T.C.
BAHÇEŞEHİR ÜNİVERSİTESİ

FAST AND EFFICIENT FREQUENT ITEMSET DETECTION BY A NOVEL CLUSTERING METHOD

Master of Science Thesis

Hüseyin ARIK

İstanbul, 2011

T.C.
BAHÇEŞEHİR ÜNİVERSİTESİ
The Graduate School of Natural and Applied Sciences
Computer Engineering

FAST AND EFFICIENT FREQUENT ITEMSET DETECTION BY A NOVEL CLUSTERING METHOD

Master of Science Thesis

Hüseyin ARIK

Supervisor: Asst. Prof. Dr. Selim Necdet MİMAROĞLU

İstanbul, 2011

T.C.
BAHÇEŞEHİR ÜNİVERSİTESİ
The Graduate School of Natural and Applied Sciences
Computer Engineering

Title of the Master's Thesis	:	Fast and efficient frequent itemset detection by a novel clustering method
Name/Last Name of the Student	:	Hüseyin Arık
Date of Thesis Defense	:	27 January, 2011

The thesis has been approved by the Graduate School of Natural and Applied Sciences.

Asst. Prof. F. Tunç BOZBURA
Acting Director

This is to certify that we have read this thesis and that we find it fully adequate in scope, quality and content, as a thesis for the degree of Master of Science.

Examining Committee Members:

Asst. Prof. Dr. Selim Necdet MİMAROĞLU :

Asst. Prof. Dr. Tevfik AYTEKİN :

Asst. Prof. Dr. M. Alper TUNGA :

ACKNOWLEDGMENTS

I am very thankful to my family and Dr. Selim Necdet Mimaroğlu who have helped and supported me all along my studies.

Hüseyin Arık
27 January, 2011

ABSTRACT

FAST AND EFFICIENT FREQUENT ITEMSET DETECTION BY A NOVEL CLUSTERING METHOD

ARIK, Hüseyin

Computer Engineering

Supervisor: Asst. Prof. Dr. Selim Necdet Mimaroğlu

January 2011, 83 Pages

Advancements in database and storage technology have led the enterprises and organizations collect large amounts of data in a short time. The methods and the techniques for obtaining information from these datasets should be more advanced compared to the traditional data analysis techniques. Therefore, traditional data analysis techniques have evaluated in efficiency and performance point of view and new methods are developed and introduced.

Frequent item set detection is one of these subjects which are used for dealing large datasets and revealing the information. Frequent item set detection is an important problem having numerous applications in marketing, sales, telecommunication, medicine, biology, and engineering. Depending on the application context, an item may be a measurement, good, service etc. A data set with n distinct items can have 2^n frequent item sets. Therefore, finding all the frequent item sets constitutes a complex information system with exponential time complexity. In this study, we present a novel algorithm for quickly finding preponderant part of the frequent item sets in a data set. This method is a new clustering algorithm designed for detecting frequent item sets. Advantages of clustering for frequent item set detection are twofold. First one is to produce results in considerably short amount of time by reducing the time complexity. Second advantage is to obtain frequent item sets that are similar to each other. Both of these advantages are vital, since in a complex information system generating all the frequent item sets takes a lot of time and there are many redundant frequent item sets.

Keywords: Data Mining, Clustering, Frequent Item Set Detection, Association Rule Mining, Binary Methods

ÖZET

YENİ BİR KÜMELEME METODU İLE HIZLI VE ETKİLİ ŞEKİLDE SIK RASTLANAN ÖĞE SETİ BULUNMASI

ARIK, Hüseyin

Bilgisayar Mühendisliği

Tez Danışmanı: Yrd. Doç. Dr. Selim Necdet Mimaroğlu

Ocak 2011, 83 Sayfa

Veritabanı ve depolama teknolojilerindeki ilerlemeler firmaları ve kurumları kısa zaman içerisinde büyük miktarlarda veri depolamasına olanak sağlar hale getirmiştir. Saklanan bu veri kümelerinin içerisinden şirket ve kurumlara faydalı olacak bilgiye ulaşmak için kullanılacak tekniklerin geleneksel veri analizi metotlarından ileri durumda olması gerekmektedir. Bu yüzden geleneksel veri analizi metotlarının ilerletilerek etkinlik ve performans penceresinden bakıldığında çok daha ileri olan teknikler geliştirilmeye çalışılmıştır. Veri kümelerinin ele alınarak bu kümelerden bilgiyi ortaya çıkarma çalışmalarından birini sık rastlanan öğe kümesi tespiti konusu oluşturmaktadır.

Sık rastlanan öğe kümeleri tespiti pazarlama, satış, telekomünikasyon, tıp, biyoloji ve mühendislik alanlarındaki pek çok uygulamada karşılaşılan bir problemdir. Uygulama içeriğine bağlı olarak bir öğe ölçü, ürün, servis vb. olabilir. Birbirinden farklı n adet öğe içeren bir veri kümesi, 2^n adet sık rastlanan öğe kümesine sahiptir. Bu nedenle tüm sık rastlanan öğe kümelerini bulmak üssel zaman karmaşıklığı içeren karmaşık bir bilgi sistemi oluşturur. Bu çalışmada, bir veri kümesi içerisinde sık rastlanan öğe kümelerinin önemli bir kısmının hızlı bir şekilde bulunması için yeni bir algoritma tanıtmaktayız. Bu metot sık rastlanan öğe kümelerini tespit etmek için tasarlanmış yeni bir kümeleme algoritmasıdır. Sık rastlanan öğe kümeleri tespiti için kümeleme yapmanın iki avantajı vardır. İlki, zaman karmaşıklığını azaltarak oldukça kısa bir sürede sonuçlarının elde edilmesidir. İkincisi ise birbirine benzer sık rastlanan öğe kümeleri elde etmektir. Karmaşık bilgi sistemi içerisinde sık rastlanan öğe kümelerinin üretilmesi çok uzun zaman alması ve gereğinden fazla sık rastlanan öğe kümelerinin bulunması nedeniyle bu avantajların her ikisi de çok önemlidir.

Anahtar Kelimeler: Veri Madenciliği, Kümeleme, Sık Rastlanan Öğe Kümeleri Tespiti, Birleşme Kuralları Madenciliği, İkili Değer Metotları

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

This chapter provides information on data mining concepts.

1.1 INTRODUCTION TO DATA MINING

Data Mining is an important subject in business and scientific area because of large availability of huge amounts of data and expanding need for transforming such data into practical information and knowledge. It is possible to use the acquired information and knowledge in all kinds of applications from market basket analysis, fraud detection, and customer retention, to production control and science exploration.

Data storage and database systems are assessed throughout the years and these assessments resulted in various periods in database technology and data mining as shown in Figure 1.1. The aim of all of these assessments is an efficient mechanism for data storage, data retrieval, also transaction processing. A new method using the traditional data analysis basis uses advanced algorithms for processing large amounts of data. While the data and database systems are being assessed, firms, establishments, government authorities have created their own data storage systems. This advancement in 1960s was shifting from basic file processing to complex and powerful database systems. As the researches became developments in database systems, hierarchical and network database systems became relational database systems, data modeling tools and indexing accessing methods. From the user's point of view, these years resulted in flexible and eligible data access skills with query languages, user interfaces, optimized query processing and transaction management (Han and Kamber 2005).

Moreover, domain specific data modeling and database systems came to light as spatial, active, stream and scientific databases. The increasing capability of distributing and sharing the data and the major effect of www plays a major role on broadening the utilization of information technologies. From the hardware point of view, the capability of collecting and storing data in durable and powerful environment made the stakeholders to build big storage spaces for data analysis and information gathering throughout the decades. Thus, all of these led the administrative decision makers to gather their data in a joint design at a single site in order to establish own Data Warehouses (Han and Kamber 2005).

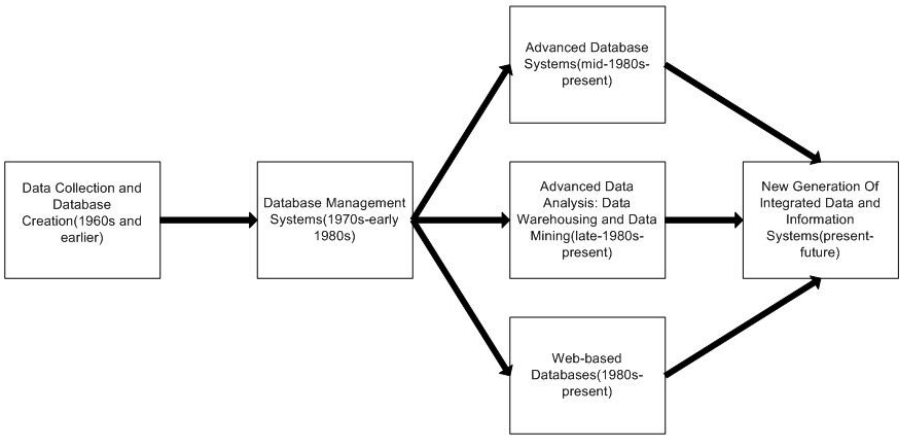


Figure 1.1: The evolution of database system technology
Source: Han and Kamber (2005)

Even though OLAP tools support multidimensional analysis and decision making, there is a need for additional data analysis tools, such as classification of data, clustering and characterization of data changes over time, in order to conduct thorough analysis. Data analysis becomes a challenging task due to the change in the data over time and the data surplus in addition to the databases and data storages such as www and spatial data. Therefore, due to powerful tools not being available in order to analyze the rapidly expanding data storages, these problems make such data sets untrusted nodes for analyzing data and retrieving information. The analysis has no wholly visited data in order to take into account of every single point for being able to valuable for the result. As a result, it is possible to conduct scientific subjects and medical researches using data mining methods in order to acquire significant data patterns and perform data analysis that is able to support business. To give a solid example regarding this issue, consider a major market chain and the data that can be collected from the shoppers in a short period of time. It is possible that this data shall become very large in time. It is possible to use data mining methods on these data in order to study shopper profiles, targeted marketing, workflow management, store design and detection of fraud. It is possible to answer a number of business questions by using these methods, e.g. “Who are the most profitable customers?”, “What products can be cross-sold or sold-up?” Another good example is the NASA’s satellite systems that supply huge amount of data on constantly monitored domains. For example: land surfaces, oceans and the atmosphere.

The reason behind the traditional data set analysis methods not being suitable for managing such large data is that, data mining methods are able to answer scientists' questions; "What is the relationship between the frequency and intensity of ecosystem disturbances such as droughts and hurricanes to global warming?", "How is land surface precipitation and temperature affected by ocean surface temperature?" (Han and Kamber 2005).

1.2 WHAT IS DATA MINING

Data mining is extracting or obtaining beneficial information from large amount of data. That is to say; acquiring the knowledge from rich data but poor information systems by using complex methods. It is also possible to describe the data mining as the process of finding useful patterns from big data storages in moderate periods of time automatically.

Figure 1.2: Data mining process
Source: Tan et al. (2005)

In the preprocessing stage, selecting the features, reducing the dimensionality, normalization and data subsetting are performed with the objective of transforming raw data into a proper format to be analyzed later on. Data mining is the stage which contributes knowledge extraction from the data. The process of integrating the result of data mining into the decision support systems is the postprocessing stage. And this stage ensures incorporating only valid and useful results into the decision support system.

1.3 CHALLENGES IN DATA MINING

There are many challenges that we may encounter in case we use traditional data analysis techniques and it is possible that we may be motivated for the development of data mining. Below are the challenges that we may encounter;

1. Scalability: Generally we deal with large amounts of data such as gigabytes, terabytes etc. during data analysis. Using traditional data analysis techniques make retrieving information from these data very difficult and time consuming. However, most of the data mining algorithms have special search techniques in handling exponential search problems; they also implement novel data structures in order to access separate records in an efficient way. For instance, with big data sets which does not fit into memory is handled by scalable algorithms.

2. High Dimensionality: As the science and the technology develop quickly, the tools for supplying data generate more data as compared to before. Therefore, such data are generally the varying kind that they supply new features of events. For example, each discovery of a new gene is also a new dimension in bioinformatics. The traditional data analysis methods that were developed for low-dimensional data generally will not work properly for such high dimensional data.

3. Heterogeneous and Complex Data: Traditional data analysis techniques were developed commonly for the same kind of data and category. However, as the industry, sciences and technology developed, the need for integrating most data types has become inevitable. In doing so, the traditional data analysis methods became unsatisfactory in terms of performance and usability.

4. Data Ownership and Distribution: The opportunity to acquire data from various locations emerged the chance to collect more data for analysis as the network technology and the distributed systems are developed. This also emerged the concept of distributed data mining methods.

5. Non-Traditional Analysis: The basis of the traditional statistical approach is the hypothesize-and-test paradigm. A hypothesis is suggested, an experiment is designed to collect the data and the data is analyzed according to such hypothesis. This process is greatly labor-intensive and consumes time. The need to make the process of generating the hypothesis automatic and evaluation provided motivation for the data mining algorithms to use opportunistic data samples and non-traditional data types as well as data distributions.

1.4 DATA MINING TASKS

Data mining is conducted for various purposes which are based on the needs and perspectives. Such needs directly influence the data mining process in order to acquire the most favorable results for the purpose. It is possible to view the data mining tasks in two primary groups; *Predictive Tasks and Descriptive Tasks.*

In Predictive Tasks; the objective is to predict the values of a specific attribute based on the values of other attributes. While the attributes that are used for making the prediction are known as the explanatory or independent variables, the attribute which will be predicted is typically known as the target or dependent variable. There are two types of predictive modeling tasks: *classification* , this task is used for discrete target variables and *regression* is used for continuous target variables (Tan et al. 2005). For instance, an application detecting spam e-mails from senders is able to predict the type of the mail as valid or spam. Since the target variable is binary-value, this prediction type is a classification task. In the event that the target variable is continuous-valued like the temperature values, this prediction type is named regression. For instance, in order to predict the weather condition for a given time period in the future, the algorithm must assess the retroactive temperature values.

The objective of *In Descriptive Tasks* is to derive patterns (correlations, trends, clusters, trajectories and anomalies) that summarize the underlying relationships of data. Descriptive data mining tasks are generally exploratory in nature and frequently require post-processing techniques to validate and explain the results (Tan et al. 2005). Association analysis is used to investigate the patterns in data that the items are frequently found together. For instance, it is possible to use this analysis in order to reveal the purchasing habits of the shoppers in a supermarket. It is possible to analyze the purchased item data and find the items frequently bought together in the data set. The example given below shows the technique and the task. Cluster analysis discovers data groups that have similarities in terms of structure. For instance, in bioinformatics, in order to build groups of genes with related expression patterns that generally contain related proteins, clustering is used. In order to find the outliers, which are excluded members whose characteristics are remarkably different from other data, Anomaly Detection task is used. Let us assume that an insurance company draws a number of insurances in a day. As per the data analysis and statistics, there are the ranges and limits of the company.

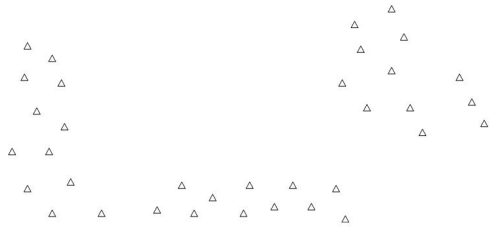
When a new insurance is received by the system, such insurance is compared against the company's ranges and limits and if necessary the administrators or other authorized personnel are warned. As the examples show the below, the objective of the data mining task is based on the requirements and the conditions.

2. CLUSTERING

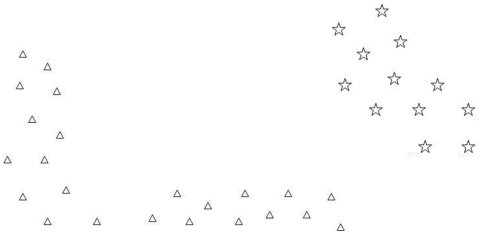
Clustering is the task of grouping the data into classes or clusters that in each group or cluster, objects have significant similarities with other objects in the same cluster and have significant dissimilarities with other objects in different groups or clusters. The similarities and dissimilarities of the objects are calculated by the distance metrics. It is also possible to say that clustering is the starting point of other purposes, such as summarization of data. Clustering plays a crucial role in a broad range of fields such as data mining, psychology, biology, statistics, machine learning and patterns recognition. In Figure 2.1, clustering is observed from initial points to finalized clusters.

2.1 WHAT IS CLUSTER ANALYSIS?

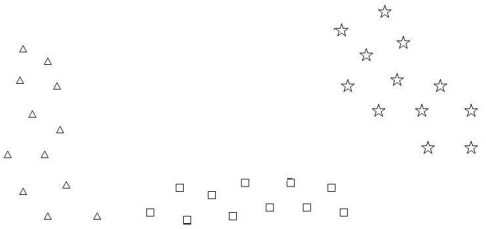
Cluster analysis groups the data objects as per their information in the data which describes them. The objective is to group objects that are similar to one another and different from the objects in other clusters. The clusters created shall be better when these similarities and distinctions are more solid. It is possible to say that cluster analysis is considered as a form of classification where each object has its class (cluster) tags. While the data definition is based on the nature of the data, such tags are generated from the data itself. This kind of classification is the progress in the reverse direction: First partition the set of data into groups depending on data similarity and assign tags to object groups which are less than the number of classifier of each model group. Segmentation and partitioning are occasionally used as synonyms for clustering; these terms are typically used for approaches outside the traditional bounds of cluster analysis. For instance, the term partitioning is frequently used with respect to the methods that divide graphs into subgraphs and that are not well connected to clustering. Segmentation generally refers to the division of date into groups using simple methods. For instance, it is possible to divide the people into groups as per their earnings.



(a) **Initial points**



(b) **Two clusters**



(c) **Final clusters**

Figure 2.1: Exploring and separating clusters

2.2 APPLICATION OF CLUSTERING

Below you may find the application areas of clustering;

Market Research: Cluster analysis is extensively used in market research when studying multi-variable data from surveys and test panels. Cluster analysis is used by market researchers in order to group the general population of consumers into market segments and to improve the relationship between various consumer groups and potential consumers.

Medicine: It is possible to use cluster analysis in medical imaging, for example PET (Positron Emission Tomography) scans, in order to differentiate between different types of tissue and blood in a three dimensional image. Actual position is not important in this application, however, the voxel intensity is deemed as a vector, with a dimension for each image that was taken in a period of time. For instance, this method allows accurate measurement of a radioactive tracer's rate that is sent to the area of interest, without requiring additional arterial blood sampling, which is one of the most common intrusive techniques of today.

Biology: Biologist created a taxonomy of all living things (hierarchical classification): kingdom, phylum, class order, family, genus and species. In this way, initial cluster analysis study investigated to create a discipline of mathematical taxonomy that was able to find such classification structures automatically. Lately, biologists have applied clustering for analyzing huge amounts of genetic information that are available today. For instance, clustering was being used in order to find genes groups with similar functions.

Educational Research: The clustering data can be students, parents, sex or test score in educational research analysis. Clustering is a significant technique in understanding and the utility of cluster in educational research. It is possible to discover the data exploration, cluster confirmation and hypothesis testing in educational research using the cluster analysis. Data exploration is used when there is little information about the schools or students that will be grouped together. The aim of this method is to discover any meaningful clusters of units depending on the measures on a set of response variables. Cluster confirmation is used in to confirm the previously reported cluster results. And hypothesis testing is used to arrange the structure of the cluster.

Data Mining: Most of the data mining applications involve dividing data items into relevant subsets; aforementioned marketing applications are representative examples. Division of documents, such as www pages, into genres is another typical application.

Software evolution: As clustering helps to reduce legacy properties in code by reforming the dispersed functionality, it is useful in the development of the software. It is a form of restructuring and therefore it is a direct preventative maintenance method.

2.3 TYPES OF CLUSTERING

2.3.1 Hierarchical Clustering

Hierarchical clustering creates a cluster hierarchy that may be represented as a tree structure, namely a dendrogram. The root of the tree has a single cluster containing all observations, and the leaves equal to the individual observations. Hierarchical clustering algorithms are typically either agglomerative, in which an algorithm emerges at the leaves, and then joins clusters together; or divisive, in which an algorithm emerges at the root and repetitively divides the clusters. It is possible to use any valid metric as a measure of similarity between observations pairs. Choosing which clusters shall be joint or divided is determined by a linkage criterion, which is a function of the pairwise distances between observations.

