DE TTK

1949

A Scalable Parallel Algorithm for Decision Support from Multidimensional Sequence Data

Egyetemi doktori (PhD) értekezés

Szerző: Esmaeili, Mahdi
Témavezető: Dr. Fazekas, Gábor

DEBRECENI EGYETEM
Természettudományi Doktori Tanács
Matematika és Számítástudományok Doktori Iskola
Debrecen, 2011.

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.
Hungary, Debrecen, June 2011

Mahdi Esmaeili

This is to certify that the thesis entitled "A Scalable Parallel Algorithm for Decision Support from Multidimensional Sequence Data" submitted by Mahdi Esmaeili to University of Debrecen for the award of the degree of Doctor of Philosophy is a bona fide record of the research work carried out by him under my(our) supervision and guidance. The content of the thesis, in full or parts have not been submitted to any other Institute or University for the award of any other degree or diploma.
Hungary, Debrecen, June 2011

Dr. Fazekas Gabor

A Scalable Parallel Algorithm for Decision Support
from Multidimensional Sequence Data

Értekezés a doktori (Ph.D.) fokozat megszerzése érdekében
a *matematika és számítástudományok* tudományágban

Írta: **Esmaeili, Mahdi** okleveles informatikus

Készült a Debreceni Egyetem *Matematika és Számítástudományok* doktori iskolája
(*Számítástudomány* programja) keretében

Témavezető: Dr. Fazekas Gábor

A doktori szigorlati bizottság:

elnök:	Dr. Maksa Gyula		....
tagok:	Dr. Kis Attila		....
	Dr. Várterész Magda		....

A doktori szigorlat időpontja: 2011. május 6.

Az értekezés bírálói:

	Dr.	....
	Dr.	....
	Dr.	....

A bírálóbizottság:

elnök:	Dr.	....
tagok:	Dr.	....
	Dr.	....
	Dr.	....
	Dr.	....

Az értekezés védésének időpontja: 201… . ……………… … .

Table of Contents

Abstract

Data mining is the task of discovering interesting patterns from large amounts of data. In other words, data mining is the process of extracting implicit knowledge and discovery of interesting patterns that are not explicitly represented in the databases. There are many data mining tasks, such as classification, clustering, association rule mining, and sequential pattern mining. Sequential pattern mining finds sets of data items that occur together frequently in some sequences. Sequential pattern mining, which extracts frequent subsequences from a sequence database, has attracted a great deal of interest during the recent data mining researches because it is the basis of many applications, such as: web user analysis, stock trend prediction, DNA sequence analysis, finding language or linguistic patterns from natural language texts, and using the history of symptoms to predict certain kind of disease.

The diversity of the applications may not be possible to apply a single sequential pattern model to all these problems. Each application may require a unique model and solution. A number of research projects were established in recent years to develop meaningful sequential pattern models and efficient algorithms for mining these patterns.

Besides mining sequential patterns in a single dimension, mining multidimensional sequential patterns can give us more informative and useful patterns. In a multidimensional sequence dataset each event depends on more than one dimension.

The search space is quite large and the serial algorithms are not scalable for very large datasets. Due to the huge increase in data volume and also quite large search space, it is necessary to study scalable parallel implementations of sequence mining algorithms.

In this work, we describe a multidimensional sequence model and then represent a parallel algorithm. We improve the primary parallel algorithm with two modification rules. Two approaches follow the level-wise approach and all participating processors or workers generate candidate sequences and count their supports independently.

Our experiments show good load balancing and scalable and acceptable speedup over different processors and problem sizes and demonstrate that our approach can works efficiently in a real parallel computing environment.

Keywords: Data Mining, Sequential Pattern Mining, Parallel Algorithm, and Multidimensional Sequence Data

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Overview

Frequent sets play an essential role in many data mining tasks that try to find interesting patterns from databases, such as association rules, correlations, sequences, episodes, and classifiers.

Data may be sequential or non sequential in nature. Non sequential data are those data where order of occurrence is not important. Sequential data are those data where order of occurrence is important to consider. Sequential data can be ordered with respect to time or some other dimension such as, space.

