A Study of two Problems in Data Mining – Projective Clustering and Multiple Tables Association Rules Mining

Ng Ka Ka

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science & Engineering

August, 2002

A Study of two Problems in Data Mining – Projective Clustering and Multiple Tables Association Rules Mining

submitted by

Ng Ka Ka

for the degree of Master of Philosophy

at the Chinese University of Hong Kong

Abstract

With a large data pool, data mining techniques become essential in uncovering the implicit and potentially useful high-level knowledge hidden in the data, which is previously unknown to the users. Both the effectiveness and efficiency of these data mining techniques are crucial, since users are expecting to have accurate mining results within a reasonable time. In this thesis, we study two problems and related techniques in data mining, namely projected clustering, and multiple tables association rules mining. We also propose various algorithms to tackle the problems, and verify the results with experiments.

Clustering has been studied in statistical and database research for a long time. This technique is useful in getting insight into the distribution of a data set. However, traditional clustering techniques are not suitable to be applied to data with high dimensionality. We study the causes of this so called "curse of dimensionality" phenomenon. With high dimensional data, it is observed that natural clusters seldom exist in the full dimensional space. Instead, different natural clusters exist in different subspaces, formed by a set of correlated dimensions with respect to each cluster. Discovering the correlated dimensions

as well as the location of clusters is known as the projective clustering problem.

We propose a new algorithm EPC (Efficient Projective Clustering) to detect clusters when the data of high dimensionality is projected to different subspaces (formed by sets of correlated dimensions). This algorithm uses a 1-d histogram to model data densities when the objects are projected to each dimension. Compared to the best previous method to our knowledge, EPC has the following advantages: (1) It does not require the input of the number of natural clusters, and the average cardinality of the subspaces. (2) It can discover clusters of irregular shapes. (3) From our experimental results, it produces better clustering quality. (4) It is scalable to a very large database, and faster than previous method.

We further extent and generalize EPC to develop a framework which makes use of multidimensional histogram. In particular, we implement EPC2, an enhanced version of EPC to model data distributions by 2-d histograms. Experiments show that EPC2 produces better clustering result than EPC in data sets, in which projections of data objects are distributed along directions not parallel to original axes. The trade-off is the increases in time and space complexity.

Association rules mining is useful in planning for marketing programs and strategies. Various efficient mining algorithms for association rules have been proposed. However, most of them are based on a single table in market–basket database, in which a record typically consists of different items purchased in a transaction. In relational database, data can be stored in multiple *Dimension Tables* related by a *Fact Table* in **star schema**. Data need not to be in binary representation (which is assumed in market–basket database, an item whether purchased or not in a transaction) in these Dimension Tables, and previous approaches cannot be applied in these multiple–table scenario.

One straightforward approach is to join the Dimension Tables globally into a single large table before performing the mining. However, multi-fold increase

in both size and dimensionality in this large table presents a huge overhead to the already expensive frequent itemset mining step. Therefore, we propose a "mining-then-joining" approach, which applies various techniques to combine results obtained from mining the rules individually from each tables locally. From our experimental results, this approach produces the same result with much less computational time.

有關數據採掘的兩項研究：尋找投影叢集的方法，及於多個數據表中尋找關聯規則

作者：吳家嘉

修讀學位：哲學碩士

香港中文大學計算機科學及工程學部

論文摘要

數據採掘技術是用於發現隱含於大量數據當中的高階知識。 所採掘的知識存有應用價值，並於被採掘前不爲用者所知道。效用及效率是這些數據採掘技術的關鍵，用者期望這些技術可以合理之時間內找出準確的解答。在這論文中，我們研究兩項數據採掘中的有關技術 – 尋找投影叢集的方法， 及於多個數據表中尋找關聯規則。我們並提出多個計算法，以及利用實驗去驗證這些計算法得出的解答是否準確。

尋找叢集這個問題已於統計學及數據庫處理兩個範籌中研究已久。這種技術多用於洞察數據的分佈特性。但傳統之尋找叢集技巧往往不適用於高維度數據。我們研究這種被稱爲"高維度之災害" (curse of dimensionality)的現象。 於高維度數據中，我們察覺到叢集極少出現於全維度空間。相對地，不同之叢集會出現於不同的子空間，而不同的子空間是由不同組合而有相互關系的次元所組成。投影叢集問題就是去發現這些由不同組合而有相互關係的次元組成的子空間，及存在於這些子空間中的叢集。

我們提出一項新的算法，名爲 EPC，去探出這些由高維度數據投影於不同子空間而影成的叢集。這新算法採用一次元柱狀圖去表現當數據投影於不同次元後的分佈特性。與我們所知中以往最佳的算法去比較，EPC 有以下之優勝處:1.這算法不需要用者輸入叢集的數量，及組成子空間之次元的數量。2.這算法可以探出不規則形狀的叢集。3.這算法提供的解答更準確。4.這算法可應用於大型數據庫及比之前的算法需要更短之計算時間。

我們擴展 EPC 所採用的方法，以應用多次元柱圖去處理更複雜的數據。當中，我們完成了 EPC2 的計算程式。EPC2 是 EPC 的擴展版本。它利用二次元柱狀圖去表現數據的分佈。實驗結果顯示，當數據分佈的方向並不平衡於原軸線時，EPC2 能計算出比較準確的解答。相對地計算時間與所需空間的複雜性也較高。

採掘關聯規則是一項可應用於市場推廣之計畫與安排的技術。多年來有各項快速的關聯規則採掘方法被提出及發展。可是，大部份提出的計算法是應用於市場籃數據庫(market-basket database)的單一數據表中。這些數據庫裏，每一項單一資料儲存了一宗交易中含有的項目。在關係數據庫中，數據經常被分散儲存於多個次元數據表(dimension table)， 並由一個事件數據表(fact table)儲存當中的關係。這種儲存模式被稱爲星狀格式(star schema)。數據未必需要以二元代表法存在於這些次元數據表中(因以往市場籃數據庫中 ， 每一項目在一交易中以二元代表法記錄有購買與否)， 而以往的計算法亦不適用當數據分散存放於多個次元數據表。

於多個次元數據表中採掘關聯規則的最直接方法，是首先把各次元數據表結合(join)成一個大的單一數據表後，才運用其他採掘關聯規則的計算法。可是，這結合的操作需要長的運算時間及大的記憶空間。 而對於這結合後的大的單一數據表，採掘所需的時間亦相對應地較長。所以，我們提出了數個技巧，首先對各單一數據表各自進行採掘，然後把各單一數據表得出的關聯規則聯合而得到所需的關聯規則。比較起先把單一數據表結合的方法，"採掘後結合"可於更少的時間，得出一樣的關聯規則。

Acknowledgment

I would like to express my grateful thank to my supervisor, Prof. Ada Wai-chee Fu. Without her warm care and support, I may not be able work with excitement, cheerfulness, keep up my spirit and motivation all the time on different research problems in database and data mining fields within these two years. With her supervision and encouragement, I could have various chances and practises to improve the writing and presentation skills, which I think would be very useful in my whole life. She also gives many valuable suggestions for my future career.

Many thanks go to Prof. Man Hon Wong and Prof. Irwin King as my thesis markers. Their advices given during term presentations help me to further enhance my research work. In addition, with the enlightening and interesting discussions with Prof. Ke Wang, I can have the opportunity to work with him in the problem of mining association rule. I would also like to thank him for his graciously consent to be my external marker.

It's also a pleasure for me to thank my fellow colleagues: Yin-ling Cheung, who provides me many useful resources and discussions in the problem of mining association rules, and unselfish helps for the experiments; Lai Mei Chiu, who gives assistances and helps in generating the data in the problem of clustering; Chi Ho Lam and Chi Kan Cheung, who work with me as TA in the Database course. Certainly, there are many others whom I cannot thank all here, but definitely their words and sharing with me in these two years will always ring in my mind and warm my heart.

Contents

List of Tables

List of Figures

Part I

Projective Clustering

Chapter 1

Introduction to Projective Clustering

Clustering is often an important initial step in the data mining process. It is being applied to many practical problems such as image segmentation, pattern recognition, trend analysis etc. Here, we state the problem of traditional clustering as in [16] : *Given a number of objects, each of which is described by a set of numerical measures, devise a scheme for dividing the objects into a number of groups such that objects within the same group are similar in some respect and unlike those from other groups. The number of groups and the characteristics of each group are to be determined.*

Extensive surveys on traditional clustering techniques and concepts can be found in [25]. Traditional clustering algorithms are different in their terminologies, cluster representations, assumptions for the components of the clustering process and the contexts in which clustering is used. They are often be classified as *hierarchical clustering, partitional clustering, optimization techniques and density search techniques.* However, most of them are originally aimed for tackling clustering problem with low dimensional data.

However, with the advances of technology, we are collecting more and more data everyday. We see a growing number of attributes for data objects (typical relational database contains twenty up to hundreds of attribute), some of

which make the data objects more informative, but some of which may add unnecessary details and thus "hide" the valuable information. For example, for a customer file, we may include many attributes of each customer, such as sex, age, income, address, family information, job information, purchase orders, etc. However, correlation or patterns typically exist in small subsets of these attributes. If we consider numerical attributes and treat each attribute as a dimension, each data record is a data vector in a high–dimensional space. Then Patterns can only be discovered when we consider projections of the data vectors in different subspaces, instead of the vectors in the full dimensional space.

In addition to the application needs, recent theoretical results [8] have also shown that in high dimensional space, distances between every pair of data objects are almost the same for a wide variety of data distributions and distance functions. Empirical results show that this can occur for as few as 10 to 15 dimensions. As a consequence, the concept of proximity and neighborhood can hardly be applied in high dimensional space, the farthest neighbor of a point is expected to be almost as close as its nearest neighbor [22], and thus natural cluster does not exist in the full dimensional space.

Clearly, the traditional understanding of clustering problem is not sufficient to meet the challenges as described above, where patterns typically exist in small subsets of attributes. Feature selection techniques [32] or methods such as Principal Component Analysis (PCA) [29, 17, 30] are proposed to reduce the dimensionality of the data, by projecting all the data objects on a subspace which minimize the information loss. However, in real life applications, correlations between dimensions are often localized to different clusters. Different patterns can only be uncovered when we consider projections of the data objects on different subspaces. In such case, any attempt to reduce the dimensionality of the whole database would bring substantial information loss.

Therefore, we study the problem of ***projective clustering*** originally suggested by [2] , which would be stated as the following:

*A **projected cluster** is a subset C of data objects together with a subset D of dimensions, which we call a subspace, such that the data objects in C are closely clustered in the subspace of dimensions in D. For the projected clustering algorithm, it outputs a $(k+1)$-way partition $\{C_1, ..., C_k, O\}$, so that the data objects in each partition except the last form a projected cluster, while the last partition contains outliers. Each cluster $C_i, 1 \leq i \leq k$, is associated with a possibly different subspace D_i, so that data objects in C_i are correlated with respect to dimensions in D_i.* $|D_i|$ denotes the cardinality (number of dimensions) of the corresponding subspace, and it need not be the same for different cluster. In our approach, each partition would be represented as an n-dimensional hyper-rectangle, where $n = |D_i|$. The hyper-rectangle locates the cluster in the corresponding subspace.

Figure 1.1 [2] shows a simple example which illustrates the idea of projective clustering. Consider the case for a 3-dimensional space **XYZ**. The left hand side of Figure 1.1 is a projection of the data set on the cross section on X-Y plane, and the right hand side is the one on the cross section on X-Z plane. We can find a projected cluster in subspace **XY**, and another projected cluster in subspace **XZ**. The regions within the dotted lines are the 2–d hyper rectangles that contain the clusters in the corresponding subspace. However, if we consider the full dimensional space **XYZ**, we can not discover any clusters, as data objects in pattern 1 spread along Z axis, and data objects in pattern 2 spread along Y axis. If we do a feature selection and consider dimension pairs **XY** or **XZ** only, either one of the patterns cannot be discovered, since each dimension is relevant to at least one of the patterns.

The difficulties of projective clustering lies in the fact that there are two dependent sub-problems embedded in the original problem, namely discovering the subspaces of each projected clusters, and finding the locations of them.

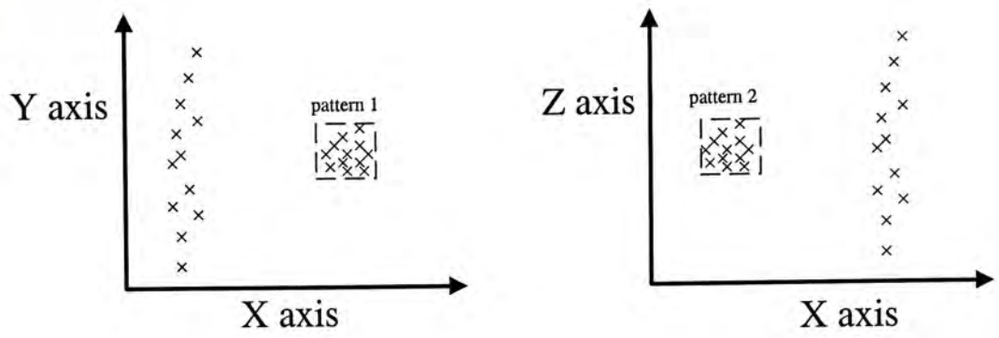


Figure 1.1: Projected clusters on subspace **XY** and **XZ**.

Given the sets of data objects belonging to the same cluster, the corresponding subspaces can be discovered by examining which of the dimensions the data objects within the same clusters are located close together. On the other hand, given the subspaces, we can apply traditional clustering techniques on different known associated subspaces. However, without knowing the subspaces and the cluster locations, there is no simple way to solve these two sub-problems at the same time. PROCLUS [2] uses a k–mean like approach to iteratively solve these two sub-problems. However, it requires the user to know the number of clusters, the average cardinality of each subspaces, and various parameter setting before applying the clustering algorithm. Detail description can be found in Chapter 2. Furthermore, other than pattern discovery, projective clustering techniques have also been successfully used for indexing high dimensional data. [10]

Other than the curse of dimensionality, clustering itself is not a well-defined problem despite its long history in statistics literatures, pattern recognition and database communities. Different clustering algorithms assume different definitions of clusters, and thus result in clusters with different properties and shapes. For example, methods based on minimizing distances between data points and their belonging cluster representative object, such as K-mean , CLARANS [37], BIRCH [54], tend to return clusters in convex shape. However, they would face difficulties in locating clusters of irregular shape, as well

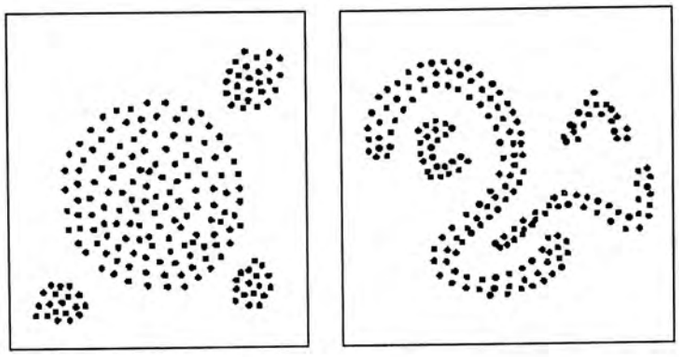


Figure 1.2: Cases that distance-based clustering algorithms do not work well

as the situation when one small cluster is located closely to another large cluster. In such case, some of the data objects belonging to the large cluster would be wrongly recognized as belonging to the small cluster, since these data objects are located nearer to the representative object of the small cluster. Figure 1.2 shows example where these algorithms cannot produce correct clustering result.

Chapter 2

Related Work to Projective Clustering

A survey on traditional clustering methods can be found in [21, 16, 27, 26]. [25, 18] also provides reviews and analysis of recent work on clustering. A lot of good clustering methods have been identified in the past. However, most of them do not aim at clustering high-dimensional data. They become computationally expensive and also ineffective with growing dimensionality. Recently there are various enhanced clustering approaches, including DBSCAN [14, 15] and OPTICS [34], which use R^* Tree to efficiently perform region queries and order the data objects to represent its density–based clustering structure; BIRCH [54], which is based on the Cluster–Feature–tree, and STING [53], which uses a quadtree–like structure containing additional statistical information. They give promising results for low dimensional data. However, because of the performance degeneration of R^* tree for high dimensionality, and ineffectiveness for condensation-based approaches [23] (BIRCH and STING) in high dimensional space, none of them are able to produce satisfactory clustering result for high–dimensional data. As as pointed out by [23], the curse of dimensionality has a severe impact on their resulting clustering quality, and continues to pose a challenge to clustering algorithms at a fundamental level. Here, we review some of the most remarkable researches on clustering in recent

years.

2.1 CLARANS - Graph Abstraction and Bounded Optimization

The study of CLARANS [37], which is based on PAM and CLARA developed previously [31], views the clustering problem as a graph abstraction. For k-medoids based methods, after we can determine the best set of medoids, we can cluster the data points to different partitions by a single scan of the database. Data points are assigned to cluster where their nearest medoid located. The most difficult part is how we can find the best set of medoids which can cluster data points well. CLARANS models the problem of finding the sub-optimal set of k medoids as the following:

2.1.1 Graph Abstraction

Given n data points, there are ${}_nC_k$ ways to choose k data points as the set of medoids. Each selection can be represented as a node in a graph $G_{n,k}$. There are totally ${}_nC_k$ nodes in the graph $G_{n,k}$. Two nodes are neighbors (connected by an arc) *iff* their sets differ by only one data point. For each node, there are $k(n-k)$ neighbors, and each move of a node to its neighbor represents a replacement of one selected medoid in the set with another data point.

Since each move (replacement) can be quantified by how much the resulting clustering quality could be improved, a cost function can be computed according to some pre-defined rules. If we choose a negative value of the cost function to indicate a better choice, the clustering problem can now be viewed as searching for a minimum on the graph $G_{n,k}$.

Although examining every node in the graph guarantees that the best clustering could be found, it would become computationally expensive as the total

number of nodes are ${}_nC_k$. Therefore, CLARANS employs a bounded optimized random search to the graph $G_{n,k}$ to find the sub-optimal solution.

2.1.2 Bounded Optimized Random Search

Instead of inspecting every neighbors, CLARANS selects a number of neighbors to be examined, in which the number is bounded by the parameter *maxneighbour*. Although this greatly reduces the computational complexity, the searching would thus be easily trapped in local minima. Therefore, once the searching reaches a local minima, another arbitrary chosen point would be chosen to start the searching again. The number of searches is bounded by another parameter *numlocal*.

This represents a random search in the graph, which tries to optimize the resulted clustering quality by continuously replacing bad medoid with a better data point. This optimization is not complete and it is bounded by the two parameters, *maxneighbour* and *numlocal*, for the gain in better efficiency.

Although this algorithm can cluster data set with a large number of data points efficiently, it does not address the problem for high dimensional data. Nevertheless, the modelling of the clustering problem as optimization (minimization) gives a good direction for further studies and investigations.

2.2 OptiGrid - Grid Partitioning Approach and Density Estimation Function

Another commonly used approach for clustering is to divide the whole space into cells using some cutting planes. Grid cells containing dense regions are located and connected in order to form clusters. However, if the cutting planes cut across the clusters, the natural neighborhood would be lost. This problem would become more severe when the number of dimensions is high.

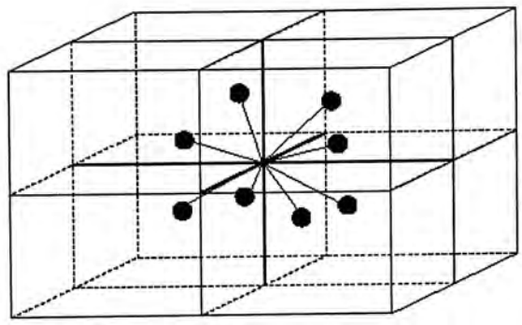


Figure 2.1: Example of empty space phenomenon, $d = 3$.

2.2.1 Empty Space Phenomenon

Consider a cluster formed by normally distributed data points in $[0, 1]^d$ with $(0.5, ..., 0.5)$ as center point, where d is the number of dimensions. Suppose we do the splitting at 0.5 in each dimension. The number of different directions from the center point would be directly proportional to the number of grid cells partitioned, which is exponential in d (that is 2^d). Although the distances between data points and the center follow a Gaussian curve, since the choice of direction is totally random, as a consequence, most data points fall into separate grid cells. Especially for high dimensional data, even though the data points form a spherical cluster, they are unlikely to be located very near to the center point, and most of them fall into separate grid cells. As a result, those adjacent grid cells can hardly be discovered because none of them are dense enough. Figure 2.1 [23] shows the scenario for $d = 3$.

The main reason for the poor performance and the ineffectiveness of grid-based clustering when the dimension is high is mainly caused by cutting planes which partition natural clusters into a number of grid cells. Therefore, the main objective of OptiGrid [23] is to construct an optimal grid-partitioning, such that they partition data in low density region and discriminate clusters as much as possible. To achieve this, OptiGrid employs concept of Density Estimation Function which can be found in statistics literature. [48].

2.2.2 Density Estimation Function

Consider the density function f for a Random Variable X.

$$P(a < X < b) = \int_a^b f(x)\,dx.$$

Given this density function of a data distribution, we can calculate how many data points are located in a specific region. However, during clustering, we do not know the density function of the data set in advance. Therefore, we employ an inverse process to estimate the underlying data distribution when we are given a set of n observed data points with position $X_1, X_2, ..., X_n$, by constructing the following density estimation function.

$$\hat{f}(x) = \frac{1}{nh}\sum_{i=1}^{n} K(\frac{X_i - x}{h})$$

where $K(x)$ satisfying $\int_{-\infty}^{\infty} K(x)\,dx = 1$, is the *kernel function*, and h is called the *smoothing factor*.

Intuitively, we can consider each data observations give its *influences* to the underlying data density distribution. The amount of influences received in each position is determined by the kernel function K, and the distance to the observations. Figure 2.2 [48] illustrates this idea. Here each influence is in a "bump" shape, and the density estimator is the summation of "bumps" placed at every data observation. The kernel function K determines the shape of the estimated distribution, while the width of the "bump" determines the smoothness level.

When we are given a set of n observed data points $X_1, X_2, ..., X_n$ and each data points is a d-dimensional vectors $X_i = (X_{i_1}, ..., X_{i_d})^T$, the univariate density estimator can be generalized to multivariate density estimator as the following function

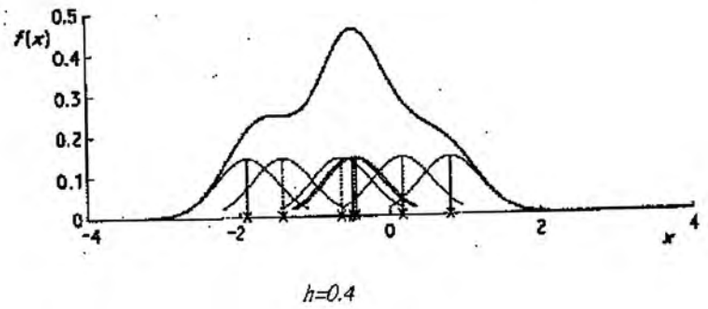


Figure 2.2: Kernel estimator showing individual kernels. $h = 0.4$.

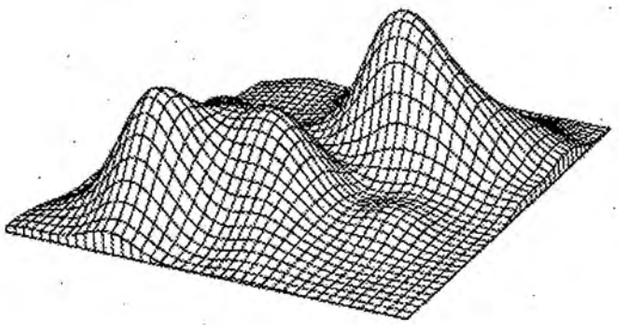


Figure 2.3: Multivariate density estimator for observations from bivariate normal mixture.

$$\hat{f}(x) = \frac{1}{nh^d} \sum_{i=1}^{n} K(\frac{X_i - x}{h})$$

Figure 2.3 [48] gives the picture of this multivariate density estimator. In OptiGrid [23], clusters is defined as subset of data points *density-attracted* by x^*, where *density-attracted* is defined by the gradient, and x^* is the local maxima of the density function which are above a certain noise level.

2.2.3 Upper Bound Property

A cutting plane is a (d-1)-dimensional hyperplane which partitions $\mathcal{R}^d$ into two half spaces. Many grid-based clustering algorithm uses cutting planes to partition the whole space into a number of grids, and detect clusters by

discovering dense grids. However, when the dimensionality is high, those algorithms suffer from the fact that the natural clusters could be easily partitioned into different grids by cutting planes, and thus lose the natural neighborhood. OptiGrid [23] first define a contracting projection for a given d-dimensional data space S and an appropriate metric $\|.\|$ as a linear transformation P defined on all points $x \in S$, such that $P(x) = Ax$ with $\|A\| = max_{y \in S}(\frac{\|Ay\|}{\|y\|}) \leq 1$. With certain definitions of clusters, it is found that the density at any point in the contracting projection serves as an upper bound for the density on the original space. OptiGrid [23] applies this idea to safely put the cutting planes at locations where their density on the contracting projection is lower than a certain threshold, because it guarantees that the corresponding density in the original full dimensional space cannot be higher. This *upper bound property* is actually a generalization of the Monotonicity Lemma, as stated in the paper discussing CLIQUE [6] which we will discuss next.

OptiGrid makes use of the *upper bound property* of the projection of data. Specifically, it constructs density estimation function with lower dimensionality, and places the cutting plane in low density region. The *upper bound property* ensures that the density in the full dimensionality where the cutting plane is placed would be no higher than the density in that lower dimensionality, and thus avoiding the lose of natural neighborhood.