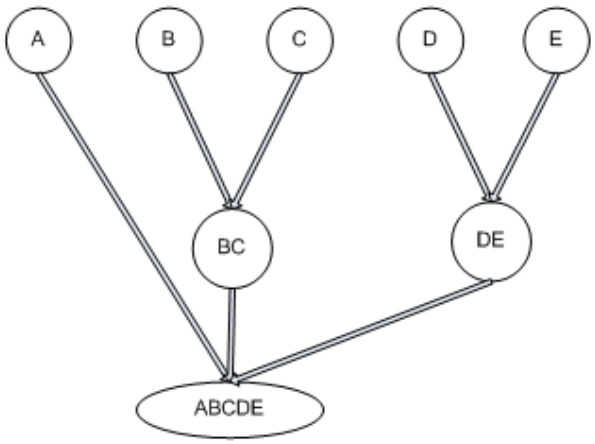


Figure 2.2: Hierarchical clustering

2.3.2 Partitional Clustering

Partitioning methods: In a database of n objects or data tuples, a partitioning technique builds k partitions of the data, in these partitions each partition denotes a cluster and $k \leq n$. In other words, the technique classifies the data into groups of k, and these groups meet the following requirements: (1) there must minimum one object in each group, and (2) each object must belong to one group only. Notice that it is possible to expand the latter in a number of fuzzy partitioning methods. Given k, the amount of partitions to be built, a partitioning technique creates an initial partitioning. Then the technique uses a recursive relocation method and attempts to develop the partitioning by moving objects from one group to another. The general criterion of a good partitioning is that, while the objects from different clusters are "distant from each other" or very different, the objects found in the same cluster are "close" or related to each other. There are different kinds of other criteria for judging the partition quality. Exhaustive enumeration of all of the possible partitions would be required in order to reach the global optimality in partitioning-based clustering. Rather than this, better part of the applications adopt a single popular heuristic method from a few available, such as (1) the $k-$ means algorithm, in which each cluster is denoted by the average value of the objects in the cluster, and (2) the $k-$ medoids algorithm, in which each cluster is denoted by one of the objects found next to the center of the cluster. These heuristic clustering methods work well when used to find spherical-shaped clusters in small to medium-sized databases. Partitioning-based methods need to be extended in order to find clusters with complex shapes and for clustering very large data sets (Han and Kamber 2005).

Density-based methods: Most partitioning methods cluster objects based on the distance between objects. It is possible for these methods to find spherical-shaped clusters only and face difficulty when discovering arbitrarily shaped clusters. Other clustering methods are also developed based on this density notion. The general idea of such methods is to continue growing the given cluster provided that the density (number of objects or data points) in the "neighborhood" goes above some limit; in other words, for each data point in a given cluster, the neighborhood of a given radius must contain at least a minimum number of points. It is possible to use such method in order to filter out noise (outliers) and discover arbitrarily shaped clusters. The typical density-based methods that grow clusters as per a density-based connectivity analysis are DBSCAN and its extension, OPTICS. DENCLUE is a method that clusters objects based on the analysis of the value distributions of density functions (Han and Kamber 2005).

Grid-based methods: A natural way to define the density of a grid cell (or a more typically shaped region) is as the number of points divided by the volume of the region. That is, density is the number of points per space quantity, without regard to the dimensionality of that space. Specific, low-dimensional density examples are the number of road signs per mile (one dimension), the number of eagles per square kilometer of habitat (two dimensions), and the number of molecules of a gas per cubic centimeter (three dimensions). As mentioned, however, using grid cells with the same volume and by this way the number of points per cell being a direct measure of the cell's density is a common approach (Tan et al. 2005).

Model-based methods: Model-based methods make an assumption of a model for every single cluster and find the best fit of the data for the model in question. It is possible to locate clusters by constructing a density function that reflects the spatial distribution of the data points by using a model-based algorithm. This approach also leads to a way of determining the number of clusters based on standard statistics automatically, taking "noise" or outliers into account and by this way yielding robust clustering methods. An algorithm performing expectation-maximization analysis based on statistical modeling is called EM. A conceptual learning algorithm performing probability analysis and taking concepts as a model for clusters is called COBWEB. A neural network-based algorithm clustering by mapping high dimensional data into a 2-D or 3-D feature map, which is also useful for data visualization is called SOM (namely, self-organizing feature map). The choice of clustering algorithm is based on both on the available data type as well as on the application's particular purpose. In the event that cluster analysis is used as a descriptive or exploratory tool, several algorithms can be tried on the same data in order to see what the data may reveal.

Some of the clustering algorithms integrate the notions of several clustering methods, and thus, classifying a specific algorithm as uniquely belonging to a single clustering method category is sometimes difficult. Moreover, it is possible that some applications may have clustering criteria that require the integration of a number of clustering techniques. Besides the aforementioned clustering method categories, there are two classes of clustering tasks that require special attention. One of them is clustering high-dimensional data, and the other one is constraint-based clustering. Since many applications require the analysis of objects containing a large number of features or dimensions, clustering high-dimensional data is especially a crucial task in cluster analysis. For instance, it is possible that text documents contain thousands of terms or keywords as features, and

DNA microarray data may provide information on the expression levels of thousands of genes under hundreds of conditions. Due to the curse of dimensionality, clustering high-dimensional data is difficult. Many dimensions may be irrelevant. As the number of dimensions increases, the data become increasingly scattered and thus, the distance measurement between pairs of points loses its meaning and the mean density of points anywhere in the data is expected to be low. So, another clustering methodology for high-dimensional data must be developed. CLIQUE and PROCLUS are two influential subspace clustering methods searching the clusters in subspaces (or subsets of dimensions) of the data, instead of over the whole data area. One of other clustering methodologies, the frequent pattern-based clustering extracts distinct frequent patterns among subsets of dimensions occurring frequently. This methodology uses these patterns for grouping the objects and generating clusters with meaning. For instance, pCluster is a frequent pattern-based clustering that groups objects based on their pattern similarity (Han and Kamber 2005).

Constraint-based clustering is a clustering approach that conducts clustering by incorporating constraints set by the user or incorporating application-oriented constraints. A constraint represents the expectation of a user or describes properties of the desired clustering results, additionally; it constitutes an effective means to communicate with the process of clustering. It is possible for either a user to specify various kinds of constraints, or as per application requirements (Han and Kamber 2005).

2.4 WIDELY USED CLUSTERING ALGORITHMS

2.4.1 k-Means

$k-$means is one of the earliest and most broadly used clustering algorithm. This algorithm focuses to partition n observations to k clusters in which each observation belongs to the cluster having the nearest mean. $k-$means tries to find the center of clusters in the data and n the recursive refinement approach utilized by the algorithm as well. k-means takes the number of clusters to be observed, k, as the initial input parameter. k objects are selected randomly from the objects in the data set as the initial center or mean. Later, each point in the data set is allotted to a closest centroid. And then, by selecting the centroids randomly and assigning the points to the centroids, initial clusters are created.

Following the initial assignment, each centroid of the cluster is updated as per the points in the cluster. This process is repeated until every point remains stable and does not change its cluster. $k-$means algorithm is indicated in the following clustering method.

Input: k: The number of clusters, D: Data set containing n objects(points)
Output: k Clusters
Randomly select k initial cluster centroids from the D ;
while *objects change cluster* **do**
    assign each object to the closest centroid ;
    recalculate the cluster centroids;
end

Algorithm 1: k-means Algorithm

The algorithm is illustrated in the following figure;

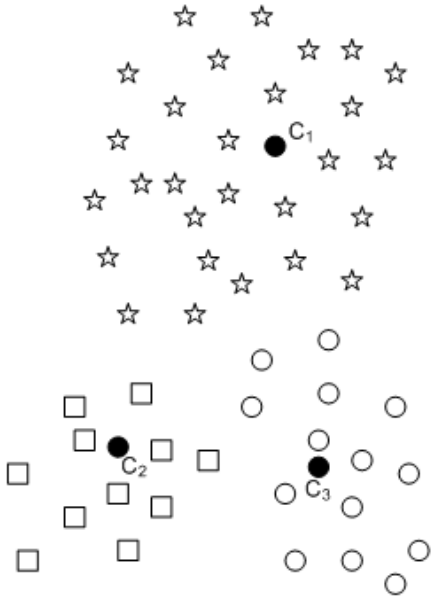


Figure 2.3: k**-means algorithm finding clusters**

Centroids of the clusters are selected randomly at the beginning of the algorithm and the points are assigned to the closest centroids. In this way, the algorithm creates the initial clusters. Following the repetitions over the algorithm, the cluster centers are re-determined and the objects are re-assigned to the centroids. Once all the objects become fixed as per the centroids, the final clusters are observed and the algorithm ends.

Applications of the k-means algorithm

The $k-$means clustering algorithm is typically used in computer vision as an image segmentation form. Segmentation results are used to help detection of the border and object recognition. In this sense, the standard Euclidean distance is generally insufficient in cluster formations. Rather than this, a weighted distance measure utilizing pixel coordinates, RGB pixel color and/or intensity, and image texture are generally used.

2.4.2 DBSCAN

The $DBSCAN$ algorithm is introduced by Ester, et al first. [Ester1996], and depends on a density-based cluster idea. Since this algorithm finds a number of clusters starting from the anticipated density distribution of relevant nodes, it is a density-based clustering algorithm. $DBSCAN$ grows clusters based on a density-based connectivity analysis. It identifies the clusters by checking the density of points. While the regions with a high density of points represent the existence of clusters, the regions with a low density of points denotes clusters of noise or clusters of outliers. $DBSCAN$ is one of the most general clustering algorithms and most referred to in scientific literature as well. This algorithm is especially appropriate for dealing with huge datasets and noise, as well as is capable of identifying different sized and shaped clusters.

DBSCAN Algorithm

This algorithm classifies the points subject to the data set into three groups as per their positions as points interior of the dense regions (core points), on the edge of the dense regions (border points), outside of the dense regions (noise or background points). Concepts of the core, border and noise points are illustrated in the following Figure 2.4. Below are the descriptions of the concepts and definitions;

The neighborhood within a radius ε of a given object is called the ε-neighborhood of the object. An object p is density-reachable from object q with respect to ε and *MinPts* in a set of objects, D, if there is a chain of objects $p_1, \ldots, p_n$, where $p_1 = q$ and $p_1 = p$ such that p_{i+1} is directly density-reachable from p_i with respect to ε and MinPts, for $1 \leq$ i $\leq$ n, p_i ε D.

An object p is density-connected to object q with respect to ε and *MinPts* in a set of objects, D, if there is an object o ε D such such that both p and q are density-reachable from o with respect to ε and *MinPts*.

Core Points: These points are located inside a density-based cluster. A point is a core point provided that the number of points within a given neighborhood around the point as determined by the distance function and a distance parameter specified by the user, *Eps*, exceeds another specific limit parameter specified by the user, *MinPts*. In Figure 2.4, point p refers to a core point, for the indicated *radius(Eps)* if *MinPts* ≤ 3.

Border Points: A border point is not a core point, however, located in the neighborhood of a core point. In Figure 2.4, point q is a border point. A border point can be located in the neighborhoods of multiple core points.

Noise Points: A noise point is any point that is not a core point or a border point. In Figure 2.4, point n is a noise point. m and p of the labeled points are core objects since each point is in an ε-neighborhood containing at least three points. It is possible to reach q directly in terms of density from m. It is possible to reach m directly in terms of density from p and vice versa. Since it is possible to reach q directly in terms of density from m and to reach m directly in terms of density from p, it is not possible to reach q (indirectly) in terms of density from p. However, it is not possible to reach p in terms of density from q since q is not a core object.

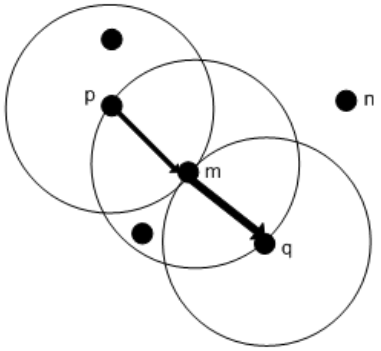


Figure 2.4: Core points, Border points, Outliers in DBSCAN algorithm
Source: Han and Kamber (2005)

Input: K: the number of clusters desired, D: data set containing n objects(points)
Output: K clusters
Label all points as core, border or noise points. ;
while *there are points to assign to clusters* **do**
    Create cluster with core points ;
    Collect directly density-reachable objects from core objects into clusters;
end

Algorithm 2: The DBCAN Algorithm

$DBSCAN$ requires two parameters: ε *(eps)* and the minimum number of points required to form a cluster *(minPts)*. It starts with a unvisited arbitrary starting point. This point's ε-neighborhood is obtained, and in the event that such point contains many points sufficiently, a cluster is started. If not, the point is tagged as noise. Notice that it is possible

to find this point in a sufficiently sized ε-environment of another point and therefore, became a part of a cluster. In the event that a point is found to be part of a cluster, its ε-neighborhood is a part of that cluster as well. Therefore, all points that are found within the ε-neighborhood are added, as is their own ε-neighborhood. This process keeps on going until the cluster is completely found. Afterwards, a new unvisited point is obtained and processed, leading to the discovery of a further cluster or noise.

3. FREQUENT ITEMSET DETECTION

3.1 PROBLEM DEFINITION

Frequent itemset detection is one of the most essantial task in Data Mining. Frequent itemset detection aims to find frequent patterns such as association rules, itemsets, correlations, sequences, clusters. Frequent patterns reside together frequently in a data set. For example, bread and milk appear frequently together in a transaction data set according to market basket analysis. These kind of itemsets are called frequent itemsets. For a concrete example, examine the following table that is commonly known as market basket transactions. Each row in the table represents a transaction which has a unique identifier as TID and a set of items purchased by the customer. These transactions are analyzed by the retailers to find out the purchasing behaviors of the customers and use the results in marketing promotions, inventory management and customer relationship management with the aim of increasing the profit of the enterprise.

Table 3.1: Sample market basket transactions

TID	Items
1	{Bread}
2	{Bread, Milk}
3	{Beer, Milk}
4	{Bread, Milk, Coke}
5	{Diaper}
6	{Bread, Beer, Milk, Diaper}

From the Table 3.1, it can be extracted that there is a strong relationship between bread and milk. Many customers who buy bread also buy milk according to the transactions.

$$\{Bread\} \longrightarrow \{Milk\}$$

To evaluate this information, retailers can lead to increased sales by helping retailers do selective marketing and plan their shelf space, they can use this kind of rules to help them identify new opportunities for cross-selling their products to the customers.
Besides market basket data, association analysis is also applicable to other application domains such as bioinformatics, medical diagnosis, Web mining, and scientific data analysis. There are two key issues that need to be addressed when applying association analysis to

market basket data. First, discovering patterns from a large transaction data set can be computationally expensive. Second, sum of the discovered patterns are potentially spurious because they may happen simply by chance (Tan et al. 2005).
Since its introduction in 1993 by Agrawal et al. (1993), the frequent set mining problem has received a great deal of attention. Hundreds of research papers have been published, presenting new algorithms or improvements to solve this mining problem more efficiently. In this chapter, we will explain the conventional frequent itemset detection algorithms in the following sections.

3.2 THE APRIORI PRINCIPLE

Apriori algorithm is introduced by R. Agrawal and R. Srikant in 1994 for mining frequent itemsets for Boolean associates. Apriori algorithm is name by the fact that it uses the prior knowledge along the algorithm. Apriori applies a method known as level-wise search where k-items are used to explore $(k+1)$-itemsets. Apriori algorithm firstly finds frequent 1-itemset which are the singleton items satisfying minimum support count in the database. This resulting set is denoted L_1. For finding L_2, L_1 is used as prior knowledge and after finding L_2, it is used for finding L_3 and so on. This process ends when mo more k-frequent itemsets can be found. For finding each L_k, a full scan of database is required. To lower the search space and accelerate the algorithm, an important property called Apriori property is presented.

Apriori property: All nonempty subsets of a frequent itemset must also be frequent. The Apriori property is based on the following observation. By definition, if an itemset *I* does not satisfy the minimum support threshold, *min_sup*, then *I* is not frequent; that is, *support(I)* $<$ *min_sup*. If an item *A* is added to the itemset *I*, then the resulting itemset (i.e., $I \cup A$) cannot occur more frequently than *I*. Therefore, $I \cup A$ is not frequent either; that is, *support(I)* $<$ *min_sup* (Han and Kamber 2005). This property brings a special category of properties called antimonotone. This property depicts that if a set can not pass the test, satisfying minimum support count in this case, all the superset of this set fails the test also. To explain how is the property is used let us see how L_{k-1} is used to find L_k. A two-step process is followed, consisting of join an prune actions.

The joining step: To find L_k, a set of candidate k-itemsets is generated by joining L_{k-1} with itself. This set of candidates is denoted C_k. Let l_1 and l_2 be itemsets in L_{k-1}. The notation $l_i[j]$ refers to the *j*th item in l_i (e.g., $l_i[k-2]$ refers to the second to the last item in l_i). By convention, Apriori assumes that items within a transaction or itemset are sorted in lexicographic order. For the (k-1)-itemset, l_i, this means that the items are sorted such that $l_i[1] < l_i[2] < \cdots < l_i[k-1]$. The join, $L_{k-1} \bowtie L_{k-1}$, is performed, where members of L_{k-1} are joinable if their first *(k-2)* items are in common. That is, members l_1 and l_2 of L_{k-1} are joined if $(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2] \wedge \cdots \wedge (l_1[k-2] = l_2[k-2] \wedge (l_1[k-1] < l_2[k-1])$. The condition $l_1[k-1] < l_2[k-1]$ simply ensures that no duplicates are generated. The resulting itemset formed by joining l_1 and l_2 is $l_1[1]$, $l_1[2]$, $\cdots$, $l_1[k-2]$, $l_1[k-1]$, $l_2[k-1]$ (Han and Kamber 2005).

The pruning step: C_k is a superset of L_k, that is, its members may or may not be frequent, but all of the frequent *k*-itemsets are included in C_k. A scan of the database to determine the count of each candidate in C_k would result in the determination of L_k (i.e., all candidates having a count no less than the minimum support count are frequent by definition, and therefore belong to L_k). C_k, however, can be huge, and so this could involve heavy computation. To reduce the size of C_k, the Apriori property is used as follows. Any (*k*-1)-itemset that is not frequent cannot be a subset of a frequent *k*-itemset. Hence, if any (*k*-1)-subset of a candidate *k*-itemset is not in L_{k-1}, then the candidate cannot be frequent either and so can be removed from C_k. This subset testing can be done quickly by maintaining a hash tree of all frequent itemsets (Han and Kamber 2005).

Input: D : a database of transactions, min_sup : the minimum support count threshold.

Output: L : frequent itemsets in *D*

L_1 = find frequent 1-itemsets(*D*);

for $(k = 2; L_{k-1} \neq \emptyset; k++)$ **do**

  C_k = **apriori_ gen**(L_{k-1});

  foreach *transaction t* $\in \mathcal{D}$ **do**

    C_t = subset(C_k, t); // get the subsets of t that are candidates

    foreach *candidate* $c \in \mathcal{C}_t$ **do**

      c.count++;

    end

  end

  $\mathcal{L}_k = \{c \in \mathcal{C}_k | c.count \geq min\_sup\}$;

end

return L= $\cup_k L_k$

procedure apriori_gen(L_{k-1}:frequent (k-1)-itemsets)

foreach *itemset* $l_1 \in L_{k-1}$ **do**

  foreach *itemset* $l_2 \in L_{k-1}$ **do**

    if *(*$l_1[1]$ = $l_2[1]$*)* $\wedge$ *(*$l_1[2]$ = $l_2[2] \wedge \cdots \wedge$ *(*$l_1[k-2]$ = $l_2[k-2] \wedge$ *(*$l_1[k-1] <$ $l_2[k-1]$*)* **then**

      c = l1 $\bowtie$ l2;

      if *has_infrequent_subset(c,*L_{k-1}*)* **then**

        delete c;

      else

        add c to C_k**;**

      end

    end

  end

end

return L= $\cup_k L_k$

has_infrequent_subset(c: candidate *k*-itemset; L_{k-1}**: frequent (*k*-1)-itemsets)**

foreach *(k-1)-subset s of c* **do**

  if *s* $\notin$ L_{k-1} **then**

    return TRUE;

  end

  return FALSE;

end

Algorithm 3: Apriori algorithm

The algorithm above gives the Apriori algorithm with its related algorithms. The algorithm executes as; firstly Apriori finds frequent 1-itemsets, L_1. Frequent 1-itemsets are used to generate candidates C_k for finding L_k for $k \geq 2$. For eliminating subsets that do not satisfy minimum support count, the Apriori property is used for. The **apriori_gen** procedure generates the candidates. After generating all candidates, the database is scanned and for each transaction the candidates are scanned and the support of each candidate is determined. Finally, the candidates which support minimum support count constitute *L*. A procedure can then be called to generate association rules from the frequent itemsets. The **apriori_gen** procedure performs two kinds of actions, namely, join and prune, as described above. In the join component, L_{k-1} is joined with L_{k-1} to generate potential candidates. The prune component employs the Apriori property to remove candidates that have a subset that is not frequent. The test for infrequent subsets is shown in procedure **has_infrequent_subset** (Han and Kamber 2005).