A sequential pattern is a subsequence that appears frequently in a sequence database. Mining sequential patterns in large databases has become an important data mining task with broad applications, such as business analysis, web mining, customer shopping sequences, security, and bio-sequences (discovery of motifs in DNA sequences) analysis [1].

Mining sequential patterns in sequence databases has been demonstrated to be useful and technically feasible in several application areas and it becomes every day more important in many applications. Although research in this area has been going on for more than one decade; today, mining such patterns is still one of the most popular methods in knowledge discovery and data mining.

In the last decade, a number of algorithms and techniques have been proposed to deal with the problem of sequential pattern mining [2][3][4][5][6]. The main approaches to sequential pattern mining, namely *Apriori*-based and pattern-growth methods are being used as the basis for other structured pattern mining algorithms. However, and despite the fact that pattern-growth algorithms have shown better performance in the majority of the situations, its advantages over *Apriori*-based methods are not sufficiently understood.

It is interesting and useful to mine sequential patterns associated with multidimensional information. In the other words, if sequential pattern mining can be associated with multidimensional information, it will be more effective. This situation exists in many practical application and real world. While many studies have contributed to find sequential patterns from sequence dataset, there is no significant number of works to mine patterns from multidimensional sequence dataset. This motivates our research and thesis of sequential pattern in multidimensional sequence data.

Although efficiency of mining the complete set of sequential patterns have been improved substantially, in many cases, sequential pattern mining still faces tough challenges in both effectiveness and efficiency.

There could be a large number of sequential patterns in a huge database, especially multidimensional sequence database. A user is often interested in only a small subset of such patterns. Presenting the complete set of sequential patterns may make the mining result hard to understand and hard to use. Besides, although efficient algorithms have been proposed, mining large datasets requires powerful computational resources. In fact, data mining algorithms working on very large datasets take a very long time on conventional computers to get results. One approach to reduce response time is sampling. But, in some cases reducing data might result in inaccurate models, in some other cases it is not useful.

Another approach is parallel computing. High performance computers and parallel data mining algorithms can offer a very efficient way to mine very large datasets by analyzing them in parallel.

1.2 Approach and Contributions

This section describes the contributions of this work to develop a model for multidimensional sequence data and parallelize this model. Parallelize multidimensional sequential pattern mining is the main subject of our thesis.

The main goal is to introduce data mining techniques on parallel architectures and to show how large scale data mining and knowledge discovery applications can achieve scalability by using systems, tools and performance offered by parallel processing systems. Our approach to efficient sequence mining is parallelization, where the whole computation is broken up into parallel tasks. Two common ways to partition computation are task and data partitioning. We explore data parallelism to extract patterns from sequence dataset.

The contributions made by this research are the following:

1. We present a framework for representing multidimensional sequence dataset. Since relational model is one the most popular way to store data, we identify a table as the main dataset with four basic groups of attributes. This model is simple and comprehensible and originally introduced in [7]. We published this result first in [8] and discussed regarding them.

2. Developing one parallel method that utilizing data parallelism. We present a framework to partition data and we propose a parallel approach for extracting sequential patterns in multidimensional sequence dataset. From a data mining viewpoint, our method has several main advantages over task parallelism. This simplifies programming and leads to a development time significantly smaller than one associated with task parallel programming, because a lot of previously written serial code can be reused. Our algorithm has a higher degree of machine architecture independence, in comparison with task parallelism. In most applications, the amount of data can increase arbitrarily fast, while the number of lines of code typically increases at a much slower rate. In our published study [9], we demonstrated that the primary parallel algorithm scales well.
3. We improve primary approach with two modification rules. Using a suitable distribution of dataset and an approximate way to find patterns led to have good performance for improved algorithm. In our published papers [8][9], we demonstrated that algorithm can greatly reduce the number of scans through the sequence dataset and a good work loading as well. In fact, it tries to optimize local mining at processors while minimizing the data transfer among them.
4. Primary parallel approach and improved method that we propose in this thesis are implemented and simulated into MATLAB software tools grouped in the Parallel Computing toolbox software and MATLAB Distributed Computing Server software. This toolbox will allow coordinating and executing independent MATLAB operations simultaneously on a cluster of computers, speeding up execution of large

MATLAB jobs. The more information about our experience of using MATLAB toolbox has been published in technical report [10].