2.3 CLIQUE and ENCLUS - Subspace Clustering

The automatical discovery of interesting subspace is first studied as the subspace clustering problem in CLIQUE [6]. Because natural clusters seldom exist in the full dimensional space as data points are usually sparsely distributed, CLIQUE aims to identify subspaces which allow better clustering of the data

points. At the same time, CLIQUE also maximizes the dimensionality of the identified subspace, since clustering on subspaces with higher dimensionality would give us more information then with lower dimensionality.

CLIQUE employs a bottom-up approach similar to Apriori [5] algorithm for mining association rules. It first partitions the data space into non–overlapping rectangular *units.* A unit is dense if the fraction of data points it contains exceeds a certain density threshold. A cluster is defined as a maximal set of connected dense units in k–dimensions. This approach bases on the following *monotonicity* property to discover higher dimensionality subspaces which contain clusters.

2.3.1 Monotonicity Property of Subspaces

If a collection of data points S is a cluster in k-dimensional space, S must also be part of a cluster in any (k-1) dimensional projections of this space.

In other words, space ABC can form clusters only if all of its subspace, that is A, B, C, AB, AC, BC can form clusters. If one of these subspaces fails to form cluster, it is guaranteed that no cluster could exist in the original space ABC. As a consequence, we can consider only higher dimensional spaces where all of its subspaces form clusters. Using bottom-up approach, we can prune out a lot of subspaces once we know any of their subspaces cannot form cluster. Examining all the possible subspaces requires $O(2^n)$ where n is the dimensionality, and the pruning can thus saves a lot of computational time. It is a special case of *Upper Bound Property Lemma* as mentioned in 2.2.3.

Following that, ENCLUS [12] extends the idea of subspace clustering by using entropy to further prune away uninteresting subspaces. However, these approaches do not aim to partition the data set. Rather, they report "dense" regions in each discovered interesting subspace. The large overlap among the

reported dense regions makes these approaches not very suitable for applications which need a clear partition of data objects [2].

2.4 PROCLUS Projective Clustering

PROCLUS [2] is developed to produce a clear partition of data objects, based on their corresponding subspaces. PROCLUS employs *Manhattan Segmental Distance* as similarity measurement to quantitatively describe how good a projected cluster is. Specifically, for any two d-dimensional data points $x_1 = (x_{1,1}, \ldots, x_{1,d})$ and $x_2 = (x_{2,1}, \ldots, x_{2,d})$, and for any subset S of the set of dimensions, $|S| \leq d$, the Manhattan segmental distance between x_1 and x_2 relative to S is given by $d_S(x_1, x_2) = \frac{(\sum_{i \in S} |x_{1,i} - x_{2,i}|)}{|S|}$. This distance metric is useful when comparing points in two different clusters that are associated with subspaces with varying number of dimensions.

With this distance metric, PROCLUS tries to minimize the intra-cluster distance, which is the sum of distances between data objects in a cluster and the centroid of the cluster. It works like a k–mean like algorithm, extended with the idea of projective clustering. First, a greedy approach is applied to select a set of potential medoids (the centers of the clusters). With the set of medoids, it estimates the correlated subspace for each cluster. To do this, the *locality* of a medoid is examined. Locality is defined as the set of data objects in a neighborhood region in the full dimensional space. The projections of these data objects on different dimensions are examined to find those dimensions that have closer average distances to the corresponding medoids. These are chosen as the dimensions of the correlated subspace. After the estimation of subspaces, data objects are assigned to its closest medoid with the distance measured with respect to the corresponding subspace. The quality of clustering, which is the sum of intra-cluster distance, is evaluated. Medoids are replaced in a hill-climbing approach, which targets to give an improvement on the clustering

quality.

This approach tackles the two sub–problems, namely discovering the subspaces and locating the position of clusters, iteratively until no further improvements can be made. One problem we can see is that the full dimensionality is used in forming the *locality.* This may not include the real neighbors in the correlated subspace and may include unrelated points in terms of the subspace. In fact, according to [8] it makes little sense to look for neighbors in the high-dimensional space. Another problem is that it may not be accurate to use the average distance of data points in the locality from the medoid on each dimension in finding the subspace, since the average distance can hide information about multiple clusters. PROCLUS takes the value of k, that is the number of projected clusters, and the average cardinality l of subspaces as input. Both parameters can greatly influence the quality of the clustering results. These pose some limitations when users do not have a good idea of k or l.

2.5 ORCLUS - Generalized Projective Clustering

ORCLUS [3] further extends and generalizes the idea of projective clustering. The difference between PROCLUS and ORCLUS is that PROCLUS is targeted to find clusters with subspaces formed by axis-parallel vectors, while ORCLUS is to find clusters existing in arbitrarily oriented subspaces. Often the data points may tend to get aligned along arbitrarily skewed and elongated shapes in lower dimensional space. ORCLUS can discover clusters with these kinds of data points, and also the hidden subspaces, which exist because of inter-attribute correlations.

2.5.1 Singular Value Decomposition SVD

ORCLUS makes use of Singular Value Decomposition (SVD), which is a well known technique to transform high dimensional data into a lower one, with the least loss of information. SVD transforms the data to a new coordinate system in which the correlations in the data are minimized.

First, *covariance matrix* of the data set is constructed, and the corresponding eigenvalues are calculated. The eigenvectors for which the corresponding eigenvalues are the largest can define the subspace in which the data will be projected. Because the data does not show much variance if the corresponding eigenvalue is large, the projection to the resulting subspaces would incur the least loss of information.

On the contrary, ORCLUS chooses the eigenvectors with minimum spread to do the projection, so that the greatest amount of *similarity* among the data points in the clusters can be detected.

Although ORCLUS solves a more general problem, the computation involved is more complex. In its merging stage, the eigenvectors with least spread for each pairs of the remaining clusters have to be computed, which takes $O(d^3)$ by using ECF (Extended Cluster Feature Factor), where d is the dimensionality of the original space. This would be prohibitively expensive for very large database and high dimensionality.

2.6 An "Optimal" Projective Clustering

There are two objectives at odd with each other in the projective clustering problem. We would like to discover clusters with as many data objects as possible, so that the discovered patterns are strongly relevant. On the other hand, we also want the dimensionalities of uncovered associated subspaces to be as high as possible, so that it captures more amount of correlations among dimensions.

However, these two objectives are conflicting to each other. Consider if one associated subspace contains all the dimensions, in such high dimensional subspace, according to [8], data objects tend to be very sparse, there should be only few data objects which are close together to form cluster. On the other hand, if we group all the data objects into a single cluster, we cannot find any dimensions where all the data objects are closely located, and thus the dimensionality of the subspace would tend to be 0.

Recently, [35] propose a concept of an "optimal projective cluster". Specifically, it first defines an α-dense projective cluster as a set of data objects, such that the number of data objects exceeds αN, where N is the total number of data objects, and the maximum distance on a related dimension between any pair of data objects in this set is less than another parameter ω. The quality of a projected cluster is defined as $\mu(|\mathcal{C}|, |\mathcal{D}|)$, where $|\mathcal{C}|$ is the number of data objects, $|\mathcal{D}|$ is the dimensionality of the subspace, and μ is a monotonically increasing function. With this function, more number of data objects and higher dimensionality of the associated subspace indicates a better cluster.

To measure the trade-off between the two objectives of increasing the number of data objects being included, and increasing the dimensionality of associated subspace, it employs a so called "β-balanced" to restrict the function μ. μ is β-balanced *if* $\mu(a, b) = \mu(\beta a, b+1) \forall a > 0, b \leq 0$. One example of such function is $\mu(\beta a, b+1)$.

We did not follow this definition of "optimal" projective cluster, because both the parameters α and ω are difficult for user to estimate. Furthermore, clusters are often with varying spanning area, number of data objects being included, and thus the density. Using a fixed values of α and ω restricts the spanning area and number of data objects being included of the discovered clusters. However, the objectives of increasing number of data objects and dimensionality of associated subspaces in each cluster, as well as the balancing between these two odd objectives are strongly relevant.

Chapter 3

EPC : Efficient Projective Clustering

3.1 Motivation

In the high dimensional space, not all dimensions are equally important. For some dimensions, data is just randomly distributed and does not exhibit any special patterns. These dimensions are considered not important, and not interesting to the users. In the clustering process, we want to find these kinds of dimensions as soon as possible, so that we need not to consider them. This does not only help to speed up the process, but at the same time helps to remove impurities.

When a natural cluster exists in certain subspace, for example subspace **XYZ**, the density within the area spanned by this cluster, with respect to subspace **XYZ**, should be higher than its neighbor area. In many cases, the projection of this cluster on each dimension **X,Y,Z** would also demonstrate a higher density. From this observation, if we consider projections parallel to the original axis, in this example, that is subspaces formed by any subset of **XYZ**, density of the area spanned by the projected cluster in these subspaces should also be higher than area with no projected clusters.

Projection to subspaces of lower dimensionality may incur loss of information. For example, we may not be able to locate the boundary of the cluster in the original subspace. Consider Figure 3.1, there are 3 reported hyper–rectangles containing irregular shaped clusters. We can know from the hyper–rectangles where the clusters are located in its associated subspace, but we may not outline the exact shape of the cluster. For example, in hyper–rectangle 1, we can not tell whether point A belongs to the cluster enclosed in this hyper–rectangle, and in hyper–rectangle 3, it includes two projected clusters with the same associated subspace.

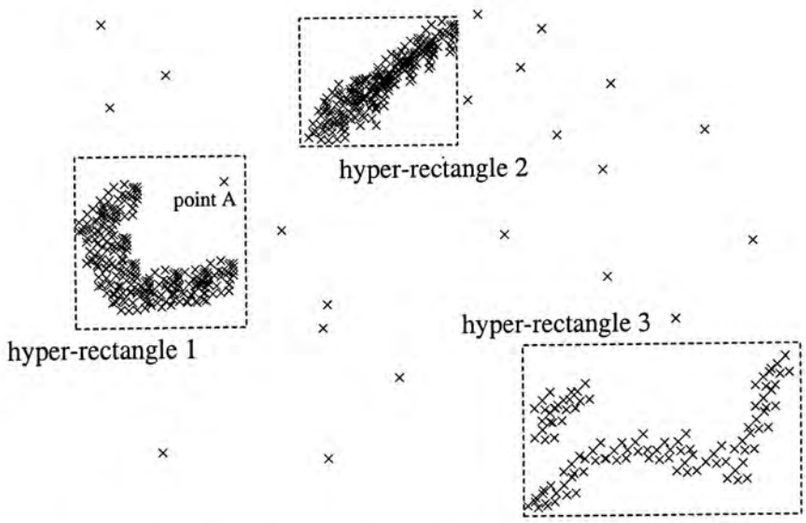


Figure 3.1: 2-d hyper-rectangle contains irregular shaped clusters.

However, since we are mostly interested in solving the more difficult problem of finding the associated subspaces of each cluster, and in which area the clusters roughly locate, the loss of information actually helps to remove the unnecessary details. Detail shape of the original cluster can be found later, by feeding data objects within the hyper-rectangle together with the discovered subspace, to other clustering algorithm which works in the full dimensional space, such as OPTICS [34].

Therefore, we propose a technique of projective clustering called EPC. EPC makes use of equi-width histograms to model and represent the data distribution in database system. Histogram is a commonly used technique in current

RDBMS to approximate distribution of values in the attributes of relations. These domain knowledge can help to estimate query result sizes and access plan costs[24]. Many RDBMS keeps updating the histograms frequently, in such case the histograms are always ready to be used by our method without additional computation. If the histograms have not been constructed in apriori, (as in our experiments), they can be built quickly in real time. This property enable us to develop very fast projective clustering algorithm scalable to very large database.

3.2 Notations and Definitions

N	number of data objects
D	dimensionality of the original space
K	number of natural clusters
m	maximum number of clusters that user is interested to find out
l	Average Dimensionality of the associated subspaces of clusters

Table 3.1: Symbols used in EPC.

Definition 1 A *data object* x_i is a single data item. It is represented by a vector of D measurements in the D-dimensional space : $x_i = (x_{i,1}, ..., x_{i,D})$, where $x_{i,j}$ is an **attribute** of x_i.

Definition 2 A **subspace** S_i with dimensionality d_i is defined by a set of d_i orthogonal vectors, where $d_i < D$.

Definition 3 A **dense region** $DR_m^{S_i}$ associated with subspace S_i is an intersection of intervals from the dimensions in S_i, in which the data density inside the intersection area is larger than a certain threshold. This is intended to contain the projection from one or more projected clusters on the subspace. m is the **dense_region id** of the dense region.

Notice that we do not require the user to specify the threshold value, it will be derived from the data set.

Definition 4 A **projected cluster**, or simply a cluster, C_i associated with subspace S_i is a set of data objects, which are closely clustered when projected in the subspace S_i.

Definition 5 We call the subspace of a projected cluster the **associated subspace** of the cluster. We call the dimensions being included in the associated subspace the **bounded dimensions** of the cluster, and others as **unbounded dimensions**. Similarly, we call any subset of the associated subspace **bounded subspace**. We would use **A, B, C** etc. to denote dimension, and **AB**, **BC** etc. to denote subspace.

EPC first models the density of data objects along each dimension. A most commonly used method in statistics ([48], [51]) is by means of the **density estimation function**.

3.2.1 Density Estimation Function

We shall adopt the concept of density estimation functions in statistical literature.

Suppose we are given a set of 1-dimensional N data points $O_1, O_2, \ldots, O_N$ with positions $X_1, X_2, \ldots, X_n$. The **density estimation function** f can be constructed by [48]:

$$\hat{f}(x) = \frac{1}{Nh} \sum_{i=1}^{N} K(\frac{X_i - x}{h}) \tag{3.1}$$

where $K(x)$ satisfying $\int_{-\infty}^{\infty} K(x)\,dx = 1$, is the *kernel function*, and h is called the *smoothing factor*. A simple K function is

$$K(x) = \begin{cases} 0.5 & if|x| < 1 \\ 0 & otherwise \end{cases}$$

In computing $\hat{f}(x)$, this definition of K gives a weight of 0.5 to the data points within a distance of h from x, and zero weights to all other data points. Therefore, the value of $\hat{f}(x)$ is **influenced** only by the values of $X_1, ..., X_n$ that are at a h neighborhood of x.

Intuitively, every data object has its influence to the overall density estimation function. In general, we can set different K and h, the shape of the influence of data on f would be determined by the kernel function, while the width is determined by the smoothing factor. Typically data points at a closer neighborhood would have a higher influence.

3.2.2 1-d Histogram

One simple way to construct the density estimation function on every dimension is by using **histograms**, which could help us to identify regions where the data objects are densely located. To build the histogram, we divide each dimension into a number of *equi-width intervals.* Let i be the number of intervals, $I_{A1}, I_{A2}, \ldots, I_{Ai}$ denote the $1^{st}, 2^{nd}, \ldots, i^{th}$ intervals of dimension **A**, and h be the width of the *equi-width intervals.* If the domain range of dimension **A** is $[x, y]$, and there are i intervals, then I_{A1} is the interval of $[x, x+h)$, I_{A2} is $[x+h, x+2h)$,... I_{Ai} is $[x+(i-1)h, x+ih]$.

With the equi-width intervals on dimension **A** we can determine the **density estimation function for dimension A** as

$$\hat{f}^A(x) = \frac{1}{16N}(4|\mathcal{X}_{Ai}| + 3(|\mathcal{X}_{A(i+1)}| + |\mathcal{X}_{A(i-1)}|) + 2(|\mathcal{X}_{A(i+2)}| + |\mathcal{X}_{A(i-2)}|) +$$

$$(|\mathcal{X}_{A(i+3)}| + |\mathcal{X}_{A(i-3)}|))$$

where the value of x is located in interval I_{Ai}, and $|\mathcal{X}_{Ai}|$ is the number of data objects whose projection on dimension **A** are located in I_{Ai}. This equation can be understood as adding up the **influences** of data objects. The interval

containing x would have more influences on $\hat{f}^A(x)$, and neighboring intervals farther away from x would have less influences. Note that this definition follows the concept of influences as defined by the Kernel function and the smoothing factor but does not follow the format of the Kernel function.

Example 1 In a give data set, suppose the domain of dimension **A** ranges from -100 to 100, and the value of h for interval width is 1 for this dimension. Consider a data object $\mathcal{O}$ with a projection on dimension **A** equal to 5.57. $\mathcal{O}$ is located in the 106^{th} interval on **A**. The influences of $\mathcal{O}$ on the neighborhood in the 1-d histogram is shown in Figure 3.2.

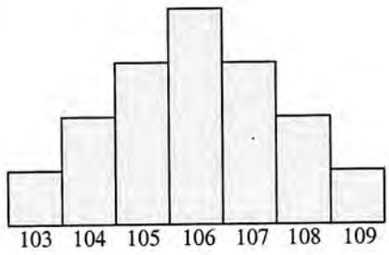


Figure 3.2: The influence of a data object in the 1-d histogram.

The function $\hat{f}^A(x)$ effectively defines the values in the 1-d histogram for dimension **A**. In the 1-d histogram, we need to define the a value $\mathcal{H}_A(I_{Ai})$ for each interval I_{Ai}. The value of $\mathcal{H}_A(I_{Ai})$ is set to be the value of $\hat{f}^A(x)$ for any data point x that lies within the interval I_{A_i}. We shall refer to the histogram as $\mathcal{H}_A$.

Example 2 Consider a data set with 10 data objects, four of them fall on the 5^{th} interval on dimension **A**, another four of them fall on the 6^{th} interval on dimension **A**, the last two falls within the 14^{th} interval. The histogram on **A** can be built as in Figure 3.3. Let us examine the influences of the first 8 data objects. For $\mathcal{H}_A(I_{A4})$ (histogram value at interval 4), influences of data objects located in the 5^{th} interval is $\frac{1}{16\times10}(3\times4) = 0.075$, and influences of data objects located in the 6^{th} interval is $\frac{1}{16\times10}(2\times4) = 0.05$. Therefore, $\mathcal{H}_A(I_{A4})$ is $0.075 + 0.05 = 0.125$. Values in other intervals can be calculated similarly.

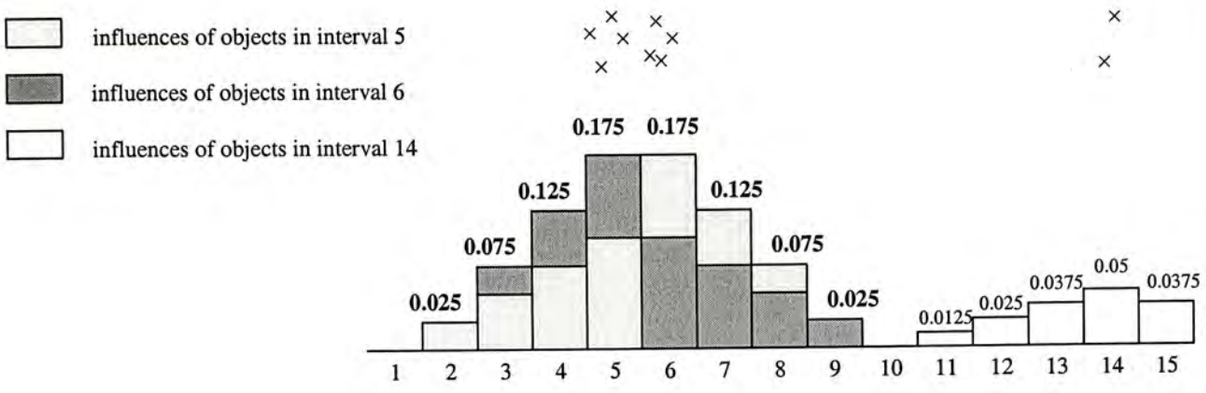


Figure 3.3: Example of an 1-d histogram.

3.2.3 1-d Dense Region

We shall find the regions where data objects are densely located along each dimension, since these regions are potentially the 1-d projections of some natural clusters. We would next define what we mean by a "dense region". Given a 1-d histogram $\mathcal{H}_A$, The set of 1-d **dense regions** of dimension **A**, denoted by $\mathbf{DR}^A$, is defined as a list of ordered pair (p, q) satisfying:

(1) $\mathcal{H}_A(I_{A(p-1)}) < \zeta$

(2) $\mathcal{H}_A(I_{A(q+1)}) < \zeta$

(3) $\mathcal{H}_A(I_{Ai}) \geq \zeta$ for all $p \leq i \leq q$

Therefore p and q represent lower and upper boundary *equi-width intervals*, such that the densities in between intervals are higher than a given threshold ζ for dimension D. There can be more than one such pair of boundary intervals on a dimension. We denote $\mathbf{DR}_i^A$ as the i^{th} such ordered pair for the dense regions on dimension **A**.

Example 3 Consider dimension **A** with 10 equi-width intervals, its histogram is shown in Figure 3.4. The dotted line represents the value of ζ, based on the previous definition, $\mathbf{DR}_1^{\mathbf{A}}$ is (2,4) and $\mathbf{DR}_2^{\mathbf{A}}$ is (7,9).

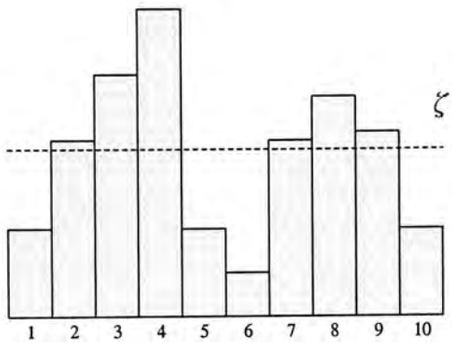


Figure 3.4: Discovering dense region with a given threshold.

3.2.4 Signature Q

When we have uncovered sets of dense regions for all dimensions, we can determine for each data point, whether it is in the scope of each of such a region. In each dimension, a data point p can lie inside one or zero dense region. When we combine this information for all dimensions, we get a **signature** for the data point p.

Given the dense regions $\mathbf{DR}^A$, the signature of a data object, denoted by Q, is an ordered list of k entries, where the j^{th} entry represents the dense region, if any, where the data object is located in dimension j. Specifically,

$$Q = [Q_1, Q_2, \ldots, Q_d]$$

$$\text{where } Q_i = \begin{cases} 0 & \text{if the object does not fall into any region in } DR^j \text{ in dimension } j, \\ i & \text{if the object is located in } DR_i^j \text{ in dimension } j. \end{cases}$$

Each Q_i in the above is called a **component** of the signature. Each data object can be assigned a signature according to the generated $\mathbf{DR}_A$. The signature can be interpreted as the compressed form of the representation of the data object and it helps to remove the unnecessary details of each individual object when we are targeted to discover the underlying clustering structure.

Data objects within the same projected cluster, which probably are located in the same dense region on the correlated dimension, would most likely be

assigned the same signature. We keep an **object count** for each signature, which is the number of data objects that are assigned to the signature. Therefore, by identifying the signatures with high object counts, we can find the corresponding subspaces and locations of the clusters.

Example 4 Suppose there are 1000 data objects located in a 3 dimensional space, ranging from -100 to 100 in the coordinates. Each of the three dimensions, **A**, **B**, and **C**, consists of 10 *equi-width intervals*. Figure 3.5 shows the 1-d histograms for each of the dimensions.

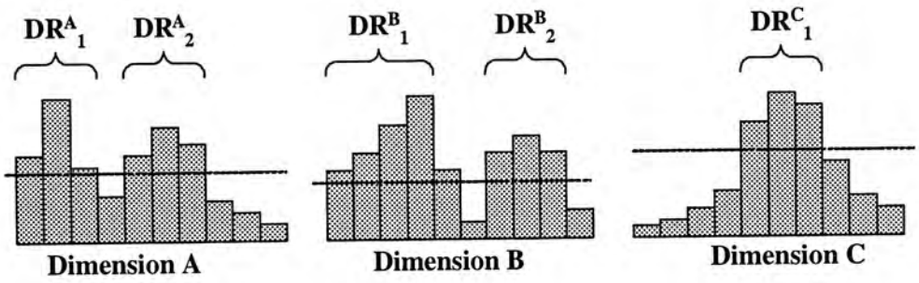


Figure 3.5: Histograms for 3-d data objects projected on each dimension.

From these histograms and the given threshold (the dotted line), the dense regions are

$\mathbf{DR}^A = ((1,3),(5,7))$

$\mathbf{DR}^B = ((1,5),(7,9))$

$\mathbf{DR}^C = ((5,7))$

Consider 5 data objects: O_1 = (-100, 70, -80), O_2 = (50, 10, 40), O_3 = (-80, 80, 40), O_4 = (30, -50, 20), O_5 = (-60, 50, -50). Their corresponding signatures would be [1,2,0], [0,0,0], [1,2,0], [2,1,1], [1,2,0]. From their signatures, object O_1, O_3, and O_5 probably belong to the same projected cluster, in subspace **AB**. Object O_2 should be an outlier as it does not fall into any dense regions in all dimensions.

3.3 The overall framework

EPC models the densities of the data objects on each dimension by constructing 1-d histograms, and determines the dense regions where data objects are located closely together. Signature of each data object can be derived from the discovered dense regions. With a high chance, objects in the same projected cluster would also be in the same dense regions along each axis-parallel 1-d projections of its associated subspace. Data objects with the same and "similar" signatures are grouped together. Thus, a *signature group* with a large number of data objects corresponds to certain combinations of dense regions in each dimension, which help us to locate a projected cluster. When we have a signature with a large object count, the non-zero entries in the signature correspond to the correlated dimensions in a cluster, and the exact values of these entries help to determine the location of the cluster in its associated subspace.

EPC needs only one input, *max_no_cluster*, which we denote as m. In case the number of natural clusters is smaller than m, the algorithm will return only the discovered clusters. Otherwise, it will return the top m clusters with the largest object counts. EPC always output the discovered projected clusters according to the order of the corresponding object counts. From experiments, this is the same as the ordering of natural projected clusters in descending number of data objects in most of the cases.

The output of EPC would be the number of discovered clusters, the associated subspace for each cluster, as well as the hyper-rectangle regions which contain the clusters.Figure 3.6 shows the pseudo code of the algorithm EPC.

