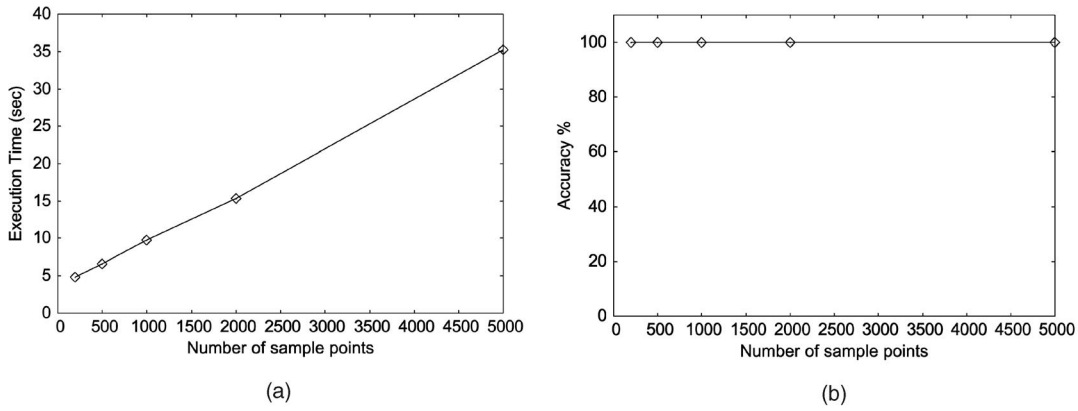

Fig. 12. Effects of sampling on FPC. (a) Effect on execution time. (b) Effect on accuracy percentage.

4 Experimental Evaluation

In this section, we experimentally evaluate the effectiveness and efficiency of our method by comparing it with FASTDOC and PROCLUS, under various experimental settings, for synthetic and real data.

The synthetic data used in our experiments was generated by randomly creating a number of synthetic clusters and their associated subspaces, as in [1]. The data generation parameters that we use are the same as in [12]. The domain of values of each dimension are $[0, 100]$. Unless otherwise stated, the number of records N of the data set is 100K and the dimensionality is 100. The average number of relevant attributes per cluster is 40. Five percent outliers are generated and the remaining points are distributed to $k = 5$ clusters. The ratio of smallest cluster size to largest cluster size is set to 0.5. Clusters of imbalanced sizes generated as clusters having similar sizes are not likely in a real-life case. The cluster C_{i+1} has 50 percent of its relevant dimensions chosen from those of cluster C_i. This models the fact that different clusters are likely to have common dimensions.

For FASTDOC, we set $MAXITER = d^2$, as suggested in [12]. For the randomly generated data (unless otherwise stated), parameters α and β were tuned to $\alpha = 0.10$ and $\beta = 0.25$, after using the heuristics of [12]. For fairness, we provide as input to PROCLUS the correct values for the parameters k number of clusters and l number of average relevant dimensions per cluster. w is tuned using a similar heuristic to that of [12]. For each dimension i, we sort the values p_i for each $p \in S$ and slide a window of $\alpha|S|$ elements along the sorted sequence of values. From each position of the window, we compute the value difference between the first and the last element of the window, and average this difference over all window positions in w_i. w is then computed by averaging w_i over all dimensions i. This technique gives good accuracy on both synthetic and real data sets. For the synthetic data, it results in a value close to 10, which is the optimal value for w, as all clusters were generated in projected hyperboxes of width 10 in all dimensions.

In the comparison study, we use the same definition of accuracy as in [12]. For each output cluster C_i, we identify the input (natural) cluster which shares with C_i the largest number of points. These points are considered correctly labeled in C_i, whereas the remaining points in C_i are considered incorrectly labeled. The output and input outlier sets are processed similarly. Then, accuracy is defined as the percentage of points correctly labeled. In the experiments that involve synthetic data, the results of all algorithms are averaged over 10 runs in order to smooth the effects of randomness in the data input. All algorithms were implemented in C++. The experiments were run on a PC with a Pentium 4 CPU of 2.3GHz and 512MB RAM.

4.1 Effectiveness and Efficiency of Sampling

In the first experiment, we test the effects of applying FPC on a database sample instead of the whole database. Fig. 12a shows the effects of the sample size on the execution time of the algorithm. As expected, the execution time is linearly proportional to the number of sample points. Fig. 12b shows the effects of the sample size on accuracy. The accuracy remains 100 percent even for low sample size (200). In the subsequent experiments, we use 1,000 as the sample size. For fairness, unless otherwise stated, we also use sampling-based versions of all algorithms in the comparison study that follows.