These contributions are presented in the following chapters of this thesis. A brief summary of these chapters is provided in the next section.

1.3 Organization of Thesis

In this section, we summarize the content of the remaining chapters of this thesis. The chapters are described briefly as follows:

Chapter 2 provides an introduction to knowledge discovery process. Kind of data to be mined plays important role to choose preprocessing step. We briefly describe some types of datasets, data mining tasks, and several data mining applications.

Chapter 3 discusses previous attempts to parallelize algorithms to data mining field. We also survey challenges and future researches in parallel data mining.

Chapter 4 introduces sequence mining and related works. We provide a brief overview of models of sequential patterns and theoretically we show kinds of sequential patterns and some properties of them. In published work [11], we show that there is fewer propositions in deal with multidimensional sequence datasets and it can be one of the most important motivations and advantages of our approach.

Chapter 5 describes a model for multidimensional sequence data and based on it we present a parallel framework for mining patterns. Then we introduce two modification rules to improve the main parallel algorithm. This chapter explores data parallelism in model and presents experimental results.

Chapter 6 summarizes the whole thesis and gives directions for future works.

Chapter 2

Knowledge Discovery

Traditional methods of data analysis, based mainly on a person dealing directly with the data, do not scale to enormous datasets. While database technology has provided us with the basic tools for the efficient storage and retrieve of large datasets, the issue of how to help humans analyze data remains a difficult and unsolved problem.

However, it is clear that the manual analysis of such amounts of data is impossible. For this reason, it is important to develop tools that can access and analyze such data to extract interesting knowledge that can be helpful in the decision making process. Knowledge discovery is a technique that can discover interesting information from such large databases. ”Knowledge discovery in databases (KDD) is the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data” [12]. The process is nontrivial means that it is not straightforward and some methodologies need to be used to obtain patterns. The extracted patterns should be novel or in other words unknown previously. They also should be useful, this usefulness implies that the patterns can play important role in the decision making process. Finally, they must be understandable at least for the data analyst.

2.1 The Knowledge Discovery Process

The KDD process is interactive and iterative, involving a number of steps with many decisions made by the user [12]. The knowledge discovery process begins by understanding the application domain and the goals of the data mining process. Here we list the main steps of the knowledge discovery process [1]:

- Data Preprocessing or Data Preparing: Main tasks of this step including data cleaning, data integration, data reduction, and data transformation. Data cleaning algorithms eliminate noise and inconsistent data. In data integration methods target datasets are gathered from multiple data sources for example from heterogeneous databases and data warehouses and may be combined. The dataset may be huge and data reduction ways try to select relevant data from datasets to analyze. The data are transformed or consolidated in a suitable manner for mining by data transformation algorithms.
- Data Mining: Data mining is the most important step in the knowledge discovery process in which different techniques are applied to extract interesting patterns. In this step, the appropriate data mining algorithms are decided to be applied on the target data.
- Pattern Evaluation: In this step, based on some interestingness and predefined measures, all truly interesting patterns in which represent meaningful knowledge are identified.

- Knowledge Presentation: To better understand and analyze the discovered knowledge different visualization techniques are used to present the mined knowledge to the user.

Most of these steps are often repeated iteratively to reach some satisfying level of the expected results. For example, in the data mining step the data analyst may repeat the data mining process by using another method to obtain better and refined results of the knowledge discovery process. The data analyst can also jump to different steps. For example, if the data analyst is using some data mining algorithms and needs different data format, then he/she may go back to the data preprocessing step in order to convert the target data to the suitable format. In the next sections, we discuss data mining as the most important step in the knowledge discovery process.

2.2 Data Mining

Data mining is considered the most important step in the knowledge discovery process. Data mining is the process of extracting interesting patterns from large amounts of data [1]. Data mining is about solving problems by analyzing data already present in datasets. It provides tools for automated learning from historical data and developing models to predict future trends and behaviors. The essence of data mining is the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data.