Algorithm 3.1 Pseudo Code of EPC

```
1  /*Histogram Generation*/
2  for each data object
3      for each dimension
4          add influence of the data object to the corresponding interval
5
6  /*Adaptive discovery of dense region*/
7  for each dimension
8      repeat until (no more dense regions are discovered or m dense regions have
       been discovered)
9          scan the histogram to set the new threshold value
10         scan the histogram to locate and store the identified dense regions in a
           linked list structure
11         remove the identified dense regions as well as their densities from the
           histogram
12     end until
13
14 /*Derive and update signature*/
15 for each data object
16     for each dimension
17         derive the signature component from identified dense regions
18     search the entry of the signature by a hash function
19     if found
20         update the object count
21     else
22         add a new entry with an initial object count of 1
23
24 /*Find the most popular signatures*/
25 sort all the derived signatures in descending order according to their object count
26 keep 3m signatures with the largest counts and discard the remaining
27
28 /*Refine the top 3m signatures*/
29 for each pair of different signatures
30     count the number of differing components in the signatures
31     if (the number of differing components < D/10)
32         merge the signatures by discarding the one with less count and add its
           count to the other one
33 discard any signature if all the components are 0, or the object count < N/10m
34 transform the remaining signatures to their corresponding subspaces and hyper-
   rectangles
35
36 /*Partition the data objects*/
37 for each data object
38     assign it to clusters or outliers based on the subspaces and hyper-rectangles
39 return the subspaces, hyper-rectangles and partitioning
```

Figure 3.6: The EPC Algorithm.

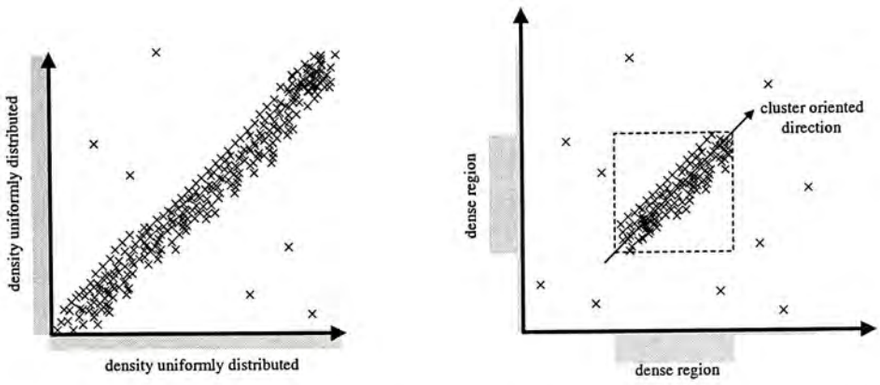


Figure 3.7: (a) 1-d projection of the cluster is uniformly distributed across the whole dimension, (b) density of the 1-d projection is higher than background noise for cluster oriented in the direction not parallel to the axis.

3.4 Major Steps

There are various technical details that are important for the efficiency or the quality of the results. They are discussed in this subsection.

3.4.1 Histogram Generation

This step can be understood as compressing the information about all the data objects to D 1-dimensional histograms, which enables us to discover the interesting subspace very quickly. This compression introduces information loss. Clusters that is not dense enough in any one of dimensions could not be discovered. For example, when the the 1-d projection of the cluster spans across the whole dimension, as shown in Figure 3.7 (a) where the 1-d projection is uniformly distributed across the whole dimension. In general, even the cluster is oriented in the direction not parallel to the axis, it can still be discovered providing that its densities of its 1-d projection is higher than background noise. Figure 3.7 (b) shows this case.

One important parameter of a histogram is the bin width h, which is the width of the *equi-width interval* introduced in Section 3.2.2. It controls the trade-off between undersmoothing or oversmoothing the true distribution. In

statistics literatures, there are several rules suggesting how to set an appropriate value of h, or the number of *equi-width interval*. Sturges' rule [51] suggests that if the data follows a normal distribution, the number of *equi-width interval* for an ideal frequency histogram should be $1 + log_2 N$. If the data are not normal, but are skewed or kurtotic, additional bins may be required. Furthermore, it was proposed to increase the number of bins to $log_2(1 + \hat{\gamma}\sqrt{\frac{n}{6}})$, where $\hat{\gamma}$ is an estimate of the standardized skewness coefficient.

In addition, it has been shown [50] that the optimal histogram bin size W can be obtained when $W = 3.49\sigma N^{-1/3}$, where σ is the standard deviation of the distribution. However, we usually do not know the values of $\hat{\gamma}$ and σ before constructing the histogram; we cannot have an accurate estimation without data analysis with high complexity. Therefore, we have used a greater number in Sturges' rule, since we would like to cater for other kinds of data distributions. In the experiments, we set the number of *equi-width interval* to be 200 for dimension ranges from -100 to 100. Since the computational complexity is linear to this number, using a greater number is highly feasible.

3.4.2 Adaptive discovery of dense regions

The main challenge of the discovery of dense regions is how we can determine the best threshold value to locate the projections of natural clusters, such that (1) it does not require the user to input, (2) it can distinguish two kinds of *equi-width interval*; those contains the projection of clusters, and those contains only random noise. We allow values of the thresholds along different dimensions to be different, which is driven by the data set distribution. Clusters vary with their underlying distribution, spanning area, and number of data objects being contained.

We explain with the help of a single cluster projection. Often a cluster is more dense in its center, and less dense in the boundary, in where the

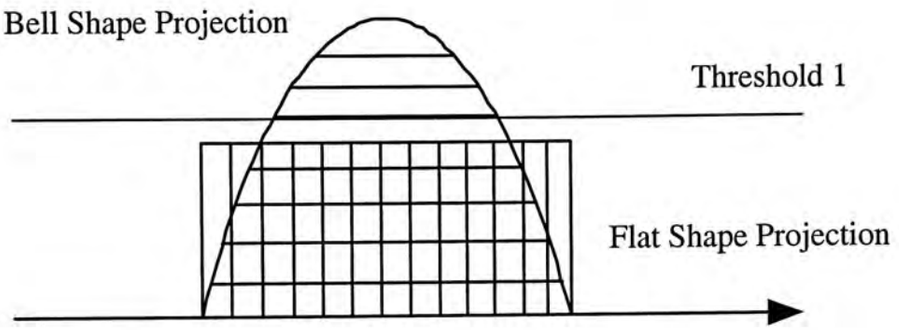


Figure 3.8: Bell shape cluster and flat cluster.

density approaches random noise. Consider two extreme cases in Figure 3.8. One projection of the cluster establish a "bell" shape which is denser in the center, while another one form a "flat cluster" in which density inside the cluster is constant. Even though the two clusters contain the same number of data objects, the latter is more difficult to be discovered by a single threshold value. Threshold 1 can detect the "bell" cluster, although missing some data objects in the boundary, but will miss the "flat" cluster. In general projections with uniform density are more difficult to be detected, since there are no peak regions with the highest density in the projection. Therefore, we use cluster in "flat" shape in the following analysis.

Consider Figure 3.9, the total number of data objects is N, and the total number of histogram intervals in the subspace is H. There is a projection in the subspace which contains F of N data objects, where $0 \leq F \leq 1$. The projection spans f of H histogram intervals, where $0 \leq f \leq 1$. We establish a lemma which can tell us how to set the appropriate value of the threshold to detect dense projections, based on the mean μ, standard deviation σ of the data distribution, and the users' expectation on how large a cluster's projection would span. Suppose the 1–d histogram for a dimension D is given by $\mathcal{X} = \{X_1, X_2, ..., X_p\}$. Let the mean value ($\frac{1}{p}\sum_{i=1}^{p} X_i$) be μ. The standard deviation σ is given by $\sqrt{\frac{1}{p}\sum_{i=1}^{p}(X_i - \mu)^2}$. The **density** of a projection is defined as number of data objects falling onto the projection divided by number

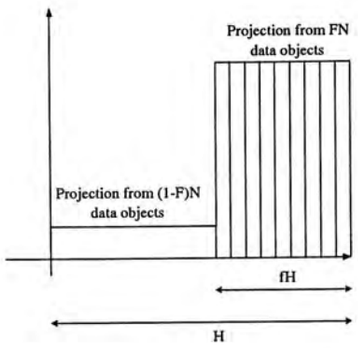


Figure 3.9: Setting the appropriate threshold value.

of intervals the projection spans.

Lemma 1 If H is the number of histogram intervals on a subspace and a projection on this subspace consists of an uniform high density region spanning f of H, and a low density region spanning $(1-f)$ of H, let ρ be the higher density of the projection, μ be the mean, σ be the standard deviation of the distribution, $\rho = \mu + \sqrt{\frac{1}{f} - 1}\sigma$.

Proof: Without loss of generality, we move the region covered by the projection with the highest density to the right, leaving all remaining random noise to the left of the histogram, as in Figure 3.9. Let $(X_1, ..., X_n)$ be the histogram values. Mean of the histogram values $\mu = \frac{N}{H}$. For the high density region, $\rho = \frac{FN}{fH}$, since there are fH intervals with high density. For the low density region, its density is $\frac{(1-F)N}{(1-f)H}$, since there are $(1-f)H$ intervals with the low density. The square of the standard deviation σ can be expressed in terms of f, F, N, H as follows:

$$\begin{aligned}
\sigma^2 &= \frac{\sum(X-\mu)^2}{H} \\
&= \frac{[(\frac{FN}{fH} - \frac{N}{H})^2 fH + (\frac{(1-F)N}{(1-f)H} - \frac{N}{H})^2 (1-f)H]}{H} \\
&= \frac{N^2}{H^2} \frac{(F-f)^2}{f(1-f)}
\end{aligned}$$

Now, we can derive a relationship of ρ with respect to μ, σ as follows:

$$\begin{aligned}(\rho-\mu)^2 &= (\frac{FN}{fH}-\frac{N}{H})^2 \\ &= \frac{N^2}{H^2}(\frac{F}{f}-1)^2 \\ &= \frac{N^2}{H^2}(\frac{F-f}{f})^2\end{aligned}$$

$$(\rho-\mu)^2 = \sigma^2(\frac{1}{f}-1)$$

$$\rho = \mu + \sqrt{\frac{1}{f}-1}\sigma$$

■

Hence ρ can be expressed as $\mu + c\sigma$, and value of c depends on the spread of the high density region in the histogram projection. Note that the high density occurrence can be in multiple intervals, i.e. if there are a number of clusters and their projections in the histogram are not continuous, the lemma still applies.

With this lemma, we can set the threshold to be lower than $\mu + c\sigma$ to detect the projections with the highest density in the subspace. The value of c is set according to the users' expectation on the number of intervals the most spreading projections, that is the projections spanning the largest number of histogram intervals on the subspace, should span. For example, if the most spreading projections should span at most $\frac{1}{2}$ of the total number of intervals on the dimension, we can set $c < \sqrt{\frac{1}{f}-1} = \sqrt{\frac{1}{\frac{1}{2}}-1} = 1$. If it should span at most $\frac{1}{3}$ of intervals, we can set $c < \sqrt{\frac{1}{\frac{1}{3}-1}} = \sqrt{2}$.

In the case when there are multiple projections on the subspace, we adopt

an adaptive approach to iteratively lower the threshold value until no more dense regions are discovered.

Specifically, we set the initial threshold value to be $\mu + c\sigma$ according to lemma 1. In the first iteration, projections with densities lower than the threshold would be treated as random noise. After detecting projections with the highest density, their densities would be removed from the histogram. The new μ, σ and threshold would be calculated . This process is repeated until either no more dense regions are discovered, or at most m time, since the user expects to find at most m projected clusters. This method can effectively detect the peak values and extract the corresponding dense regions, without pre–defining the threshold. This idea is illustrated in Figure 3.10. In the figure, the dotted lines indicate the threshold values used in the iterations.

In the example, we can also see another detail in the algorithm where adjacent dense regions are merged to form bigger dense regions. For example in Iteration 1, the 5th interval is considered dense, in Iteration 2, intervals 3, 4, and 6, 7 are considered dense. So in Iteration 2, a bigger dense region of $DR1$ is formed which ranges from interval 3 to interval 7.

We aim to find no more than m dense regions in each dimension. If at the last iteration, totally more than m dense regions are found, we shall keep only the m regions with the greatest number of data points.

Lemma 1 helps to explain why our method works very well in many cases. A natural cluster typically does not span the entire domain when projected to some low dimensional subspace. Lemma 1 shows that when projected to one dimension, as long as the cluster does not span more than half the domain, the density threshold of $\mu + \sigma$ is able to detect it. If the cluster spans less of the domain, a higher threshold can be set accordingly.

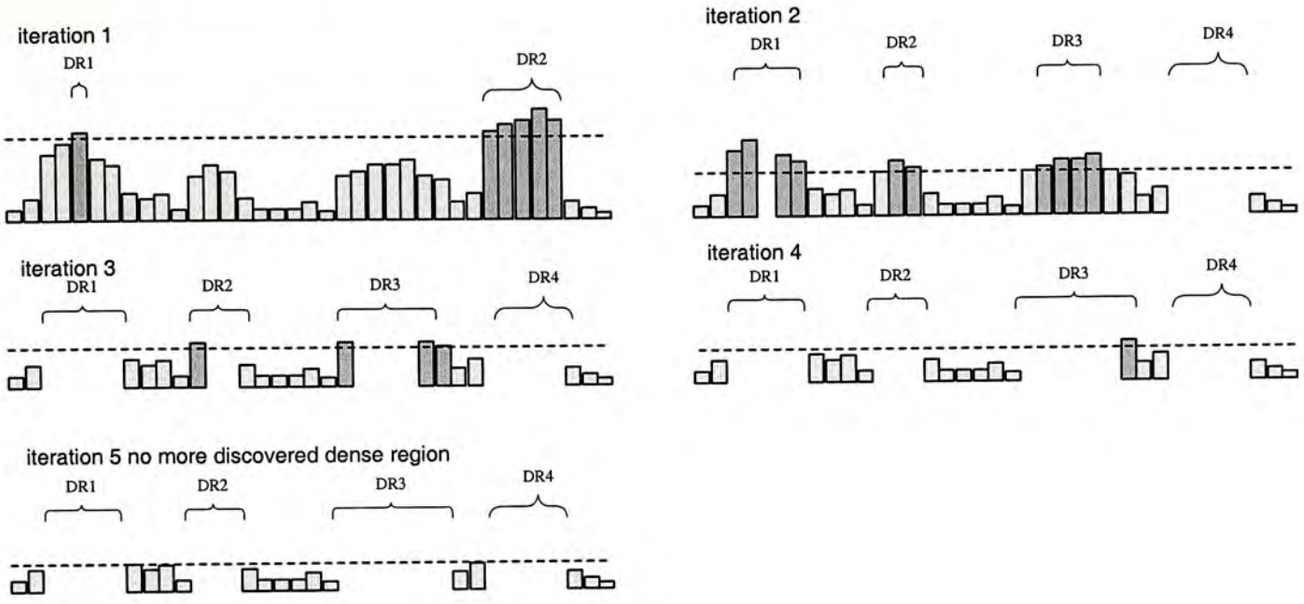


Figure 3.10: Adaptive approach to locate dense regions.

3.4.3 Count the occurrences of signatures

Given a data object, after generating its signature, we have to efficiently search if the signature has been recorded, and if so update the object count. If the number of dense intervals in each dimension is small, we can simply use $[Q_1, ..., Q_d]$ as the index. In other cases, to speed up this searching we use a **hash function** to map the signatures to the memory buckets. Suppose we have a signature $Q = [Q_1, Q_2, ...Q_d]$ the hash function will map this to a bucket with an address of $Q_1 + Q_2 + ... + Q_d$. If two signatures are hashed to the same address, then a linked list is used to keep the overflowing buckets.

When there exists an underlying clustering structure, many data objects should share the same signatures. Thus there should only be a small number of distinct signatures compared with the total number of data objects.

3.4.4 Find the most frequent signatures

All the signatures can be ranked in descending order of object counts, by a classical quick sort algorithm. Those top ranked signatures tell us the locations and the associated subspaces of clusters containing the largest number of data

objects. We are only interested in discovering at most m clusters, so it seems we need only keep m signatures. However, some of the signatures can be 'similar', and they may actually refer to the same projected cluster (this relationship would be discussed in the following subsection). Therefore we have to keep more than m signatures. From experiments, it is sufficient to keep $3 \times m$ of the top ranked signatures and discard the remaining ones.

3.4.5 Refine the top $3m$ signatures

If we find a signature Q with a large object count, and there are occurrences of some similar signatures with smaller counts, we may want to merge the similar signatures to Q and hence their counts. The reason for this can be explained with an example. Suppose in a 3-dimensional space **ABC**, we have a projected cluster with subspace **AB**, and **C** is an unrelated dimension. As expected, most data objects should have the same signature, say [1,2,0]. Occasionally, as data in the cluster are randomly distributed in dimension **C**, some objects may fall in some other dense regions in this dimension, which is contributed by other projected cluster with **C** as a correlated dimension. As a result, some data objects in this cluster may have signatures [1,2,2], [1,2,5], etc. We observe that if signature [1,2,0] has a high object count, while signatures [1,2,2], [1,2,5] have lower object counts, probably [1,2,2] and [1,2,5] refer to the same cluster as [1,2,0].

The question is how we can determine if two signatures are 'similar'. Currently we employ the heuristic that two signatures are similar if their number of differing components $\leq \frac{D}{10}$.

There may be cases where two clusters are located closely in the common set of correlated dimensions, and would be merged as a single cluster. We can identify such cases by post-processing using other traditional clustering methods. However, for large data set with sparsely distributed objects and

high dimensionality, this case rarely occurs.

3.5 Time and Space Complexity

Let s be the number of different signatures generated. Let p be the number of equi-width intervals at each dimension. For time complexity, it takes $O(DN)$ time in the first step for histogram generation (Lines 1–4 in Algorithm 3.1), since it sums influences of each data object for each dimension. In the second step adaptive discovery of dense region (Lines 7 – 12 in Algorithm 3.1), it takes $O(mpD)$ time as it takes at most m iterations, and in each iteration, each of the D 1–d histogram is scanned twice. By the way we uncover dense regions, the number of dense regions at each dimension is $O(m)$.

In the third step for deriving and updating signatures (Lines 15 – 22 in Algorithm 3.1), for each data object, it needs to check $O(m)$ dense regions on each of the D dimensions, and thus it takes $O(mND)$ time for generating all signatures. Once a signature is generated, the address of the bucket storing this signature entry is calculated by a hash function, which takes constant time.

Let s be the number of remaining signatures. To find the most popular signatures (Lines 25–26 in Algorithm 3.1), it takes $O(slogs)$ time on the average for using a quick sort.

For signature merging (Lines 29 – 34 in Algorithm 3.1), we need to compare every pair of signatures to determine whether they are "similar", this takes $O(s^2D)$ time. To remove the outlier groups and transform the remaining signatures to corresponding subspaces and hyper–rectangle, it takes at most $O(sD)$ time. Finally, to partition the data objects (Lines 37 – 39 in Algorithm 3.1, we have to check each data object against all cluster configurations, it takes $O(mND)$.

Since we can assume $N >> p$, the total running time would be $O(mND)+$

$O(s^2D)$. We can also assume that $mN > s^2$, so that the running time is dominated by $O(mND)$. Therefore the time complexity is linear in the number of data objects, the number of dimensions and the number of clusters.

For space complexity, storage needed to store the D 1–d histograms is $O(pD)$. The linked list used to store all discovered dense regions needs $O(mD)$ space, since there are $O(m)$ dense regions for each dimension. We use a hash table to keep signatures and their object counts. Since at each dimension we uncover $O(m)$ dense regions, so the number of possible values for $Q_i, 1 \leq i \leq D$ is $O(m)$, for a signature $Q = [Q_1, Q_2, ..., Q_D]$. We need $O(s)$ space for handling overflowing buckets. Hence the space needed to store the hash table for signatures with the counts is $O(mD + s)$.

Chapter 4

EPCH: An extension and generalization of EPC

4.1 Motivation of the extension

EPC outperforms PROCLUS in terms of clustering quality and actual running time, from the experimental results presented in Chapter 5. However, when we exam data sets with more complicated data distribution, we observe that there are three types of associated subspaces, as shown in Figure 4.1. Type I is the set of subspaces formed by axes parallel vectors, and the spreading directions of data objects are also parallel to original axes. Type II is a superset of Type I, which includes subspaces formed by axes parallel vectors, but we relax the restriction that the spreading of data objects can be in arbitrary directions on the corresponding subspace. Type III is the most relax one, it includes subspaces formed by any sets of orthogonal vectors, in which the vectors need not to be parallel to the original axes. For example, for a data set in the 3-d space **ABC**, there exists a projected cluster with 2-d subspace, composed by 2 orthogonal vectors $v_1 = 0.5A + 0.2B + 0.4C$ and $v_2 = 0.2A + 0.5B - 0.5C$. Neither v_1 nor v_2 is parallel to the original axes. Notice that even the subspace is formed by 2 vectors, both vectors are contributed by all the three dimensions..

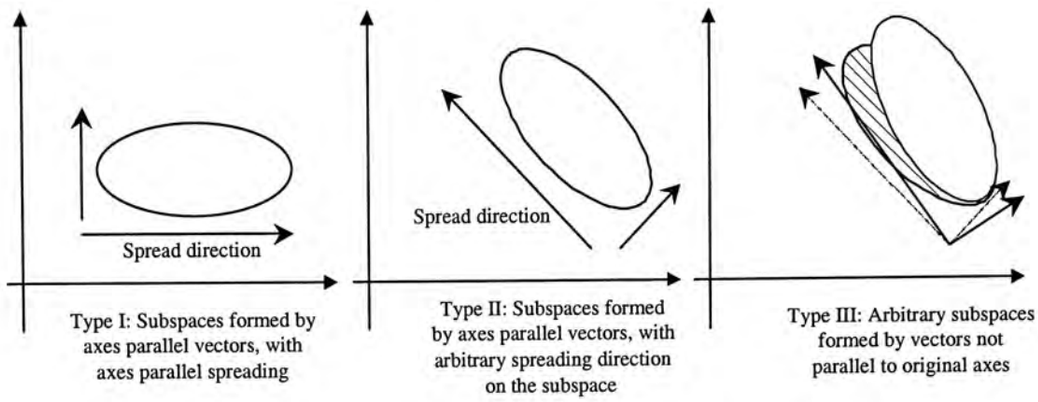


Figure 4.1: Subspaces detected by different algorithms

Both PROCLUS and EPC is good in uncovering the Type I subspaces, in which the data objects are located closely together in each of the bounded dimensions. PROCLUS uses *Manhattan segmental distance* as measurement of whether two data objects are close in the corresponding subspace, and EPC uses 1-d histogram to model data projections on any single dimension. In both cases, they favor in discovering clusters which contain data objects spreading along certain directions parallel to original axes.

ORCLUS solves the most general problem. It can detect arbitrarily oriented subspaces formed by any set of orthogonal vectors, which is the Type III subspace. In some problem domains, applications need to uncover arbitrarily oriented subspace. However, as pointed out in [35], in many real life applications, original coordinate axes do have special meaning. Patterns naturally exist in subspaces formed by axes parallel vectors. For example, a customer database contains three attributes: age, income and number of family members. We may discover one projected cluster in the subspace [age, income], and another projected cluster in the subspace [age, number of dependent], with the following interpretation: a typical group of customers with age in range A, income with range B and independent of number of dependent is interested in the new product; an interesting buying behavior and correlation exist between age in range C and number of dependent in range D. Furthermore, when the

dimensionality is growing to very high, pattern typically exists with a small subset of dimensions. i.e. subspaces where patterns exist are commonly formed by a small subset of axes parallel vectors, but not vectors contributed by all dimensions.

As a consequence, we target to discover patterns which can be observed after projecting onto Type II subspace. The vectors forming the subspace are only related to a small subset of bounded dimensions, but not all dimensions. Certain correlations exist among the bounded dimensions, such that the spreading direction may be arbitrary. EPC may miss some of these kinds of patterns, as the arbitrary spreading may not produce dense projection on any single dimension, as in Figure 3.7. Also, EPC would combine two projected clusters, if their projections on a single dimension are overlapping.

Furthermore, all PROCLUS, ORCLUS and EPC are *hard clustering techniques*, which assign a class label l_i to each data object x_i, identifying its class. For many applications in customer segmentation and trend analysis, a partition of data objects is required. Partition also provides clearer interpretability of the results. However, consider figure 4.2. Unlike traditional cluster, a data object can be fallen into more than one cluster under different projections. In such case, it would be difficult to assign a single class label to it. Even in the synthetic generated data set, a data object is targeted to be generated from projected cluster A can be actually nearer to projected cluster B. Therefore, in the context of projective clustering, *fuzzy clustering* is more prefer. Fuzzy clustering can assign to each data object x_i a fractional degree of membership f_{ij} in each output cluster j.

Therefore, we extend EPC to uncover Type II subspace, as well as to provide fuzzy clustering result. EPC is generalized to use multi-dimensional histograms to model the data distribution, with the trade-off between increasing clustering quality and increasing time and space complexity. In this chapter, we present the idea in the generalized framework, which we denote as EPCH

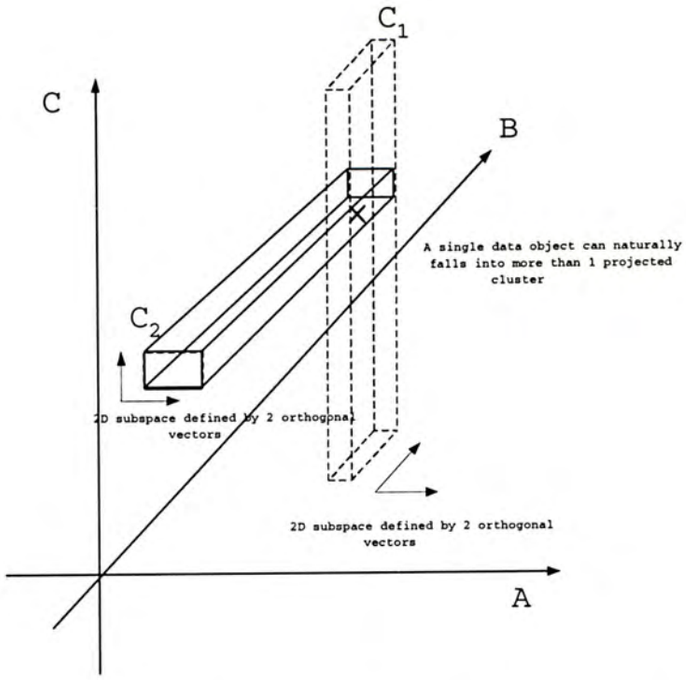


Figure 4.2: A data object falls into more than one cluster under different projections

(Efficient Projective Clustering using Histogram). We denotes d as the dimensionalities of the histograms that we construct. In particular, we describe an implementation of EPC2, which we set $d = 2$. In the following discussion, We follow the general notations used in Chapter 3.