4.2 Efficiency of Subspace Mining Methods

Our next experiment compares the efficiency of the $\mu Growth$ and $\mu dEclat$ subspace mining techniques. For each experimental instance, we choose a medoid and then construct a set of itemsets as in Fig. 5b. To avoid effects of different medoids, the same set of itemsets (from the same medoid) is passed to the subspace mining algorithms $\mu Growth$ and $\mu dEclat$. Four data sets of subspace dimensionality from 20 to 80 were tested.

Fig. 13 shows the running time of $\mu Growth$ and $\mu dEclat$ as a function of subspace dimensionality. $\mu dEclat$ becomes slower compared to $\mu Growth$ as the dimensionality of the discovered cluster increases. As discussed in Section 3.4, $\mu dEclat$ constructs at least quadratic number of diffsets in terms of subspace dimensionality, so it is expensive for mining large subspaces. On the other hand, $\mu Growth$ is fast and robust to subspace dimensionality.

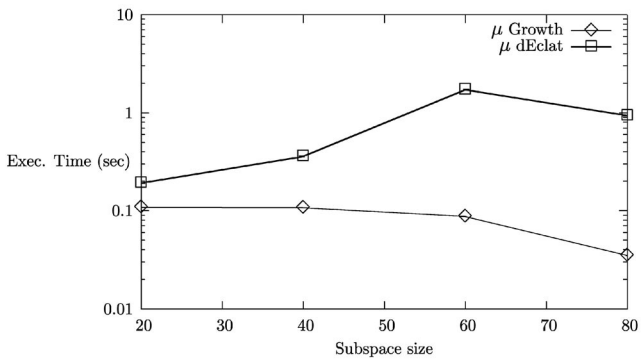

Fig. 13. Running time as a function of subspace dimensionality.

Algorithm	Accuracy(%)	Time(sec)
FPC	100.0	8.9
CFPC	100.0	6.2
FASTDOC	100.0	70.6
PROCLUS	95.6	12.6

Fig. 14. Comparison of projected clustering algorithms.

4.3 Accuracy on Synthetic Data

In the next set of experiments, we compare the accuracy of FPC, CFPC, FASTDOC, and PROCLUS for various types of generated data and input parameter values.

Fig. 14 compares the accuracies of FPC, CFPC, FASTDOC, and PROCLUS when the standard generator parameters are used. FPC, CFPC, and FASTDOC discover the clusters with 100 percent accuracy; however, FPC and CFPC are one order of magnitude faster than FASTDOC. PROCLUS, on the other hand, has lower accuracy but is faster than FASTDOC. As CFPC is faster than FPC but it has similar accuracy to FPC, we use CFPC instead of FPC in the following experiments.

In the next experiment, we test the sensitivity of CFPC and FASTDOC to the input parameters α, β, and w (Fig. 15). Fig. 15a shows how the accuracy varies with α on the same synthetic data set. The accuracies of both CFPC and FASTDOC decrease as the value of α increases to values larger than 0.1. This happens because small clusters are missed for large values of α. Fig. 15b shows how the accuracy varies with β on the same synthetic data set. CFPC is not sensitive to β because the $\mu Growth$ algorithm is employed to discover the best subspace in a deterministic way. The accuracy of FASTDOC is sensitive to β. It decreases significantly for values of β larger than 0.25 because a larger discriminating set needs to be picked. It is more likely for a large discriminating set to have records from different clusters. As a result, only small subspaces can be discovered, which are usually common subspaces of some clusters. Fig. 15c shows how w affects accuracy on the same synthetic data set. Recall that the best value for w is $w = 10$. Note that both CFPC and FASTDOC have high accuracy for a wide range of w, especially for values larger than 10. When w is too small, the original clusters become non α-$dense$, thus CFPC chooses smaller subspaces that are common and close in different clusters, which causes the accuracy to decrease. Although FASTDOC does not check whether the discovered clusters are α-dense, it chooses subspaces of smaller size which cause some clusters to be mixed at low w. In order to avoid too small values for w, it is safe to multiply the estimated value of w by a small factor (e.g., $\times\ 1.5$), as suggested in [12].