Table 3.2: Market basket data

TID	Items
T1	{A, B, C, D}
T2	{A, C, D, E}
T3	{A, B, C, D, E}
T4	{A, C, D}
T5	{A, B, C}

In Table 3.2, an example dataset is given. The dataset contains 5 transactions, $|D|$ = 5. If we follow Algorithm 3, the following steps are done.

1. In the first iteration of the algorithm, each item is a member of the set of candidate 1-itemsets as indicated in Figure 3.1, C_1. The algorithm scans all of the transactions in order to count the number of occurrences of each item.

2. Suppose that the minimum support count required is 3, that is, min_sup = 3. The set of frequent 1-itemsets, L_1, can then be determined. It consists of the candidate 1-itemsets satisfying minimum support. All of the candidates except {E} in C_1 satisfy minimum support.

3. To find the set of frequent 2-itemsets, L_2, the algorithm uses the join L_1 on L_1 to generate a candidate set of 2-itemsets, C_2. C_2 consists of $\binom{|L_1|}{2}$ 2-itemsets.

4. Next, the transactions in D are scanned and the support count of each candidate itemset in C_2 is accumulated, as shown in Figure 3.2.

5. The set of frequent 2-itemsets, L_2, is then determined, consisting of those candidate 2-itemsets in L_2 having minimum support.

6. The generation of the set of candidate 3-itemsets,C_3, is detailed in Figure 3.3.

7. The transactions in D are scanned in order to determine L_3, consisting of those candidate 3-itemsets in C_3 having minimum support.
8. Thus, $C_4 = 0$, and the algorithm terminates, having found all of the frequent itemsets.

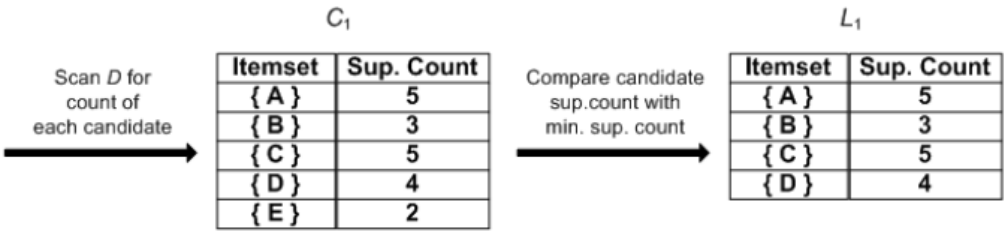


Figure 3.1: Generation of level-1 candidate itemsets and frequent itemsets

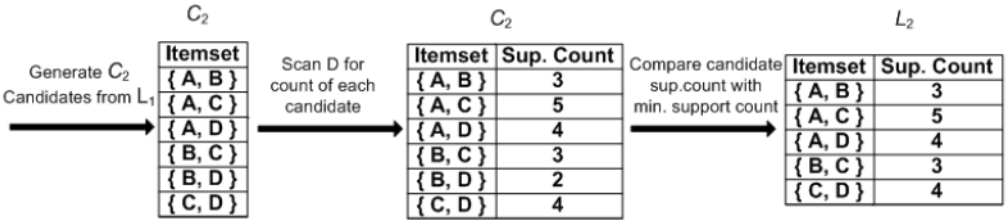


Figure 3.2: Generation of level-2 candidate itemsets and frequent itemsets

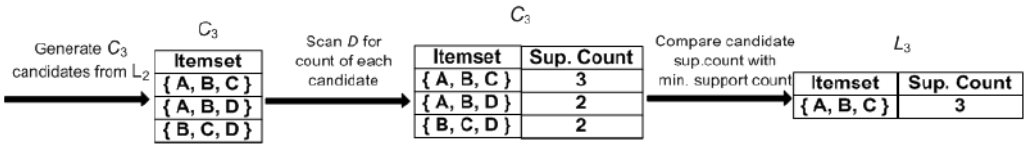


Figure 3.3: Generation of level-3 candidate itemsets and frequent itemsets

3.3 FP-GROWTH ALGORITHM

In the previous section we have examined Apriori algorithm and detailed the algorithm in a concrete example. Although Apriori can gain from performance by reducing the candidates counts, there are two nontrivial costs that Apriori suffers:
1. Apriori generates huge number of candidate sets. For example, if there are 10^4 frequent 1-itemsets, the Apriori algorithm will need to generate more than 10^7 candidate 2-itemsets. Moreover, to discover a frequent pattern of size 100, such as $\{ a_1, \ldots, a_{100} \}$, it has to generate at least 2^{100} - 1 $\approx 10^{30}$ candidates in total.
2. It may scan the database repeatedly for matching patters. Each transaction is checked for determining support count of each candidate sets.

"Can we design a method that mines the complete set of frequent itemsets without candidate generation" An interesting method in this attempt is called frequent-pattern growth, or simply FP-growth, which adopts a divide-and-conquer strategy as follows. First, it compresses the database representing frequent items into a frequent-pattern tree, or FP-tree, which retains the itemset association information. It then divides the compressed database into a set of conditional databases (a special kind of projected database), each associated with one frequent item or pattern fragment and mines each such database separately. You will see how it works with the following example (Han and Kamber 2005). FP-growth (finding frequent itemsets without candidate generation). We will use the following transaction database, *D*, for analysing FP-Growth in Table 3.3.

Table 3.3: Market basket transactional data

TID	**Items**
T1	{I1, I2, I5}
T2	{I2, I4}
T3	{I2, I3}
T4	{I1, I2, I4}
T5	{I1, I3}
T6	{I2, I3}
T7	{I1, I3}
T8	{I1, I2, I3, I5}
T9	{I1, I2, I3}

Source: Han and Kamber (2005)

For deriving frequent 1-itemsets and their support counts FP-Growth applies the same method like Apriori. Let the minimum support count is 2. The set of frequent items is sorted in the order of descending support count. This resulting set or list is denoted L. Thus, we have $L = \{\{I2 : 7\}, \{I1, 6\}, \{I3, 6\}, (I4, 2), \{I5, 2\}\}$.

An FP-tree is then constructed as follows. First, create the root of the tree, labeled with *null*. Scan database *D* a second time. The items in each transaction are processed in *L* order (i.e., sorted according to descending support count), and a branch is created for each transaction. For example, the scan of the first transaction, T1: I1, I2, I5 which contains three items (I2, I1, I5 in *L* order), leads to the construction of the first branch of the tree with three nodes, $\langle I2, 1\rangle$, $\langle I1, 1\rangle$, and $\langle I5, 1\rangle$, where I2 is linked as a child of the root, I1 is linked to I2, and I5 is linked to I1. The second transaction, T2, contains the items I2 and I4 in *L* order, which would result in a branch where I2 is linked to the root and I4 is linked to I2. However, this branch would share a common prefix, I2, with the existing path for T1. Therefore, we instead increment the count of the I2 node by 1, and create a newnode,$\langle I4, 1\rangle$,which is linked as a child of$\langle I2, 2\rangle$. In general, when considering the branch to be added for a transaction, the count of each node along a common prefix is incremented by 1, and nodes for the items following the prefix are created and linked accordingly. To make tree traversal easier, an item header table is built so that each item points to its occurrences in the tree via a chain of node-links. The tree obtained after scanning all of the transactions is shown in Figure 3.4 with the associated node-links. In this way, the problem of mining frequent patterns in databases is transformed to that of mining the FP-tree (Han and Kamber 2005).

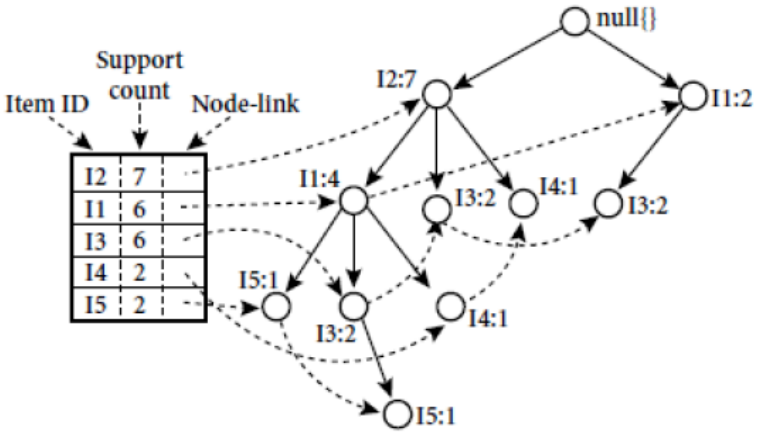


Figure 3.4: An FP-tree registers compressed, frequent pattern information
Source: Han and Kamber (2005)

Item	Conditional Pattern Base	Conditional FP-tree	Frequent Patterns Generated
I5	{{I2, I1: 1}, {I2, I1, I3: 1}}	⟨I2: 2, I1: 2⟩	{I2, I5: 2}, {I1, I5: 2}, {I2, I1, I5: 2}
I4	{{I2, I1: 1}, {I2: 1}}	⟨I2: 2⟩	{I2, I4: 2}
I3	{{I2, I1: 2}, {I2: 2}, {I1: 2}}	⟨I2: 4, I1: 2⟩, ⟨I1: 2⟩	{I2, I3: 4}, {I1, I3: 4}, {I2, I1, I3: 2}
I1	{{I2: 4}}	⟨I2: 4⟩	{I2, I1: 4}

Figure 3.5: Mining the FP-tree by creating conditional (sub-)pattern bases
Source: Han and Kamber (2005)

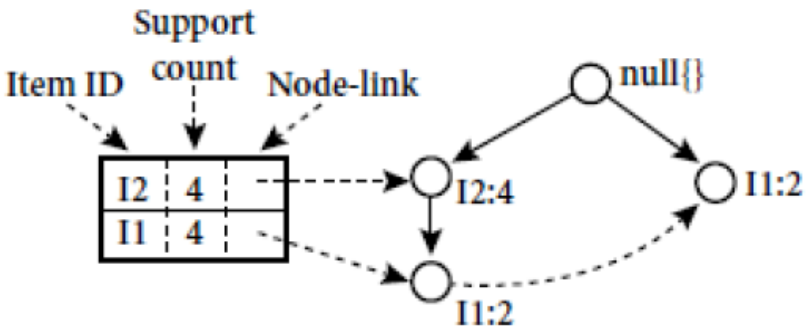


Figure 3.6: The conditional FP-tree associated with the conditional node I3
Source: Han and Kamber (2005)

The work on the FP-tree is executed as follows. 1. As an initial suffix pattern, start from each frequent length-1, 2. Construct its subdatabase, which consists of the set of prefix paths in the FP-tree co-occurring with the suffix pattern 3. Construct its conditional FP-tree 4. Perform recursive mining on such a tree. By the concatenating of the suffix pattern with the frequent patterns generated from a conditional FP-tree the pattern growth is achieved .

In Figure 3.5 mining of the FP-tree is summarized and detailed as follows. We first consider I5, which is the last item in *L*, rather than the first. The reason for starting at the end of the list will become apparent we explained the FP-tree mining process. I5 occurs in two branches of the FP-tree of Figure 3.4. (The occurrences of I5 can easily be found by following its chain of node-links.) The paths formed by these branches are $\langle I2, I1, I5 : 1\rangle$ and $\langle I2, I1, I3, I5 : 1\rangle$. Therefore, considering I5 as a suffix, its corresponding two prefix paths are $\langle I2, I1 : 1\rangle$ and $\langle I2, I1, I3 : 1\rangle$, which form its conditional pattern base. Its conditional FP-tree contains only a single path, $\langle I2 : 2, I1 : 2\rangle$;I3 is not included because its support count of 1 is less than the minimum support count. The single path generates all the combinations of frequent patterns: $\langle I2, I5 : 2\rangle$, $\langle I2, I1, I5 : 1\rangle$, $\langle I2, I1, I5 : 2\rangle$. For I4, its two prefix paths form the conditional pattern base, $\langle I2, I1 : 1\rangle$,

$\langle I2 : 1\rangle$, which generates a single-node conditional FP-tree,$\langle I2 : 2\rangle$, and derives one frequent pattern, $\langle I2, I1 : 2\rangle$. Notice that although I5 follows I4 in the first branch, there is no need to include I5 in the analysis here because any frequent pattern involving I5 is analyzed in the examination of I5. Similar to the above analysis, $I3$'s conditional pattern base is $\langle I2, I1 : 2\rangle$, $\langle I2 : 2\rangle$, $\langle I1 : 2\rangle$. Its conditional FP-tree has two branches,$\langle I2 : 4\rangle$, $\langle I1 : 2\rangle$ and $\langle I1 : 2\rangle$, as shown in FigureAlgorithm 3.6, which generates the set of patterns, $\langle I2, I3 : 4\rangle$, $\langle I1, I3 : 4\rangle$, $\langle I2, I3 : 2\rangle$ (Han and Kamber 2005). Finally, $I1$'s conditional pattern base is $\langle I2 : 4\rangle$, whose FP-tree contains only one node, $\langle I2 : 4\rangle$, which generates one frequent pattern, $\langle I2, I1 : 4\rangle$. This mining process is summarized in Algorithm 4.

The FP-growth algorithm reduces the resources for searching by transforming the issue of finding long frequent patterns to searching for shorter ones recursively and the concatenating the suffix. For large databases building the FP-tree consumes much memory that it might be unrealistic to realize. To alternate this problem, partitioning the database and executing the main approach on each divided part might be a solution. The studies showed that FP-Growth algorithm works better on large and small databases than Apriori algorithm.

Input: D : a transaction database, $minsup$: the minimum support count threshold

Output: The complete set of frequent patterns

procedureFP_growth*(Tree, α)*

if *Tree contains a single path P* **then**

  foreach *combination (denoted as β) of the nodes in the path P* **do**

    generate pattern $\beta \cup \alpha$ with *support_count = minimum supprt count* of nodes in β;

  end

else

  foreach a_i *in the header of Tree* **do**

    generate pattern $\beta = a_i \cup \alpha$ with support_count = a_i.*support_count*;

    construct β's conditional pattern base and then β's conditional FP_tree $Tree_\beta$;

    if $Tree_\beta \neq \emptyset$ **then**

      call FP_growth($Tree_\beta$,β)

    end

  end

end

Algorithm 4: FP growth

3.4 OTHER METHODS

In (Hu et al. 2008), an algorithm introduced which generate maximum length frequent itemsets by adapting a pattern fragment growth methodology based on the FP-tree structure and some other optimization techniques have been exploited to prune the search space.

For efficiently obtaining frequent itemsets by a special frequent items ultrametric tree (FIU-tree) and to supply some advantages such as reducing the I/O cost are represented in (Tsay et al. 2009). For efficiently detecting frequent itemsets in data streams, Chernoff bound based false-negative oriented algorithms are reported in (Yu et al. 2006).

In (Zhong 2007), methods designed for both vertically and horizontally partitioned data sets which maintain privacy in distributed mining of frequent itemsets are proposed. For mining frequent itemsets and generating association rules (Shen et al. 1999) presents efficient parallel algorithms.

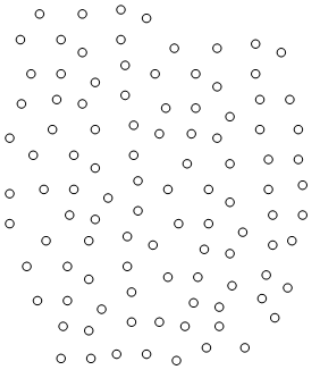
4. COFISA: CLUSTERING FOR FREQUENT ITEM SET DETECTION ALGORITHM

The approximative frequent itemset algorithm (AFISA) (Mimaroglu and Simovici 2007) is an practical algorithm that identifies the frequent itemsets approximately instead of finding all of the frequent itemsets in a dataset.

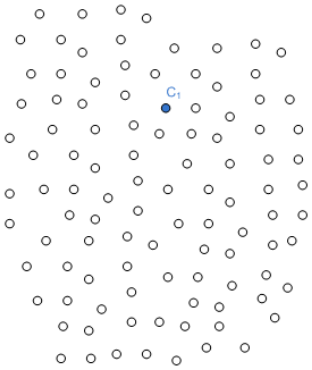
4.1 COFISA

Input: T: Transaction Data Set, μ: Minimum Support
Output: μ-Frequent Itemsets
Initialize T_μ to contain all items i_q, such that $supp_T(i_q) \geq \mu$;
$k = 0$;
while *there are unvisited items in* T_μ **do**
    `// Choose a new cluster center among unvisited items`
    Randomly select an unvisited item i_r as a cluster center, and mark i_r as visited ;
    $k++$;
    Create a cluster C_k with cluster center i_r;
    // $\mathbf{b}^{\mathbf{c_k}}$ bit vector represents C_k cluster
    // $\mathbf{b}^r$ is the characteristic bit vector of i_r
    $\mathbf{b}^{\mathbf{c_k}} = \mathbf{b}^r$;
    `// add elements to the cluster`
    while $supp_T(\boldsymbol{b}^{\mathbf{c_k}}) \geq \mu$ **do**
        forall the *items* $i_j \notin C_k$ **do**
            // δ is Jaccard-Tanimoto distance metric
            $s = argmin_j\ \delta(i_r, i_j)$;
            if $supp_T(\boldsymbol{b}^{\mathbf{c_k}} \wedge \boldsymbol{b}^s) \geq \mu$ **then**
                $\mathbf{b}^{\mathbf{c_k}} = \mathbf{b}^{\mathbf{c_k}} \wedge \mathbf{b}^s$;
                Assign i_s to the cluster C_k, and mark i_s as visited ;
            else
                Break while loop at 8;
            end
        end
    end
end

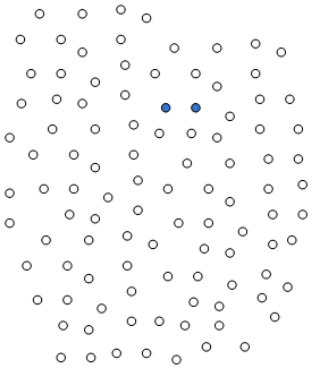
Algorithm 5: Clustering f**or F**requent **I**tem **S**et Detection **A**lgorithm (COFISA)



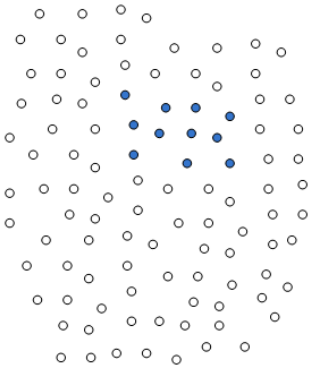
(a) **Initial phase**



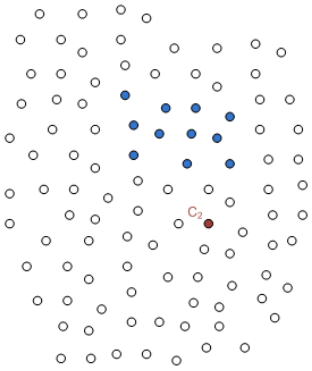
(b) **A cluster center**



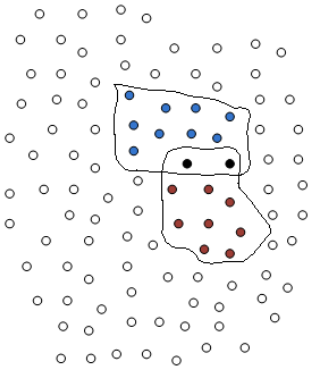
(c) **Expansion of a cluster**



(d) **Final form of the cluster**



(e) **Another cluster center**



(f) **Final form of two clusters**

Figure 4.1: Graphical representation of cluster similarities

Algorithm 5 shows the algorithm of COFISA and the main flow of COFISA is explained in Figure 4.1. COFISA randomly selects a object as shown in Figure 4.1(b) on the target data set in Figure 4.1(a) and uses this object as the cluster center for the immediate expansion of the cluster. The expansion of the cluster begins with the most similar neighbor as shown in Figure 4.1(c). The expansion of the cluster continues until the support of the cluster goes down μ. Figure 4.1(d) depicts the final form of the cluster. If there are some unvisited objects in the dataset, COFISA randomly selects another object as the cluster center as shown in Figure 4.1(e). The expansion of the new cluster takes place and the final form of the second cluster is shown in Figure 4.1(f). COFISA runs until all the objects belong to a cluster and performs complete clustering.