4.2 Distinguish clusters by their projections in different subspaces

In the discussion of EPC in Chapter 3, we consider projections of data objects on each dimension, which we can consider as 1-d subspace. A dimension is bounded with respect to an associated subspace if it is included in the subspace. With histograms in higher dimensionality, we can examine densities of projections on subspaces with higher dimensionality. The subspace is bounded

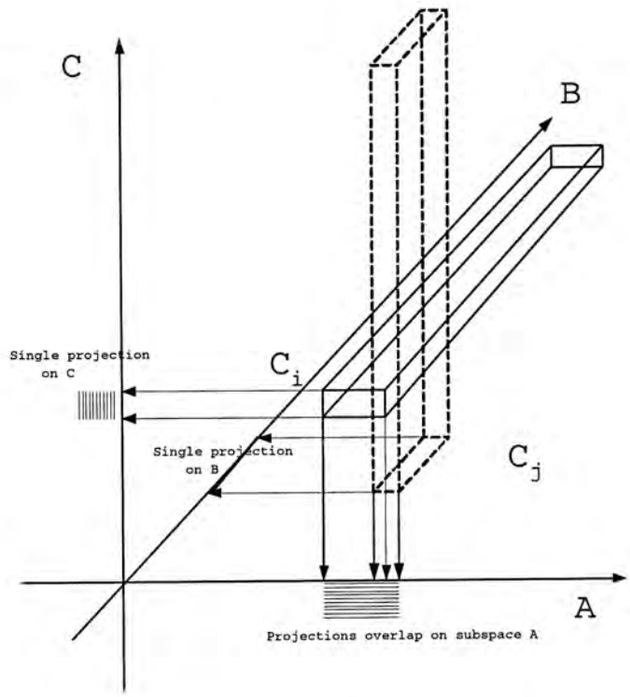


Figure 4.3: Projections overlap on subspaces with lower dimensionality.

with respect to an associated subspace if it is a subset of the associated subspace. Two projected clusters can be differentiated if 1)they produce non-overlapping projections on a bounded subspace, or 2)they produce projections on different bounded subspace. In real data set, projections of clusters often overlap on some subspaces with lower dimensionality, as in figure 4.3. C_i is associated with subspace **AC**, C_j is associated with subspace **AB**. They share a common subspace A. When we examine subspace **A**, because their projections are over-lapping, we cannot distinguish two clusters and will merge them together. In subspace **B** or **C**, we can distinguish these two clusters, because C_i would not produce dense projection on **B** as it spread along **B**, C_j would not produce dense projection on **C**.

Therefore, if there are some subspaces which contain only one projection, these subspaces can help us to identity the corresponding projected cluster. Consider example 5:

Example 5 There are four dimensions, **A, B, C, D**, five clusters with subspaces $S_1 = \mathbf{AC}, S_2 = \mathbf{AD}, S_3 = \mathbf{BC}, S_4 = \mathbf{BD}, S_5 = \mathbf{AD}$ respectively. If we

construct 5 1-d histogram to model the data density, histogram on **A** may not help to identify any cluster, since S_1, S_2, S_5 may produce overlapping projections. Similarly, histogram on **B, C, D** may not be able to identify any cluster, since subspace **B, C, D** may contain overlapping projection from more than one cluster.

However, if we construct 2-d histogram, histogram on **AC, BC, BD** can identify S_1, S_3, S_4 respectively, since these subspace would contain projection from only one cluster.

In general, a subspace can identify a cluster if it is the bounded subspace of only one cluster, and it is the unbounded subspace of other clusters. Assume the associated subspace of each cluster is independent to each other, the probability of a d-dimensional subspace to be the bounded subspace of exactly one cluster is :

$$K(\frac{{}_lC_d}{{}_DC_d})(1 - \frac{{}_lC_d}{{}_DC_d})^{K-1} \tag{4.1}$$

Let $N(D,d) =_D C_d$ be the number of d-dimensional subspaces, P be the probability obtained from Equation 4.1. The probability of at least K d-dimensional subspaces to be the bounded subspace of exactly one cluster is :

$$\sum_{i=K}^{N(D,d)} ({}_{N(D,d)}C_i(P)^i(P)^{N(D,d)-i}) \tag{4.2}$$

By Equation 4.2, building histograms with higher dimensionality gives us a higher probability that more subspaces are the bounded subspace of exactly one cluster. In general, histogram with higher dimensionality helps us to uncover cluster more easily, with the expense of increasing complexity.

4.3 EPCH: a generalization of EPC by building histogram with higher dimensionality

4.3.1 Multidimensional histograms construction and dense regions detection

Given a set of n observed data points $X_1, X_2, ..., X_n$ and each data points is a d-dimensional vectors $X_i = (X_{i_1}, ..., X_{i_d})^T$, we can model the data density in the multivariate case by extending the univariate density estimation function and kernel function as stated in Equation 3.1 to d-dimensional space as the follows:

$$\hat{f}(x) = \frac{1}{nh^d} \sum_{i=1}^{n} K(\frac{X_i - x}{h}, ..., \frac{X_{i_d} - x_d}{h_d}) \tag{4.3}$$

where $K(x)$ satisfies $\int_{R^d} K(x)dx = 1$.

To approximate $\hat{f}^d(x)$ efficiently, we first divide each of the d dimensions into a number of equi-width intervals with length h. A histogram cell is a single unit formed by an intersection of the intervals. Then, we construct multidimensional histogram in d-dimensional space as the follows:

$$\hat{f}^d(x) = \frac{1}{nh^d}(\text{no. of } X_i \text{ in same histogram bin as x}) \tag{4.4}$$

The number of d-dimensional histograms we have to construct is ${}_DC_d$, instead of D comparing to 1-d histogram in Chapter 3. Now, each d-dimensional histogram corresponds to the estimation of data distribution in the d-dimensional subspace.

Density of the projection on the d-dimensional subspace would be approximated by the number of data objects located inside the projection estimated by the histogram, over the number of cells spanned by the projection. Dense Region is the group of connected cells with densities above a certain threshold.

As similar to Chapter 3, we use $DR_m^{S_i}$ to denote the dense region on subspace S_i with *dense id* equals to m. We apply similar adaptive dense region discovery as described in Section 3.4.2, by iteratively recording and removing cells that have been detected as dense, and calculate the new μ, σ and threshold as $\mu + c\sigma$ for the next iteration.

4.3.2 Compressing data objects to signatures

Each D-dimensional data objects would be compressed to signature Q with ${}_DCd$ entries. The j^{th} entry represents the dense region, if any, where the data object is located in subspace S_j. Specifically,

$$Q = [Q_1, Q_2, \ldots, Q_L]$$

$$\text{where } Q_j = \begin{cases} 0 & \text{if the object does not fall into} \\ & \text{any region in subspace } S_j, \\ m & \text{if the object is located in } DR_m^{S_j} \\ & \text{in subspace } S_j. \end{cases}$$

Each non-zero entry corresponds to a **bounded subspace**, and each zero entry corresponds to an **unbounded subspace**.

From the signature, we can estimate the associated subspace of the projected cluster, in which the corresponding data object is likely to belong to. Here we call the estimated associated subspace as **derived subspace**, and the actual subspace of cluster the data object should belong to as **associated subspace**. Consider example 6.

Example 6 Suppose there are four dimensions **A, B, C, D**, and we have constructed ${}_4C_2 = 6$ 2-d histograms. The signatures should contain 6 entries, where the first, second ... entries corresponds to subspace **AB, AC, AD, BC, BD, CD** respectively. Consider O_1, its signature is [2 0 0 1 0 0]. Its bounded subspace is **AB** and **BC**. We estimate the associated subspace of cluster the

data object belongs to (the derived subspace) as **ABC**. However, if the associated subspace is **ABC**, all **AB, BC, AC** should be the bounded subspace. There are two possible reasons. 1. **BC** is not the bounded subspace, O_1 just occasionally fall into some dense regions in **BC**. In such case, the associated subspace should be **AB**. 2. **AC** should also be the bounded subspace, but O_1 is located in the boundary of the projection on **AC**, in which the location is not considered as dense.

Consider O_2, its signature is [2 3 0 2 0 0]. Its bounded subspace is **AB, AC** and **BC**. Its derived subspace is **ABC**. This time, since all 2-d subsets of **ABC** are bounded subspace, we are quite sure that the derived subspace should match the associated subspace. Comparing these 2 cases, we call the derived subspace **ABC** is of *high confidence level* for O_2, and it is of *lower confidence level* for O_1.

We estimate the derived subspace of a signature, by union all corresponding bounded subspace. Here, we define the confidence level of a derived subspace with l_j dimensionality, estimated by d-dimensional histograms for data object O_i as:

$$\frac{\text{number of bounded subspaces in signature of } O_i}{{}_{l_j}C_d} \tag{4.5}$$

This corresponds to the ratio between the number of bounded subspace in the signature of the data object, and the number of all possible d-dimensional subsets of the derived subspace. When this ratio approaches 1, the derived subspace should be an accurate estimation of the associated subspace.

Two signatures having the same derived subspace can be in two different projected clusters, if they are in different projection in some bounded subspace. Consider example 6, dense id of bounded subspace **AB, BC** of O_1 is 2, 1 respectively; dense id of bounded subspace **AB, AC, BC** of O_2 is 2, 3, 2 respectively. Both two data objects O_1, O_2 have the same derived subspace

ABC. However they should be in two different projected cluster, since they share a common bounded subspace **BC** but with different dense id – O_1 is in DR_1^{BC} while O_2 is in DR_2^{BC}. We call the common bounded subspace of two data objects *conflicting* if they have different dense id in this bounded subspace.

To compress the original data set, we derive signatures of every data object. Their derived subspace, dense id of bounded subspaces and the corresponding confidence level are then inserted into a signature list. Every entry in the signature list records the derived subspace, dense id of bounded subspaces from data objects contributing to this entry, and a weighting. Signatures with the same derived subspace and no conflicting bounded subspaces would be combined and inserted into the same entry of the list, where the weighting of the entry would be calculated as the summation of the confidence level of the data objects contributed to this entry.

4.3.3 Merging Similar Signature Entries

Each entry in the signature list corresponds to a group of data objects with the same derived subspace and sharing no conflicting bounded subspaces. Most likely, this group of data objects are from the same projected cluster. However, there are often cases that data objects from the same projected cluster would have different derived subspace, and thus contribute to different entry in the signature list. Consider Example 7.

Example 7 Suppose there are four dimensions **A, B, C, D**. The signatures contain 6 entries, where the first, second ... entries corresponds to subspace **AB, AC, AD, BC, BD, CD** respectively. Consider O_1, O_2, O_3, O_4, their signature are [2 0 0 2 0 0], [2 0 0 1 0 0], [2 0 1 0 3 0] and [2 0 1 0 3 2] respectively, and their derived subspace are **ABC, ABC, ABD, ABCD**. Although O_1 and O_2 have the same derived subspace, they are inserted into

different entry in the signature list because they have the conflicting subspace BC. O_1 and O_2 are totally dissimilar.

O_3 and O_4 have different derived subspaces and are in different entries, but they share many common bounded subspaces. They can be in the same projected cluster if O_3 fall in the boundary of the projection on **CD**, or O_4 occasionally fall in the projection on **CD**. We consider O_3 and O_4 are similar.

O_2 and O_4 have different derived subspaces, and they share only one common bounded subspace in the total of 5 bounded subspaces. O_2 and O_4 are not similar.

We want to merge "similar" signature entries, as they may correspond to group of data objects locating in the same projected cluster. We define the *similarity* between two signature entries as

$$S(O_i, O_j) = \frac{\text{number of common bounded subspaces}}{\text{total number of different bounded subspaces}} \tag{4.6}$$

Signature entries sharing large proportion of common bounded subspaces would have a high similarity value and can be considered as similar. We keep only $k \times m$ top ranked signature entries to speed up the merging process, where signatures with low weighting would be pruned away (in current implementation, we set k to be 50). Similarity of every pair of the remaining signature entries would be computed according to Equation 4.6. We continually combine the pair of signature entries with the highest similarity until reaching a termination criteria. One termination criterion is when the number of signatures reaches m. However, if the signatures are quite different before we reach m, then the merging is stopped when we see a sudden drop of the highest similarity among the signature, or the highest similarity is already less than a lower bound threshold (in current implementation, we use a threshold of 0.3). We do a final sorting in descending order of weighting. The remaining signature entries correspond to the clusters and subspaces detected by the algorithm.

At most *max_no_cluster* signature entries with the highest weighting would be kept after merging: if there are *max_no_cluster* or more signature entires then the top *max_no_cluster* signature entries are kept, otherwise all the signature entries are kept. Each signature entry corresponds to a projected cluster, where its derived subspace represents the associated subspace, and the recorded dense_region id locates the cluster on the subspace.

4.3.4 Associating membership degree

As illustrated in figure 4.2, because the nature of projective clustering does not produce clear partitioning, in which a data object can be contained in two or more different projected clusters, we apply the concept of fuzzy clustering. Each data object is associated a membership degree for each discovered projected clusters. When comparing the clustering quality with other projected clustering algorithms in chapter 5, an data object would be assigned to output cluster with the largest degree of membership, or as outlier if no membership degrees exceeds a certain threshold.

We define a **membership function** for data object O_m with cluster C_n, using the signature of O_m and the dense id in the bounded subspaces recorded in the signature entry corresponds to C_n:

$$member(O_m, C_n) = \frac{\text{no. of matched bounded subspaces}}{\text{no. of bounded subspaces in the signature list of } C_n} \tag{4.7}$$

Equation 4.7 measures the degree of membership in terms of similarity between the signature of the data object and the cluster. If this measurement approaches 1, that means the data object is located in all dense regions projected by the cluster. Thus, it has the strongest membership degree. On the other hand, if this approaches 0, that means the data object is not located in any dense regions projected by the cluster.

For hard clustering, we first compute the membership functions, then an data object would be assigned to output cluster with the largest degree of membership, or as outlier if no membership degrees exceeds a certain threshold.

We can also identify outliers if the membership degrees of the data object to all discovered cluster are low, (e.g. in current implementation we use 0.1), or we can specify an expected percentage of outliers and such an amount of data with the lowest degrees of membership will be considered outliers, or criterion formed by a combination of outlier percentage and the membership values.

4.3.5 The choice of Dimensionality d of the Histogram

As shown in Section 4.2, when increasing the dimensionality of the histogram we build, chances for successfully detecting the projected clusters would be increased, with the trade-off of higher complexity. For different data set with different subspace property and distribution, appropriate value of d varies. User may not be able to choose the value which give satisfactory cluster result and acceptable computational time. Here we propose a framework in which user do not need to select the d value.

First we perform a very fast clustering using $1-d$ histograms. If the $1-d$ histograms cannot detect and distinguish clusters well, data objects would be associated with low degree of membership, in which many of them would be outputted as outliers. We record the detected clusters and partitioned data objects. The unpartitioned data objects (which are outputted as outliers in this run) would be input to the next run, in which we do clustering using $2-d$ histograms. We continue this approach with higher dimensionality histograms, until the number of outliers remained is less than a certain fraction of data objects (e.g. 5% of the whole data set), which are actually random noise.

4.4 Implementation of EPC2

This section describes the implementation of EPC2, which chooses 2 to be the dimensionality of the histograms d to be constructed. The same approach can be applied for other values of d.

The overall algorithm consists of five phases. **Histogram Building phase** builds ${}_DCd$ histograms, each corresponds to one d-dimensional subspace. First, we divide each of the d dimensions in the subspace into 400 equi-width intervals. For each data objects, we identity the cell where the object locates in each subspace, and update the corresponding histogram.

Dense Region Detection phase works iteratively to identify all dense regions. In each iteration, for each histogram, we calculate the mean μ and standard deviation σ of the distribution, and set the threshold γ to be $\mu = \sigma + c\gamma$. (In EPC2, we set $c = 5$.) Cells with density above γ are recorded and removed. All the adjacent recorded cells are connected by a recursive algorithm, the connected cells are labelled as a dense region. The new μ, σ and γ are updated, and start the next iteration. Iteration would be terminated when γ does not change after one iteration, or at most *max_no_cluster* dense regions have been detected.

Signature List Construction phase examines each data object, derive its signature according to the *id* of the dense regions the object locates in each subspace. If the object does not fall into any detected dense region in a subspace, its corresponding entry in its signature would be zero. From the signature, we find the derived subspace by union all bounded subspaces, and its confidence level according to Equation 4.5. These information would be inserted into the signature list, where signatures with the same derived subspace would be combined into a single entry. The weighting of a single entry would equal to the summation of confidence levels of signatures composing it.

Merging similar Signatures phase sorts the signature entries in the list

according to descending order of their weighting, where signature entry with higher weighting corresponds to cluster with more number of data objects and clearer associated subspace. We keep only $50 \times max\_no\_cluster$ top ranked signature entries, where signatures with low weighting would be pruned away. (From experiments, $50 \times max\_no\_cluster$ should be a number large enough to hold relevant signature entries. We also try using $40 \times max\_no\_cluster$ or $60 \times max\_no\_cluster$, but the results does not vary too much.) Similarity of each two remaining signature entries would be computed according to Equation 4.6. We continuously merge two signature entries with the greatest similarity until the greatest similarity is less than 0.3. We do a final sorting in descending order of weighting. The remaining signature entries correspond to the clusters and subspaces detected by the algorithm. In case if the number exceeds *max_no_cluster*, only top *max_no_cluster* would be kept and reported to the user.

Membership Degree Assigning phase associates each data object and discovered cluster a degree of membership according to Equation 4.7. Hard clustering result can be produced by assigning data object to cluster with largest degree of membership. Outliers can be detected if the membership degree of all clusters are lower then a 0.1.

4.5 Time and Space Complexity of EPCH

In this subsection, we let $m = max\_no\_cluster$, and h be the number of equi-width intervals. In the **Histogram Building phase**, we need to scan the database once, and build ${}_DC_d$ histograms. The time complexity is $O(ND^d)$. The space needed to store the histograms would also depend on the number of equi-width intervals h, which is $O(hD^d)$.

In each iteration in **Dense Region Detection phase**, we have to scan each histogram twice to calculate the new threshold and detect dense regions,

which takes $O(hD^d)$ time. Since the number of iteration is upper bounded by m, the total time complexity in this phase is thus $O(mhD^d)$. We do not need additional space complexity in this phase.

In **Signature List Construction phase**, we scan the database once and derive signatures for each data object based on the ${}_DC_d$ histograms. The time complexity is $O(ND^d)$. For each signature, on average it corresponds to ${}_lC_d$ bounded subspace. The time for finding the derived subspace and confidence level is l^d. If the signature list is kept in a hash table, the expected time for insertion into the signature list is $O(Nl^d)$. Thus, the total time complexity is $O(N(l^d+D^d))$. The space complexity for storing the signature list is $O(NDl^d)$.

Sorting in **Merging Similar Signatures phase** while keeping only the top km signature entries takes $O(Nlog(km))$ time. Since we keep only top km signature entries, computation of similarities between each pair takes totally $O((km)^2l^d)$ time. The merging takes at most km iterations. The total time complexity is $O(Nlog(km)+(km)^3l^d)$. We can also record the similarities between each pair of signature entries, and update only the values of the newly merged entry in each iteration. This can reduce the complexity to be $O(Nlog(km)+(km)^2l^d)$, with the additional space complexity of $O((km)^2)$ for storing the similarity values.

In **Membership Degree Assigning phase**, it takes $O(Nml^d)$ to assign degree of membership between each data object and cluster pair. The space complexity for this is $O(Nm)$.

Thus, the total time complexity is $O(ND^d+mhD^d+N(l^d+D^d)+Nlog(km)+(km)^2l^d+Nml^d)$, if we consider $log(km)$ small, this can be simplified to $O(D^d(N+mh^d)+l^d((km)^2+Nm)$. The space complexity is $O(hD^d+NDl^d+Nm)$. In particular, in current implementation of EPC2, we set a fixed value of h, and d is fixed to be 2, the time complexity is thus $O(N(D^2+l^2m)+(km)^2))$. For 1-d histograms, $d=1$, the time complexity is $O(N(D+lm)+l(km)^2)$.

Chapter 5

Experimental Results

We present several experimental results and their analysis in this section. We have implemented PROCLUS, ORCLUS, EPC [36] , and EPC2 (EPCH with 2-d histograms), and would like to compare their performance differences in terms of clustering quality and running time.

5.1 Clustering Quality Measurement

There are three different quality measurements that we shall use:

- **Confusion Matrix** is often used to evaluate the clustering result of synthetically generated data. Specifically, it is a $p \times p$ matrix, where p is the number of natural clusters. Entry (i, j) records the number of data objects belonging to the natural cluster i, that is assigned to the output cluster j. Obviously, if we can observe a clear one–to–one mapping between each output cluster to a natural cluster, the clustering quality is good. The left hand side of Table 5.1 shows an example of good clustering result, in which we can find clean one–to–one mappings between each pair of output and natural clusters. The right hand side represents a bad clustering result, e.g. output cluster 1 and 2 split natural cluster 1, and output cluster 3 merges natural cluster 2 and 3.

output \ natural	Outlier	0	1	2	3
Outlier	154	46	47	2	21
0	0	0	0	465	0
1	0	900	0	0	0
2	1	0	0	0	483
3	0	0	567	0	0

output \ natural	Outlier	0	1	2	3
Outlier	198	101	54	12	3
0	45	29	77	49	25
1	32	31	123	0	0
2	21	6	254	0	0
3	12	8	12	527	497

Table 5.1: Confusion matrix of good and bad clustering result

- **Dominant Ratio**: [4] suggested a measurement *dominant ratio* to evaluate the quality of the clustering. This is the average fraction of data objects in each output cluster which are populated by the most dominant natural cluster. For each output cluster, we identify the natural cluster which contains the largest number of data objects. Then, we calculate the percentage of data objects in the output cluster which was populated by this natural cluster. Averaging this value over all output clusters yield the dominant ratio. A good clustering should have the dominant ratio close to one.

  Here, we argue that this single measurement may not be able to justify a good clustering when the number of natural clusters may not match the output clusters (either by inappropriate estimation by users, or by wrong estimation by the clustering algorithm) . Consider an extreme case where there are many output clusters so that each natural cluster becomes a number of output clusters. The inappropriate splitting of natural clusters could still produce a dominant ratio of 1.

- **Coverage ratio**: Therefore, in addition to dominant ratio, we propose another measurement called the *coverage ratio*, which in a way complements the dominant ratio. This is the average fraction of data objects in each input cluster which are covered by the most dominant output cluster. For each natural cluster, we identity the output cluster which

contains the largest number of data objects. The percentage of data objects in the natural cluster which are populated by their dominant output cluster is calculated, which would be averaged over all natural clusters to yield the coverage ratio. Similarly, there would be cases where an output cluster combines several natural clusters and still give rise to a coverage ratio close to 1.

We can evaluate the effectiveness of a clustering algorithm by the resulting dominant and coverage ratio. A good clustering result should yield both dominant ratio and coverage ratio close to 1. In this case, all output clusters can cleanly map to natural clusters, no output clusters combine several natural cluster; at the same time, no natural clusters are split by several output clusters. For clustering algorithm which often splits natural clusters, the dominant ratio would be much higher than the coverage ratio. On the other hand, if the coverage ratio is significantly larger than dominant ratio, the clustering algorithm often combines natural clusters.

5.2 Synthetic Data Generation

We generate data sets with different properties.

- **PR-Set**: follows the data generation in [2] (PROCLUS), where both the vectors forming the subspace and the spreading directions of the clusters are parallel to the original axes. (as shown in Figure 4.1a.) Associated subspaces of clusters are with varying dimensionality. Data objects follow normal distribution in bounded dimensions with small variance. Clusters in this data set should be easier to be detected, as its projections to any single bounded dimension is very dense.

- **AP-Set**: models clusters associated with arbitrary subspaces (as shown in Figure 4.1b). It follows the data generation described in [3](ORCLUS)

with some modification so that the bounded subspaces are not determined by all of the original dimensions. Specifically, dimensionalities of all associated subspaces are set to a fixed value l. For each cluster, we choose randomly l dimensions as its bounded dimension, and then generate l orthogonal vectors randomly in the subspace formed by the bounded dimensions (by finding eigenvectors in a randomly generated symmetric matrix). We select e vectors randomly from the l eigenvectors to form the orientation of the cluster. Anchor point of each cluster would be randomly chosen. Data objects distribute along e vectors defining the orientation in its associated subspace, following normal distribution with their mean at the anchor point. In other unbounded dimensions, they distribute randomly. We set l and e to be 6 in the base case.

In real life application, data often contains outliers (data object do not belong to any cluster, or random noise), and dimensionalities of different associated subspaces need not be the same. Therefore, we generate two variations on the **AP-Set**.

1. **APN-Set** is the same as **AP-Set** which includes a certain percentage of outliers. (we use 5 % of noise in the base case.)

2. **APD-Set** contains clusters with associated subspaces with varying dimensionality. In particular, for data set with 5 clusters, we set the dimensionalities of associated subspaces of clusters to be 4, 5, 6, 7, 8.

We use these two data sets to evaluate performances of different algorithms on data with these two properties.