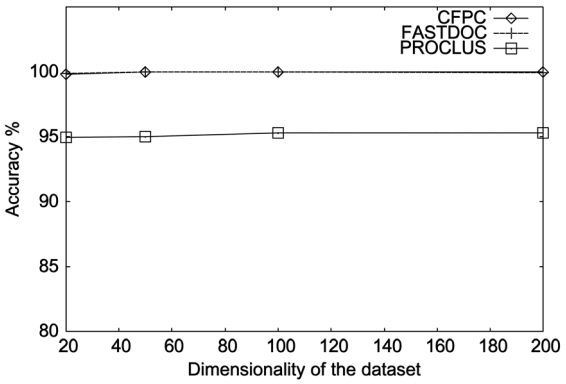

Fig. 16. Accuracy, varying dimensionality.

The next experiment compares the accuracy of CFPC, FASTDOC, and PROCLUS for various values of the data set dimensionality d. For each tested value of d, we generated five data sets with the average number of relevant attributes per cluster as $d \cdot 40$ percent. Fig. 16 shows the comparative results. All of them are insensitive to dimensionality in terms of accuracy. PROCLUS has lower accuracy because the outlier detection mechanism cannot remove all the outliers.

Fig. 17 shows how accuracy is affected by the average number of relevant dimensions in the generated clusters. five data sets were generated. Only PROCLUS has lower

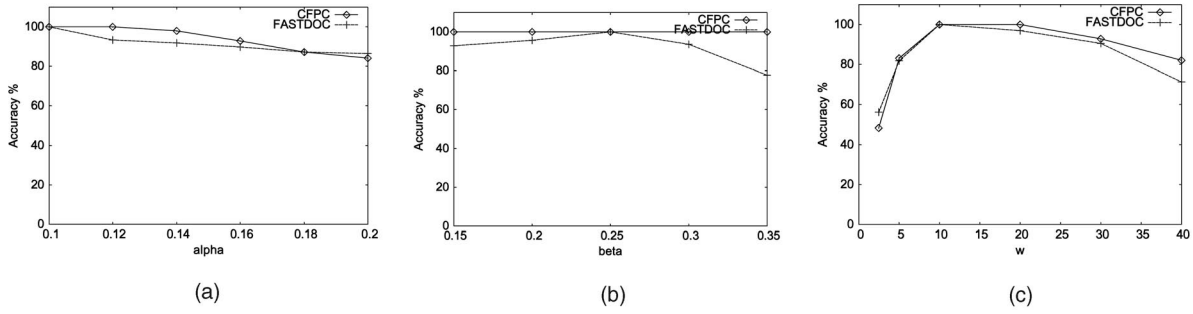

Fig. 15. Effect of α, β, and w on accuracy. (a) Dependency on α. (b) Dependency on β. (c) Dependency on w.

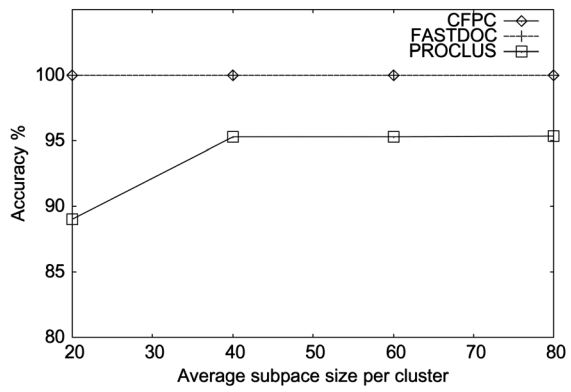

Fig. 17. Accuracy, varying cluster subspace.

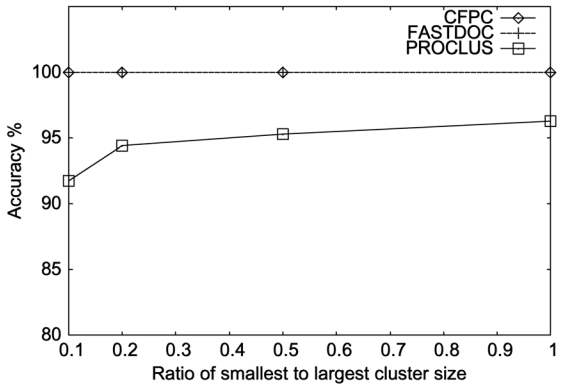

Fig. 19. Accuracy, varying cluster sizes.

accuracy for small subspace sizes. When the subspace size is small and if a few (e.g., one or two) of the relevant dimensions are not discovered, the identified subspace may be too small to distinguish records from different original clusters. On the other hand, for large subspace sizes, even if some of the relevant dimensions are not selected, the identified subspace is large enough to differentiate records from different clusters.