COFISA is a randomized binary algorithm and creates overlapping clusters. As explained in (Motwani and Raghavan 1996, 1995), randomized algorithms have two important advantages: simplicity and speed. COFISA have some similarities with k-means such as; selecting cluster centers are randomly. In k-means, number of clusters to be found is supplied to the algorithm. But, COFISA determines the cluster number automatically. All the clusters produced by COFISA and the subclusters of these clusters are always μ-frequent. If the number of items in a dataset is n, the complexity of COFISA is $O(n^2)$. COFISA finds significant number of frequent itemsets in the dataset, if not all of them, since, COFISA is an approximative frequent itemset detection algorithm. The frequent itemsets are similar to each other regarding to this distance metric. It is also important to note that selecting cluster centers randomly does not significantly affect the number of frequent itemsets found.

4.1.1 Execution Flow of COFISA

This section gives information about the execution flow of COFISA algorithm including representation of the objects, cluster center selection, cluster expansion and algorithm termination conditions.

COFISA algorithm uses binary data for representing the dataset. Each cluster (itemset) is represented by a bit vector. Therefore, each object and cluster is represented by its distinctive bit vector.

COFISA algorithm starts with selecting an object from the dataset randomly. The selected object is a pivot object which initiates a new cluster. COFISA tries to expand the initiated

cluster from starting the most similar neighbor. This expansion may continue considering all the neighbors of the pivot object. If the combined support of the objects in the cluster satisfies μ, then the neighbor object is included into the cluster. The characteristic bit vector of the cluster is calculated using the included object's bit vector and it represents all its members in the cluster. As depicted above, the clusters and its sub-clusters are always μ-frequent.

An expansion of a cluster terminates when the characteristic bit vector of that cluster goes below the support μ. This depicts that the cluster can not include any other neighbor into the cluster. If there is any other object which is not a member of any cluster created by the algorithm, COFISA randomly selects a new object (a new pivot) from the non-member objects and applies the same expansion regulations. COFISA algorithm terminates when all the objects belong to a cluster. It is important to notice that an object may be a member of multiple clusters because COFISA creates overlapping clusters.

There is no pre-condition on the input data and arbitrary shape clusters can be found by COFISA. As shown in (Mimaroglu and Simovici 2007), the probability that $\delta(\mathbf{b}^K, \mathbf{b}^L) > d$ (which is the probability that COFISA will fail to join the sets K and L) is small for values of p that are close to 0 or close to 1 because $\phi_{k,l}(p)$ is small in this case. This means that for low or high density datasets COFISA should work quite well which is effectively the case in our experiments.

4.1.2 Experimental Results

For our experiments of COFISA, we used a computer running on Windows operating system, having 2GB of main memory and having 1.8 GHz processor. For utilizing the advantages of built-in support and operations on bit vectors, we implemented COFISA in Java programming language. Java Development Kit 1.6.0_16 is used for testing COFISA, AFISA, FP-Growth and APRIORI tests.

Real data sets such as Thrombin and Zoo are used along with synthetically generated data sets. For generating synthetical data sets we used the generator which is available from IBM Almaden Research Center through the "IBM Quest Data Mining Project".

It is significant to note that APRIORI always finds all the frequent itemsets in any data set. But as depicted above, both COFISA and AFISA are approximative algorithms which means that they may not find all the frequent itemsets.

Figure A. 1, Figure A. 3, Figure A. 6, Figure A. 9, Figure A. 12, Figure A. 16 show the results of on an artificial data set having $1,000$ transactions (objects), 20 items (attributes) and 2, 8, 10, 12, 16, 18 items/transaction (on average). The results show that COFISA misses some of the frequent itemsets, but it finds more frequent itemsets than AFISA nearly in all these datasets. The results in $1K$ also show that COFISA is much more faster APRIORI and comparable to AFISA but for this kind of small datasets COFISA's run time benefits can be ignored.

Figure A. 21 shows the frequent itemset counts and execution times on a dataset having $10,000$ transactions, 100 items, and 10 items/transaction. The results on this data set show more apparent differences between COFISA and AFISA: For the support of 0.02, COFISA finds 64% more frequent itemsets than AFISA. For the same support level COFISA is 15% faster than AFISA, and COFISA is 44 times faster than APRIORI.

Figure A. 23, Figure A. 25 and Figure A. 27 show the results of on an artificial data set having $1,000$ transactions (objects), 20 items (attributes) and 5, 10, 20 items/transaction (on average). In Figure A. 23, it can be observed that for the support of 0.03, COFISA misses only 15.2% of all the frequent itemsets. COFISA is 2.3 times faster than AFISA, and 39.9 times faster than APRIORI. Also in Figure A. 27 for the support 0.2, COFISA misses 43.1% of all the frequent itemsets, but COFISA is 203.4 time faster than APRIORI.

Figure A. 29 shows the results of on an artificial data set having $500,000$ transactions, 100 items and 5 items/transaction (on average). On this dataset, COFISA is very successful by finding significant part of the frequent itemsets. Also, COFISA is very fast performing on this dataset: COFISA is at least twice faster than AFISA, and around 60 times faster than APRIORI.

The results of test on real datasets are shown in Figure A. 33. Zoo data set, which is obtained from (Asuncion and Newman 2007), contains 101 objects, and 17 attributes and Thrombin dataset, which is used in the 2001 KDD data mining competition, contains 634 objects and 100 randomly selected attributes out of the original $139,351$ attributes.

In Figure A. 36, execution time results of APRIORI, FP-Growth, and COFISA on the data set having $500,000$ transactions are shown. As depicted earlier, FP-Growth is an efficient algorithm which discovers frequent itemsets without generating any candidates. Performance-wise, COFISA is clearly superior to both FP-Growth and APRIORI.

Finally, all the frequent itemset counts and execution time results are followed by level-wise frequent itemset counts found by COFISA and APRIORI.

Figure 4.2 shows the average time spent per frequent itemset by Apriori and COFISA algorithms. COFISA's superiority is remarkable on especially large data sets.

Table 4.1 gives the properties of each dataset.

Table 4.1: Properties of datasets

Name	Num. of Transactions	Num. of Items	Average Items Per Transaction
1K_20I_2	1032	20	2
1K_20I_8	957	20	8
1K_20I_10	994	20	10
1K_20I_12	971	20	12
1K_20I_16	1000	20	16
1K_20I_18	1000	20	18
10K	9850	100	10
100K_5	83386	100	5
100K_10	98261	100	10
100K_20	99994	100	20
500K	491403	100	10
Thrombin	634	100	4
Zoo	101	21	2

Method	1K_20I_2	1K_20I_8	1K_20I_10	1K_20I_12	1K_20I_16	1K_20I_18
Apriori	6.63	3.63	1.10	0.33	3.79	0.89
COFISA	1.25	0.41	0.13	0.05	0.0005	0.000009

(a)

Method	10K	10K_100I_5	100K_100I_10	100K_100I_20	500K_100I_5
Apriori	67.12	703.13	1092.29	1604.22	3049.88
COFISA	3.85	27.83	24.52	17.73	125.35

(b)

Method	Zoo	Thrombin
Apriori	0.734	3.682
COFISA	0.019	0.089

(c)

Figure 4.2: Average time (ms) by COFISA and Apriori per frequent itemsets

4.1.3 COFISA on an Example

Table 4.2 gives an example dataset for executing COFISA and exploring clusters.

Table 4.2: Sample dataset for COFISA example

TID	I_1	I_2	I_3	I_4	I_5	I_6	I_7
T_1	0	**0**	0	1	0	0	1
T_2	0	**0**	1	1	0	0	0
T_3	0	**1**	0	0	0	1	1
T_4	0	**1**	1	0	0	1	1
T_5	1	**0**	1	1	0	0	0
T_6	0	**1**	0	1	0	0	1
T_7	1	**1**	1	0	1	1	1
T_8	0	**0**	0	0	0	1	0
T_9	0	**1**	1	0	0	1	0
T_{10}	0	**1**	1	0	1	0	1

From Table 4.2, we can observe that each object is represented by it's characteristic bit vector. These bit vectors are composed by the transactions and the binary value of that object.

The characteristic bit vector of I_2 is :

0	0	1	1	0	1	1	0	1	1

For this example we define min_sup = 3. As Table 4.3 shows, I_1 and I_5 do not satisfy min_sup, because their support counts are 2. These objects will not be included in any cluster, so no further calculations will be done for these objects.

Table 4.3: Objects which do not satisfy min_sup

TID	I_1	I_2	I_3	I_4	I_5	I_6	I_7
T_1	**0**	0	0	1	**0**	0	1
T_2	**0**	0	1	1	**0**	0	0
T_3	**0**	1	0	0	**0**	1	1
T_4	**0**	1	1	0	**0**	1	1
T_5	**1**	0	1	1	**0**	0	0
T_6	**0**	1	0	1	**0**	0	1
T_7	**1**	1	1	0	**1**	1	1
T_8	**0**	0	0	0	**0**	1	0
T_9	**0**	1	1	0	**0**	1	0
T_10	**0**	1	1	0	**1**	0	1

We use Jaccard-Tanimoto distance metric δ on two bit vectors. The distance matrix calculated between objects are shown in Table 4.4.

Table 4.4: Distance matrix of COFISA example

$Item$	I_2	I_3	I_4	I_6	I_7
I_2	0.0	0.5	0.888	0.428	0.285
I_3	0.5	0.0	0.75	0.625	0.666
I_4	0.888	0.75	0.0	1.0	0.75
I_6	0.428	0.625	1.0	0.0	0.625
I_7	0.285	0.666	0.75	0.625	0.0

As described in the section above, COFISA starts by selecting a pivot object randomly. I_2 is selected randomly and the support count of the I_2 is 6.

The objects and the distances to I_2 from closest to farthest are shown in Table 4.5.

Table 4.5: Objects and distances to I_2

$Item$	I_7	I_6	I_3	I_4
$Distance$	0.285	0.428	0.5	0.8888

The expansion starts with the closest object I_7 and the cluster bit vector is calculated in Table 4.6.

Table 4.6: Combined bit vector of I_2 and I_7

I_2	0	0	1	1	0	1	1	0	1	1
I_7	1	0	1	1	0	1	1	0	0	1
AND	0	0	1	1	0	1	1	0	0	1

The support count is 5. So, I_7 is included into the cluster. Now the cluster contains I_2 and I_7. And the expansion continues with I_6 and the cluster bit vector is calculated in Table 4.7.

Table 4.7: Combined bit vector of cluster 1

CBV	0	0	1	1	0	1	1	0	0	1
I_6	0	0	1	1	0	0	1	1	1	0
AND	0	0	1	1	0	0	1	0	0	0

The support count is 3. So, I_6 is included into *Cluster 1*. Now *Cluster 1* contains I_2, I_7 and I_6. The expansion continues with the next closest object and the cluster bit vector is calculated in Table 4.8.

Table 4.8: Combined bit vector of cluster 1 - cont.

CBV	0	0	1	1	0	0	1	0	0	0
I_3	0	1	0	1	1	0	1	0	1	1
AND	0	0	0	1	0	0	1	0	0	0

The support count is $2 < min\_sup$, I_3 is not included into *Cluster 1* and the expansion of *Cluster 1* ends. $\{ I_2, I_6, I_7 \}$ are marked as visited and they will not selected as pivot objects for further cluster explorations.

For the next pivot object, I_3 is selected randomly. The support count of I_3 is 6 and the bit vector of I_3 is:

0	1	0	1	1	0	1	0	1	1

The same steps are executed for cluster expansion, the distances from closest to farthest are shown in Table 4.9.

Table 4.9: Objects and distances to I_3

$Item$	I_2	I_6	I_7	I_4
$Distance$	0.5	0.625	0.666	0.75

The expansion starts with the closest object I_2. The combined bit vector is shown in Table 4.10.

The support count of the combined bit vector is 4. So, I_2 is included into *Cluster 2*. Now the cluster contains I_3 and I_2. The next object for expansion is I_6 and the combined bit vector is calculated as shown in Table 4.11.

Table 4.10: Combined bit vector of I_3 and I_2

I_3	0	1	0	1	1	0	1	0	1	1
I_2	0	0	1	1	0	1	1	0	1	1
AND	0	0	0	1	0	0	1	0	1	1

Table 4.11: Combined bit vector of cluster 2

CBV	0	0	0	1	0	0	1	0	1	1
I_6	0	0	1	1	0	0	1	1	1	0
AND	0	0	0	1	0	0	1	0	1	0

The support count is 3. So, I_6 is included into the *Cluster 2*. Now the *Cluster 2* contains I_3, I_2 and I_6. The expansion continues with the next closest object and the combined bit vector is calculated in Table 4.12.

Table 4.12: Combined bit vector of cluster 2 - cont.

CBV	0	0	0	1	0	0	1	0	1	0
I_7	1	0	1	1	0	1	1	0	0	1
AND	0	0	0	1	0	0	1	0	0	0

The support count is $2 < min\_sup$, *I7* is not included into *Cluster 2* and the expansion of *Cluster 2* ends.
I_4 is randomly selected as the next pivot object. The support count of I_4 is 4. The distances to I_4 are shown in Table 4.13.

Table 4.13: Objects and distances to I_4

$Item$	I_3	I_7	I_2	I_6
$Distance$	0.75	0.75	0.888	1.0

Cluster 3 will continue expansion with the closest object which is I_3. The combined bit vector is calculated in Table 4.14.

Table 4.14: Combined bit vector of I_4 and I_3

I_4	0	1	0	1	1	0	1	0	1	1
I_3	1	1	0	0	1	1	0	0	0	0
AND	0	1	0	0	1	0	0	0	0	0

The support count of combined bit vector is 2. So, I_3 is not included into the *Cluster 3*. The cluster contains only I_4 and the expansion of *Cluster 3* ends.

All the objects in the are visited and the algorithm terminates. The final clusters are as follows:

Table 4.15: Final clusters of COFISA example

Cluster Id	*Items*
Cluster 1	**I2, I7, I6**
Cluster 2	**I3, I2, I6**
Cluster 3	**I4**

Cluster 1 contains I_2, I_7, I_6. Frequent itemsets are: $\{ I_2 \}$, $\{ I_7 \}$, $\{ I_6 \}$, $\{ I_2, I_7 \}$, $\{ I_2, I_6 \}$, $\{ I_7, I_6 \}$, $\{ I_2, I_7, I_6 \}$
Cluster 2 contains I_3, I_2, I_6. Frequent itemsets are: $\{ I_3 \}$, $\{ I_3, I_2 \}$, $\{ I_3, I_6 \}$, $\{ I_3, I_2, I_6 \}$
Cluster 3 contains I_4. Frequent itemset is: $\{ I_4 \}$
The total count of frequent itemsets is **12**.

5. CONCLUSION

COFISA is a novel, binary clustering algorithm for obtaining frequent itemsets and creates overlapping clusters that an object may be members of more than one cluster. COFISA uses binary data for its operations on the dataset and each object is represented by its characteristic bit vector. COFISA is fast-performing algorithm on small or large datasets as shown in experimental results. COFISA produces frequent itemsets that are close to each other with respect to Jaccard-Tanimoto distance metric. We compared COFISA with other frequent itemset detection methods: AFISA, APRIORI and FP-Growth. We used two different implementations of APRIORI which are "ARtool" of Cristofor and "A fast APRIORI implementation" from FIMI repository. The experimental results showed that COFISA is much more faster than both of the implementations. Extensive experimental evaluations demonstrated that COFISA missed fewer frequent itemsets than AFISA. In terms of execution time, COFISA is superior to APRIORI, FP-Growth and AFISA. Remarkably short execution times, and very accurate approximative results obtained by COFISA are very important for complex information systems having large data sets.
For future work, COFISA can be implemented using other distance metrics such as XOR distance and the achievement of COFISA can be measured. Also, COFISA can be implemented to produce non-overlapping clusters and compared to other non-overlapping implementations of frequent itemset detection algorithms. For the implementation of COFISA, there is a post-processing part for counting frequent itemsets from derived clusters. This post-processing part does not affect the time for creating clusters (execution time). But in an other implementation of COFISA this post-processing time may be handled in the main part of the algorithm and explore if it has a big effect on execution time.

REFERENCES

Books

Han, J. and Kamber, M.: 2005, *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Motwani, R. and Raghavan, P.: 1995, *Randomized algorithms*, Cambridge University Press, New York, NY, USA.

Tan, P.-N., Steinbach, M. and Kumar, V.: 2005, *Introduction to Data Mining, (First Edition)*, Addison-Wesley Longman Publishing, Boston, MA, USA.

Periodicals

Hu, T., Sung, S. Y., Xiong, H. and Fu, Q.: 2008, Discovery of maximum length frequent itemsets, *Information Sciences* **178**(1), 69 – 87.

Mimaroglu, S. and Simovici, D.: 2007, Clustering and approximate identification of frequent item sets, *Proceedings of the Twentieth International Florida Artificial Intelligence Research Society Conference* pp. 502–506.

Motwani, R. and Raghavan, P.: 1996, Randomized algorithms, *ACM Comput. Surv.* **28**(1), 33–37.

Shen, L., Shen, H. and Cheng, L.: 1999, New algorithms for efficient mining of association rules, *Information Sciences* **118**(1-4), 251 – 268.

Tsay, Y.-J., Hsu, T.-J. and Yu, J.-R.: 2009, Fiut: A new method for mining frequent itemsets, *Information Sciences* **179**(11), 1724 – 1737. Including Special Issue on Chance Discovery - Discovery of Significant Events for Decision.

Yu, J. X., Chong, Z., Lu, H., Zhang, Z. and Zhou, A.: 2006, A false negative approach to mining frequent itemsets from high speed transactional data streams, *Information Sciences* **176**(14), 1986 – 2015. Streaming Data Mining.

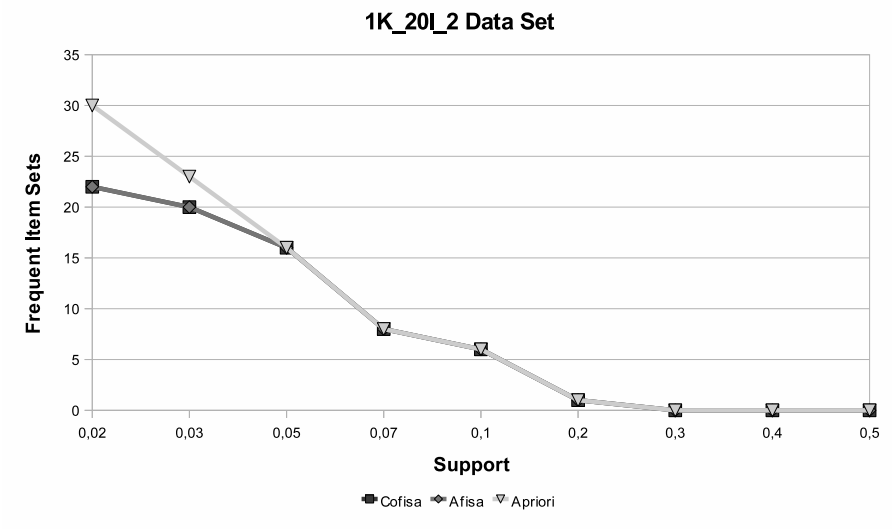
Zhong, S.: 2007, Privacy-preserving algorithms for distributed mining of frequent itemsets, *Information Sciences* **177**(2), 490 – 503.