Data mining functionalities are used to specify the kind of patterns to be found in data mining tasks. In general, data mining tasks can be classified into two categories: descriptive and predictive. Descriptive data mining models try to extract useful patterns that can describe the mined data and explore its properties and characterize the general properties of the data in the database. Predictive mining tasks perform inference on the current data in order to make predictions, in other words, try to predict unknown values or future trends or behaviors depending on other variables or historical values presented in the mined database [1].

Data mining is widely used by different companies and organizations especially in the field of medical, retail, marketing, finance, communication, and science. It enables these companies to gain information about diseases, sales behavior, customer satisfaction, and corporate profits. Using data mining, organizations can increase the profitability of their interactions with patients, customers, and improve risk management in marketing.

The extracted patterns using data mining help organizations to have better decisions. In the following sections, we examine a number of different data repositories on which mining can be performed, data mining methods and tasks, and some data mining applications.

2.3 Data Mining and Kind of Data

Data mining is not restricted to one kind of data and should be applicable to any kind of data repositories. We remember that different techniques may be suitable to be applied on one kind of data but not for other. Data mining can be applied to flat files, relational databases, data warehouses, transactional databases, and advanced data and information systems such as multimedia databases, spatial databases, time series databases, and the World Wide Web. Here we discuss briefly each of them.

- Flat Files: Flat files are the most used data source in data mining. This kind of data may be the simplest type for example, a text file with a comma separated format. It is normally a simple data file with text or binary format with a structure that can be recognized by the used data mining algorithm.
- Relational Databases: In most organizations today, data are stored in relational databases. A relational database consists of a set of tables. Each table has a number of columns (attributes or fields) and rows (tuples or records). Each tuple in a relational table represents an object or a relationship between objects and identifies by a set of attributes that represent a unique key. SQL is the most common language for this kind of data. Running data mining algorithms using relational databases can be more efficient and easier than using flat files due to the ability to take advantage of the functionalities that are provided by SQL.
- Data Warehouses: Data warehouses were conceived as a means to facilitate the compilation of regular reports on the status of the business by continuously collecting, cleaning, and summarizing the core data of the organization. According to [13], a data warehouse is defined as:"a subject oriented, integrated, time variant and nonvolatile collection of data in support of management's decision making process". A data warehouse is a repository of data collected from multiple data sources and stored under a unified schema at a single site. Data warehouses provide a clean and organized source of data for data mining. Data warehouses are usually modeled as a multidimensional data structure (data cubes), where each dimension represents a set of attributes in the schema, and each cell stores

a value of some aggregate measures [14]. A data cube provides a multidimensional view of data and allows the precomputation and fast accessing of summarized data. For this reason, data warehouse systems are well suited for online analytical processing (OLAP). OLAP allows the navigation of data at different levels of abstraction, such as, drill-down, roll-up, slice, dice, and pivot. As the data in the data warehouse are cleaned and integrated, many times do not need for the data to be further preprocessed and prepared to be mined.

- Transactional Databases: A transactional database is a set of records representing transactions, each with a unique identifier and a set of items. The transactional database may have additional information. One typical data mining application on such data is called Market Basket Analysis that uses mostly association rule mining to study the relationship between different items in a transactional database.
- Multimedia Databases: Multimedia databases store images, audio, and video. A multimedia object is high dimensional which makes data mining a challenging process. For multimedia data mining, storage and search techniques need to be integrated with standard data mining methods.
- Spatial Databases: This type of data contains spatial-related information. Examples include geographic (maps) databases, very large scale integration (VLSI) or computed aided design databases, and medical and satellite image databases.
- Time Series Databases: It is obvious that time stamp is an important attribute of each dataset, and it can give us more accurate and useful information and rules. A database consists of sequences of values or events

that change with time are called a time series database. Data mining can be applied on such kind of data to study the changing behavior of items with respect to time.