Fig. 18 shows how accuracy is affected by the number of clusters k. Again, five data sets were tested for values of k from 5 to 11. As k increases, the average size of a cluster decreases. CFPC discovers all clusters correctly, as opposed to PROCLUS and FASTDOC. The accuracy of FASTDOC decreases fast as k increases because for large k, the probability of common subspaces between clusters becomes large. Therefore, FASTDOC has increased the probability of selecting incorrect subspaces common to more than one original cluster and merge their contents. CFPC does not have this problem because it does not rely on random discriminating sets in order to identify the best subspace around a medoid p. For PROCLUS, it becomes more difficult to choose the correct set of medoids as k increases.

Fig. 19 shows how the ratio of smallest to largest cluster size affects the accuracy of the algorithms. This ratio is also related to the size of the smallest cluster. To ensure that smallest clusters are not missed, we tune the values of α accordingly for CFPC and FASTDOC in each case. Observe that CFPC and FASTDOC have high accuracy. PROCLUS is not accurate when the ratio is small. It is possible that many medoids are chosen from large clusters and few or no medoids are chosen from small clusters. In this case, large clusters may be split and small clusters may be missed. For CFPC and FASTDOC, clusters discovered early are removed from the data set. The density of small clusters with respect to the data set increases and they are more likely to be discovered in later rounds.

Fig. 20 shows how accuracy is affected by the percentage of outliers in the data set. Similar to the previous experiment, we set the appropriate values for α accordingly for CFPC and FASTDOC in each instance. Both CFPC and FASTDOC are not affected by the outliers. On the other hand, the accuracy of PROCLUS decreases as the outlier percentage increases. This shows that the outlier removal mechanism of PROCLUS is not effective.

Fig. 21 shows the percentage of relevant dimensions discovered for all clusters as a function of number of examined medoids p on the same data set. Given the same number of medoids p tested, CFPC can mine more relevant dimensions than FASTDOC. The figure has step-like structure. A rise of a step happens when a better subspace is discovered. A level of a step corresponds to a medoid for which no better subspace is discovered.

4.4 Efficiency and Scalability

In the next experiment, we compare the efficiency and scalability of the algorithms on synthetic data of various sizes $(N >= 20K - 500K)$. Fig. 22 shows their running

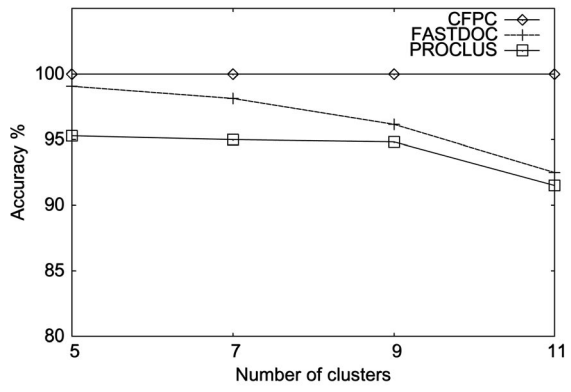

Fig. 18. Accuracy, varying number of clusters.

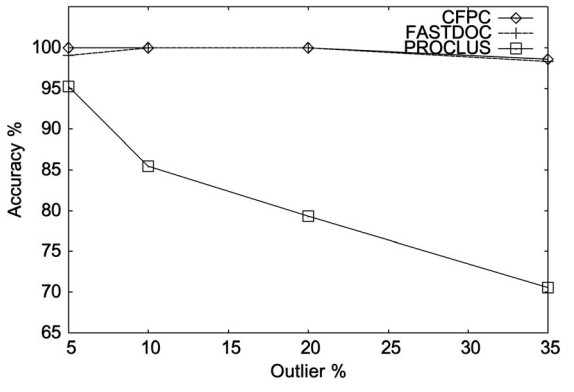

Fig. 20. Accuracy, varying outlier percentage.