Other References

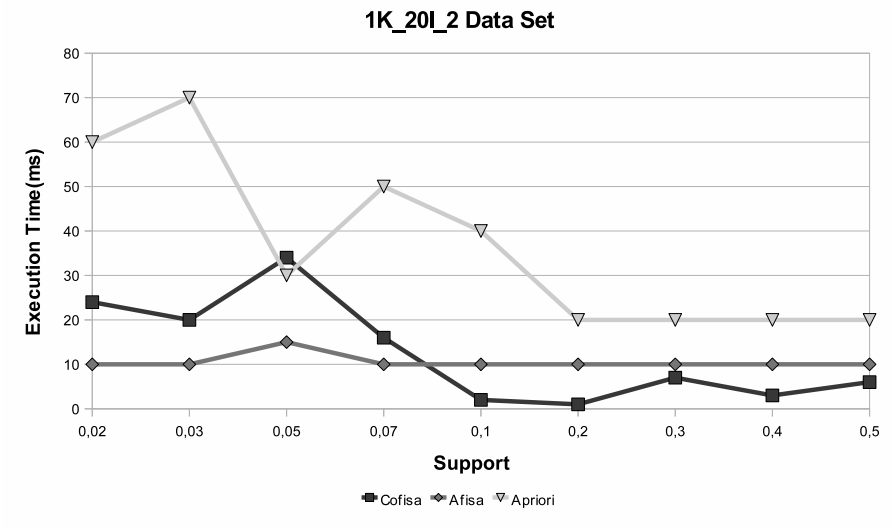
Asuncion, A. and Newman, D.: 2007, Uci machine learning repositor.
URL: *http://www.ics.uci.edu/∼mlearn/MLRepository.html*

APPENDICES

Appendix A: Experimental Results Details



(a) **Frequent itemsets**



(b) **Execution time**

Figure A. 1: Test results on a very small dataset

	Support	**0.02**
	Cofisa	**Apriori**
1-length	18	20
2-length	4	10
Total	22	30

(a)

	Support	**0.03**
	Cofisa	**Apriori**
1-length	17	17
2-length	3	6
Total	20	23

(b)

	Support	**0.05**
	Cofisa	**Apriori**
1-length	16	16
Total	16	16

(c)

	Support	**0.07**
	Cofisa	**Apriori**
1-length	8	8
Total	8	8

(d)

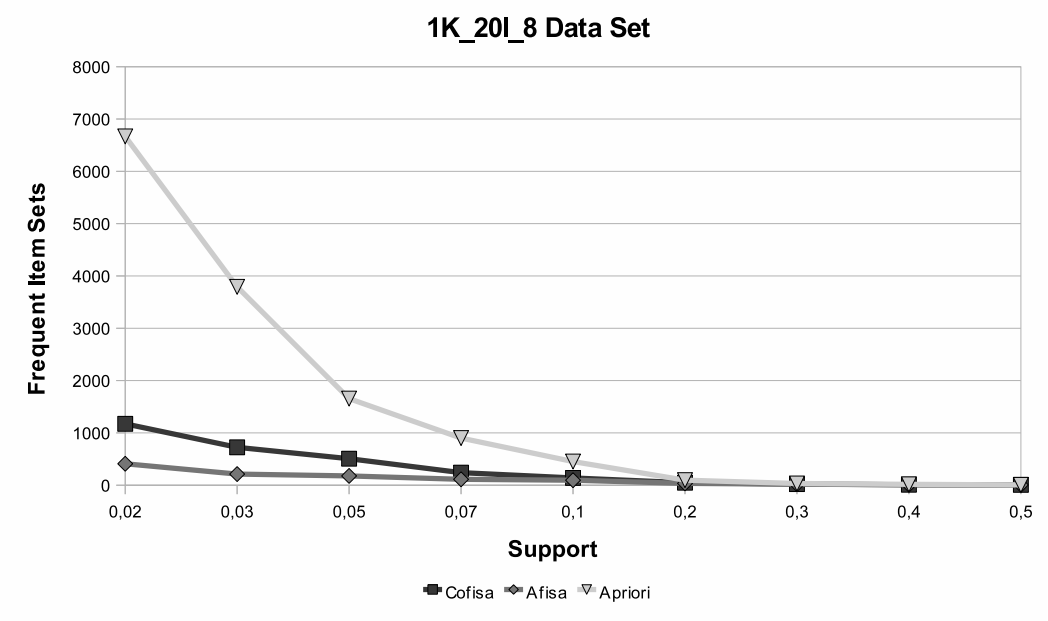
	Support	**0.1**
	Cofisa	**Apriori**
1-length	6	6
Total	6	6

(e)

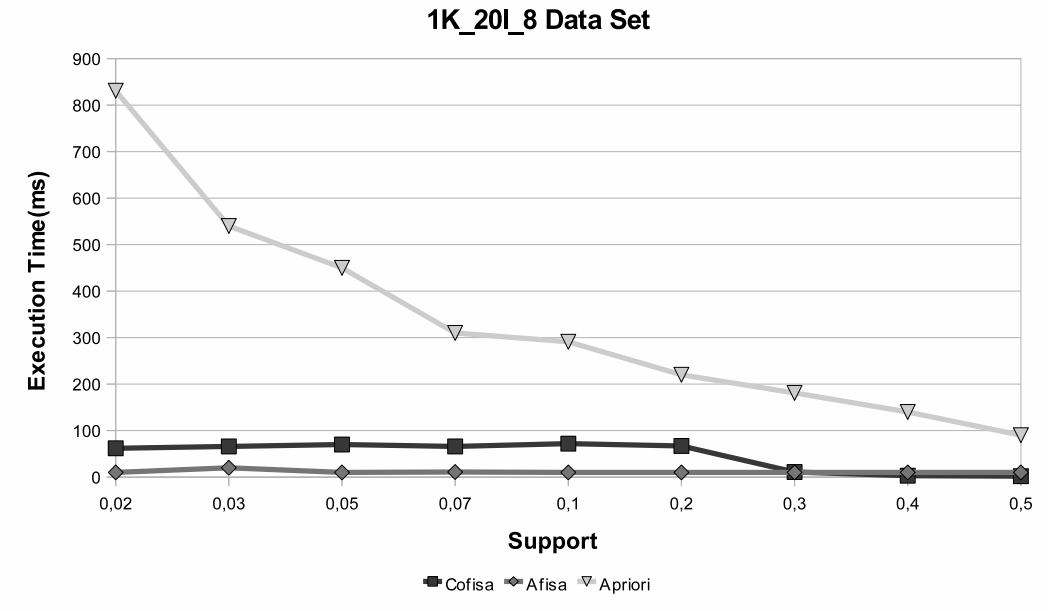
	Support	**0.2**
	Cofisa	**Apriori**
1-length	1	1
Total	1	1

(f)

Figure A. 2: Level-wise frequent itemset counts(on 1K_20I_2 dataset)



(a) **Frequent itemsets**



(b) **Execution time**

Figure A. 3: Test results on a very small dataset

	Support	0.02
	Cofisa	Apriori
1-length	19	19
2-length	92	166
3-length	225	758
4-length	326	1893
5-length	291	2291
6-length	157	1236
7-length	47	286
8-length	6	18
Total	1163	6667

(a)

	Support	0.03
	Cofisa	Apriori
1-length	19	19
2-length	81	158
3-length	170	669
4-length	206	1285
5-length	146	1173
6-length	56	436
7-length	9	51
Total	68	3791

(b)

	Support	0.05
	Cofisa	Apriori
1-length	19	19
2-length	73	141
3-length	139	487
4-length	150	678
5-length	92	273
6-length	30	57
7-length	4	4
Total	507	1659

(c)

	Support	0.07
	Cofisa	Apriori
1-length	18	18
2-length	58	133
3-length	83	356
4-length	61	275
5-length	22	114
6-length	3	5
Total	245	901

(d)

	Support	0.1
	Cofisa	Apriori
1-length	18	18
2-length	44	110
3-length	46	181
4-length	23	134
5-length	6	9
6-length	1	1
Total	138	452

(e)

	Support	0.2
	Cofisa	Apriori
1-length	16	16
2-length	20	52
3-length	10	20
4-length	3	8
Total	49	96

(f)

Figure A. 4: Level-wise frequent itemset counts(on 1K_20I_8 dataset)

	Support	**0.3**
	Cofisa	**Apriori**
1-length	11	11
2-length	8	15
3-length	3	6
Total	22	32

(g)

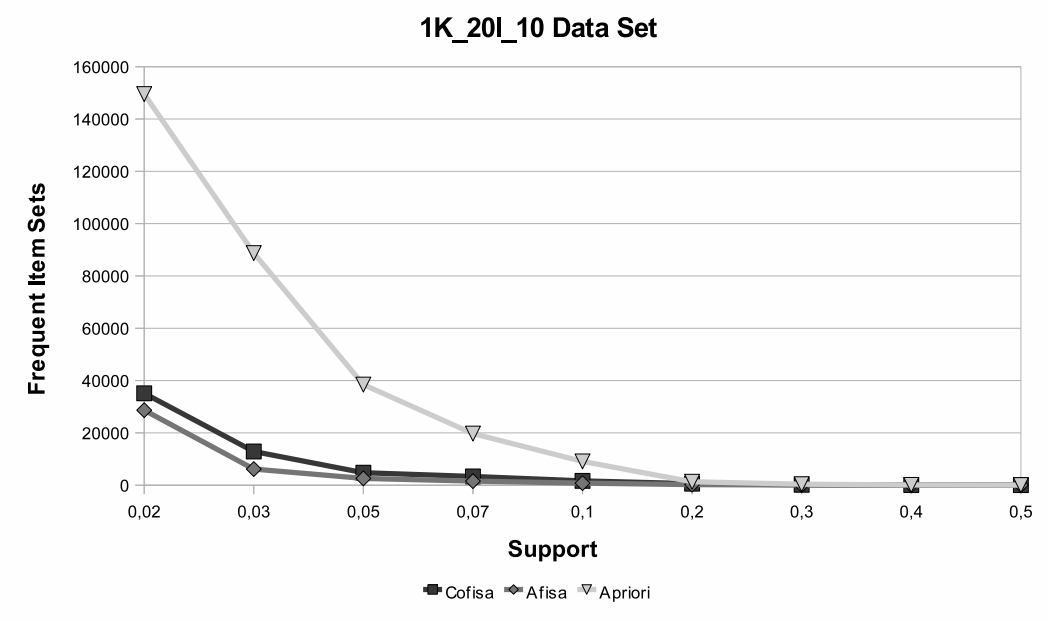
	Support	**0.4**
	Cofisa	**Apriori**
1-length	6	6
2-length	3	11
Total	9	17

(h)

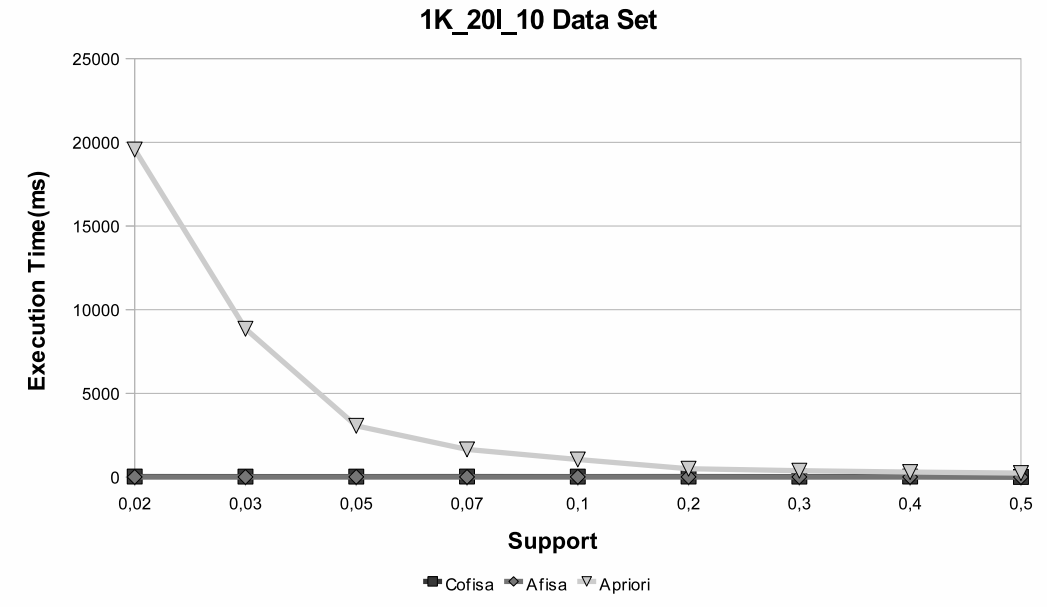
	Support	**0.5**
	Cofisa	**Apriori**
1-length	6	6
Total	6	6

(i)

Figure A. 5: Level-wise frequent itemset counts(on 1K_20I_8 dataset) - continues



(a) **Frequent itemsets**



(b) **Execution time**

Figure A. 6: Test results on a very small dataset

	Support	0.02
	Cofisa	Apriori
1-length	19	19
2-length	92	166
3-length	225	758
4-length	326	1893
5-length	291	2291
6-length	157	1236
7-length	47	286
8-length	6	18
Total	1163	6667

(a)

	Support	0.03
	Cofisa	Apriori
1-length	19	19
2-length	81	158
3-length	170	669
4-length	206	1285
5-length	146	1173
6-length	56	436
7-length	9	51
Total	68	3791

(b)

	Support	0.05
	Cofisa	Apriori
1-length	19	19
2-length	73	141
3-length	139	487
4-length	150	678
5-length	92	273
6-length	30	57
7-length	4	4
Total	507	1659

(c)

	Support	0.07
	Cofisa	Apriori
1-length	18	18
2-length	58	133
3-length	83	356
4-length	61	275
5-length	22	114
6-length	3	5
Total	245	901

(d)

	Support	0.1
	Cofisa	Apriori
1-length	18	18
2-length	44	110
3-length	46	181
4-length	23	134
5-length	6	9
6-length	1	1
Total	138	452

(e)

	Support	0.2
	Cofisa	Apriori
1-length	16	16
2-length	20	52
3-length	10	20
4-length	3	8
Total	49	96

(f)

Figure A. 7: Level-wise frequent itemset counts(on 1K_20I_10 dataset)

	Support	**0.3**
	Cofisa	**Apriori**
1-length	11	11
2-length	8	15
3-length	3	6
Total	22	32

(g)

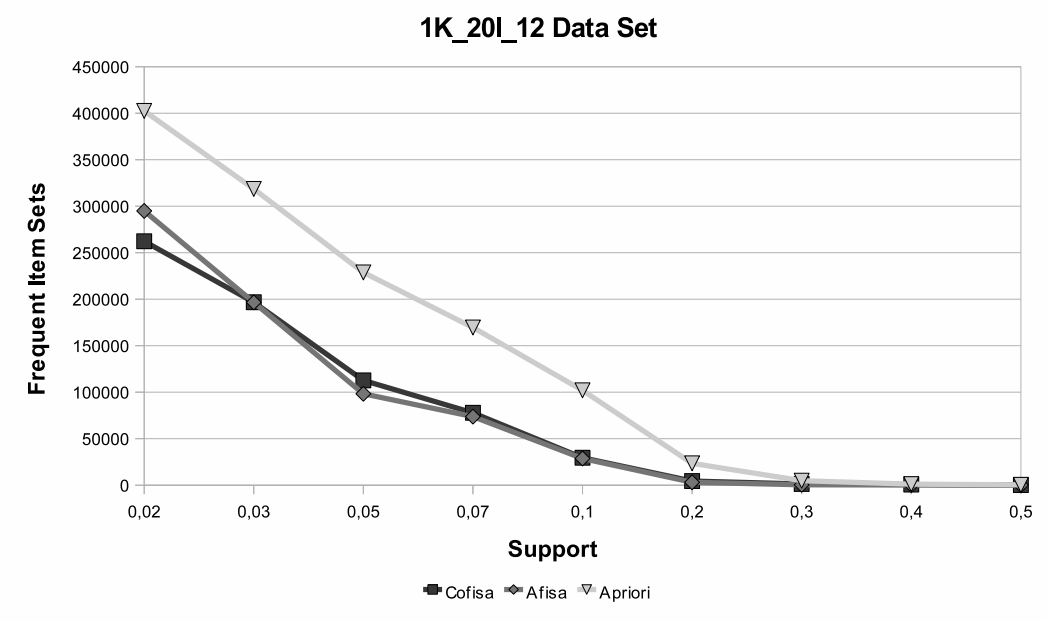
	Support	**0.4**
	Cofisa	**Apriori**
1-length	6	6
2-length	3	11
Total	9	17

(h)

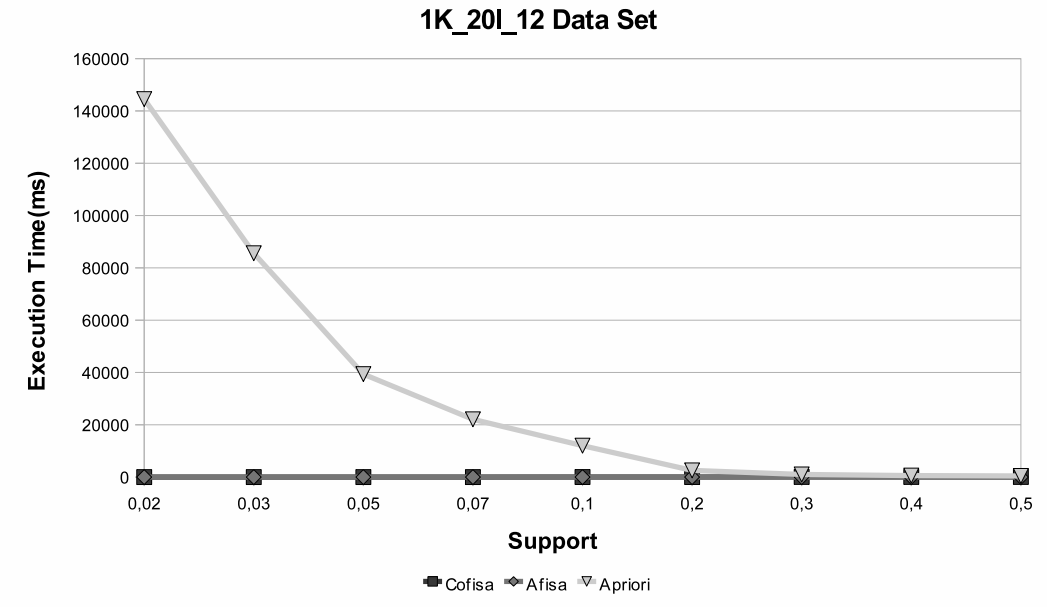
	Support	**0.5**
	Cofisa	**Apriori**
1-length	6	6
Total	6	6

(i)

Figure A. 8: Level-wise frequent itemset counts(on 1K_20I_10 dataset) - continues



(a) **Frequent itemsets**



(b) **Execution time**

Figure A. 9: Test results on a very small dataset

	Support	0.02
	Cofisa	Apriori
1-length	20	20
2-length	160	189
3-length	837	1104
4-length	3096	4455
5-length	8603	13170
6-length	18585	29480
7-length	31831	51327
8-length	43759	70867
9-length	48620	77948
10-length	43758	67847
11-length	31824	46376
12-length	18564	24825
13-length	8568	10415
14-length	3060	3414
15-length	816	856
16-length	153	155
17-length	18	18
18-length	1	1
Total	262273	402467

(a)

	Support	0.03
	Cofisa	Apriori
1-length	20	20
2-length	159	187
3-length	821	1039
4-length	2975	3985
5-length	8043	11464
6-length	16765	25331
7-length	27463	43400
8-length	35751	58225
9-length	37180	61752
10-length	30888	51942
11-length	20384	34437
12-length	10556	17694
13-length	4200	6835
14-length	1240	1887
15-length	256	344
16-length	33	37
17-length	2	2
Total	196736	318581

(b)

	Support	0.05
	Cofisa	Apriori
1-length	19	19
2-length	153	171
3-length	762	954
4-length	2643	3674
5-length	6735	10362
6-length	13013	21949
7-length	19448	35415
8-length	22737	44117
9-length	20878	43147
10-length	15015	33585
11-length	8372	20825
12-length	3549	10045
13-length	1106	3556
14-length	239	834
15-length	32	108
16-length	2	5
Total	114703	228766