5.3 Experimental setup

All the experiments have been performed on a 12 UltraSPARC-II 400 MHz machine with 8GB RAM, running Solaris 7. The algorithms are implemented

using C language and gcc compiler v2.7 without code optimization. For PROCLUS implementation, we set the number of seed points to be 2% of data objects, and number of iterations allowed for no quality improvement to be 20. For ORCLUS implementation, we use the parameter values suggested in [3], except that we set the reduction factor α of the number of clusters in each iteration be 0.8 instead of 0.5 to obtain a more accurate clustering result. For data set with outliers, we follow suggestions in [3] to add outlier handling implementation, and name it as ORCLUS_outlier. For implementation of EPC2, we set the threshold to detect the dense regions to be $\mu + 5\sigma$, and the number of top ranked signatures we would like to keep be $50 \times m$, where m represents the number of clusters users are interested to find. We report here some special properties of each clustering algorithm and general trends.

5.4 Comparison between EPC and PROCULS

Since both PROCLUS and EPC are targeted to discover clusters in **PR-Set**, and both can report the bounded dimensions of the associated subspace, we evaluate their clustering qualities, accuracy of identification of subspaces, and running time in this data set. We vary N from 3000 to 200000, and set $D = 20$. In EPC we set the number of bins in each histogram to 200, which is much greater than $1 + log_1 N$. In order to evaluate how many clusters are correctly discovered, and the number of correctly uncovered bounded dimensions in the corresponding associated subspaces, we set the criteria of clear "one-to-one" mapping in the resulted confusion matrix as follows:

Output cluster X has a clear "one-to-one" mapping to natural cluster Y, if more than 60% of data objects from Y are reported to belong to X, and X contains no more than 20% of data objects from other natural clusters. Based on this criteria, a natural cluster is said to be correctly discovered if we can find a clear "one-to-one" mapping between this cluster and an output cluster

output \ natural	Outlier	0	1	2	3	4
Outlier	302	120	156	83	122	35
0	0	0	0	0	3168	0
1	0	0	0	6	0	819
2	0	0	0	721	0	0
3	0	0	468	0	0	0
4	0	0	0	0	0	0

(a)

output \ natural	Outlier	0	1	2	3	4
Outlier	61	4	0	0	69	12
0	25	0	0	810	0	207
1	55	34	624	0	0	102
2	33	3	0	0	0	267
3	103	67	0	0	122	215
4	25	12	0	0	3099	51

(b)

Table 5.2: Confusion matrix obtained for **PR-Set** with 6000 data objects from (a)EPC, (b)PROCLUS.

	EPC			PROCLUS		
Data Set	Correctly discovered clusters	Percentage of correctly partitioned data objects	Correct dimension/ Total dimension in the correct clusters	Correctly discovered clusters	Percentage of correctly partitioned data objects	Correct dimension/ Total dimension in the correct clusters
3000	4	86	19/19	2	72	6/10
6000	4	94	18/18	2	65	7/10
10000	5	84	25/25	3	88	7/14
20000	4	92	20/20	0	0	0/0
30000	5	93	25/25	2	39	10/12
50000	3	94	15/15	2	81	6/10
65000	4	94	19/19	3	91	11/15
80000	4	94	17/19	4	88	16/20
100000	3	78	13/13	0	0	0/0
200000	5	90	25/25	2	75	8/12

Table 5.3: Summary of the comparison of results obtained from EPC and PROCLUS for **PR-Set**

from the confusion matrix. Due to the limited space, we show in detail only one of the results obtained from a data set with $N = 6000$ in Table 5.2. Other data sets show similar trends. The shaded field corresponds to the correctly discovered clusters. In this data set, EPC can discover 4 clusters. One of the good property of EPC is that it always returns clusters containing the largest number of data objects first. It is not surprising that EPC cannot discover the smallest cluster, as it contains less than 5% of data objects, which may be considered as random noise. For PROCLUS, it can discover only 2 clusters.

We calculate the percentage of correctly partitioned data objects, and also compare the reported dimensions in the associated subspaces of the output clusters, with the associated subspace of the corresponding matching natural clusters. Table 5.3 compare the results obtained from EPC and PROCLUS in **PR-Set**

Table 5.4 shows the dominant and coverage ratios of these two algorithms.

	EPC		PROCLUS	
Data Set	dominant ratio	coverage ratio	dominant ratio	coverage ratio
3000	1.000	0.908	0.812	0.717
6000	0.998	0.864	0.764	0.763
10000	1.000	0.505	0.755	0.668
20000	1.000	0.912	0.732	0.316
30000	1.000	0.902	0.838	0.469
40000	1.000	0.745	0.764	0.715
65000	1.000	0.906	0.629	0.739
80000	1.000	0.783	0.863	0.838
100000	0.993	0.736	0.930	0.163
200000	1.000	0.912	0.732	0.316

Table 5.4: Dominant and Coverage ratios of results obtained from EPC and PROCLUS with data sets varying N

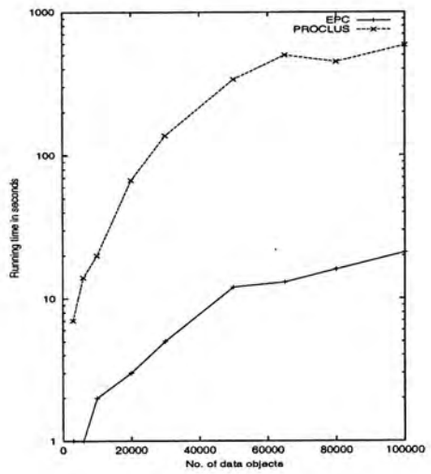


Figure 5.1: Running time against N for EPC and PROCLUS

We can see that EPC produces more accurate result than PROCLUS in nearly all cases. Figure 5.1 compares their running time with varying N. Both methods scale about linearly with N, EPC is much faster than PROCLUS. For example, for the data set with $N = 10000$, EPC takes 20 seconds and PROCLUS takes about 10 minutes to compute. Since EPC is more accurate and efficient than PROCLUS, in the following discussions, we mainly compare EPC2, ORCLUS, and EPC.

5.5 Comparison between EPCH and ORCLUS

We compare the differences between clustering qualities of the **AP-Set**, **APD-Set**, **APN-Set** and **PR-Set**. We observe that varying the number of data

	EPC2		EPC		ORCLUS/ORCLUS_outlier	
	Dominant ratio	Coverage ratio	Dominant ratio	Coverage ratio	Dominant ratio	Coverage ratio
Average over AP-Set	0.829	0.736	0.321	0.776	0.836	0.855
Average over APD-Set	0.841	0.722	0.32	0.945	0.766	0.652
Average over APN-Set	0.806	0.684	0.368	0.891	0.795	0.832
Average over PR-Set	0.914	0.936	1	0.828	0.800	0.585

Table 5.5: Average Dominant and Coverage ratios for AP-Set, APD-Set, APN-Set and PR-Set

objects would not have significance change on the clustering quality. We ran experiments for different datasets with N ranged from 3000 to 60000, and obtain similar general trend for dominant and coverage ratios. Table 5.5 shows the average of the dominant and coverage ratios over data sets with varying N.

Histograms with different dimensionalities: Both EPC and EPC2 perform very well in **PR-Set**. For other data sets where the spreading directions of clusters may not parallel to original axes, EPC2 clearly outperforms EPC. 1-d histogram can help to detect and distinguish projections from clusters with axes parallel spreading. For clusters with arbitrary spreading direction, using histograms with higher dimensionality can significantly improve the clustering quality, in which overlapping of projections to lower dimensionality often occurs.

Associated subspaces with varying dimensionality: Comparing **AP-Set** and **APD-Set**, ORCLUS performs better in the **AP-Set**, because it assumes the associated subspaces are with the same dimensionality, where **APD-Set** contains subspaces with varying dimensionality. In such case, for associated subspaces with lower dimensionality, ORCLUS would include additional unbounded subspaces and retain some of the noise from the data. For the same reason, ORCLUS does not perform well in **PR-Set**, which contains

	EPC2		ORCLUS_outlier	
% of outliers	Dominant ratio	Coverage ratio	Dominant ratio	Coverage ratio
10	0.915	0.838	0.904	0.902
15	0.913	0.855	0.806	0.855
20	0.611	0.511	0.806	0.844
25	0.774	0.855	0.694	0.766
30	0.880	0.952	0.575	0.806

Table 5.6: Dominant and Coverage ratios for data sets with 50000 data objects, 20 dimensions and increasing percentage of outliers

associated subspaces with varying dimensionalities. In contrast, EPC2 does not make this assumption. It gives satisfactory result in the **APD-Set**. The clustering quality of **APD-Set** is even slightly better than **AP-Set**, because the more variant of the dimensionalities of the associated subspaces, the easier for the merging "similar" signatures phase to distinguish different signatures corresponding to different associated subspaces.

Presence of outliers: Presence of outliers would affect accuracy of the clustering results. ORCLUS presented in [3] does not include the clustering results when the data contains outliers. From Table 5.5, qualities of clustering results from EPC2 and ORCLUS_outlier does not vary too much between **AP-Set** and **APN-Set**. We increase the percentage of outliers in the dataset, and compare its effect on EPC2 and ORCLUS_outlier, as shown in Table 5.6

When the percentage of outliers is larger than 10%, the accuracy obtained from clustering result of ORCLUS_outlier significantly decreases. This is because even the outlier handling can remove most of the outliers, remaining few of them can change the positions of intermediate centroids of the clusters and the estimated eigen vectors forming the associated subspace. Comparing to ORCLUS, the decrease in accuracy of EPC2 is not so significant. EPC2 is more outliers-resistant because outliers are often located in non-dense regions,

	EPC2		ORCLUS_outlier	
D	Dominant ratio	Coverage ratio	Dominant ratio	Coverage ratio
10	0.882	0.819	0.750	0.768
15	0.688	0.323	0.768	0.785
20	0.888	0.692	0.817	0.843
25	0.873	0.575	0.655	0.743
30	0.843	0.631	0.662	0.647
35	0.758	0.748	0.564	0.588
40	0.941	0.920	0.537	0.593
45	0.932	0.889	0.560	0.595
50	0.724	0.551	0.475	0.496

Table 5.7: Dominant and Coverage ratios for APN-Set with $N = 20000$ and $l = 6$

in which most of them have been pruned already in the dense region detection phase. Signature entries corresponding to outliers are with low weighting, because there should be few other data objects falling into the same set of dense regions on different subspaces as the outlier. Signature entries with low weighting would also be pruned away.

5.5.1 Dimensionality of the original space and the associated subspace

We observe an interesting phenomenon that ORCLUS has advantages when the difference between the dimensionality of the original space and the dimensionality of the associated subspace is small, i.e. the dimensionality of associated subspaces approach dimensionality of the original space. In contrast, performance of ORCLUS degrades when the difference is large. On the other hand, accuracy of EPC2 does not vary much with different dimensionality of the original space, and different dimensionality of the associated subspaces. This happens in both **AP-Set** and **APN-Set**. Table 5.7 shows the dominant and coverage ratio with different values of D for a **APN-Set** with $N = 20000$ and $l = 6$. Table 5.8 shows the dominant and coverage ratio with different values of l for a **APN-Set** with $N = 20000$ and $D = 20$.

From Table 5.7, we observe that ORCLUS produces good clustering quality

	EPC2		ORCLUS_outlier	
l	Dominant ratio	Coverage ratio	Dominant ratio	Coverage ratio
2	0.821	0.763	0.525	0.468
4	0.842	0.63	0.627	0.622
6	0.846	0.811	0.830	0.739
8	0.878	0.817	0.942	0.985
10	0.959	0.967	0.692	0.855

Table 5.8: Dominant and Coverage ratios for APN-Set with $N = 20000$ and $D = 20$

when $D <= 20$. However, as D increases, the accuracy of ORCLUS decreases. Also, as shown in Table 5.8, ORCLUS can produce good clustering quality only when $l >= 6$. This is because the number of iterations in ORCLUS is determined by the ratio between number of seeds and number of clusters, and a reducing factor. When these two values are fixed it takes too few steps to reduce the dimensionality of the associated subspaces if the difference between dimensionality of the associated subspace and the original space is large. This decrease the accuracy in uncovering the subspaces. EPC2 is not affected by this factor. Therefore, EPC2 shows more advantages when the dimensionality of the original space is very high, and at the same time the dimensionality of the associated subspaces is much lower. Many real data sets have such property, as the database can contain up to hundreds of attributes, but patterns typically exist within a different small subset of attributes.

5.5.2 Projection not parallel to original axes

In the **AP-Set**, we can vary the parameter e and l. If $e < l$, the projection would not be perpendicular to subspaces formed by subset of bounded dimensions. In such case, the projection would span larger area, where the density inside the spanning area would be lower. In setting the threshold $\mu + c\sigma$ for detecting dense region, we have to use a smaller value of c, which reflects the expectation that projections would span larger area. We compare clustering result for EPC2 and ORCLUS in some of this scenario. We test on **AP-Set**

		EPC2 (c=4)		ORCLUS	
e	l	dominant ratio	coverage ratio	dominant ratio	coverage ratio
2	3	0.69	0.547	0.266	0.252
3	3	0.842	0.593	0.447	0.42
3	4	0.634	0.383	0.329	0.291
4	4	0.783	0.566	0.434	0.428
4	5	0.669	0.686	0.537	0.524
5	5	0.603	0.613	0.707	0.717
5	6	0.518	0.643	0.706	0.705
6	6	0.669	0.521	0.993	0.994

Table 5.9: Dominant and Coverage ratios for AP-Set with varying e and l

with $D = 20$ and varying e and l, and set $c = 4$ for EPC2. Table 5.9 shows the result.

As explained in Section 5.5.1, ORCLUS does not perform well when e is small (e.g. when $e < 5$). In these cases, EPC2 perform better. We observe that when $e \geq 6$, ORCLUS produces better clustering result when $e < l$.

5.5.3 Data objects belong to more than one cluster under fuzzy clustering

From the fuzzy clustering result, we investigate the percentages of data objects that can belong to the projections of more than one cluster. We consider a data object belongs to more than one projected clusters if its membership degree is larger than 0.6 for these clusters. We examine the **APN-Set** with $D = 20$, $l = 6$ for varying N, the results show that there can be 0 up to 16% of data that belong to 2 clusters, and 0 up to 13% of data that belong to 3 clusters. In some synthetic data sets, even though the data object from each cluster is generated independently, there is more than 10% of data objects are reported as belong to more than 1 projected cluster. We can expect this ratio would be higher for some real data sets, in which data object can exhibit multiple properties of different patterns, in different subsets of attributes.

5.6 Scalability of EPC

The scalability of EPC is tested by varying 3 factors, the number of data objects(N), the number of dimensions(D), and the number of natural clusters. Each data point in figure 5.2 represents the result obtained from a single experimental trial with the corresponding data configuration. Figure 5.2(a) shows the user CPU time against the number of data objects (N) for different data configuration. We can see that the computation time grows approximately linearly with increasing number of data objects for data set with different dimensionality. We can obtain a set of similar linear curves for different data configuration, when we vary the number of dimensions (D) or the number of natural clusters. (see Figure 5.2(b) and Figure 5.2(c))

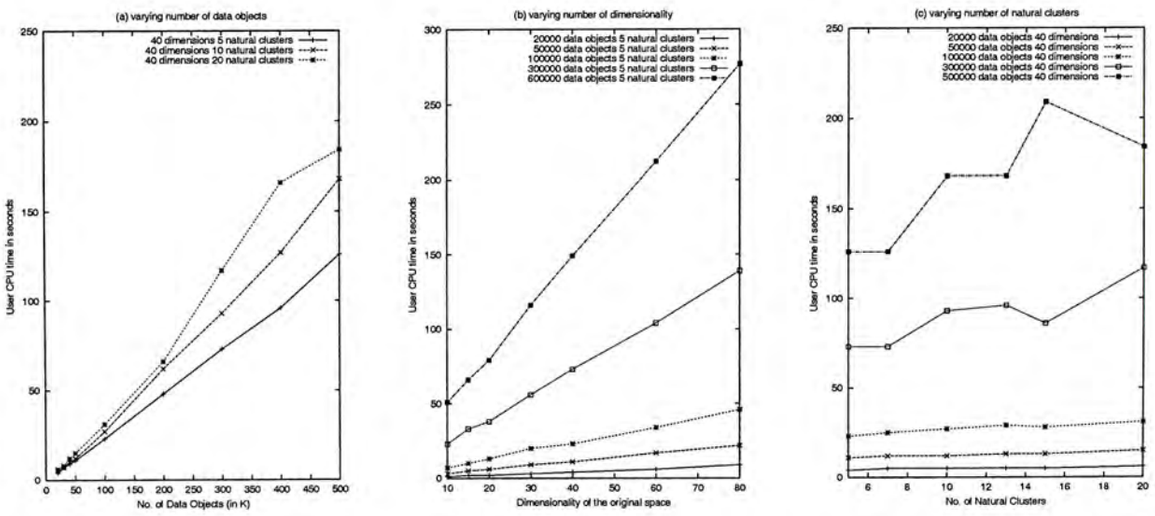


Figure 5.2: User CPU time of EPC against a)varying number of data objects, b)varying original dimensionality, c)varying number of natural clusters.

From the above results, we see that EPC grows approximately linearly with all these 3 factors. This scalability makes EPC applicable for clustering very large data sets in high dimensionality, especially when the users need to understand the underlying projective clustering structure within a short period of time.

5.7 Scalability of EPC2

We compare the user CPU time on the EPC2, EPC and ORCLUS in **APN-Set** in Figure 5.3. We input the correct values of parameters (dimensionality of subspaces and number of clusters for ORCLUS, number of clusters users interested to discover in EPC2 and EPC). Each data point represents the experimental result obtained from one single dataset. Figure 5.3(a) plots time in log scale against varying N with $D = 20$ and $L = 6$. Figure 5.3(b) plots time in log scale against varying D with $N = 50000$ and $L = 6$. Figure 5.3(c) plots time in log scale against varying L with $N = 50000$ and $D = 20$.

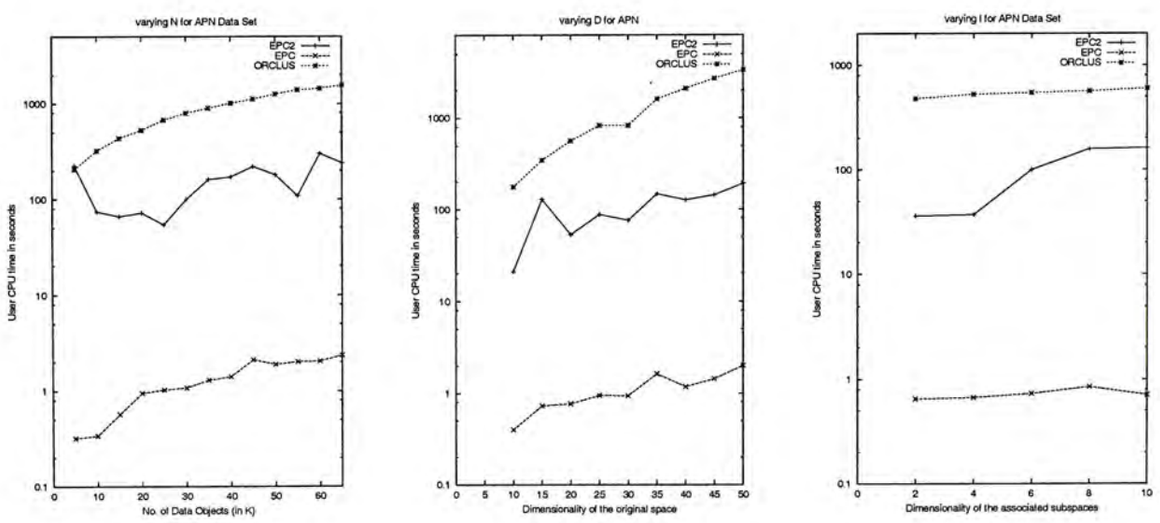


Figure 5.3: User CPU Time against a)varying number of data objects, b)varying original dimensionality, c)varying dimensionality of associated subspaces

We observe that EPC is extremely fast comparing to ORCLUS and EPC2 in all cases. (e.g. for the largest data set, EPC requires about 2 user CPU seconds, while EPC2 requires about 300s and ORCLUS takes about 1500s.) We examine the user CPU time of different phases in EPC2, and observe that it is dominated by the signatures merging phase. The theoretical running time complexity of this phase is $O(Nlog(km) + (km)^3 l^d)$, where m is the number of clusters the user binterested to discover. We set km the number of top ranked

signatures we would keep to be $50m$, and $d = 2$ for EPC2.

With fixed value of m, l and varying N, however, we do not observe a clear trend of the running time. This happens even we do a number of trials in different datasets, or average the results obtained from different trials. Although the worst case should be bounded by N, the average running time greatly depends on the similarity among data objects. The more dissimilarity among the objects, the less time required for the merging process to reach the termination criteria. Therefore, the curve of EPC2 in figure 5.3(a) cannot be smoothed by taking a number of experiment trails. In any cases, EPC2 is more efficient than ORCLUS when N is up to 65000.

With varying D, ORCLUS scales superlinearly, which agrees with the running time analysis that ORCLUS is $O(D^3)$ in [3]. It is not very feasible to apply ORCLUS to database with high dimensionality. EPC is about linear with D. EPC2 scales quadratically with D, which matches the analysis that the time complexity is $O(D^d)$ in Section 4.5.

With varying l, both EPC and ORCLUS scales linearly. EPC2 scales quadratically with l, which matches the analysis that the time complexity is $O(l^2)$ in the case of EPC2.

Chapter 6

Conclusion

Projective clustering is often preferable compared to traditional clustering for high-dimensional data. Based on the definition of projective clustering, we propose an efficient algorithm EPC, with a very different approach from the previous methods of PROCLUS and ORCLUS. EPC satisfies the following desirable properties for clustering algorithm.

(1) It needs minimum domain knowledge of the user. Many other clustering algorithms, which takes many input parameters that may not be known by the user in advance, and different setting of parameters can greatly impact the results. EPC only takes one input parameter, *max_no_cluster*. In addition, different setting of this parameter would not affect the correctness of the output clustering result.

(2) It has a high efficiency on a large database. As shown from the experiments, EPC scales approximately linearly with the number of data objects, dimensions and natural clusters. This is especially useful, when the user want to quickly identify the underlying clustering structure.

(3) The returned subspaces and the corresponding hyper-rectangles in the high dimensional space can be easily interpreted by the end-users, and gives an insight to the user about which features are important to a particular cluster, and where it roughly locates. With the partitioning result and the associated

subspace, the cluster shape can be refined by feeding only the related dimensions and the group of data objects to other traditional clustering algorithm.

(4) It is insensitive to the ordering of the data objects, and the presence of outliers.

Compared with PROCLUS, the running time of EPC is significantly faster while giving a better clustering result. It does not need many parameter setting, and the returned hyper-rectangles contain more useful information to the user.

Based on EPC, we further generalize the approach to estimate data density by histograms with higher dimensionality. We name this class of approaches as EPCH (Efficient Projective Clustering using Histogram). In particular, we describe and implement EPC2, which construct 2-d histograms to do the clustering. Empirical results show that EPC2 can give comparable results as ORCLUS, for datasets that contain clusters associated with subspaces formed by axes parallel vectors, with much less actual computational time. In some scenario, e.g. 1) data set contains outliers, 2) the dimensionality of associated subspaces is varying, 3) the dimensionality of associated subspaces is low, the clustering quality given by EPC2 outperforms ORCLUS. In any data set, running time of EPC2 is significantly faster than previous approaches.

Signatures of data objects can be conceptually understood as a transaction database. Each signature corresponds to a transaction, where each "entry-denseid" pair in the signature corresponds to an item. A group of signatures sharing common set of "entry-denseid" pairs identifies a projected cluster and its associated subspace. Therefore, the projective clustering problem can be transformed to the problem of mining frequent itemsets in the transaction database. The frequent itemset mining problem is well established and various fast algorithms have been proposed. However, most of these algorithms are developed for mining frequent itemsets with relatively low support (e.g. less than 5%), and the sizes of the maximal frequent itemsets would not be

too large. By maximal, we mean no supersets of the itemset are frequent. In the context of projected clustering, we want to discover itemsets with relatively high support (e.g. if the cluster contains 20% of data objects, the support would be 20%) and the sizes of maximal frequent itemsets should be ${}_lC_d$, where l is the dimensionality of the associated subspace, d is the dimensionality of histograms we constructed. This is a large number comparing to other mining frequent itemsets problems. Because of these two properties, the mining step requires much longer computational time than current approach. New algorithm which can do the mining efficiently with these two properties can be developed, so that it can be plugged into current approach to enhance the clustering quality and shorten the running time of the clustering process.

Part II

Multiple Tables Association Rules Mining

Chapter 7

Introduction to Multiple Tables Association Rule Mining

The problem of mining association rules over basket data was introduced in [7]. Basket data consists of a set of transactions, each transaction consists of a set of products purchased. The task is to discover interesting association rules, such as "65% of customers who buy milks would also buy bread". In this example, the association rule "$milks \Rightarrow bread$" represents a useful knowledge users are interested to find out for products promotion, shelf placement, store layout etc. In general, the rules are interesting only if they have a high *support* and high *confidence*.

Here, we give the definitions of **frequency** (also known as **support**), **frequent itemsets**, **association rule**, and **confidence** of the rule for a database D containing a set of transactions, where each transaction contains a set of *items*. The frequency of a set of items Z (we denote as $support(Z)$) is the number of occurrences of Z in D, or in other words, the number of transactions in D that contain Z. Frequent itemsets are sets of items that have frequencies above a threshold *minsup*. An association rule has the form of $X \Rightarrow Y$ where X and Y are sets of items. In such a rule, we require that $X \bigcup Y$ (we would denote as XY) is a frequent itemset, i.e. $support(XY) > minsup$. The confidence of the rule is the probability of Y given X, i.e. $\frac{support(XY)}{support(X)}$. For a rule to

be "interesting", we also require that its confidence is above another threshold $minconf$.