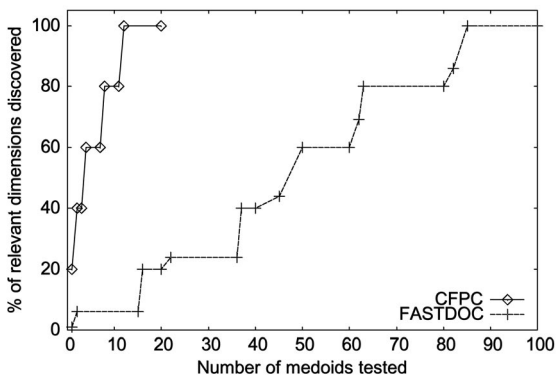

Fig. 21. Progress of CFPC and FASTDOC.

time in seconds. All algorithms are scalable to the database size. FASTDOC is the most expensive as it spends much time on choosing the best relevant dimensions of a cluster by Monte-Carlo techniques. Also, it needs to process each record at least once in each iteration. PROCLUS is more expensive than CFPC as it needs many iterations to converge to the optimal solution; however, it is much cheaper than FASTDOC as it needs less time for computing the subspace. CFPC is the fastest method due to its efficient heuristics for discovering the best cluster and subspace for a given medoid p.

Fig. 23 compares the cost of the three methods as a function of the data dimensionality. The cost of CFPC and PROCLUS is linear to the number of dimensions. On the other hand, FASTDOC is not scalable as it needs to perform d^2 inner iterations in order to achieve high accuracy. Overall, CFPC is faster and more accurate compared to FASTDOC and PROCLUS for various data characteristics.

4.5 Application to Image Recognition

In order to test the applicability of our approach and projected clustering algorithms, we compared the effectiveness of CFPC, FASTDOC, and PROCLUS on real data clustering problems. In the comparison, we also included k-medoids (denoted by KMED), a partitioning algorithm that considers all dimensions in clustering. In the first experiment, we used a real data set with image features from the UCI Machine Learning Repository [7]. This data

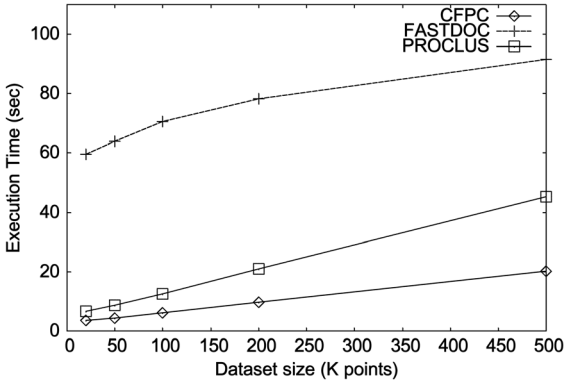

Fig. 22. Efficiency, varying database.

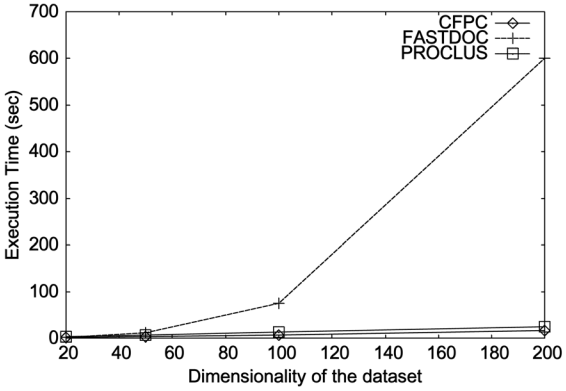

Fig. 23. Efficiency, varying dimensionality.

Name	CFPC	FASTDOC	PROCLUS	KMED
Image	69.28	64.09	62.14	60.24

Fig. 24. Accuracy on a real image data set.

set (Image Segmentation Data) contains 19-dimensional features of 2,100 outdoor images of seven types (classes): brickface, sky, foliage, cement, window, path, and grass. The number of records in different classes are roughly the same. Before clustering, we scale all attribute values to the range $[0, 1]$ using min-max normalization.