(c)

	Support	0.07
	Cofisa	Apriori
1-length	19	19
2-length	148	169
3-length	707	931
4-length	2327	3480
5-length	5578	9280
6-length	10087	18529
7-length	14080	28828
8-length	15378	35307
9-length	13222	33319
10-length	8943	23290
11-length	4720	11518
12-length	1910	3854
13-length	574	854
14-length	121	133
15-length	16	16
16-length	1	1
Total	77831	169528

(d)

Figure A. 10: Level-wise frequent itemset counts(on 1K_20I_12 dataset)

	Support	0.1
	Cofisa	Apriori
1-length	19	19
2-length	134	168
3-length	568	858
4-length	1611	3028
5-length	3263	7967
6-length	4920	15448
7-length	5652	21631
8-length	49956	21845
9-length	3388	16265
10-length	1737	9214
11-length	652	4019
12-length	169	1266
13-length6 27	233	
14-length	2	14
Total	27137	101975

(e)

	Support	0.2
	Cofisa	Apriori
1-length	18	18
2-length	110	152
3-length	358	698
4-length	724	2098
5-length	971	4383
6-length	883	6248
7-length	540	5713
8-length	213	3108
9-length	49	952
10-length	5	124
11-length	0	2
Total	3871	23496

(f)

	Support	0.3
	Cofisa	Apriori
1-length	18	18
2-length	85	128
3-length	215	532
4-length	321	1176
5-length	296	1475
6-length	166	1071
7-length	52	402
8-length	7	56
9-length	0	1
Total	1160	4859

(g)

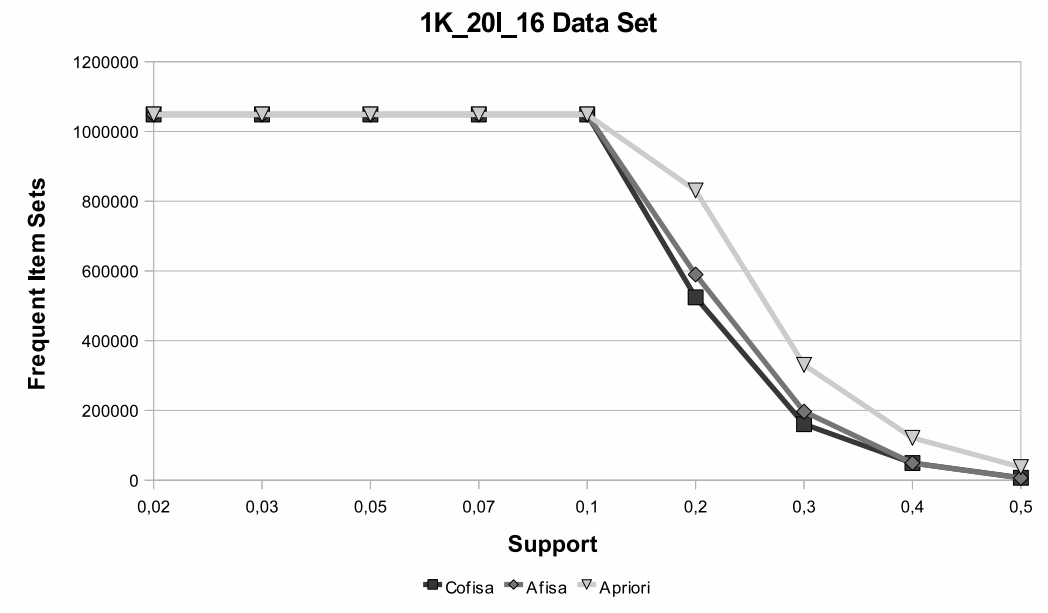
	Support	0.4
	Cofisa	Apriori
1-length	16	16
2-length	75	109
3-length	170	293
4-length	215	391
5-length	156	243
6-length	61	78
7-length 10	10	
Total	703	1140

(h)

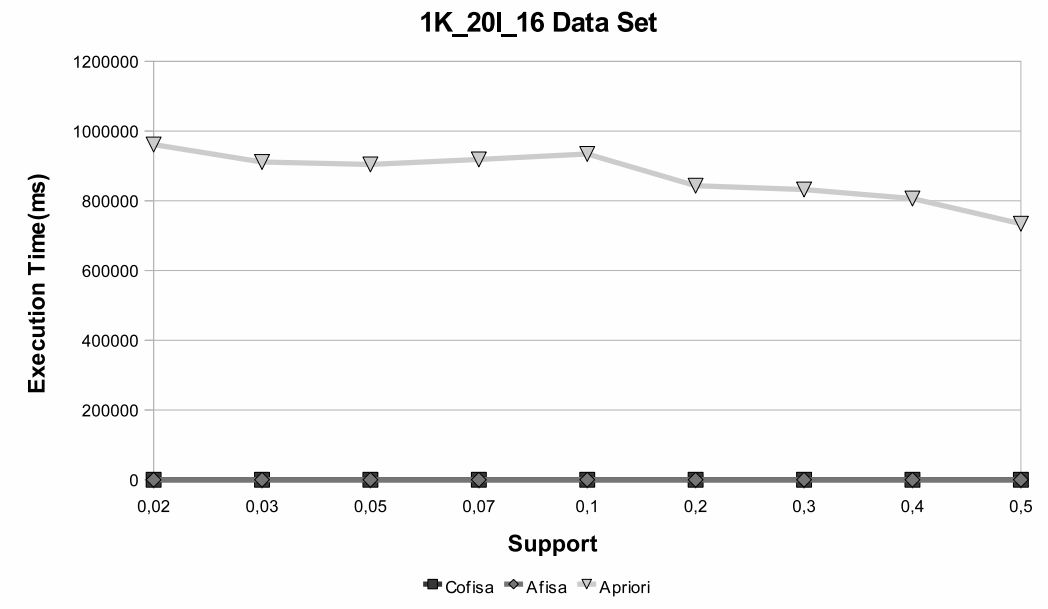
	Support	0.5
	Cofisa	Apriori
1-length	16	16
2-length	44	76
3-length	47	139
4-length	22	123
5-length	4	43
6-length	0	1
Total	133	398

(i)

Figure A. 11: Level-wise frequent itemset counts(on 1K_20I_12 dataset) - continues



(a) **Frequent itemsets**



(b) **Execution time**

Figure A. 12: Test results on a very small dataset

	Support	**0.02**
	Cofisa	**Apriori**
1-length	20	20
2-length	190	190
3-length	1140	1140
4-length	4845	4845
5-length	15504	15504
6-length	38760	38760
7-length	77520	77520
8-length	125970	125970
9-length	167960	167960
10-length	184756	184756
11-length	167960	167960
12-length	125970	125970
13-length	77520	77520
14-length	38760	38760
15-length	15504	15504
16-length	4845	4845
17-length	1140	1140
18-length	190	190
19-length	20	20
20-length	1	1
Total	1048575	1048575

(a)

	Support	**0.03**
	Cofisa	**Apriori**
1-length	20	20
2-length	190	190
3-length	1140	1140
4-length	4845	4845
5-length	15504	15504
6-length	38760	38760
7-length	77520	77520
8-length	125970	125970
9-length	167960	167960
10-length	184756	184756
11-length	167960	167960
12-length	125970	125970
13-length	77520	77520
14-length	38760	38760
15-length	15504	15504
16-length	4845	4845
17-length	1140	1140
18-length	190	190
19-length	20	20
20-length	1	1
Total	1048575	1048575

(b)

Figure A. 13: Level-wise frequent itemset counts(on 1K_20I_16 dataset)

	Support	**0.05**
	Cofisa	**Apriori**
1-length	20	20
2-length	190	190
3-length	1140	1140
4-length	4845	4845
5-length	15504	15504
6-length	38760	38760
7-length	77520	77520
8-length	125970	125970
9-length	167960	167960
10-length	184756	184756
11-length	167960	167960
12-length	125970	125970
13-length	77520	77520
14-length	38760	38760
15-length	15504	15504
16-length	4845	4845
17-length	1140	1140
18-length	190	190
19-length	20	20
20-length	1	1
Total	1048575	1048575

(c)

	Support	**0.07**
	Cofisa	**Apriori**
1-length	20	20
2-length	190	190
3-length	1140	1140
4-length	4845	4845
5-length	15504	15504
6-length	38760	38760
7-length	77520	77520
8-length	125970	125970
9-length	167960	167960
10-length	184756	184756
11-length	167960	167960
12-length	125970	125970
13-length	77520	77520
14-length	38760	38760
15-length	15504	15504
16-length	4845	4845
17-length	1140	1140
18-length	190	190
19-length	20	20
20-length	1	1
Total	1048575	1048575

(d)

Figure A. 14: Level-wise frequent itemset counts(on 1K_20I_16 dataset) - continues

	Support	0.1
	Cofisa	Apriori
1-length	20	20
2-length	190	190
3-length	1140	1140
4-length	4845	4845
5-length	15504	15504
6-length	38760	38760
7-length	77520	77520
8-length	125970	125970
9-length	167960	167960
10-length	184756	184756
11-length	167960	167960
12-length	125970	125970
13-length	77520	77520
14-length	38760	38760
15-length	15504	15504
16-length	4845	4845
17-length	1140	1140
18-length	190	190
19-length	20	20
20-length	1	1
Total	1048575	1048575

(e)

	Support	0.3
	Cofisa	Apriori
1-length	20	20
2-length	142	188
3-length	684	1089
4-length	2381	4219
5-length	6188	11845
6-length	12376	26525
7-length	19448	46770
8-length	24310	62020
9-length	24310	64247
10-length	19448	54748
11-length	12376	35689
12-length	6188	16339
13-length	2380	5329
14-length	680	1334
15-length	136	256
16-length	17	33
17-length	1	2
Total	131085	330653

(f)

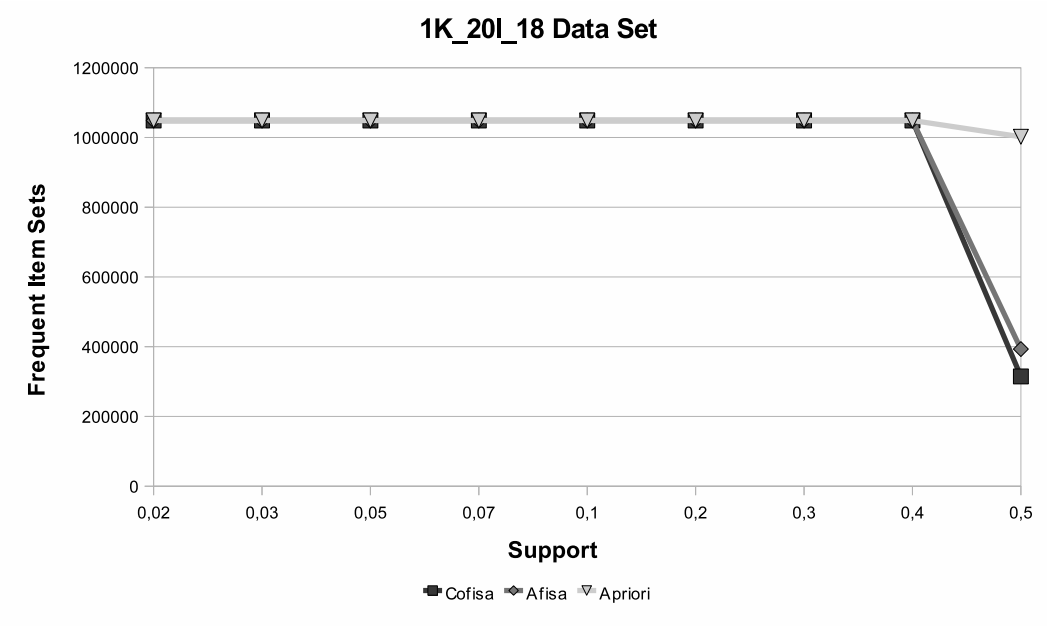
	Support	0.4
	Cofisa	Apriori
1-length	19	19
2-length	133	169
3-length	589	921
4-length	1805	3370
5-length	4089	8575
6-length	7069	16654
7-length	9467	25080
8-length	9875	27173
9-length	8009	20590
10-length	5005	11523
11-length	2366	5145
12-length	819	1816
13-length	196	415
14-length	29	48
15-length	2	2
Total	49472	121500

(g)

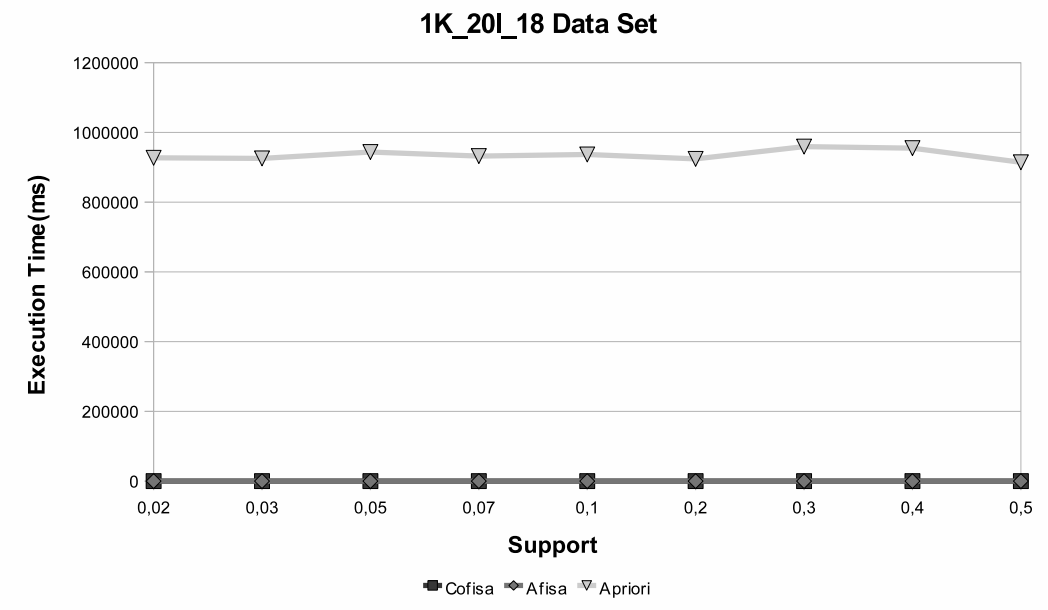
	Support	0.5
	Cofisa	Apriori
1-length	18	18
2-length	107	151
3-length	376	750
4-length	884	2399
5-length	1444	4984
6-length	1694	7575
7-length	1450	9062
8-length	905	7357
9-length	404	3770
10-length	123	1229
11-length	23	249
12-length	2	26
Total	7430	37570

(h)

Figure A. 15: Level-wise frequent itemset counts(on 1K_20I_16 dataset) - continues



(a) **Frequent itemsets**



(b) **Execution time**

Figure A. 16: Test results on a very small dataset

	Support	**0.02**
	Cofisa	**Apriori**
1-length	20	20
2-length	190	190
3-length	1140	1140
4-length	4845	4845
5-length	15504	15504
6-length	38760	38760
7-length	77520	77520
8-length	125970	125970
9-length	167960	167960
10-length	184756	184756
11-length	167960	167960
12-length	125970	125970
13-length	77520	77520
14-length	38760	38760
15-length	15504	15504
16-length	4845	4845
17-length	1140	1140
18-length	190	190
19-length	20	20
20-length	1	1
Total	1048575	1048575

(a)

	Support	**0.03**
	Cofisa	**Apriori**
1-length	20	20
2-length	190	190
3-length	1140	1140
4-length	4845	4845
5-length	15504	15504
6-length	38760	38760
7-length	77520	77520
8-length	125970	125970
9-length	167960	167960
10-length	184756	184756
11-length	167960	167960
12-length	125970	125970
13-length	77520	77520
14-length	38760	38760
15-length	15504	15504
16-length	4845	4845
17-length	1140	1140
18-length	190	190
19-length	20	20
20-length	1	1
Total	1048575	1048575

(b)

Figure A. 17: Level-wise frequent itemset counts(on 1K_20I_18 dataset)

	Support	0.05
	Cofisa	Apriori
1-length	20	20
2-length	190	190
3-length	1140	1140
4-length	4845	4845
5-length	15504	15504
6-length	38760	38760
7-length	77520	77520
8-length	125970	125970
9-length	167960	167960
10-length	184756	184756
11-length	167960	167960
12-length	125970	125970
13-length	77520	77520
14-length	38760	38760
15-length	15504	15504
16-length	4845	4845
17-length	1140	1140
18-length	190	190
19-length	20	20
20-length	1	1
Total	1048575	1048575

(c)

	Support	0.07
	Cofisa	Apriori
1-length	20	20
2-length	190	190
3-length	1140	1140
4-length	4845	4845
5-length	15504	15504
6-length	38760	38760
7-length	77520	77520
8-length	125970	125970
9-length	167960	167960
10-length	184756	184756
11-length	167960	167960
12-length	125970	125970
13-length	77520	77520
14-length	38760	38760
15-length	15504	15504
16-length	4845	4845
17-length	1140	1140
18-length	190	190
19-length	20	20
20-length	1	1
Total	1048575	1048575

(d)

Figure A. 18: Level-wise frequent itemset counts(on 1K_20I_18 dataset) - continues

	Support	**0.1**
	Cofisa	**Apriori**
1-length	20	20
2-length	190	190
3-length	1140	1140
4-length	4845	4845
5-length	15504	15504
6-length	38760	38760
7-length	77520	77520
8-length	125970	125970
9-length	167960	167960
10-length	184756	184756
11-length	167960	167960
12-length	125970	125970
13-length	77520	77520
14-length	38760	38760
15-length	15504	15504
16-length	4845	4845
17-length	1140	1140
18-length	190	190
19-length	20	20
20-length	1	1
Total	1048575	1048575

(e)

	Support	**0.2**
	Cofisa	**Apriori**
1-length	20	20
2-length	190	190
3-length	1140	1140
4-length	4845	4845
5-length	15504	15504
6-length	38760	38760
7-length	77520	77520
8-length	125970	125970
9-length	167960	167960
10-length	184756	184756
11-length	167960	167960
12-length	125970	125970
13-length	77520	77520
14-length	38760	38760
15-length	15504	15504
16-length	4845	4845
17-length	1140	1140
18-length	190	190
19-length	20	20
20-length	1	1
Total	1048575	1048575

(f)

Figure A. 19: Level-wise frequent itemset counts(on 1K_20I_18 dataset) - continues

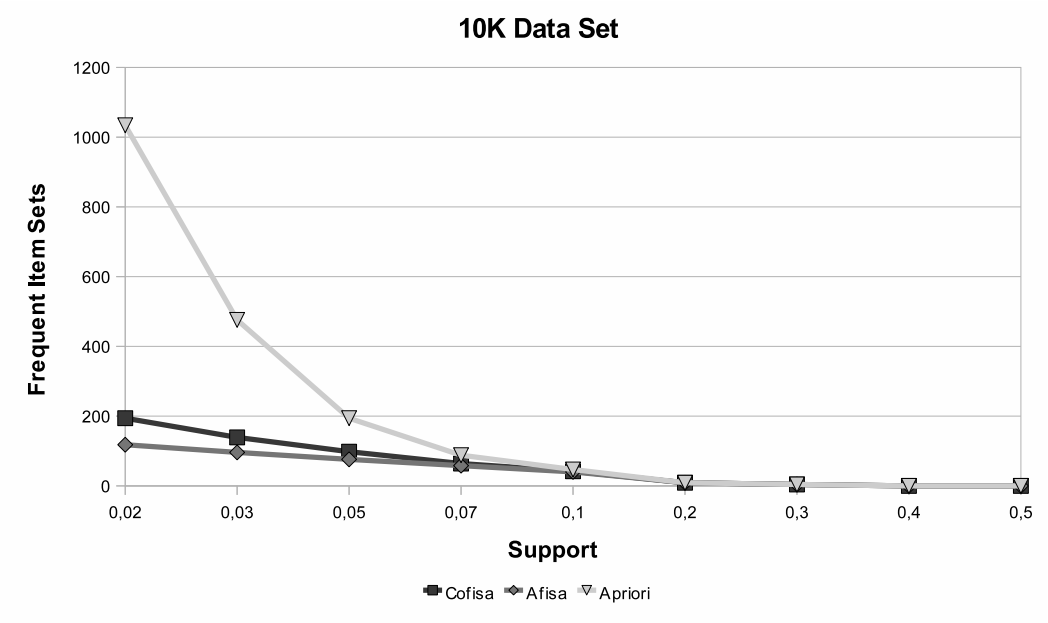
	Support	**0.3**
	Cofisa	**Apriori**
1-length	20	20
2-length	190	190
3-length	1140	1140
4-length	4845	4845
5-length	15504	15504
6-length	38760	38760
7-length	77520	77520
8-length	125970	125970
9-length	167960	167960
10-length	184756	184756
11-length	167960	167960
12-length	125970	125970
13-length	77520	77520
14-length	38760	38760
15-length	15504	15504
16-length	4845	4845
17-length	1140	1140
18-length	190	190
19-length	20	20
20-length	1	1
Total	1048575	1048575