- The World Wide Web (WWW): The WWW is a heterogeneous and dynamic data repository. A heterogeneous database consists of a set of interconnected, autonomous component databases. The WWW includes a huge number of data types varying from text, audio, video, raw data, and application. The WWW is composed of three main parts: web content, structure of the web, and usage of the web. The web content represents the documents available on the web. The structure of the web is represented by the hyperlinks and the relationships between different documents in the web. The usage of the web describes how the web documents and resources are being accessed and used.

2.4 Data Mining Tasks

Using data mining task depends on the application domain and the structure of the patterns that are expected to be extracted. We should know that there is no single method which is appropriate for all possible problems. The universal method is an idealistic fantasy. In the following we discuss basic data mining tasks.

2.4.1 Mining Frequent Patterns and Association Rule Mining

The discovery of frequent patterns, association, and correlation relationships among huge amounts of data are useful in many applications.

An association rule is an expression of the form $X \rightarrow Y$, where X and Y are itemsets and have no items in common. Rule support and confidence are two common measures of rule interestingness. A rule $X \rightarrow Y$ denotes that whenever a transaction T in transactional database D contains X then there is a probability that it contains Y too. The rule has support α in D if α% of the transactions in D contains both X and Y and holds confidence β if β% of transactions in D that contain X also contain Y.

Typically, association rules are considered interesting if they satisfy both a minimum support threshold (*Min_sup*) and also a minimum confidence threshold (*Min_conf*). Such thresholds can be set by users or domain experts. Association rule mining is finding all association rules of which support and confidence are greater than or equal to *Min_sup* and *Min_conf*, respectively.

In general, the process of extracting interesting association rules consists of two major steps. The first step is finding all itemsets that satisfy *Min_sup* threshold. The second step is generating all association rules that satisfy *Min_conf* using itemsets generated in the previous step.

Association rule mining algorithms employ one of two common approaches: Breadth First Search approach (BFS), and Depth First Search approach (DFS). In BFS approach, the support values of all k-itemsets are determined before counting the support values of the $(k+1)$-itemsets where k is a positive integer. In DFS approach the algorithm expands the $(k+1)$-itemsets before computing all of frequent k-itemsets until an infrequent node is reached. It then backtracks and continues the search from expanding remaining k-itemsets.

Many efficient and scalable algorithms have been developed for frequent itemset mining, from which association and correlation rules can be derived. These algorithms can be classified into three categories:

(1) *Apriori* based algorithms

(2) Frequent pattern growth based algorithms, such as FP-growth

(3) Algorithms that use the vertical data format

The *Apriori* based algorithms follow the BFS approach. The algorithm makes a single pass over the data to determine the support of each item which results are in the set of *1*-itemsets. Next, the algorithm will iteratively generates new candidate *k*-itemsets using the frequent *(k−1)*-itemsets found in the previous iteration. After counting their supports, the algorithm eliminates all candidate itemsets whose support count are less than *Min_sup*. It explores the level wise mining *Apriori* property that all nonempty subsets of a frequent itemset must also be frequent.

Unlike the *Apriori* based algorithms, the frequent pattern growth based algorithms, such as FP-growth follows the DFS approach. FP-Growth algorithm mines frequent itemsets without candidate generation. It encodes the dataset using a compact data structure called the FP-Tree, and extracts frequent itemsets directly from this structure. The FP-Tree [15] is created as follows: First, a root node of the tree is created (labeled"*null*"). For each transaction in the database, the items are processed in reverse order and a branch is created for each transaction. Every node in the FP-Tree stores a counter which keeps track of the number of transactions that share that node. When adding a branch for each transaction, the count of each node among the common prefix is incremented by *1*, and nodes for the items in the transaction following the prefix are created and linked accordingly. Additionally, an item header table is built so that each item points to its occurrences in the tree via a chain of node links. Each item in this header table also stores its support. The transactions in the FP-Tree are stored in support descending order which keeps the FP-Tree representation of the database as small as possible since the more frequently occurring items are arranged closer to the root of the FP-Tree and thus are more likely to be shared. Because there are often a lot of sharing of frequent items among transactions, the size of the tree is usually much smaller than its original database which avoids the costly database scans implemented by *Apriori* based algorithms.