The mining process can be divided into two main steps. In the first step, all frequent itemsets are identified. In the second step, from the sets of discovered frequent itemsets, we can generate the association rule. However, the generation of *frequent itemsets* is a more difficult problem, which has been shown to be NP-hard. Since the search space is exponential with the number of items, with millions of database objects, I/O minimization would become critical as this step contributes to the major portion of the computational time.

There has been a lot of research work [46, 39, 9, 49, 33, 38, 13, 40, 52, 41] in association rule mining in recent years. Although the first applications were found in supermarket data, the technique has been extended to work on numerical data and categorical data in more conventional databases [43, 44], some researchers have noted the importance of association rule mining in relation to relational databases [45]. Tools for association rule mining are now found in major products such as IBM DB2 Intelligent Miner [1], and SPSS's Clementine.

Many efficient algorithms have been proposed and developed in recent years to generate frequent itemsets. However, most previous frequent itemsets discovery algorithms assume items appearing as binary representation in transactions from a single table. An item is either purchased or not purchased in an transaction. In RDBMS, an attribute can take categorical values or continuous values (where continuous values can be partitioned into ranges and transformed to categorical). More importantly, data are often decomposed and stored in multiple tables in RDBMS [42]. Previous frequent itemsets mining algorithm cannot be directly applied in this scenario. Therefore, we are motivated to propose algorithms and data structures to efficiently mine frequent itemsets from multiple relational tables. In particular, we investigate relational database in a **star schema** [11].

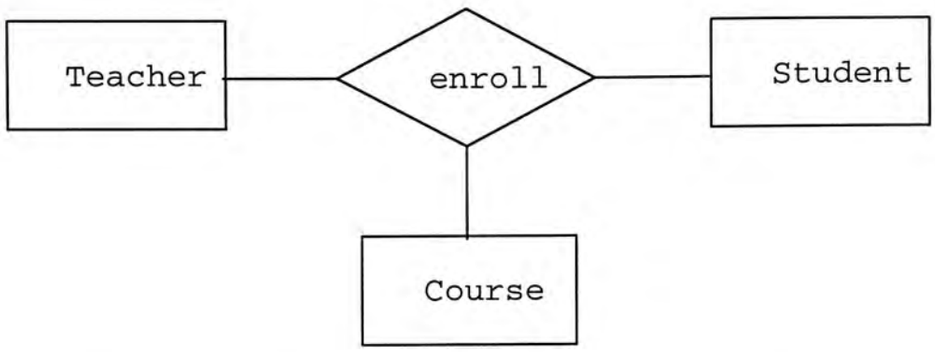


Figure 7.1: ER Diagram modelling relationship among teachers, courses, students

7.1 Problem Statement

Consider a relational database in a **star schema**. There are multiple *Dimension Tables*, which we would denote as **A, B, C**, ..., each of which contains only one primary key as *transaction id*, some other attributes and no foreign keys. We denote *tid(A)* as the transaction id of table **A**. Among the multiple Dimension Tables, there is one *Fact Table*, which we denote as *FT*. It stores the *tid(A), tid(B), tid(C)*, ... (which we denote as a_i, b_i, c_i) as foreign keys from Dimension Tables **A, B, C**, ..., and possibly some other optional attributes $O_1, O_2, ...$ In an ER model, Dimension Tables typically corresponds to an *entity set*, and Fact Table corresponds to a *relationship set*. We do not restrict the key constraints of the relationship to be many-to-many, many-to-one, or one-to-one. Figure 7.1 illustrates an example of an ER diagram modelling the entity sets teachers, courses, students, and a relationship set relating to these 3 entities. Figure 7.2 shows the corresponding tables in star schema. An example of the discovered rule could be "students with age > 25 and taking some course at the advanced level, would have a great chance to choose the teachers with age > 40".

In order to mine association rules across multiple *dimension tables*, we have to discover some sets of "attribute-value" pairs, which occur frequently in the table resulting from a natural join of these dimension tables. To do so, we can

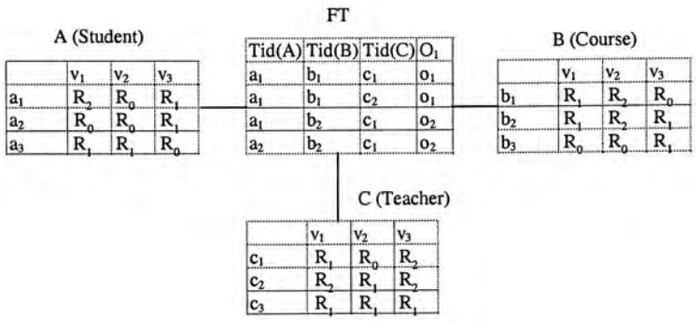


Figure 7.2: Star Schema with 3 Dimensional Table (teachers, courses, students) and 1 Fact Table

represent each "attribute-value" pair as an *item.* With this association, we are interested to mine frequent itemsets containing items across the multiple Dimension Tables, where the frequency of the itemsets are counted by the occurrence in the table resulting from the natural join of the multiple Dimension Tables and the Fact Table.

At first glance it may seem easy to join the tables and then do the mining process on the joined result. However, when multiple tables are joined, the resulting table will increase in size many folds. There are two major problems. Firstly, in large applications, often the join of all related tables cannot be realistically computed because of the many-to-many relationship blow up, large dimension tables, and the distributed nature of data.

Secondly, even if the join can be computed, the multi-fold increase in both size and dimensionality presents a huge overhead to the already expensive frequent itemset mining step:

(1) The number of columns of the joined table will be close to the sum of the number of columns in the individual tables. As the performance of association rule mining is very sensitive to the number of columns (items), the mining on the resulting table can take much longer computation time compared to mining on the individual tables.

(2) If the join result is stored on disk, the I/O cost will increase significantly for multiple scanning steps in data mining.

(3) For mining frequent itemsets of small sizes, a large portion of the I/O cost is wasted on reading the full records containing irrelevant dimensions.

(4) Each tuple in a dimension table will be read multiple times in one scan of the joined table because of duplicates of the tuple in the joined table.

Therefore, instead of "joining-then-mining", we can exploit the characteristics of relational tables in star schema, and apply "mining-then-joining" strategy in which the "joining" part is much less costly. We are motivated to mine association rules from the start schema, using the "mining-then-joining" approach.

Specifically, given three different kinds of input, namely

- Dimension Tables **A, B, C**, ..., where we assume attributes in the dimension tables are unique and will not appear in two different tables,
- a Fact Table *FT*, where we assume the optional attributes do not appear in any of the dimension tables,
- *minsup*, which specifies the minimum number of occurrence of "attribute-value" pairs in order to be frequent, in the natural join of all the given tables ($FT \bowtie A \bowtie B \bowtie C...$), where the joining conditions are given as $FT.Tid(A) = A.tid$, $FT.Tid(B) = B.tid$, $F.Tid(C) = C.tid$,...

we would like to mine all *frequent itemset* in the table ($FT \bowtie A \bowtie B \bowtie C...$), without performing the actual joining.

Chapter 8

Related Work to Multiple Tables Association Rules Mining

8.1 Aprori - A Bottom-up approach to generate candidate sets

For n items, there can be potentially 2^n *frequent itemsets.* The computational time would become exponential if we search for every possibilities. Therefore, the Apriori algorithm is proposed [5], which is a bottom-up approach to generate a $sizek$ candidate itemsets given all mined frequent itemsets with size $k-1$. Specifically, all $size-1$ frequent itemsets are found by simply counting the occurrences. Each subsequence pass k is divided into two phases. In the first phase, the previous mined itemsets with size $k-1$ is given. Any two size $k-1$ mined itemsets which are different from only the last item is combined to form a potential candidate itemset with size k. For example, $x_1x_2x_3$ can be combined with $x_1x_2x_4$ to form potential candidate $x_1x_2x_3x_4$. After that, any potential candidate size k itemsets are deleted if some of its (k-1)-subset is not the size $k-1$ frequent itemset. For example, If any of the size-2-subset of $x_1x_2x_3x_4$, that is $x_1x_2x_3, x_1x_2x_4$ (actually we do not need to check these two), $x_2x_3x_4$ is not frequent, $x_1x_2x_3x_4$ can be pruned away. In the second phase,

the supports of all remaining size k candidates are counted by one scan of the database. Candidates with enough support would become the size k frequent itemsets.

Aprori makes use of the monotonicity property of itemsets, which is similar to the monotonicity property of subspace as discussed in section2.3.1. For a given itemset, its support is higher than a certain threshold *only if* all of its subsets have enough support. In other words, if the support of an itemset is lower than the threshold, all of its supersets must not have enough support. This serves as a powerful pruning in two different ways. (1) We only need to do a counting for a candidate if all of its subsets are frequent. (2) When a itemset is not frequent, we need not to consider any of its superset in the future passes. In doing so, the searching time for frequent itemset which number is potentially exponential could be greatly reduced.

However, the disadvantages of Apriori is that it requires multiple database scans, as many as the longest frequent itemset. And thus, various optimization techniques are proposed to minimize the number of database scans, and so as the I/O time.

8.2 VIPER - Vertical Mining with various optimization techniques

Apriori and most other association rules mining algorithms are designed for use with *horizontal* database layout. In such representation, each row corresponds to a transaction with unique *tid* in terms of the items that were purchased in that transaction. Alternatively, VIPER [47] uses the *vertical* data representation where each *item* is associated with a column of values representing the transactions that contain the *item*.

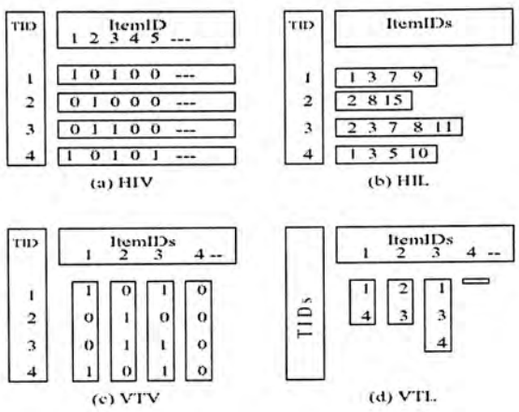


Figure 8.1: 4 different data representations for market-basket database

8.2.1 Vertical TID Representation and Mining

There are four different ways for representing the market-basket database, namely **HIV** (Horizontal item-vector), **HIL** (Horizontal item-list), **VTV** (Vertical tid-vector), and **VTL** (Vertical tid-list), as shown in Figure 8.1 [47].

HIV Each row corresponds to a transaction with an unique *tid*. For n items, there would be n entries in each row. Each entry represents whether the corresponding item is present in the transaction (1), or absence (0).

HIL Similar to HIV, except for each row, it records only the *iid* (**item identifier**) of the corresponding items that are present in the transaction.

VTV Each column corresponds to an item with unique *iid*. For m transactions, there would be m entries in each column. Each entry represents whether the corresponding transaction contains the item (1), or not (0).

VIL Similar to VIL, except for each column, it records only the *tids* corresponding to the transactions which contain the item.

VIPER proposed to apply vertical mining instead of horizontal. When the database is represented in vertical layout, supports of itemsets need not to be counted by scanning the whole database. Instead, it involves only the *intersection* of tid-lists or tid-vectors. Furthermore, only the tid-lists of the

interested itemset need to be stored in memory and accessed from disk, where in horizontal layout, all information of a transaction, including the uninterested items, need to be processed.

8.2.2 FORC

One of the optimization used in VIPER is **FORC** (Fully ORganized Candidate-generation). It is based on the technique called *equivalence class*, which groups all $size - k$ itemsets sharing a common prefix of length $k - 1$. For each class, the common prefix is stored in a hash table, and the last elements is stored in a lexicographically ordered list, called *extList*.

Similar to Apriori, FORC generates $size - k$ candidate frequent itemsets given size $k - 1$ itemsets. They both prune away candidates in which some of its subsets are not frequent. The main difference is that Apriori determines the subset status on a sequential (one candidate after another) basis, and FORC optimizes it by a simultaneous search. For example, $ABCD$ and $ABCE$ is generated and we have to check whether all of their subsets are frequent. In particular, ABC, ABD, ACD, BCD are checked for $ABCD$, and ABC, ABE, ACE, BCE are checked for $ABCE$. We totally need to check whether a given itemsets are frequent for 8 times. However, after grouping previous frequent itemsets into *equivalence classes*, actually ABC, ABD, ABE are from the same *equivalence class* prefixed with AB, ACD, ACE are from the same class prefixed with AC, BCD, BCE are from the same class prefixed with BC. In FORC, while examining class prefixed with AC, both ACD and ACE could be checked, and thus candidates $ABCD$ and $ABCE$ can be checked for pruning simultaneously.

8.3 Frequent Itemset Counting across Multiple Tables

Recently, [28] proposes an algorithm for mining frequent itemsets in decentralized data. Their work is closely related to our proposed solution, in which the joined table T is computed but without being materialized. When each row of the joined table is formed, it is processed and thereby storage cost for T is avoided. In the processing of each row in the table, an array that contains the frequencies for all candidate itemsets is updated. As pointed out by the authors, all itemsets are counted in one scan and there is no pruning from one pass to the next as in the apriori-gen algorithm in [5]. Therefore there can be many candidate itemsets and the approach is expensive in memory costs and computation costs. The empirical experiments in [28] compare their approach with a base case of applying the apriori algorithm on a materialized table for T. It is shown that the proposed method needs only 0.4 to 1 times the time compared to the base case. However, there are new algorithms in recent years such as [20, 47] which are shown by experiment to often run many times faster than the apriori algorithm. Therefore, the approach in [28] may not be as efficient as our proposed method.

Chapter 9

The Proposed Method

9.1 Notations

- **A, B, C** denotes the Dimension tables. x_i, y_i, z_i denotes the items from table **A, B, C** correspondingly. $x_i x_j$ denotes the **itemset** composed of item x_i, x_j, which are from the same Dimension Table. $x_i y_i$ denotes the itemset composed of item x_i, y_i from different Dimension tables. We assume an *ordering* of Dimension tables and an *ordering* of items which is adopted in any transactions and itemset. to facilitate our algorithm. E.g. x_1 would always appear before x_2 if they exist together in any transactions or itemset. x_2 would always appear before y_1 in an itemset.

- X denotes an itemset composed of item from **A** only. Similarity, Y, Z denotes an itemset composed of items from **B, C** only. We call the *size* of X is i, if X contains i items from **A**. XY denotes *itemset* composed of items from **A** and **B**.

- a_i, b_i, c_i denote the **transaction id** (*tid*) of **A, B, C** respectively.

- *tid_list* is a data type that we shall use in our algorithm. It is an ordered list of elements of the form *tid*(*count*), where *tid* is a transaction ID, and **count** is a non-negative integer.

- Given two tid_lists L_1, L_2, the **union** $L_1 \cup L_2$ is the list of $tid(count)$, where tid appears in either L_1 or L_2, and the count is the sum of the counts of tid in L_1 and L_2.

  The **intersection** of two tid_lists L_1, L_2 is denoted by $L_1 \cap L_2$, which is a list of $tid(count)$, where tid appears in both L_1 and L_2 and the $count$ is the smaller of the counts of tid in L_1 and L_2.

- $tid_A(x_i)$: a tid_list for x_i where x_i is an item in **A**. In each element $tid(count)$ in the list, tid is the transaction in **A** that contains x_i, and $count$ is the number of occurrence of the tid in FT. If the tid of the transaction that contains x_i does not appear in $tid_A(x_i)$, the count of it is 0 in FT.

  E.g. $tid_A(x_3) = \{a_1(5), a_3(2)\}$. Transactions containing x_3 in A are a_1 and a_3; a_1 appears 5 times in the FT, and a_3 appears 2 times.

- $tid_A(X)$ where X is an itemset with items from , **A**, it is similar to $tid_A(x_i)$ except x_i is replaced by X. If X is $x_i x_j$, $tid_A(x_i x_j)$ can be obtained by $tid_A(x_i) \cap tid_A(x_j)$.

- $B\_key(a_n)$: Given a tid a_n from **A**, $B\_key(a_n)$ denotes a tid-list of $tid(count)$. For each entry of $tid(count)$, tid is a tid from B and $count$ is the number of occurrences of tid together with a_n in FT.

  E.g. $B\_key(a_1) = \{b_3(4), b_5(2)\}$. It means that $a_1 b_3$ occurs 4 times in FT, and $a_1 b_5$ occurs 2 times.

- $B\_tid(x_i)$: Given an item x_i in **A**, $B\_tid(x_i)$ denotes a tid-list, of $tid(count)$. For each entry $tid(count)$, tid is a tid of B, and $count$ is the number of times the tid appears together with any tid a_j of A such that transaction a_j contains x_i in A.

  E.g. Suppose in A, only transactions a_2 and a_3 contain x_3. Let b_1 be

a tid of B. If in FT, a_2b_1 occurs 4 times, and a_3b_1 occurs 2 times, and there are no other occurrences of a_2 and a_3, then $B\_tid(x_3) = \{b_1(6)\}$

- $B\_tid(X)$: similar to $B\_tid(x_i)$ except item x_i is replaced by an itemset X from **A**.

- F^A denotes the set of frequent itemsets with items from **A**, F^{AB} denotes the set of frequent itemsets with items from tables **A** and/or **B**. F^{AB_k} denotes the set of frequent itemsets of the form XY, where X is either empty set or an itemset from **A**, and Y is either an empty set or an itemset from **B** with size k.

  E.g. $F^{AB} = \{x_1, y_1, x_1y_1\}$. E.g. $F^{AB_2} = \{y_1y_2, y_2y_3, x_1y_1y_2, x_1y_2y_3\}$.

- F^{A_k} denotes the set of frequent itemsets of size k from **A**. $F^{A_iB_j}$ denotes the set of frequent itemsets in which the subset of items from **A** has size i and subset of items from **B** has size j.

  E.g. $F^{A_2B_1} = \{x_1x_2y_1, x_3x_4y_2\}$.

9.2 Converting Dimension Tables to internal representation

Here we assume attributes in the input Dimension Tables take categorical values, where numerical values can be partitioned and transformed to categorical,the **domains** for each attributes are known, and same attribute will not appear in two different tables. (If initially two tables have common attributes, renaming can make them different.) We will transform the input Dimension Table to **HIL** representation as in 8 on-the-fly. Specifically, for each "attribute-value" pair in the original Dimension Tables, we will associate an *item* to represent it. For example, consider *dimension table*A in Figure 9.1, a_1, a_2, a_3 are *tid*, v_1, v_2, v_3 are attributes names, and the value of attribute v_1

A (Student)

	v_1	v_2	v_3
a_1	R_2	R_0	R_1
a_2	R_0	R_0	R_1
a_3	R_1	R_1	R_0

conversion

	$v_1=R_0$	$v_1=R_1$	$v_1=R_2$	$v_2=R_0$	$v_2=R_1$	$v_3=R_0$	$v_3=R_1$
	x_1	x_2	x_3	x_4	x_5	x_6	x_7
a_1	0	0	1	1	0	0	1
a_2	1	0	0	1	0	0	1
a_3	0	1	0	0	1	1	0

Figure 9.1: Convert the input Dimension Tables to **hil** representation on-the-fly

for transaction a_1 is R_2. In the conceptual *HIL* representation, we associate this pair with the *item* x_3. In our remaining discussion, we base on this *HIL* representation and associate an *item* for each "attribute-value" pair. We call each row of the Dimension Tables in the *HIL* representation as a **transaction**. In implementation, we do not actually compute and store the conceptual *HIL* representation. We have two implementations which adopt a modified *VTL* and *VTV* representations respectively as in 8, in which we can retrieve the *tid_list* of a given *item* easily, where *tid_list* is defined in Section 9.1. Specifically, suppose there are T_A, T_B, T_C transactions in table **A, B, C** respectively. In *VTL* implementatino, for each frequent item x in **A**, we store a *tid_list* which records the transactions in **A** that contains x. In *VTL* implementation, we store a column of T_A bits, the i^{th} bit is 1 if item x is contained in transaction i, and 0 otherwise. We also keep an array of T_A entries where the i^{th} entry corresponds to the frequency of tid i in FT.

We aim to mine frequent itemsets composed of items from individual Dimension tables as well as across multiple Dimension tables. We apply the same threshold *minsup* to itemsets from single or multiple Dimension Tables. The frequency of an itemsets is counted from the occurrence of the corresponding transactions that contain the items in FT.

9.3 The idea of discovering frequent itemsets without joining

We present a simple example to show the idea of discovering frequent itemsets across multiple Dimension tables without actually performing the join operation.

Example 8 Suppose we have a star schema for a number of dimension tables related by a fact table FT. Figure 9.2 shows 2 of the Dimension tables **A** and **B**, and the projection of FT on the two columns that contains transaction ID's for **A, B**, but without removing duplicate tuples. We do not assume any ordering in the FT, and some of the tid from the Dimension tables may not appear in FT (e.g. a_2, b_1).

Suppose *minsup* is set to 5. We first mine frequent itemsets from **A** and **B** individually. The itemset are frequent if they appear at least 5 times in FT. For example, x_1 and x_3 appear together in a_1, a_3, and their total count in FT is 6. Hence, x_1x_3 is a frequent itemset. Next we check if a frequent itemset from **A** can be combined with another frequent itemset from **B** to form a frequent itemset of greater size.

x_1x_3 is one of the frequent itemsets from **A**, y_1y_6 is another frequent itemsets from **B** (y_1y_6 appear together in b_2, b_5, whose count in FT is 5). We want to check if x_1x_3 can be combined with y_1y_6 to form a frequent itemset, the steps are outlined as follows:

Tid	Items
a_1	x_1, x_3, x_5
a_2	x_2, x_3, x_6
a_3	x_1, x_3, x_6
a_4	x_1, x_4, x_6

Tid(DA)	Tid(DB)
a_1	b_5
a_1	b_3
a_1	b_2
a_3	b_2
a_3	b_5
a_1	b_5

Tid	Items
b_1	y_1, y_3, y_5
b_2	y_1, y_3, y_6
b_3	y_2, y_4, y_6
b_4	y_1, y_4, y_5
b_5	y_1, y_4, y_6

Figure 9.2: Example for discovering *frequent itemsets* across dimension tables

1. $tid_A(x_1x_3) = tid_A(x_1) \cap tid_A(x_3) = \{a_1(4), a_3(2)\}$, $tid_B(y_1y_6)tid_B(y_1) \cap tid_B(y_6) = \{b_2(2), b_5(3)\}$.

2. $B\_key(a_1) = \{b_2(1), b_3(1), b_5(2)\}$, $B\_key(a_3) = \{b_2(1), b_5(1)\}$.

3. $B\_tid(x_1x_3) = B\_key(a_1) \cup B\_key(a_3) = \{b_2(2), b_3(1), b_5(3)\}$.

4. $B\_tid(x_1x_3) \cap tid(y_1y_6) = \{b_2(2), b_5(3)\}$.

5. The combined frequency = total count in the list $\{b_2(2), b_5(3)\} = 5$.

Hence the itemset $x_1x_3y_1y_6$ is frequent. ∎

In general, to examine the frequency for an itemset S that contains items from two dimension tables **A** and **B**, we do the following.

We examine table **A** to find the set of transactions T_1 in **A** that contain the A items in S.

Next we determine the transactions T_2 in B that appear with those transactions in T_1 in FT. Note that this is similar to the derivation of an intermediate table in a **semi-join** strategy, where the result of joining a first table with the key of a second table are placed, the key of the second table is a **foreign key** in this intermediate table.

In the mean time, the set of transactions T_3 in B that contain the B items in X are identified. Finally T_2 and T_3 are intersected, and the resulting count is obtained. Note that when we utilize $B\_tid()$ and $B\_key()$ for the above computation, we are using the primary key of table **B** as a **foreign key** for an intermediate semijoin table where A is "joined" with FT for the particular itemsets X.

The use of tid_list is a compressed form of recording the occurrence of tid's in the fact table. Multiple occurrences of transaction id's such as a_1 or a_1b_2 in FT are condensed as one single entry in a tid_list with the count associated.

Initial Step: In order to do the above, we need to have at least some initial information about $tid_A(x_i)$ for each item x_i in each Dimension Table **A**. One scan of a dimension table can give us the list of transactions for all items. In one scan of FT we can determine all the counts for all transactions in all the dimension tables. In the same scan, we can also determine $B\_key(a_i)$ for each *tid* a_i in each Dimension Table.

9.4 Overall Steps

For simplicity, let us first assume that there are 3 Dimension Tables **A, B, C**. The overall steps of our method are:

Step 1 : Preprocessing

Read the Dimension Tables, covert them into VTV or VTL representation with counts as described in Section 9.2.

Step 2 : Local Mining

Perform local mining on each dimension table.

Step 3 : Global Mining

Step 3.1 : Scan the Fact Table

Scan the Fact Table FT and record the information in some data structures.

We set an ordering for **A, B, C**. First we handle tables **A** and **B** with the following 2 steps:

Step 3.2 : Mining size-two itemsets

This step examines all pairs of frequent items x and y, which are from the two different Dimension Tables.

*Step 3.3 : Mining the rest for **A** and **B***

Repeat the following for $k = 3, 4, 5.....$ Candidates are generated by the union of pairs of mined itemsets of size $k-1$ differing only in the last item. The technique of generation would be similar to FORC [47]. Next count the frequencies of the candidates and determine the frequent itemsets.

After Steps 3.2 and 3.3, the results will be all frequent itemsets formed from items in tables **A** and/or **B**. This can be seen as the frequent itemset mined from a single dimension table **AB**. Similar steps as Steps 3.2 and 3.3 are then applied for the tables **AB** and **C** to obtain all frequent itemsets from the star schema.

9.5 Binding multiple Dimension Tables

In general there can be more than 2 dimension tables in the star schema. We can easily generalize the overall steps above from 3 Dimension Tables to N Dimension Tables. Suppose there are totally N dimension tables and a fact table FT in the star schema. We start with two of the Dimension Tables, say **A** and **B**. We apply Steps 3.2 and 3.3 above to mine all frequent itemsets with items from **A** and/or **B** without joining the tables with FT to find F^{AB}. We call Steps 3.2 and 3.3 a **binding** step of **A** and **B**. After binding, we treat **A** and **B** as a single table **AB** and begin the process of binding **AB** with another table, this is repeated until all N dimensions are bound.