(g)

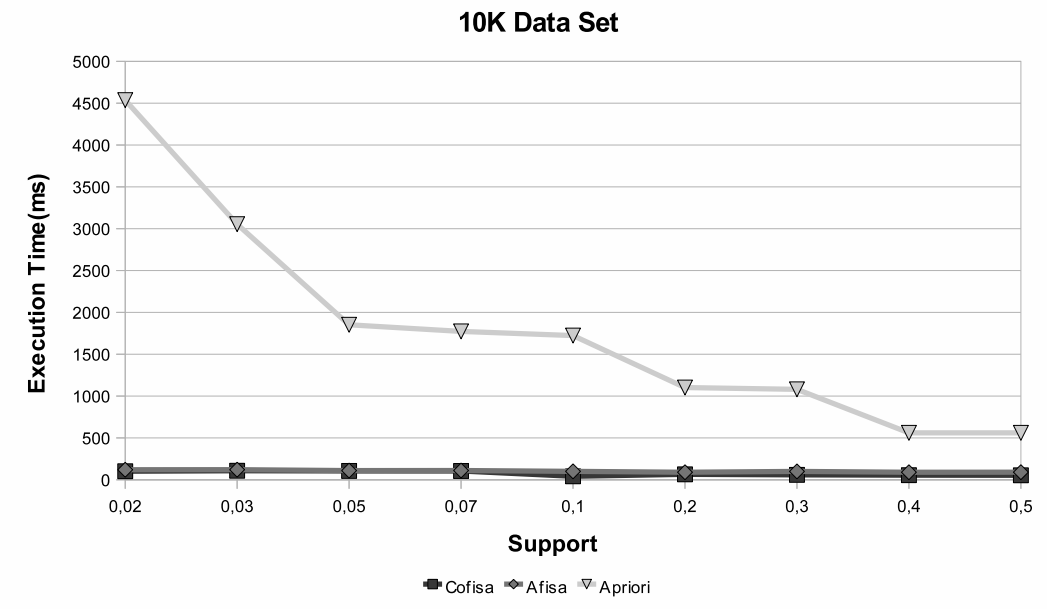
	Support	**0.4**
	Cofisa	**Apriori**
1-length	20	20
2-length	190	190
3-length	1140	1140
4-length	4845	4845
5-length	15504	15504
6-length	38760	38760
7-length	77520	77520
8-length	125970	125970
9-length	167960	167960
10-length	184756	184756
11-length	167960	167960
12-length	125970	125970
13-length	77520	77520
14-length	38760	38760
15-length	15504	15504
16-length	4845	4845
17-length	1140	1140
18-length	190	190
19-length	20	20
20-length	1	1
Total	1048575	1048575

(h)

Figure A. 20: Level-wise frequent itemset counts(on 1K_20I_18 dataset) - continues



(a) **Frequent itemsets**



(b) **Execution time**

Figure A. 21: Test results on a small dataset

	Support	**0.02**
	Cofisa	**Apriori**
1-length	87	87
2-length	78	736
3-length	29	210
4-length	1	1
Total	195	1034

(a)

	Support	**0.03**
	Cofisa	**Apriori**
1-length	80	80
2-length	53	361
3-length	6	35
Total	139	476

(b)

	Support	**0.05**
	Cofisa	**Apriori**
1-length	68	68
2-length	32	123
3-length	1	3
Total	101	194

(c)

	Support	**0.07**
	Cofisa	**Apriori**
1-length	54	54
2-length	11	34
Total	65	88

(d)

	Support	**0.1**
	Cofisa	**Apriori**
1-length	38	38
2-length	4	9
Total	42	47

(e)

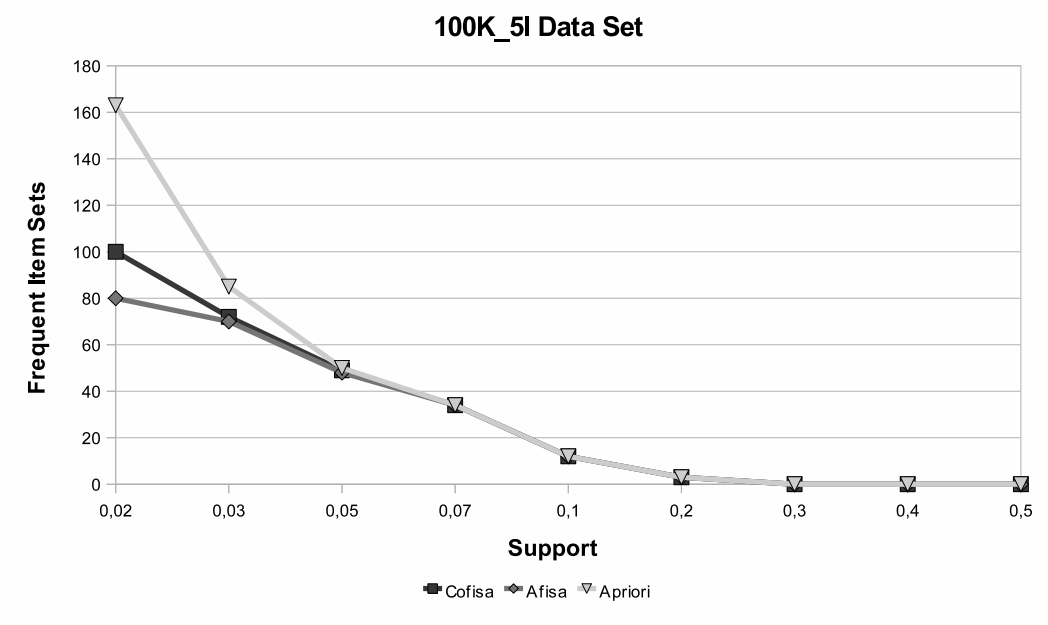
	Support	**0.2**
	Cofisa	**Apriori**
1-length	9	9
Total	9	9

(f)

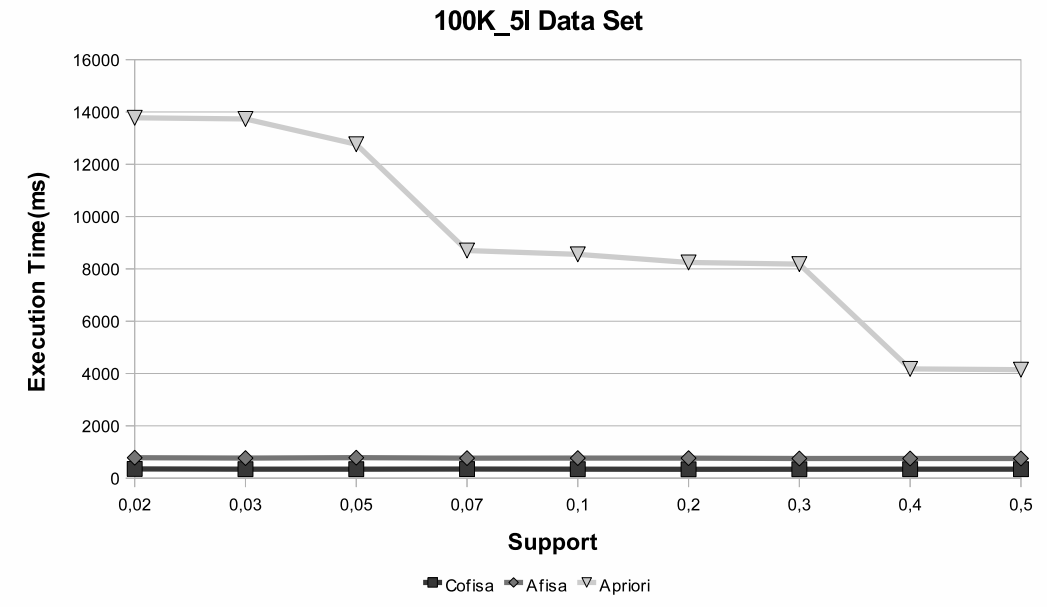
	Support	**0.3**
	Cofisa	**Apriori**
1-length	4	4
Total	4	4

(g)

Figure A. 22: Level-wise frequent itemset counts(on 10K dataset)



(a) **Frequent itemsets**



(b) **Execution time**

Figure A. 23: Test results on a medium size dataset

	Support	**0.02**
	Cofisa	**Apriori**
1-length	76	76
2-length	24	87
Total	100	163

(a)

	Support	**0.03**
	Cofisa	**Apriori**
1-length	66	66
2-length	7	19
Total	73	85

(b)

	Support	**0.05**
	Cofisa	**Apriori**
1-length	47	47
2-length	2	3
Total	49	50

(c)

	Support	**0.07**
	Cofisa	**Apriori**
1-length	34	34
Total	34	34

(d)

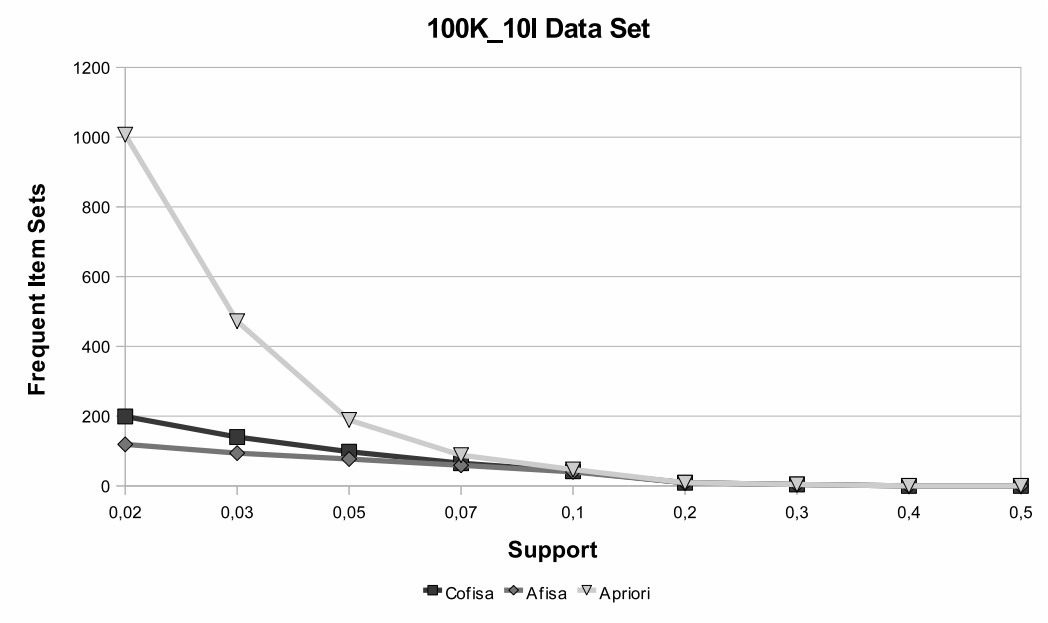
	Support	**0.1**
	Cofisa	**Apriori**
1-length	12	12
Total	12	12

(e)

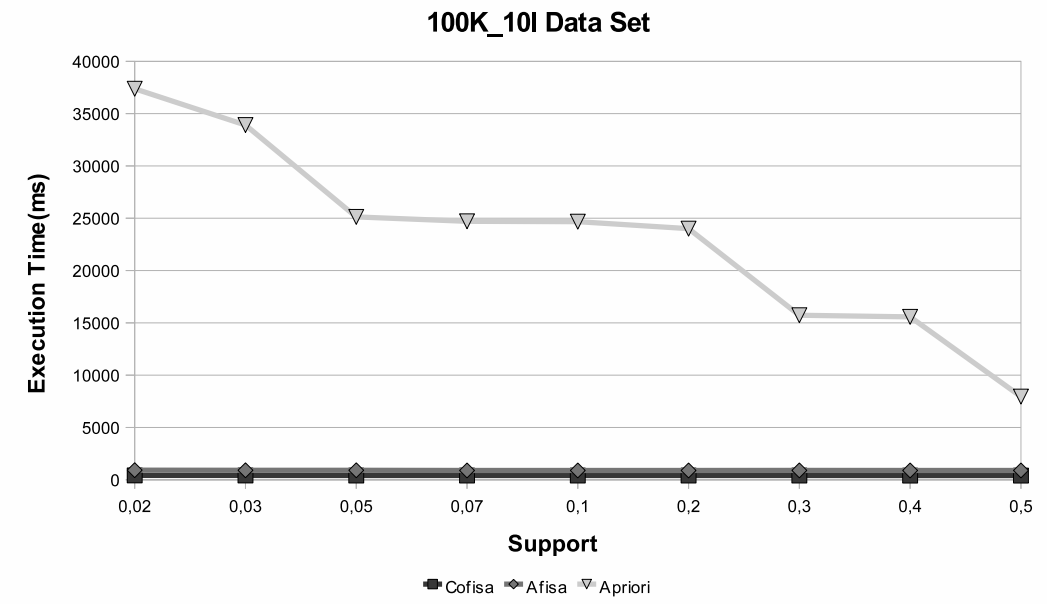
	Support	**0.2**
	Cofisa	**Apriori**
1-length	3	3
Total	3	3

(f)

Figure A. 24: Level-wise frequent itemset counts(on 100K_5 dataset)



(a) **Frequent itemsets**



(b) **Execution time**

Figure A. 25: Test results on a medium size dataset

	Support	**0.02**
	Cofisa	**Apriori**
1-length	88	88
2-length	84	726
3-length	25	192
4-length	0	1
Total	197	1007

(a)

	Support	**0.03**
	Cofisa	**Apriori**
1-length	78	78
2-length	54	360
3-length	6	34
Total	138	472

(b)

	Support	**0.05**
	Cofisa	**Apriori**
1-length	68	68
2-length	31	118
3-length	1	3
Total	100	189

(c)

	Support	**0.07**
	Cofisa	**Apriori**
1-length	55	55
2-length	9	33
Total	64	88

(d)

	Support	**0.1**
	Cofisa	**Apriori**
1-length	38	38
2-length	4	9
Total	42	47

(e)

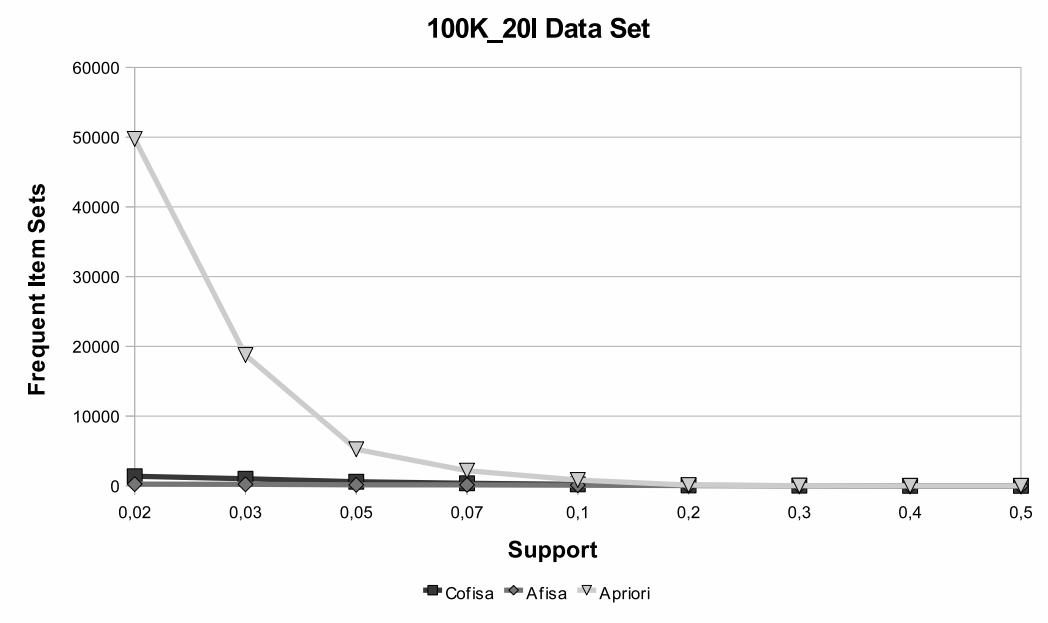
	Support	**0.2**
	Cofisa	**Apriori**
1-length	9	9
Total	9	9

(f)

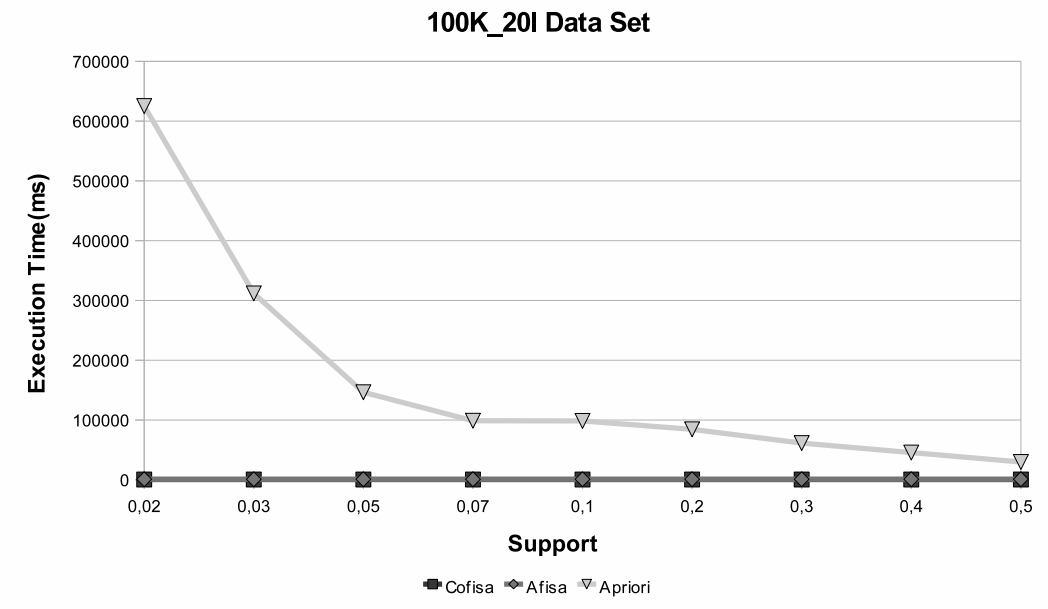
	Support	**0.3**
	Cofisa	**Apriori**
1-length	4	4
Total	4	4

(g)

Figure A. 26: Level-wise frequent itemset counts(on 100K_10 dataset)



(a) **Frequent itemsets**



(b) **Execution time**

Figure A. 27: Test results on medium size dataset

	Support	0.02
	Cofisa	Apriori
1-length	91	91
2-length	282	2641
3-length	450	16294
4-length	378	22728
5-length	162	7510
6-length	28	476
Total	1391	49740

(a)

	Support	0.03
	Cofisa	Apriori
1-length	90	90
2-length	251	2059
3-length	348	8461
4-length	242	6920
5-length	81	1209
6-length	10	21
Total	1022	18790

(b)

	Support	0.05
	Cofisa	Apriori
1-length	86	86
2-length	193	1308
3-length	200	2794
4-length	96	1003
5-length	18	56
Total	593	5247

(c)

	Support	0.07
	Cofisa	Apriori
1-length	77	77
2-length	143	837
3-length	109	1051
4-length	34	204
5-length	2	3
Total	365	2172

(d)

	Support	0.1
	Cofisa	Apriori
1-length	69	69
2-length	89	447
3-length	47	301
4-length	8	21
Total	213	838