FP-Growth has several advantages than any other *Apriori* based algorithms, especially when the dataset contains many patterns and/or when the frequent patterns are long. The *Apriori* based algorithms outperform the FP-Growth when the *Min_sup* value is small.

Mining frequent itemsets using vertical data format (ECLAT) is a method that transforms a given dataset of transactions in the horizontal data format into the vertical data format [16]. Experiments show that in certain situations, such as when the dataset contains many dense and long patterns, this technique can substantially reduce the total cost of vertical format mining of frequent itemsets.

2.4.2 Sequential Pattern Mining

The problem of mining sequential patterns was first proposed by Agrawal and Srikant [2]. Mining sequential patterns is highly similar with mining association rules. The main difference between the sequential patterns and the association rules is that the time element is taken into account (order of events). Sequential patterns indicate the correlation between transactions while association rules represent intra transaction relationships.

Sequential patterns can be widely used in different areas, such as mining user access patterns for the web sites, using the history of symptoms to predict certain kind of disease, also by using sequential pattern mining, the retailers can make their inventory control more efficient.

2.4.3 Clustering

Clustering divides a dataset into different groups. The process of grouping objects into clusters such that the objects from the same cluster are similar and objects from different clusters are dissimilar. Most efforts to produce a rather simple group structure from a complex data necessarily require a measure of closeness or similarity. In clustering data objects have no class label and when we start clustering we do not know what the resulted clusters will be. For this reason, clustering is also called unsupervised learning. Clustering is used in many areas, including artificial intelligence, biology, customers relationship management, image processing, machine learning, marketing, medicine, and statistics.

Scalability, ability to deal with different data types, discovery clusters with arbitrary shapes, no need to input parameters, robustness, and interpretability are the desirable features of a clustering algorithm. Clustering has many different methods and techniques. In the following we discuss the main clustering methods, briefly [1].

- Partitioning Methods [17][18][19][20][21]: In partitioning methods the dataset is divided into non overlapping clusters such that each data object belongs to exactly one cluster. Objects in one cluster are similar or close to each other, but are far away from objects in other clusters. The K-Means and the K-medoids are two major and well known algorithms in this category. In the K-Means, each cluster is represented by the mean value of the objects in the cluster and in the K-medoids, each cluster is represented by one of the objects located near the center of the cluster. To find clusters with complex shapes and for clustering very large datasets, partitioning based methods need to be extended.

- Hierarchical Methods [19][22][23][24][25]: If we permit a cluster to have sub-clusters, then we obtain hierarchical clusters. Hierarchical clusters are sets of nested clusters that are organized as a tree, where each node in the tree represents a cluster and its sub-nodes represent the sub-clusters. The root of the tree represents the cluster that contains all the objects. Hierarchical methods can be classified as being either agglomerative or divisive, based on how the hierarchical decomposition is formed. The agglomerative (bottom-up) approaches, starts with each object as a single cluster, and then repeatedly merge the two closest clusters until all clusters are merged in one cluster or a termination condition holds. In contrast to agglomerative approach, the divisive approach(top-down) starts with a one big cluster and repeatedly splits each cluster into smaller clusters until reaching the point that each object is in one cluster, or until a termination condition holds. The BIRCH is well known hierarchical clustering algorithms.
- Density based Methods [26][27]: These methods group neighboring objects into clusters based on density conditions. In other words, a cluster is a dense region of objects that is surrounded by a region of low density. The DBSCAN, OPTICS, and DENCLUE are the famous density based clustering algorithms.
- Grid based Methods [28]: the methods quantize the space into a finite number of the cells that form a grid structure, and then perform clustering on the grid structure. The STING is a typical example of a grid based method.

- Model based Methods [29][30][31][32]: Model based methods hypothesize a model for each of the clusters and find the best fit of the data to the given model. Typical model based methods involve statistical approaches (COBWEB, CLASSIT, AutoClass) or neural network approaches (Competitive learning and Self-organizing maps).

2.4.4 Classification

Classification and prediction are two forms of data analysis that can be used to extract models describing important data classes or to predict future data trends. Classification also known as supervised learning is the process of finding a set of models or functions that describe and distinguish data classes or concepts where the models derived based on a set of training data. Decision tree classifiers, Bayesian classifiers and rule based classifiers are basic and well known techniques for data classification.