After performing "binding", we can treat the items in the combined itemsets as coming from a single Dimension Table. For example, after "binding" **A** and **B**, we *virtually* **combine A** and **B** into a single Dimension Table **AB**, and all items in F^{AB} are from the new Dimension Table **AB**. We always "bind" 2 Dimension Tables at each step, and iterate for $N-1$ times if there are totally N Dimension Tables. At the end all frequent itemsets will be discovered.

Figure 9.3 shows a possible ordering of the "bind" operations on four dimension tables: **A, B, C, D**.

We need to do two things to combine two Dimension tables: (1) To assign each combination of *tid* from table **A** and *tid* from table **B** in FT a new *tid*, and (2) to set the *tid* in the *tid_lists* for items in **AB** to the corresponding new *tid*.

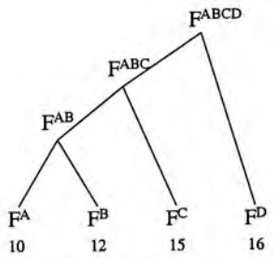


Figure 9.3: An example of "binding" order

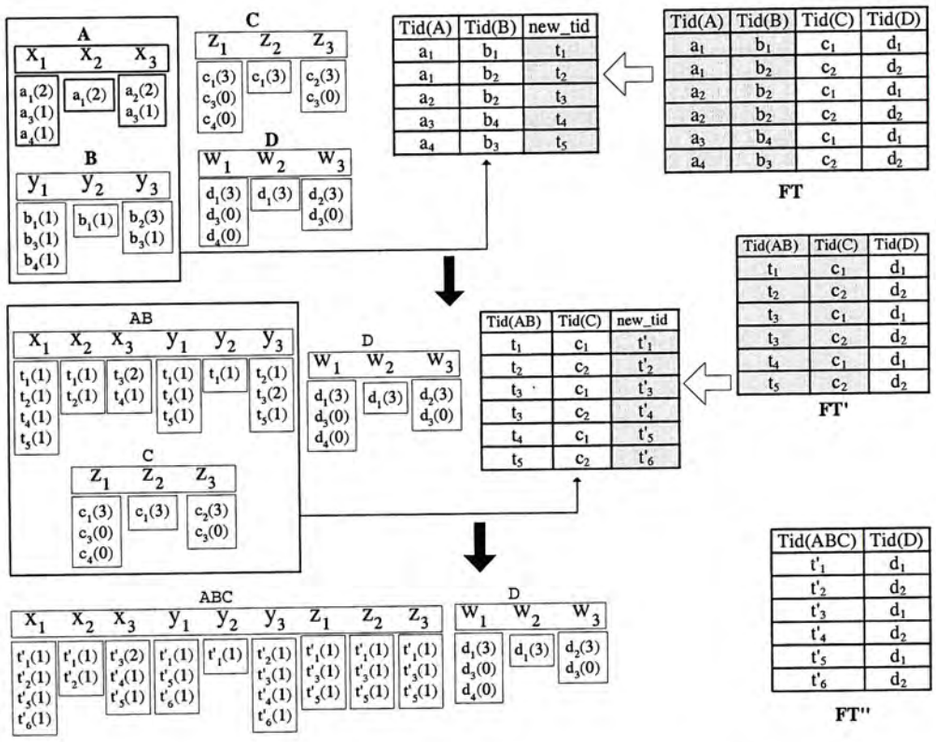


Figure 9.4: Concatenating tids after "binding"

Consider an example in Figure 9.4, for a FT relating to 4 Dimension Tables **A, B, C, D**, after "binding" **A** and **B**, the columns storing tid(A) and tid(B) would be concatenated. Each combination of tid(A) and tid(B) would be assigned a new tid. **A** and **B** would be combined into **AB**. For example, before "binding", item x_1 appears in transactions a_1, a_3 and a_4. $tid_A(x_1) = \{a_1(2), a_3(1), a_4(1)\}$. After "binding", since a_1 corresponds to new tid t_1 and t_2, a_3 corresponds to new tid t_4, a_4 corresponds to new tid t_5. Therefore $tid_A(x_1)$ is updated to $tid_{AB}(x_1) = \{t_1(1), t_2(1), t_4(1), t_5(1)\}$. In this example, $tid\_list(x_1)$ becomes longer, but the total count is 4, which is the same as before.

Similarly, when F^{AB} is then "bound" with F^C, **AB** is combined with **C** and FT would be updated again.

Note that in Figure 9.4, the tables with attribute *new_tid* and the multiple fact tables are not really constructed as tables, but instead stored in a structure which is a prefix tree.

We always bind a given Dimension Table with the result of the previous binding because the tid of the Dimension Table allows us to apply the technique of a foreign key as described in the previous section. The ordering can be based on the estimated result size of natural join of the tables involved, which can in turn be estimated by the Dimension Table sizes. A heuristic is to order the tables by increasing table sizes for binding.

9.6 Prefix Tree for FT

In Step 3.1 of the overall steps, the fact table FT is scanned once and the information is stored into a data structure which can facilitate the mining process. The data structure is in the form of a *prefix tree.* Each node in the prefix tree has a label (a tid) and also a counter.

We need only scan FT once to insert each tuple into the prefix tree. Suppose we have 3 dimensions **A, B, C**, and a tuple is a_3, b_2, c_2. First, we enter at the root node and go down a child with label a_3, from a_3 we go down to a child node with label b_2, from b_2 we go to a child node labelled c_2. Every time we visit a node, we increment the counter there by 1. If any child node is not found, it is created, with the counter initialized to 1. Hence level n of the prefix tree corresponds to tid's of the n^{th} Dimension table that would be "bound". When searching for a foreign tid_list, we can go down the path specified by the prefix. In this way, the *foreign key* and the global frequency in the i^{th} iteration can be efficiently retrieved from the $i + 1^{th}$ level of the *prefix tree.* Figure 9.5 shows how a fact table is converted to a prefix tree.

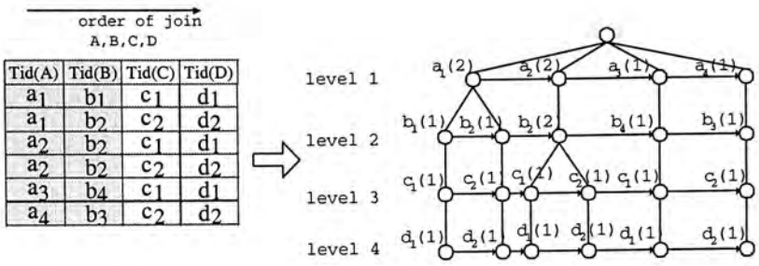


Figure 9.5: Prefix Tree structure representing FT

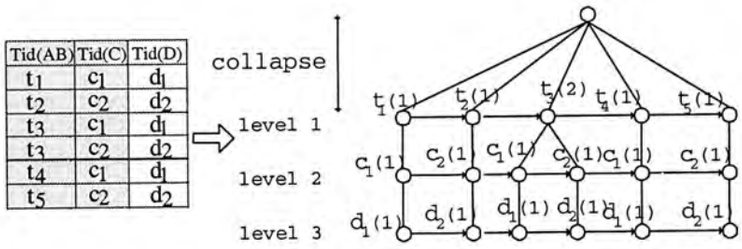


Figure 9.6: Collapsing the prefix tree

Use of the prefix tree – the foreign key: The prefix tree is a concise structuring of FT which can facilitate our mining step. When we want to "bind" F^A with F^B, we have to check whether an itemset (e.g. x_1) in F^A can be combined with an itemset in F^B (e.g. y_1). We need to obtain the information of a foreign key in the form of *tid_list* (e.g. $B\_tid(x_1)$). Let $tid_A(x_1) = \{a_1(2), a_2(1)\}$. We can find $B\_key(a_1)$ by searching the children of a_1 which are labelled $b_1(1)$, $b_2(1)$, similarly let $B\_key(a2) = \{b_2(2)\}$. As a result, $B\_tid(x_1y_1) = B\_key(a_1) \cup B\_key(a_2) = \{b_1(1), b_2(3)\}$.

Collapsing the prefix tree: Suppose **A** and **B** are bound, AB is the derived dimension. If **B** is not the last dimension to be bound, we can collapse the prefix tree by one level. A new root node is built, each node at the second level of the original tree becomes a child node of the new rootnode. The subtree under such a node is kept intact in the new tree. Figures 9.5 and 9.6 illustrate the collapse of one level in a prefix tree.

To facilitate the above, we create a horizontal pointer for each node in the same level so that the nodes form a linked list. A unique AB_tid is given to

each of the nodes in the second level, which corresponds to the bound table **AB**. These unique tids at all the levels can be assigned when the prefix tree is first built.

Updating tid: We need to do the following with the collapse of the prefix tree. After binding two tables **A** and **B**, a "derived dimension" **AB** is formed, we would update the tid_lists stored with the frequent itemsets and items that would be used in the following iteration, so that all of them are referencing to the same (derived) Dimension Table. For example, $tid_A(X)$ or $tid_B(Y)$ are updated to $tid_{AB}(X)$ or $tid_{AB}(Y)$.

9.7 Maintaining frequent itemsets in FI-trees

In both local mining and global mining (Steps 2 and 3), we need to keep frequent itemsets as they are found from which we can generate candidate itemsets of greater sizes. We keep all the discovered frequent itemsets of the *same size* in a tree structure which we call an **FI-tree** (FI stands for Frequent Itemset). Hence itemsets of $F^{A_3B_1}$ is mixed with itemsets of $F^{A_2B_2}$, the first one belongs to F^{AB_1}, the second belongs to F^{AB_2}. Let us call the FI-tree for size k itemsets H_k.

In the FI-tree, we also keep the relevant *tid_lists* for the frequent itemsets at the leaf nodes. Figure 9.7 shows an FI-tree. An itemset such as $x_1x_2y_2y_3$ is kept in a path from the root node to a leaf node (recall that we assume an ordering on the items, so the items follow this order in the path). The corresponding tid_list for $x_1x_2y_2y_3$ is kept at the leaf node. Itemsets with the same prefix share the same path. If the number of items is large, an hash function can be applied at each node to allow mapping of multiple items to the same child. This will be similar to the hash tree structure in [5]. The FI-tree will also be used in the generation of candidate itemsets. The technique

of generation would be similar to FORC [47] so that the same leaf node and tidlist in the FI-trees need not to be examined repeatly.

Let us examine the overall steps again. In Step 2, we find the set of frequent itemsets F^A, for each **A**. This mining can be based on any existing algorithm with a slight modification that the single count of each transaction a_i is replaced by the count of it in FT, which is kept in the prefix tree for FT. We store each frequent itemsets X of size k together with $tid_A(X)$ into FI-tree T_k.

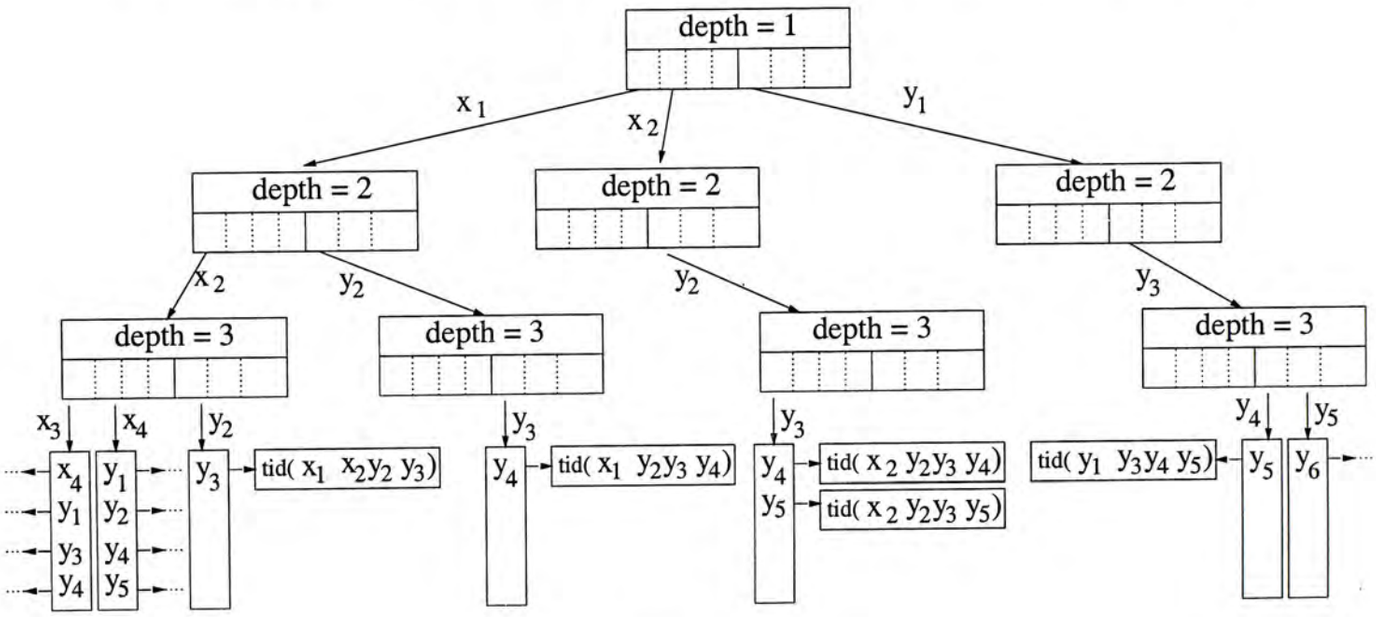


Figure 9.7: An FI-tree

In Steps 3.2 and 3.3, in each iteration, we generate candidate itemsets in an order of increasing itemset sizes, starting with size 2. In each case, we first generate a size k candidate by joining 2 $(k-1)$-itemsets with similar prefix. After this joining step we prune the candidates if any subset of them is not frequent. For example, for candidate $x_1x_2y_1y_2y_3$, if $x_1x_2y_2y_3$ is not frequent, this candidate have to be removed.

Suppose we have found the frequent size 2 itemsets of F^{A_2}, F^{B_2} and $F^{A_1B_1}$, we can generate F^{A_3}, F^{B_3}, $F^{A_1B_2}$, and $F^{A_2B_1}$. Let us examine each of the four cases. For F^{A_3}, we can generate the candidate sets by joining from F^{A_2} and the pruning is also by examining F^{A_2}. For F^{B_3} the case is also very similar.

For $F^{A_1B_2}$ we first generate the itemsets by joining two itemsets with similar prefix in $F^{A_1B_1}$. Then we prune the itemsets by examining both $F^{A_1B_1}$ and F^{B_2}. For $F^{A_2B_1}$ we first generate the itemsets by joining two itemsets with similar prefix, one from F^{A_2} and the other from $F^{A_1B_1}$. Then we prune the itemsets by examining both $F^{A_1B_1}$ and F^{A_2}. To perform the above candidate generation, we look up the size two frequent itemsets of F^{A_2}, F^{B_2} and $F^{A_1B_1}$ all together in the FI-tree H_2, and if a candidate is found and the frequency is above the threshold *minsup*, it is stored in H_3. This will be similar for itemsets of larger sizes.

Example 9 $x_1x_2y_1y_2$ and $x_1x_2y_1y_3$ (from $F^{A_2B_2}$) are frequent itemsets in H_4. These two can be used to generate new candidate $x_1x_2y_1y_2y_3$ (candidate for $F^{A_2B_3}$). In order to determine whether $x_1x_2y_1y_2y_3$ is frequent or not, we look up $F^{A_2B_2}$ and $F^{A_1B_3}$ to check whether $x_1x_2y_1y_3$, $x_2y_1y_2y_3$, and $x_1y_1y_2y_3$ exist in these sets or not (from H_4). If they are and the frequency of $x_1x_2y_1y_2y_3$ is above the threshold *minsup* (the frequency count methods would be described in Section 9.8), the new candidate $x_1x_2y_1y_2y_3$ is inserted into H_5.

In the FI-tree, we try to maintain the boundaries between say F^{AB_1} and F^{AB_2}. We assume an ordering of tree nodes in terms of the Dimension Tables. For example among the child nodes of any given node, the nodes corresponding to values of table **A** always appear before those of table **B**. We record in the parent node a pointer to the first child node that is for table **B**.

In this way, when we want to find elements for F^{AB_1}, we go to the level above leaf node level and for each node, find the first child node that corresponds to **B**, the following child nodes are also of interest.

9.8 Frequency Counting

In the overall Step 3.3, after a set of candidate itemsets from two tables **A** and **B** is generated, the frequencies of the itemsets are counted in order to identify the frequent itemsets. We consider two itemsets X_i and Y_j containing items from two different Dimension Tables respectively. Here we examine some algorithms for counting the frequency of the itemset $X_i \cup Y_j$.

Frequency Counting: Algorithm 10.1

Assume that we have found the values of $tid_A(x_i)$ for all items x_i, and $B\_key(a_j)$ for all tid's a_j in A. The first mechanism of checking the frequency of itemset $X_i \cup Y_j$ is outlined in Algorithm 9.1.

Algorithm 9.1 count_global_support(X_m, Y_n)
1 Let x_i be the i^{th} item in X_m, for $1 \leq i \leq m$
2 y_j be the j^{th} item in Y_n, for $1 \leq j \leq n$
3 Let X_m contain items of table A, Y_n contain items from table B.
4 $tid(X_m) = tid(x_1); tid(Y_n) = tid(y_1); B\_tid(X_m) = \{\};$
5 **for** $(i = 2$ to $m)$
6 $\quad tid(X_m) = tid(X_m) \cap tid(x_i);$
7 **for** $(j = 1$ to $n)$
8 $\quad tid(Y_n) = tid(Y_n) \cap tid(y_j);$
9 **for** $(i = 1$ to size of $tid(X_m)$ $)$
10 $\quad B\_tid(X_m) = B\_tid(X_m) \cup B\_key($ tid of the i^{th} entry of $tid(X_m)$ $)$
11 $tlist(X_m, Y_n) = B\_tid(X_m) \cap tid(Y_n);$
12 return (count $(tlist(X_m, Y_n)));$

The steps in the algorithm are similar to those in Example 8. One can examine the example to see how it works. The correctness of the above algorithm can be established by the following lemma but we skip the proof for interest of space.

Lemma 2 Let T be the table resulting from a natural join of all the Dimension Tables and FT. Suppose that $X \cup Y$ appears in the set of tuples T_{XY} in T. $tlist(X, Y)$ at the end of Algorithm 9.1 gives us the tid's of **B** in T_{XY} in a

compressed form, meaning multiple appearances of the same tid's are recorded by the tid with a count.

Proof: With the for-loop at Lines 5,6, we generate $tid(X_m)$ which is the tid-list that contains all the **A** tids for tuples that contains X_m in **A** together with the count of such tids in the fact table. With the for-loop at Lines 7,8, we generate $tid(Y_n)$ which is the tid_list that contains all the **B** tids for tuples that that contains Y_n in **B** together with the count of such tids in the fact table. These correspond to the **B** tid values at tuples T_Y in T where Y_n occurs. At Lines 9,10, for each **A** tid in $tid(X_m)$, we find the **B** tids in the fact table which corresponds to the **B** tid values at tuples T_X in T where X_m occurs, resulting in $B_{tid}(X_m)$. At Line 11, $tid(Y_n) \cap B_{tid}(X_m)$ is computed. Since X_m and Y_n both occurs in T iff X_m occurs (in T_X) and Y_n occurs (in T_Y), the result at Line 11 corresponds to the **B** tid values for exactly the tuples T_{XY} in T where both X_m and Y_m occurs. ∎

Frequency Counting: Algorithm 10.2

Algorithm 9.1 suffers from repeated intersection of *tid_lists* when computing the support, that is, $tid(x_1x_2t_3)$ is computed from scratch though $tid(x_1x_2)$ has been computed at an earlier time. In our second algorithm we try to eliminate this.

Suppose we need to count the support of XY, where X is an itemset from **A** and Y is an itemset from **B**. We assume that $B\_tid(X)$ is kept for all frequent itemsets X, where X is from table **A**. We assume $tid(Y)$ is kept for all frequent itemsets Y, where Y is from table **B** (this is kept in an FI-tree)

Algorithm 9.2 count_global_support(X_m, Y_n)

1 $tlist(X_m, Y_n) = B\_tid(X_m) \cap tid(Y_n)$;

2 return (count ($tlist(X_m, Y_n)$));

We can determine $B\_tid(X)$ when X is created. $B\_tid(X)$ is obtained from $B\_key(a_i)$ for $a_i \in X$. $B\_key(a_i)$ is obtained from the prefix tree for FT.

$B\_tid(X)$ is then recorded in a linked list structure.

Frequency Counting: Algorithm 10.3

Algorithm 10.3 is applicable if the *tid_lists* associated with the subsets that generate a candidate are related to the same "derived" dimension, such as AB. For example, if x is from **A**, y is from **B**, we update $tid_A(x)$ to $tid_{AB}(x)$, and $tid_B(y)$ is updated to $tid_{AB}(y)$. The support of candidate xy is computed by intersecting the *tid_lists* ($tid_{AB}(x)$ and $tid_{AB}(y)$ in this case) associated with the two parents that generate the candidate.

Applying Algorithms 10.2 and 10.3

In our mining steps, if dimension tables **A**, **B**, and **C** are bound in that order, we apply Algorithm 10.3 for the binding step of **A** and **B**, and Algorithm 9.2 for the binding of **AB** and **C**.

Chapter 10

Experiments

10.1 Synthetic Data Generation

We generate synthetic data sets in a similar way as the generator in [19] to test our proposed methods. First, we generate each Dimension Table individually, in which each record consists of a number of attribute values within the attribute domains, and model the existence of frequent itemsets. Specifically, We have the following parameters:

D	number of dimensions
n	number of transactions in each Dimension Table
m	number of attributes in each Dimension Table
s	largest size of frequent itemset
t	largest number of transactions with a common itemset
d	domains of attributes (we assume same for all attributes)
p	probability of attributes sharing common values in transactions containing frequent itemset

Table 10.1: Parameters used in synthetic data generation in single Dimension Table

The domain size d of an attribute is the number of different values for the attribute, which is the number of items derived from the attribute-value pairs for the attribute. We can imagine the generation process as placing a

number of *rectangles* [1] with widths within the range from 1 to s, and heights within the range from 1 to t in the Dimension Table, such that attributes in the transactions within the rectangles would very likely share common values. That is, frequent itemsets exist within the rectangle. Each attribute in the rectangle in those transactions contain the same value with probability p, where p should be set near to 1. Outside the rectangles, values of attributes are chosen randomly. (see Figure 10.1). Parameter p, which is the inverse of the *corruption level* described in [5], models the situation that all the items in a frequent itemset do not always appear together. The algorithm for generating transactions in a single Dimension Table is outlined in algorithm 10.1.

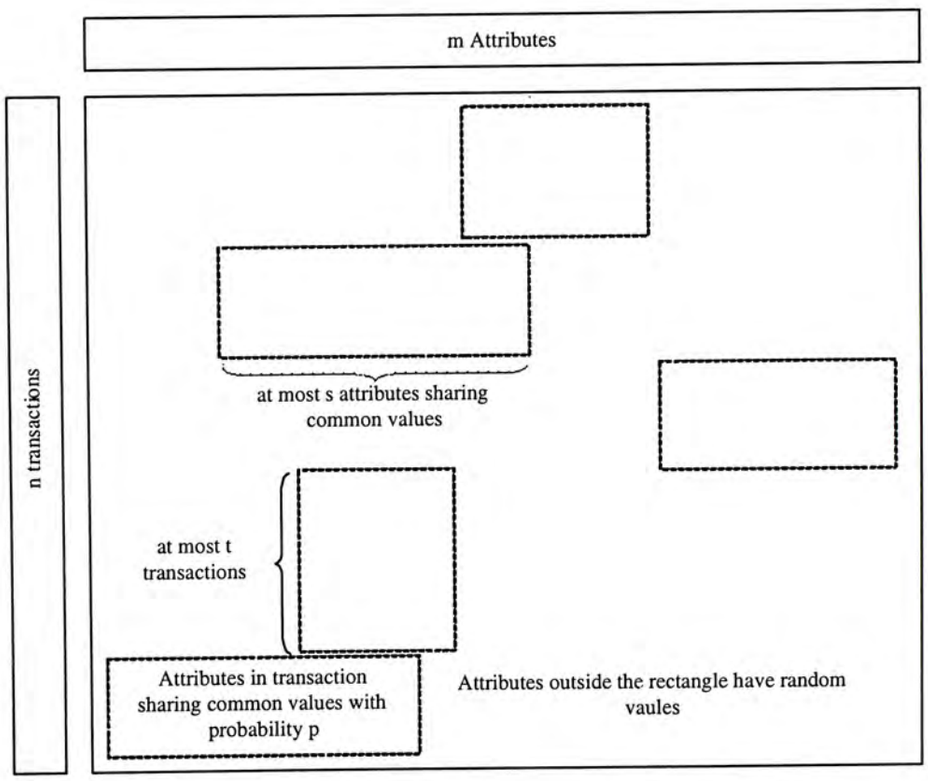


Figure 10.1: Idea of generating transactions with itemset in a Dimension Table

Algorithm 10.1 gen_dimension(n, m, s, t, p, q)

1 $k = 0$;

2 $no\_rect = 0$;

[1]Note that the "rectangles" may not look like rectangles: the attributes in a rectangle may not be consecutive in the relation layout, and the rows also may not be consecutive

R	No. of Related Dimensions
Br	Branching Factor
sup	target frequency of the association rules
$\|L\|$	number of maximal potentially frequent itemsets
N	number of noise transactions

Table 10.2: Parameters used for generating FT

3 **repeat**
4 $height[no\_rect] = rand() * t$;
5 $width = rand() * s$;
6 select $x_{i_1}, \ldots, x_{i_{width}}$ randomly from $x_1, \ldots, x_m$;
7 **for** ($l = 0$ to $height[no\_rect]$)
8 **if** ($k \geq n$)
9 break;
10 **end if**
11 for $Attribute_{x_{i_1}}$, ..., $Attribute_{x_{i_{width}}}$, assign a common value with probability p
12 for other Attributes, randomly select a value
13 $k++$;
14 **end for**
15 no_rect++;
16 **until** ($k \geq n$);
17 return ($height[0], \ldots height[no\_rect]$,$tuple_0, \ldots, tuple_{n-1}$);

After generating transactions for each Dimension Table, we generate FT based on the following parameters:

In constructing FT, there can be correlations among two or more Dimension Tables so that some frequent itemsets contain items from multiple dimensions. For the case of two dimensions, we want the *tids* associated with the same "rectangle" from one Dimension Table to appear at least *sup* times

together with another group of *tid* sharing common frequent itemsets from another dimension table. In doing so, frequent itemsets across dimensions from these 2 groups would appear with a frequency count greater than or equal to *sup*, after joining the two Dimension Tables and FT. We repeat this process for $|L|$ times, so that $|L|$ maximal potentially frequent itemsets would be formed (by **maximal**, we mean that no superset of the itemset is frequent).