As this data set is not large, we do not use the sampling technique in this experiment. In addition, as there are no outliers, we turn off the outlier handling mechanism in PROCLUS. For PROCLUS, the number of clusters k is set to the number of classes and the average subspace dimensionality l is set to 15, the average subspace dimensionality of the projected clusters found by CFPC. For CFPC and FASTDOC, we tuned $\alpha = 0.13$, $\beta = 0.25$, and $w = 0.25$. In order to have fair comparison with PROCLUS, α was set to a value such that the number of clusters found is similar to the number of classes. The accuracy of each of the methods is shown in Fig. 24. k-medoids is less accurate than the projected clustering algorithms because some dimensions are not relevant for some clusters. Observe that CFPC is more accurate than PROCLUS and FASTDOC; however, the accuracy is not as high as in the synthetic data case. The reason is that the class labels do not always reflect the cluster properties. In other words, the features do not always determine the class labels of the objects.

To illustrate this, we show the confusion matrix of CFPC in Fig. 25. Next to the label (C1–C6) of each discovered cluster we enclose its dimensionality in brackets.

	grass	path	window	cement	foliage	sky	brickface
C0 (18)	0	0	109	98	9	0	238
C1 (14)	0	0	0	0	0	295	0
C2 (18)	0	236	3	36	0	0	0
C3 (17)	274	0	0	0	0	0	0
C4 (17)	1	0	97	1	183	0	6
C5 (10)	25	2	88	20	83	0	56
C6 (9)	0	62	0	141	20	0	0
$\mathcal{O}$	0	0	3	4	5	5	0

Fig. 25. Confusion matrix of CFPC on the image data set.

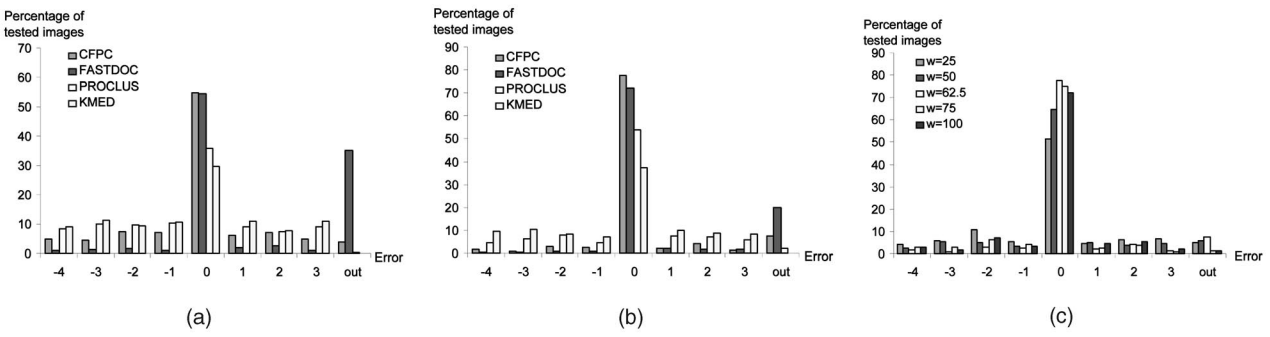

Fig. 26. Accuracy of face rotation estimation based on projected clustering. (a) Ten clusters/rotation class. (b) Twenty clusters/rotation class. (c) Twenty clusters/class, varying w.

$\mathcal{O}$ corresponds to the class of unclustered images (outliers). Note that the discovered clusters are indeed projected, since on the average the subspace size is less than the data dimensionality. For instance, 295 out of 300 sky images were correctly identified considering only 14 out of the 19 features. Although the accuracy of CFPC is not very high, we can discover some interesting properties about the real data sets from the confusion matrix. For the image data set, grass and sky classes are accurately discovered in clusters C1 and C3, respectively. Clusters C0, C2, C4, and C6 contain a large percentage of instances from their class labels and, in general, only instances of few (two or three) classes are confused in the same cluster. Thus, mainly window and cement images are mixed in C0, mainly window and foliage images are mixed in C4 and C5, and path, cement, and foliage instances are mixed in C6. The small number of outliers implies that the discovered projected clusters cover most of the instances.