- Decision Tree Classifiers [33][34][35][36]: The decision tree classification method is one of the most useful methods in data mining. Classification trees are methodologies to classify data into discrete ones using the tree structured algorithms. The decision tree nodes are test values that need to be decided or questions that need to be answered in order to decide the branches that should be followed depending on the decided value. Each node may have some branches and each of them will lead to another decision node or to the bottom of the tree. Decision tree algorithms such as ID3, C4.5 and CART are used in classification to make predictions of current and future values of some attributes.

- Bayesian Classifiers [37][38][39]: Bayesian classification is based on Bayes' theorem. This method assumes that the effect of an attribute value on a given class is independent of the values of the other attributes.
- Rule based Classifiers [40][41][42][43]: A rule based classifier uses a set of IF-THEN rules for classification. Rules can be extracted from a decision tree. Rules may also be generated directly from training data using sequential covering algorithms and associative classification algorithms.

2.5 Data Mining Applications

Data mining is an important technology that has been integrated in many industrial, governmental, and academic applications. Data mining made substantial contributions to the success of different industries and application fields. In the following, we examine the usage of data mining in some application domains.

- Data Mining in Medicine: The first general application area we will explore is medical informatics. The reason for treating it first is that in no other discipline has statistical and data mining applications been applied for a longer time, and it is an area in which analytical results can have such profound effects in both positive and negative directions. In bioinformatics it can be used to analyze DNA sequences to find the relationship between some specific gene sequences and a specific disease. In other words, the change of the DNA sequence can give an indication of the high probability of the development of some diseases. Data mining is used also in medical decision support. Patient records include patient demographic information, symptoms, blood measurements and laboratory test results. Mining those data can find the correlation between two different diseases, or the relationship between social and environmental issues and some diseases, or how a group of patients react to a specific drug.

- Data Mining for Security Applications: Ensuring the integrity of computer networks, both in relation to security and with regard to the institutional life of the nation in general, is a growing concern. In addition, we need techniques to detect security breaches. Intrusion detection involves processing and learning a large number of examples in order to detect intrusions. Such a process becomes computationally costly and impractical when the number of records to train against grows dramatically. Eventually, this limits our choice of which data mining technique to apply.
- Data Mining in Customer Relationship Management: The crushing practical needs of business to extract knowledge from data that could be leveraged immediately to increase revenues required new analytical techniques that enable analysis of highly nonlinear relationships in very large datasets with an unknown distribution. Customer Relationship Management (CRM) tries to use all measures to understand customer behaviors and invest the extracted knowledge to implement marketing strategies, control production, and coordinate the supply chain. Data mining is one of those tools used by CRM to study customer behavior and interests and use the extracted knowledge to manage customer life cycle, attract new customers, and retain good customers.
- Data Mining in Marketing: Most computer databases in business were designed for the efficient storage and retrieval of account or product

information. Market basket analysis uses data mining techniques to analyze customer transactions in a retail transactional database.

- Data Mining in Insurance: Companies in the insurance industry collect enormous amounts of data about their clients such as information about customer behavior, activities, and preferences. Applying data mining on such data might be useful in detecting and predicting fraud or risky customers and predicting which customers will get benefit from new services and policies.
- Web Mining: Applying data mining techniques to extract interesting pattern from web data is called web mining. Improvements in web search engines can also be achieved using predictive models.
- Data Mining in Fraud Detection: In telecommunication companies, huge amount of data are collected daily varying from transactional data to other customer data such as billing information or customer profiles, and additional data such as network load. Fraud detection techniques can also help in finding stolen mobile phones or phone cards.

Chapter 3

Parallel Data Mining

With the availability of inexpensive storages and the programs in data capture technology, many organizations have created ultra huge database, and this trend is expected to grow. Since the databases to be mined are likely to be very huge, there is a crucial need to investigate methods for parallel data mining techniques. Without parallelism, it is generally difficult for a single processor system to provides reasonable response time. We have now reached a point in terms of computational power, storage capacity and cost that enable us to gather, analyze and mine unprecedented amounts of data [44].