The parameters R and Br model the situation where frequent itemsets composed of items from only a subset of all the Dimension Tables. There are different degree of correlations among different Dimension Tables. For example, we have table **A**, **B** and **C**. **A** and **B** are strongly related, and **C** is not related to **A** and **B**. In such case, frequent itemsets may contain items only from **A** and **B** but not **C**.

Parameter R specifies the number of related Dimension Tables, which can be equal or smaller than D. Parameter Br specifies for each pair of *tid* that contains frequent itemsets from **A** and **B**, how many transaction from **C** it would be related to in FT. For example, x_1y_1 form a frequent itemset. $tid\_list(x_1) = \{a_1\}, tid\_list(y_1) = \{b_2\}$. We set *sup* as 5 and Br as 3. That means we generate at least 5 pairs of $\{a_1b_2\}$ in FT. For each of such pair, it would be related to any 3 *tids* from C in FT. If all the dimension tables are related, we set Br to 1.

In order to generate some random noise, transactions which do not contain frequent itemset are generated. N rows in FT are generated, in which each tid from the dimension tables is picked randomly. The algorithm for constructing FT is outlined in algorithm 10.2.

Suppose as in Figure 10.2 we want to generate FT with 2 dimension tables, A and B. A has 3 groups of transactions sharing common frequent itemsets, $A[0], A[1], A[2]$ with size 3, 3, 4 respectively. B has 2 groups of transactions, $B[0], B[1]$ with size 2, 5 respectively. If we want frequent itemsets to exist in $A[0]B[0]$, $A[1]B[1]$, and $A[2]B[1]$, with support near s_1, s_2, s_3 respectively, we

Algorithm 10.2 gen_ft(D, R, Br, *sup*, $|L|$)

```
1  for (i = 0 to |L|)
2      for (k = 0 to R)
3          randomly select one group G_k from dimension table k
4          (G_k is a set of transactions corresponding to a rectangle in table k)
5      for (r = 0 to Br)
6          for (l = 0 to sup)
7              for (k = 0 to R)
8                  randomly select a tid from G_k and store it in a tuple t
9              for (k = R to D)
10                 randomly select a tid from dimension table k
11         write the tuple t to FT
12 for (i = 0 to N)
13     for (j = 0 to D)
14         randomly select a tid from dimension table k
```

can select randomly s_1 *tid* from A[0], s_1 *tid* from B[0] and put them in FT. We repeat similar processes for $A[1]B[1]$, $A[2]B[1]$.

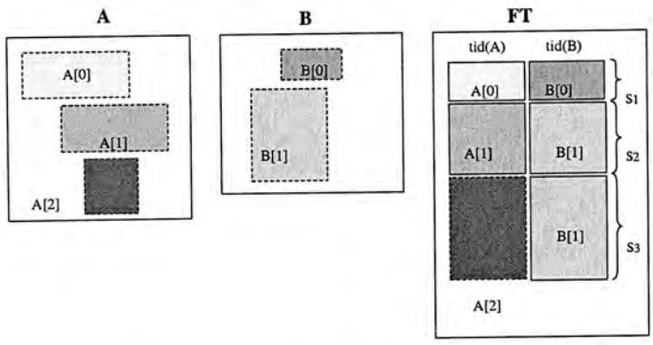


Figure 10.2: Constructing FT

10.2 Experimental Findings

We compare our proposed method with the approach of applying FP-tree algorithm [20] on top of the joined table. We call the result of joining all the tables in the star schema the **joined table**. We assume the joined table T is kept on disk and hence requires I/O time for processing. FP-tree requires two scanning of the table T during the FP-tree itemsets mining. The I/O time required is up to 200 seconds in our experiments. It turns out that the table

join time is not significant compared to the mining time.

All experiments are carried out on a SUN Ultra-Enterprise Generic_106541-18 machine with SunOS 5.7 and 8192MB Main Memory. The average disk I/O rate is 8MB per second. Programs are written in C++. We calculate the total execution time of mining multiple tables as the sum of required CPU and I/O times, and that of mining a large joined table as the CPU and I/O times for joining and FP-tree mining.

In the experiments, we compare the running time of **masl** (our proposed method, implementing *tid_list* as a linked list structure in VTL representation), **masb** (our proposed method, implementing *tid_list* as a fixed-size bitmap and an array of count in VTV representation), and **fpt** (the join-before-mine approach with FP-tree) with different data setting in 3 Dimension Tables **A, B, C** and a *FT*. In most cases, *masb* runs slightly faster than *masl*, but needs about 10 times more memory.

Dataset 1

In the first dataset, we model the situation that items in **A** and **B** are strongly related, such that frequent itemsets contain items across **A** and **B**, while items in **C** are not involved in the frequent itemsets. In such cases, transactions containing frequent itemsets from **A** and **B** can be related to *Br* transactions in **C** randomly. *Br* is set to 100 in all of our experiments reported here. (We have varied the value of *Br* and discovered little change in the performance.) The default values of other parameters used in generating the dataset are :

When we increase the number of attributes, the running time of *fpt* increases steeply, while that of both *masl* and *masb* would increase almost linearly. Running time of FP-tree grows exponentially with the depth of the tree, which is determined by the maximum number of items in a transaction. In this case, performance of our proposed method outperforms FP-tree, especially when the number of items in each transaction is large. (see Figure 10.3(a))

Parameter	Default Value
number of transactions in the joined table	50K
number of attributes in each dimension table	10
size of each attribute domain	10
total number of items in all tables	300
random noise	10%
max. size of potentially maximal frequent itemset	8

Table 10.3: Parameter Table

When the number of transactions in the joined table increases, running time of both methods would increase greatly. *masl* and *masb* are about 10 times faster than *fpt* (see Figure 10.3(b)). We also vary the percentage of random noise being included in the datasets, (see Figure 10.3(c)), both *masl* and *masb* are faster than *fpt*.

Dataset 2

In the second group of dataset, we model the case that **A**, **B**, **C** are all strongly related, so that maximal frequent itemsets always contain items from all of **A**, **B** and **C**. Compared with the previous group of data set, performance of our approach does not vary too much, while the running time of *fpt* is faster in some cases (Figures 10.3(b) and (c)). The reason is that with the strong correlation, there would be less different patterns to be considered and the FP-tree will be smaller. However, we believe that in real life situation such a strong correlation will be rare. Our approach still has advantages when the number of items in each transaction in the joined table is large, which happens when we "join" more tables together. (see Figure 10.3(a))

In real life application, there are often mixtures of relationships across different dimension tables in the database.

Dataset 3

In the third group of dataset, we present data with such mixture. In

particular, 10% of transactions contain frequent itemsets from only **A**, **B**, **C**, respectively, 15% contain frequent itemsets from **AB**, **BC**, **AC** respectively, 10% contain frequent itemsets from **ABC**, and 15% are random noise. We investigate how the running times of *masl* and *fpt* vary against increasing number of items in each transaction, and increasing number of transactions in the joined table.

In Figure 10.4(a), we vary the number of transactions from 20K to 60K, while keeping the number of attributes in each dimension table to be 30 (domain of attributes has size 10). In Figure 10.4(b), we vary the number of attributes in each dimension table from 10 to 60 (domain of attribute has size 10), and keep the number of transactions to be 30K. In this case, running time of *fpt* grows much faster than our approach. This demonstrates the advantage of applying our method. This advantage will be even more significant when we have more dimension tables so that the number of items in the joined table will be large.

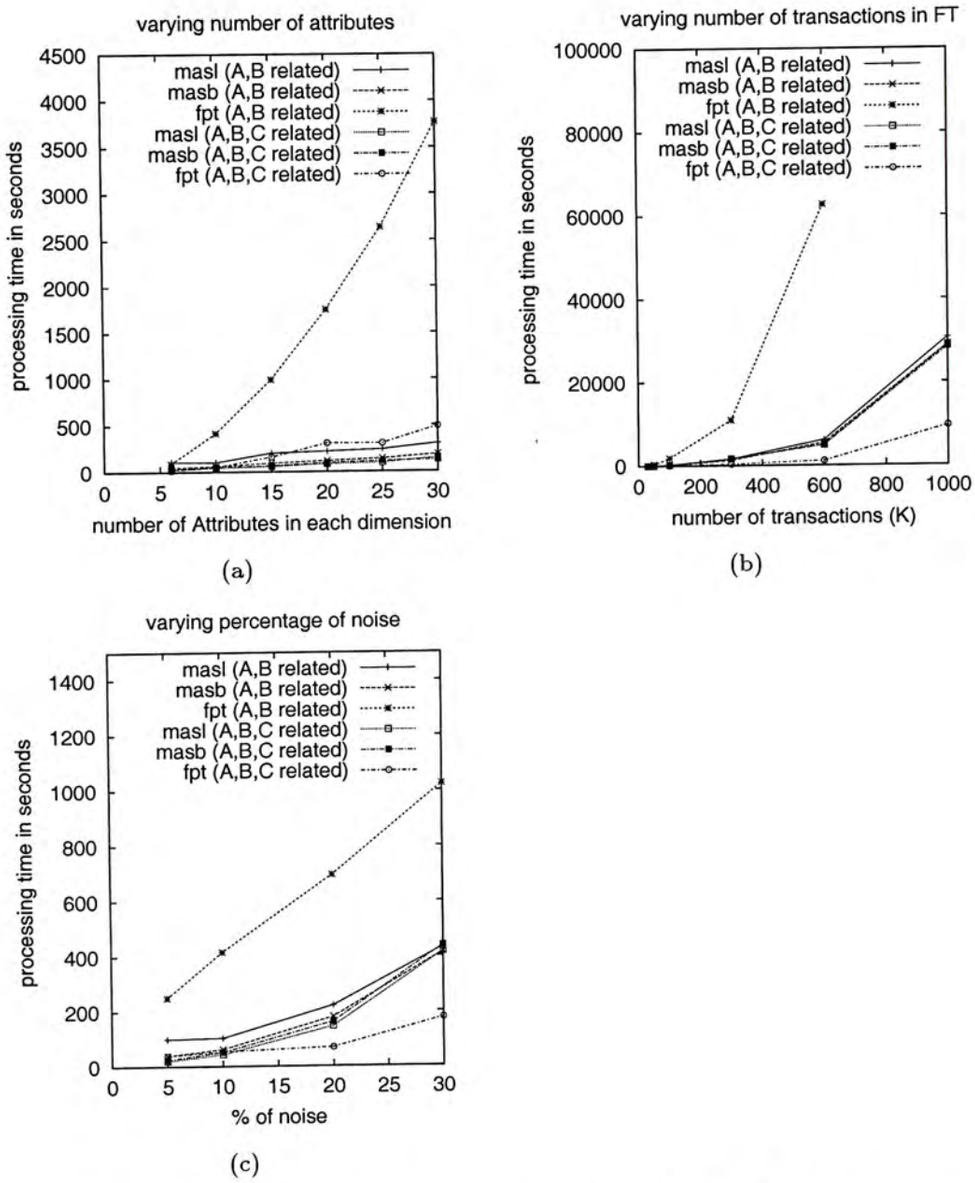
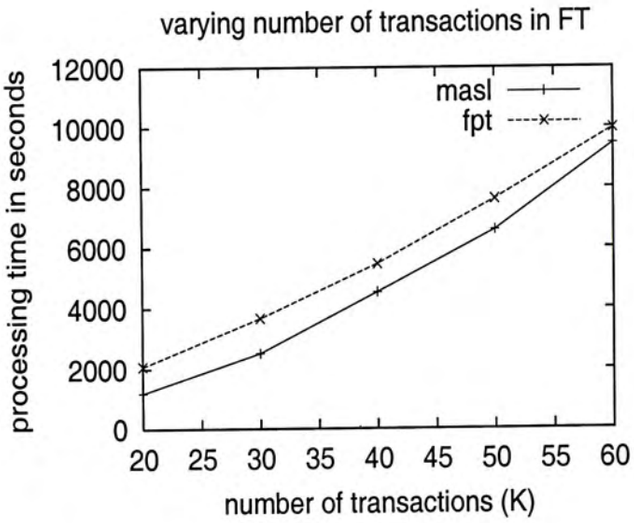
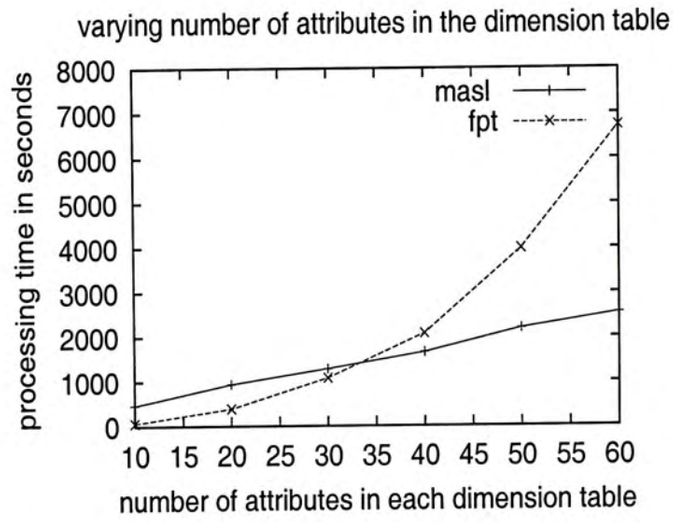


Figure 10.3: Running time for (A,B) related and (A,B,C) related datasets



(a)



(b)

Figure 10.4: Running time for mixture datasets

Chapter 11

Conclusion and Future Works

The problem of mining association rules across multiple tables is studied. Currently, there are few research works on developing algorithms for this problem. We propose methods to combine frequent itemsets mined locally from individual tables to avoid the expensive joining operations and mining from the large resulting table. In particular, we assume the input tables form a **star schema.**

These methods employ the ideas of vertical *tid_list* mining, where frequency of itemsets from different Dimension Tables can be counted by simple *tid_list intersections.* We also make use of the *monotonicity property* of frequent itemsets, which prune unfrequent itemsets at the earlier stage, and thus we do not need to consider any supersets of the pruned itemsets. Various data structures are designed to mine and retrieve itemsets across multiple Dimension Tables efficiently.

Synthetic data generator is used to generate data sets with various parameters. Extensive experiments have be carried out, and it has been shown that in many scenarios, the newly proposed method can greatly outperform a method based on joining all tables even when the naive approach is equipped with a state-of-the-art efficient algorithm.

Our proposed method can be generalized to be applied to a **snowflake** structure, where there is a star structure with a fact table FT, but a Dimension

Table can be replaced by another fact table FT' which is connected to a set of smaller Dimension Tables. We can consider mining across Dimension Tables related by FT' first. We then consider the mined result as from a single derived dimension, and continue to process the star structure with FT. This means that we always mine from the "leaves" of the snowflake.

Suppose there are five Dimension Tables, namely $\mathbf{A_1}$, $\mathbf{A_2}$, **B**, **C**, **D**. $Tid(B), tid(C), tid(D)$ form a candidate key in FT. We can relate **B, C, D** by a new FT', and assign a id g_n for each combination of $tid(B), tid(C), tid(D)$ in FT'. Then we can use g_n to replace the corresponding combination of $tid(B), tid(C), tid(D)$ in the original FT. In this case, **B, C, D** is related by FT', and we can treat them as a single entity and this single entity related with $\mathbf{A_1}, \mathbf{A_2}$ in FT by the key g_n, as shown in Figure 11.1. We mine dimensions **B, C, D** with FT' first. After we have computed the frequent itemsets F^{BCD}, we then consider the result as a single derived dimension, and continue to process the star structure with FT, $\mathbf{A_1}$ and $\mathbf{A_2}$.

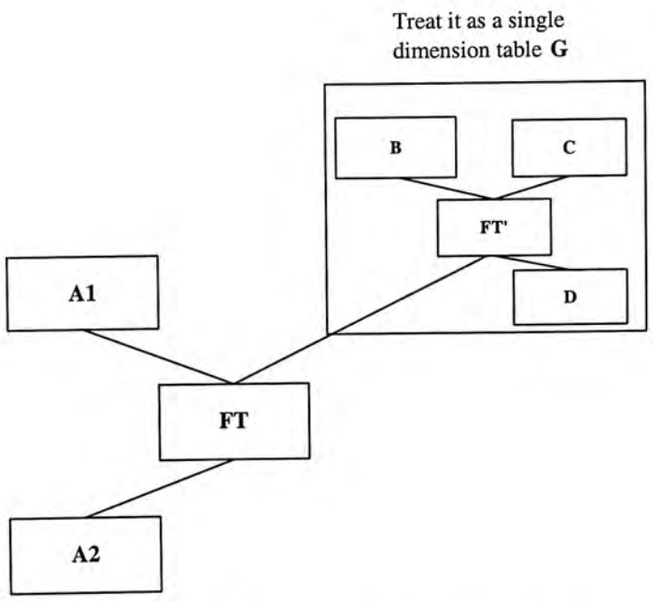


Figure 11.1: Snowflake Structure

Bibliography

[1] IBM software: Database and data management: Intelligent miner family: Overview. In *http://www-3.ibm.com/software/data/iminer/.*

[2] Charu C. Aggarwal, Cecilia Procopiuc, Joel L. Wolf, Philip S. Yu, and Jong Soo Park. Fast algorithms for projected clustering. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 61–72, 1999.

[3] Charu C. Aggarwal and Philip S. Yu. Finding generalized projected clusters in high dimensional space. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 70–81, 2000.

[4] Charu C. Aggarwal and Philip S. Yu. Redefinig clustering for high-dimensional applications. In *IEEE Transactions on knowledge and Data Engineering, Vol. 14, No. 2*, 2000.

[5] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of International Conference of Very Large Data Bases, VLDB*, 1994.

[6] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining application. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 94–105, 1998.

[7] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 207–216, 1993.

[8] Goldstein J. Beyer K., Ramakrishnan R., and Shaft U. When is nearest neighbor meaningful? In *Proceedings of ICDT Conference*, 1999.

[9] Sergey Brin, Rajeev Motwani, and Craig Silverstein. Beyond market baskets: generalizing association rules to correlations. pages 265–276. SIGMOD Record (ACM Special Interest Group on Management of Data), 1997.

[10] K. Chakrabarti and S. Mehrotra. Local dimensionality reduction: A new approach to indexing high dimensional spaces. In *Proccedings of International Conference of Very Large Data Bases, VLDB*, pages 89–100, 2000.

[11] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. In *ACM SIGMOD Record, Vol.26 No.1*, pages 65–74, March 1997.

[12] Chung-Hung Cheng, A. Fu, and Yi Zhang. Entropy-based subspace clustering for numerical data. In *Proceedings of Internationl Conference on Knowledge Discovery and Data Mining (KDD'99)*, pages 84–93, 1999.

[13] Edith Cohen, Mayur Datar, Shinji Fujiwara, Aristides Gionis, Piotr Indyk, Rajeev Motwani, Jeffrey D. Ullman, and Cheng Yang. Finding interesting associations without support pruning. volume 13, pages 64–78, 2001.

[14] M. Ester, H-P. Kriegel, J. Sander, and X.Xu. A density–based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of ACM SIGKDD*, 1996.

[15] M. Ester, H-P. Kriegel, J. Sander, and X.Xu. Density–connected setsand their application for trend detection in spatial database. In *Proceedings of ACM SIGKDD*, 1997.

[16] Brian Everitt. Cluster analysis, second edition. Halsted Heinemann, 1980.

[17] C. Faloutsos and K.-I. Lin. A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1995.

[18] D. Fasulo. An analysis of recent work on clustering algorithms. In *Technical Report UW-CSE-01-03-02, University of Washington*, 1999.

[19] Jiawei Han, Jian Pei, Guozhu Dong, and Ke Wang. Efficient computation of iceberg cubes with complex measures. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 1–12, 2001.

[20] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 1–12, 2000.

[21] John A. Hartigan. Clustering algorithms. Wiley, 1975.

[22] A. Hinneburg, C. C. Aggarwal, and D. A. Keim. What is the nearest neighbor in high dimensinoal spaces? In *Proceedings of International Conference of Very Large Data Bases, VLDB*, pages 506–515, 2000.

[23] Alexander Hinneburg and Daniel A. Keim. Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. In *The VLDB Journal*, pages 506–517, 1999.

[24] Yannis E. Ioannidis and Viswanath Poosala. Balancing histogram optimality and practicality for query result size estimation. In *Proceedings*

of the ACM SIGMOD International Conference on Management of Data, 1990.

[25] A. K. Jain, M. N. Murty, and P.J. Flynn. Data clustering: A review. In *ACM Computing Surveys, Vol.31, No.3*, 1999.

[26] Anil K. Jain and Richard C. Dubes. Algorithms for clustering data. Prentice Hall, 1988.

[27] M. Jambu and M-O. Lebeaux. Cluster analysis and data analysis. North-Holland, 1983.

[28] V.C. Jensen and N. Soparkar. Frequent itemset counting across multiple tables. In *Proceedings of Pacific-Asia Conference on Konwledge Discovery and Data Mining (PAKDD)*, pages 49–61, 2000.

[29] I. T. Jolliffe. Principal component analysis. Springer–Verlag, New York, 1986.

[30] K. Ravi Kanth, D. Agrawal, and A. Singh. Dimensionality reduction for similarity searching in dynamic databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1998.

[31] L. Kaufman and P. Rousseeuw. Finding groups in data: An introduction to cluster analysis. John Wiley and Sons, 1990.

[32] R. Kohavi and D. Sommerfield. Feature subset selection using the wrapper method: Overfitting and dynamic search space topology. In *The First International Conference on Knowledge Discovery and Data Mining*, pages 192–197. AAAI Press, Menlo Park, California, August 1995.

[33] D. Lin and Z.M. Kedemt. Pincer–search: A new algorithm for discovering the maximum frequent set. In *Proccedings of Conference on Extendnig Database Technology (EDBT)*, 1998.

[34] Ankerst M., Breunig M. M., Kriegel H-P., and Sander J. Optics: Ordering points to identify the clustering structure. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 46–49, 1999.

[35] Cecilia M., Michael J, Pankaj K, and T. M. Murlali. A monte carlo algorithm for fast projective clustering. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 2002.

[36] Eric Ka Ka Ng and Ada Wai chee Fu. An efficient algorithm for projected clustering. In *Proceedings of International Conference of Data Engineering (ICDE)*, 2002 (poster paper).

[37] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *20th International Conference on Very Large Data Bases, September 12–15, 1994, Santiago, Chile proceedings*, pages 144–155, 1994.

[38] R. T. Ng, L. V. S. Lakshmanan, J. Han, , and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 13–14, 1998.

[39] J.S. Park, M. S. Chen, and P.S. Yu. An efficient hash based algorithm for mining association rules. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 175–186, 1995.

[40] J. Pei and J. Han. Can we push more constraints into frequent pattern mining? In *Proceedings of ACM SIGKDD*, pages 350–354, 2000.

[41] J. Pei, J. Han, and L. V. S. Lakshmanan. Mining frequent itemsets with convertible constraints. In *Proceedings of International Conference of Data Engineering (ICDE)*, 2001.

[42] Raghu Ramakrishnan and Johannes Gehrke. Database management system, second edition. McGRAW-HILL International Editions, 2000.

[43] Rakesh Agrawal Ramakrishnan Srikent. Mining quantitative association rules in large relational tables. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1–12, 1996.

[44] R. Rastogi and K. Shim. Mining optimized association rules with categorical and numeric attribute. In *Proceedings of Internatinoal Conference on Data Engineering, ICDE*, pages 503–512, 1998.

[45] R. Agrawal S. Sarawagi, S. Thomas. Integrating association rule mining with relational database systems: Alternatives and implications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 343–354, 1998.

[46] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *Proceedings of Inernational Conference of Very Large Data Bases, VLDB*, pages 432–444, 1995.

[47] Pradeep Shenoy, Jayant R. Haritsa, S. Sudarshan, Gaurav Bhalotia, Mayank Bawa, and Devavrat Shah. Turbo-charging vertical mining of large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2000.

[48] B. W. Silverman. Density estimation for statistics and data analysis. Chapman and Hall, 1986.

[49] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *Proccedings of ACM SIGKDD*, pages 67–73, 1997.

[50] Scott D. W. On optimal and data-based histograms. Biometrika, 66:605-610, 1979.

[51] Scott D. W. Multivariate density estimation. Wiley & Sons, 1992.

[52] K. Wang, Y. He, and J. Han. Pusing support constraints into frequent itemset mining. In *Proceedings of Inernational Conference of Very Large Data Bases, VLDB*, 2000.

[53] Wei Wang, Jiong Yang, and Richard Muntz. Sting: A statistical information grid approach to spatial data mining. In *Proceedings of Inernational Conference of Very Large Data Bases, VLDB*, pages 186–195, 1997.

[54] WT. Zhang, R.Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1996.