Next, we demonstrate a real application of projected clustering. Facial rotation estimators can be used to approximately determine the degree by which a face is rotated. Procopiuc et al. [12] demonstrate how projected clustering can be used to train a facial rotation estimator. We performed a similar experiment to compare the effectiveness of CFPC, FASTDOC, PROCLUS, and k-medoids

For this experiment, we use 1,521 facial gray-scale images from the BioID Face Database [8], available at http://www.humanscan.de. From each image, we extract the face and scale it to a fixed size image (16×16 pixels). The domain of a pixel is $[0, 255]$. Then, we discretize the rotation angle into eight classes, each spanning 45 degrees. The original images are frontal upright face images (class 0). The other seven classes are generated by rotating the corresponding upright images by $c \times 45$ degrees, where c is the class label. For each rotation class, 90 percent of the images (training set) are distributed in 10 projected clusters. Note that images in the same cluster always belong to the same rotation class independently of the algorithm used. Each cluster in a rotation class captures some relevant dimensions of the images (e.g., people with mustaches), which cannot easily be captured by a full dimensional cluster. Now, for a given query image, we can estimate its rotation class by checking in which cluster it belongs and then using the class label of the cluster.

A total of 10×8 projected clusters are computed. For efficient storage and query processing, each cluster is represented by minimum hyperbox that encloses its contents in all relevant dimensions. This representation is also applicable for the clusters discovered by PROCLUS and k-medoids.

For fairness to both PROCLUS and k-medoids, CFPC and FASTDOC are tuned to compute disjoint clusters. We set $\alpha = 0.1$, $\beta = 0.25$, and $w = 62.5$ for both CFPC and FASTDOC. The average dimensionality of the projected clusters found by CFPC is 40 (out of 256), which is also used as the average subspace dimensionality l of PROCLUS.

The remaining 10 percent of the images are used as queries to test the effectiveness of clustering. To estimate the rotation class of a test image, we find the projected clusters containing it (if any) and label it as the rotation class with the most clusters whose hyperbox contains the corresponding point. Accuracy is measured as follows: If a face in rotation class i is estimated as in the rotation class $(i + \delta)$ mod 8, the error is δ. Fig. 26a shows accuracy when using 10 clusters per rotation class. Test images not found in any clusters could not be classified, thus they are shown over the "out" label. Observe that CFPC and FASTDOC are more accurate than PROCLUS and k–medoids, although FASTDOC is slightly less accurate than CFPC and cannot classify a significant portion of test images. This can be attributed to the fact that FASTDOC does not check whether the discovered clusters are α–dense, thus it may discover too small clusters which cannot easily contain the test images.

We also tested the effect of using 20 clusters per rotation class. Fig. 26b shows that the accuracy of all algorithms increases. In general, more clusters per rotation class can capture finer properties of the images and improve the accuracy of rotation class estimation. On the other hand, many clusters also imply higher cost towards class estimation for a query image. Note that in both experiments of Fig. 26a and Fig. 26b, the projected clustering algorithms perform much better than k-medoids, a conventional full-dimensional clustering algorithm. Finally, we validate the accuracy of CFPC for various values of w around the estimated one ($w = 62.5$). Fig. 26c shows that the result of CFPC is not affected too much by w taking values around the estimated one, especially for larger values (similar conclusions can be derived from Fig. 15c).

5 Conclusion

In this paper, we presented an efficient and effective projected clustering algorithm. FPC is built on DOC/FASTDOC, a Monte-Carlo algorithm, but it replaces the inner randomized part of the algorithm by a systematic search method based on mining frequent itemsets.

We first identified the similarity between mining frequent itemsets and discovering the best projected cluster that includes a random point p. Then, we proposed an adaptation of FP–growth [9], called $\mu Growth$, that gracefully exploits the properties of the μ function to efficiently discover the projected cluster for a given p. This technique employs branch-and-bound to reduce the search space of the original mining algorithm significantly.

We showed how $\mu Growth$ can be embedded into an iterative process, called FPC (for Frequent-Pattern-based Clustering), which efficiently finds the best projected cluster among a number of medoids p. FPC is extended to CFPC, a method that concurrently finds multiple clusters at a single instance of the iterative process. Both FPC and CFPC employ branch and bound to utilize the best clusters found so far in order to reduce the space of future search.

We discussed how another mining algorithm based on diffsets [17] can be adapted for our problem; however, we showed experimentally that adapting this method is not as appropriate for the optimization problem of finding the best subspace cluster around p, compared to $\mu Growth$. Next, we discussed heuristics that may improve the quality of the discovered clusters by 1) assigning points close to some cluster, 2) pruning small clusters of low quality, and 3) merging clusters close to each other with similar subspaces.