(e)

	Support	0.2
	Cofisa	Apriori
1-length	38	38
2-length	23	64
3-length	3	7
Total	64	109

(f)

	Support	0.3
	Cofisa	Apriori
1-length	25	25
2-length	4	9
Total	29	34

(g)

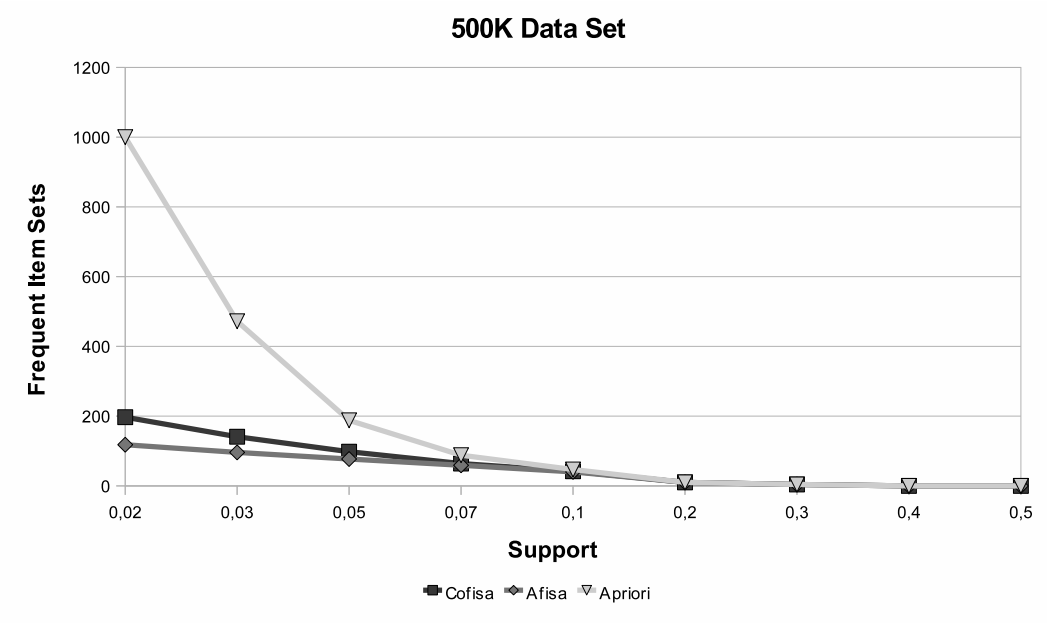
	Support	0.4
	Cofisa	Apriori
1-length	7	7
2-length	1	1
Total	8	8

(h)

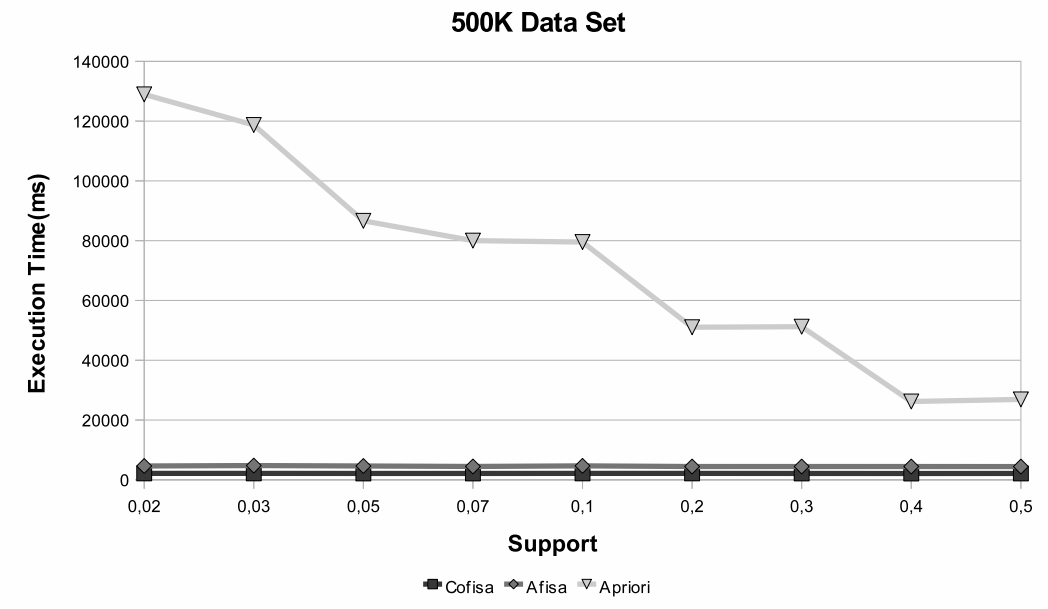
	Support	0.1
	Cofisa	Apriori
1-length	5	5
Total	5	5

(i)

Figure A. 28: Level-wise frequent itemset counts(on 100K_20 dataset)



(a) **Frequent itemsets**



(b) **Execution time**

Figure A. 29: Test results on a large data set

	Support	**0.02**
	Cofisa	**Apriori**
1-length	87	87
2-length	82	730
3-length	25	182
4-length	0	1
Total	194	1000

(a)

	Support	**0.03**
	Cofisa	**Apriori**
1-length	78	78
2-length	56	361
3-length	6	33
Total	140	472

(b)

	Support	**0.05**
	Cofisa	**Apriori**
1-length	68	68
2-length	31	118
3-length	1	2
Total	100	188

(c)

	Support	**0.07**
	Cofisa	**Apriori**
1-length	55	55
2-length	10	33
Total	65	88

(d)

	Support	**0.1**
	Cofisa	**Apriori**
1-length	38	38
2-length	3	9
Total	41	47

(e)

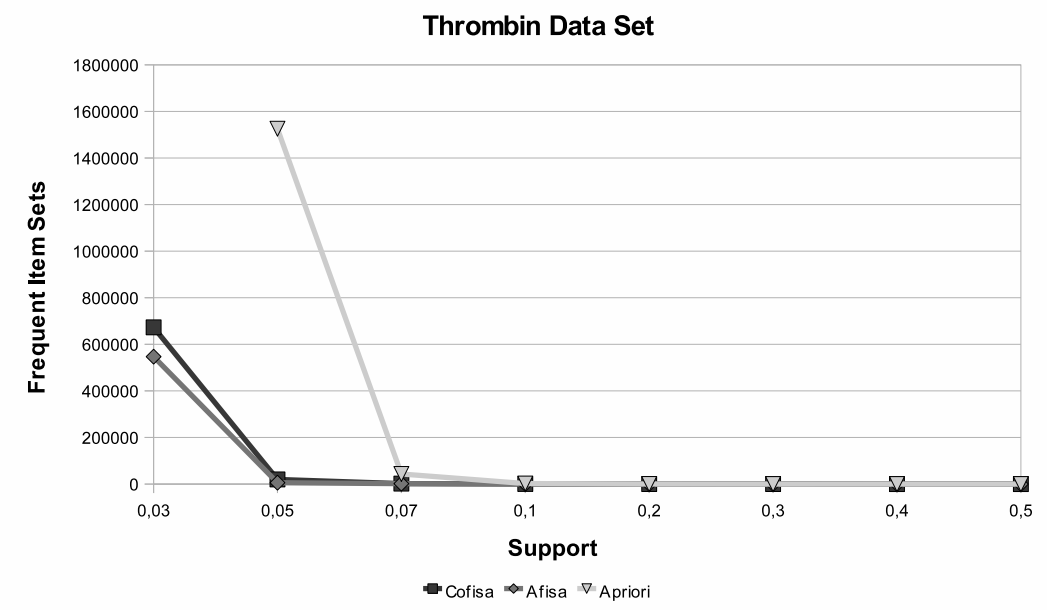
	Support	**0.2**
	Cofisa	**Apriori**
1-length	10	10
Total	10	10

(f)

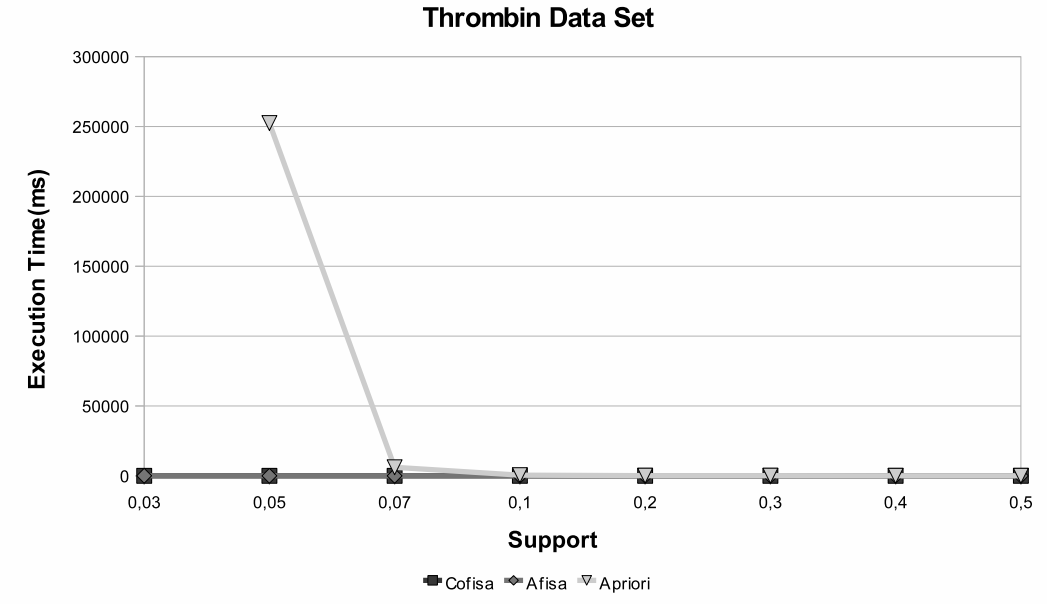
	Support	**0.3**
	Cofisa	**Apriori**
1-length	4	4
Total	4	4

(g)

Figure A. 30: Level-wise frequent itemset counts(on 500K dataset)



(a) **Frequent itemsets**



(b) **Execution time**

Figure A. 31: Test results on real data sets

	Support	0.05
	Cofisa	**Apriori**
1-length	63	63
2-length	254	1145
3-length	753	8827
4-length	1646	37769
5-length	2767	107161
6-length	3660	214585
7-length	3842	310844
8-length	3187	331420
9-length	2059	262826
10-length	1012	155474
11-length	365	68151
12-length	91	21718
13-length	14	4843
14-length	1	704
15-length	0	58
16-length	0	2
Total	19714	1525590

(a)

	Support	0.07
	Cofisa	**Apriori**
1-length	55	55
2-length	164	685
3-length	316	2860
4-length	436	6750
5-length	438	10486
6-length	319	10853
7-length	163	7388
8-length	55	3186
9-length	11	794
10-length	1	98
11-length	0	4
Total	1958	43159

(b)

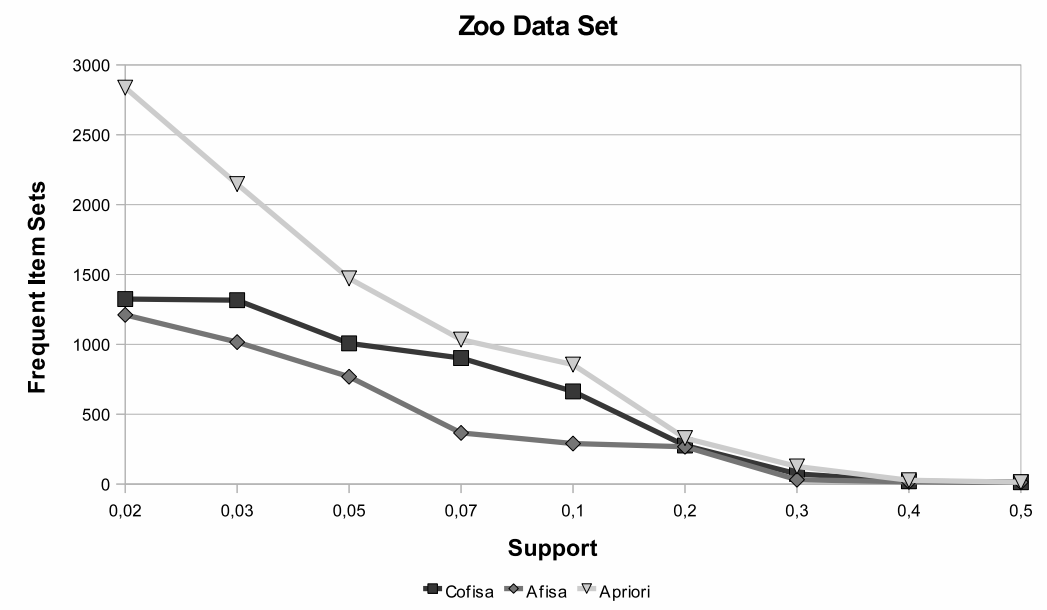
	Support	0.1
	Cofisa	**Apriori**
1-length	45	45
2-length	70	241
3-length	70	594
4-length	42	726
5-length	14	426
6-length	2	101
7-length	0	7
Total	243	2140

(c)

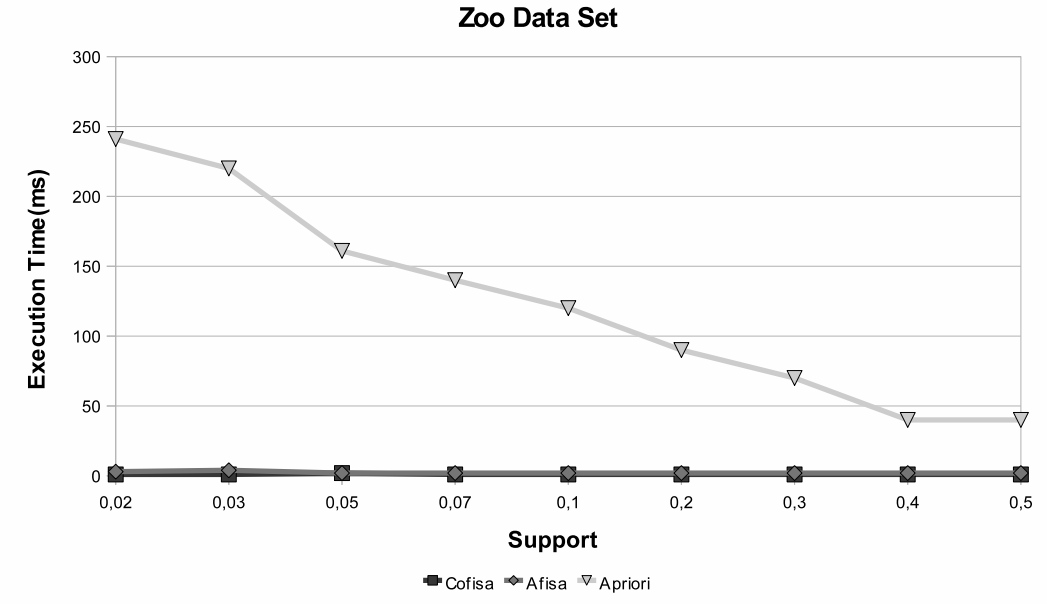
	Support	0.2
	Cofisa	**Apriori**
1-length	11	11
2-length	1	4
Total	12	15

(d)

Figure A. 32: Level-wise frequent itemset counts(on thrombin dataset)



(a) **Frequent itemsets**



(b) **Execution time**

Figure A. 33: Test results on real data sets

	Support	**0.02**
	Cofisa	**Apriori**
1-length	19	19
2-length	99	126
3-length	247	395
4-length	365	699
5-length	348	769
6-length	218	535
7-length	87	231
8-length	20	57
9-length	2	6
Total	1405	2837

(a)

	Support	**0.03**
	Cofisa	**Apriori**
1-length	19	19
2-length	91	122
3-length	229	346
4-length	350	561
5-length	342	568
6-length	217	358
7-length	87	139
8-length	20	31
9-length	2	3
Total	1000	2147

(b)

	Support	**0.05**
	Cofisa	**Apriori**
1-length	19	19
2-length	83	107
3-length	191	280
4-length	265	411
5-length	238	368
6-length	140	204
7-length	52	69
8-length	11	13
9-length	1	1
Total	1000	1472

(c)

	Support	**0.07**
	Cofisa	**Apriori**
1-length	19	19
2-length	77	90
3-length	171	214
4-length	230	288
5-length	203	238
6-length	119	128
7-length	45	46
8-length	10	10
9-length	1	1
Total	875	1034

(d)

	Support	**0.1**
	Cofisa	**Apriori**
1-length	17	17
2-length	66	80
3-length	137	181
4-length	176	233
5-length	153	190
6-length	92	105
7-length	37	39
8-length	9	9
9-length	1	1
Total	688	855

(e)

	Support	**0.2**
	Cofisa	**Apriori**
1-length	14	14
2-length	36	49
3-length	63	86
4-length	72	85
5-length	56	58
6-length	28	28
7-length	8	8
8-length	1	1
Total	278	329

(f)

Figure A. 34: Level-wise frequent itemset counts(on zoo dataset)

	Support	0.3
	Cofisa	Apriori
1-length	11	11
2-length	25	34
3-length	29	44
4-length	18	28
5-length	6	8
6-length	1	1
Total	90	126

(g)

	Support	0.4
	Cofisa	Apriori
1-length	9	9
2-length	9	13
3-length	3	5
Total	21	27

(h)

	Support	0.1
	Cofisa	Apriori
1-length	6	6
2-length	5	5
3-length	2	2
Total	13	13

(i)

Figure A. 35: Level-wise frequent itemset counts(on zoo dataset) - continues

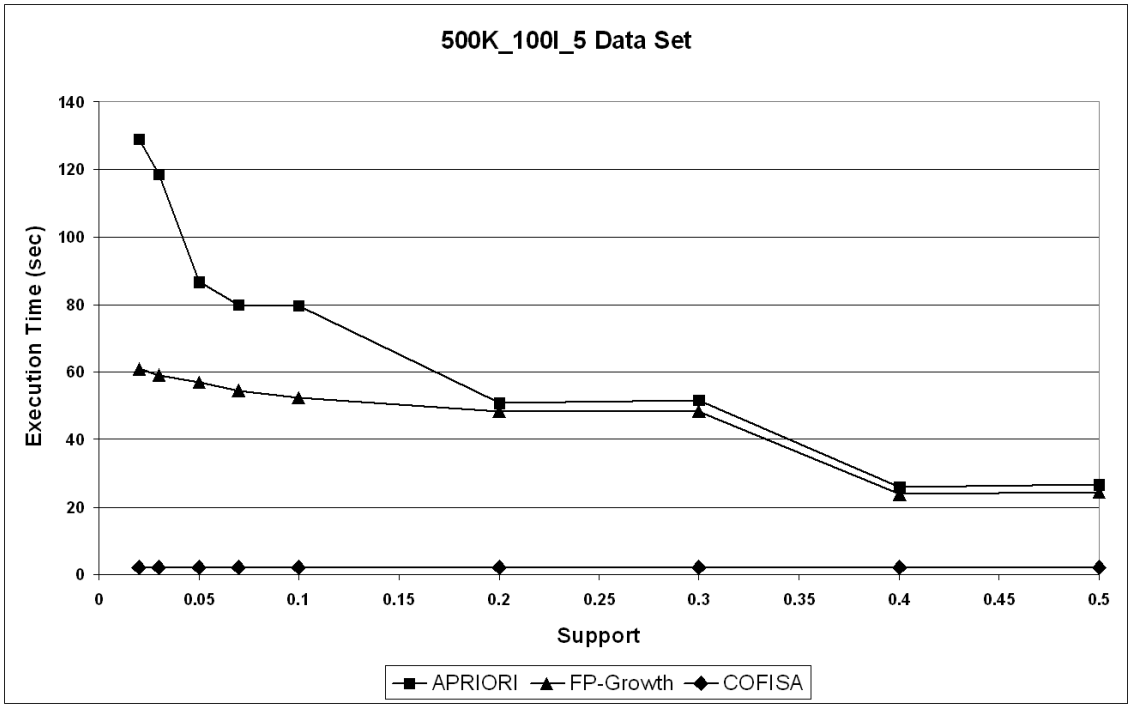


Figure A. 36: Execution time comparison of APRIORI, FP-Growth, and COFISA