Finally, we evaluated the efficiency and effectiveness of our technique by comparing it with FASTDOC [12] and PROCLUS [1], using synthetic and real data, under various problem conditions. Our method can discover clusters of high quality since it systematically finds the best cluster that encloses the current p. In addition, it is much faster than previous approaches due to the branch-and-bound technique employed.

Acknowledgments

This work was supported by grant HKU 7380/02E from Hong Kong RGC.

References

[1] C.C. Aggarwal, J.L. Wolf, P.S. Yu, C. Procopiuc, and J.S. Park, "Fast Algorithms for Projected Clustering," *Proc. ACM SIGMOD,* 1999.

[2] C.C. Aggarwal and P.S. Yu, "Finding Generalized Projected Clusters in High Dimensional Spaces," *Proc. ACM SIGMOD,* 2000.

[3] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, "Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications," *Proc. ACM SIGMOD,* 1998.

[4] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," *Proc. Int'l Conf. Very Large Data Bases (VLDB '94),* 1994.

[5] F. Beil, M. Ester, and X. Xu, "Frequent Term-Based Text Clustering," *Proc. ACM SIGKDD,* 2002.

[6] K.S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is 'Nearest Neighbor' Meaningful?" *Proc. Int'l Conf. Database Theory (ICDT '99),* 1999.

[7] C. Blake and C. Merz, "UCI Repository of Machine Learning Databases," *Univ. of Calif., Irvine, Dept. of Information and Computer Sciences,* http://www.ics.uci.edu/~mlearn/MLRepository.html, 1998.

[8] R. Frischholz and U. Dieckmann, "BioID: A Multimodal Biometric Identification System," *Computer,* vol. 33, no. 2, Feb. 2000.

[9] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation," *Proc. ACM SIGMOD,* 2000.

[10] A. Hinneburg, C.C. Aggarwal, and D.A. Keim, "What is the Nearest Neighbor in High Dimensional Spaces?" *Proc. Int'l Conf. Very Large Data Bases (VLDB '00),* 2000.

[11] J. Pei, X. Zhang, M. Cho, H. Wang, and P.S. Yu, "Maple: A Fast Algorithm for Maximal Pattern-Based Clustering," *Proc. Int'l Conf. Data Mining (ICDM '03),* 2003.

[12] C.M. Procopiuc, M. Jones, P.K. Agarwal, and T.M. Murali, "A Monte Carlo Algorithm for Fast Projective Clustering," *Proc. ACM SIGMOD,* 2002.

[13] H. Toivonen, "Sampling Large Databases for Association Rules," *Proc. Int'l Conf. Very Large Data Bases (VLDB '96),* 1996.

[14] H. Wang, W. Wang, J. Yang, and P.S. Yu, "Clustering by Pattern Similarity in Large Data Sets," *Proc. ACM SIGMOD,* 2002.

[15] J. Yang, W. Wang, H. Wang, and P.S. Yu, "δ-*Clusters*: Capturing Subspace Correlation in a Large Data Set," *Proc. Int'l Conf. Data Eng. (ICDE '00),* 2002.

[16] M.L. Yiu and N. Mamoulis, "Frequent-Pattern Based Iterative Projected Clustering," *Proc. Third IEEE Int'l Conf. Data Mining (ICDM '03),* Nov. 2003.

[17] M.J. Zaki and K. Gouda, "Fast Vertical Mining Using Diffsets," *Proc. ACM SIGKDD,* 2003.

[18] M.J. Zaki, S. Parthasarathy, W. Li, and M. Ogihara, "Evaluation of Sampling for Data Mining of Association Rules," *Research Issues in Data Eng. (RIDE),* 1997.

Man Lung Yiu received the BEng degree in computer engineering from the University of Hong Kong, China, in 2002. He is currently a PhD candidate in the Department of Computer Science at the University of Hong Kong. His research interests include databases and data mining.

Nikos Mamoulis received the BS degree in computer engineering and informatics from the University of Patras, Greece, in 1995 and the PhD degree in computer science from the Hong Kong University of Science and Technology in 2000. He is an assistant professor in the Department of Computer Science at the University of Hong Kong. In the past, he worked as a research and development engineer at the Computer Technology Institute, Patras, Greece, and as a post doctoral researcher at the Centrum voor Wiskunde en Informatica (CWI), Netherlands. His research interests include spatial, spatio-temporal, multimedia, object-oriented, semi structured databases, and constraint satisfaction problems.