There are some other ways to face against high volume of data such as sampling, but even when sample size is small, in some applications may be computationally intensive and will therefore benefit from parallelism.

Data mining often is a computing intensive and time requiring process. For this reason, several data mining systems have been implemented on parallel computing platforms to achieve high performance in the analysis of large datasets. Exploring useful information from huge data will require efficient parallel algorithms running on high performance computing systems with powerful parallel I/O capabilities. When data mining tools are implemented on these computers, they can analyze massive databases in a reasonable time.

We get higher I/O bandwidth, larger memory, and computational power than the limits of existing serial systems, all these factors leading to lower response time and improve scalability to larger datasets. The common drawback is that algorithm and application design become more complex in order to enjoy higher performance. We need to devise algorithms and techniques that distribute the I/O and the computation in parallel, minimizing communication and data transfer to avoid wasting resources [45].

3.1 Parallel Computing and Data Mining Techniques

In the Internet era, many companies rely on the storage and processing of huge amounts of data, with high speed computation making it feasible to analyze these datasets. The examination of large databases in order to generate new information is referred to as data mining, which involves several processes including classification, association rule, clustering, and sequence mining [46].

With the availability of huge datasets in many applications like medical informatics, financial analysis, scientific data analysis, retailing, and marketing, it is becoming increasingly important to execute data mining tasks in parallel. Since serial algorithms are not scalable, high performance parallel and distributed computing becomes essential.

Using multiple processors to compute in parallel may allow us to obtain a solution sooner and if each processor has its own memory, partitioning the data among the processors may allow larger problems to be handled that could not be handled on a single processor. By adding additional processing elements, more data can be processed and can be improved performance. Performance improvement is main goal of using parallel computing technologies in the data mining field. In fact, a successful approach tries to use parallel techniques for optimizing the local mining at a site, and uses distributed techniques for construction global patterns or model, while minimizing the amount of results communicated.

Throughput and response time are two main measures of performance improvement, which are normally quantified by the following metrics: speed up and scale up. The first metric can be measured by dividing the run time of a job on single processor with the run time of that job on multiprocessors. In parallel systems, equal workload among processors plays the critical role to achieve linear speed up. Loading one processor, heavier than others is certainly undesirable. The second metric is scale up, which can be measured by dividing the run time on a small system and the run time on a large system. We would like to maintain system performance when the number of tasks and size of the data to be mined increase.

There are some factors which work against speed up and scale up such as initialize and communication cost, shared resources competition, work load unbalancing, and consolidation. In parallel environment, there is a start-up cost associated with initiating each process. In this situation, often one process for communication with others may be forced to wait. System resources are limit and both speed up and scale up are affected by sharing these resources. Performance of parallel system is dependent upon how the workload distribution, uniform workload among processors is most desirable. In parallel processing, it is necessary to gather the results produced by each processor and system performance is affected by this phenomenon.

Parallel and distributed data mining algorithms can be classified in term of five main components [47]:

- Distributed versus shared memory systems
- Data versus task parallelism
- Static versus dynamic load balancing
- Complete versus heuristic candidate generation
- Horizontal versus vertical data layout

In distributed memory architecture each processor has a private memory and a message passing mechanism needs to be employed for exchanging data among processors. On the other side, in shared memory architecture, all processors can access a common memory. In this architecture different processors may simultaneously read the same memory location, but may not simultaneously write to the same memory. A typical shared memory machine consists of several CPUs ranging from *2* to *16* CPUs. A larger number of CPUs is not too common due to the scaling up limitation. Hybrid architecture is a mixture between shared memory and distributed memory architectures. Communication between the processors is crucial in distributed memory systems and I/O can become a bottleneck in shared memory systems. Communication costs can be balanced by the improved knowledge that a data mining algorithm can get from parallelization. Figure 3.1 shows these three types.

With data parallelism, the database is partitioned among the processors and same operation is executed on multiple partitions at the same time. With task parallelism each processor has or needs access to the entire database and multiple operations are executed concurrently.

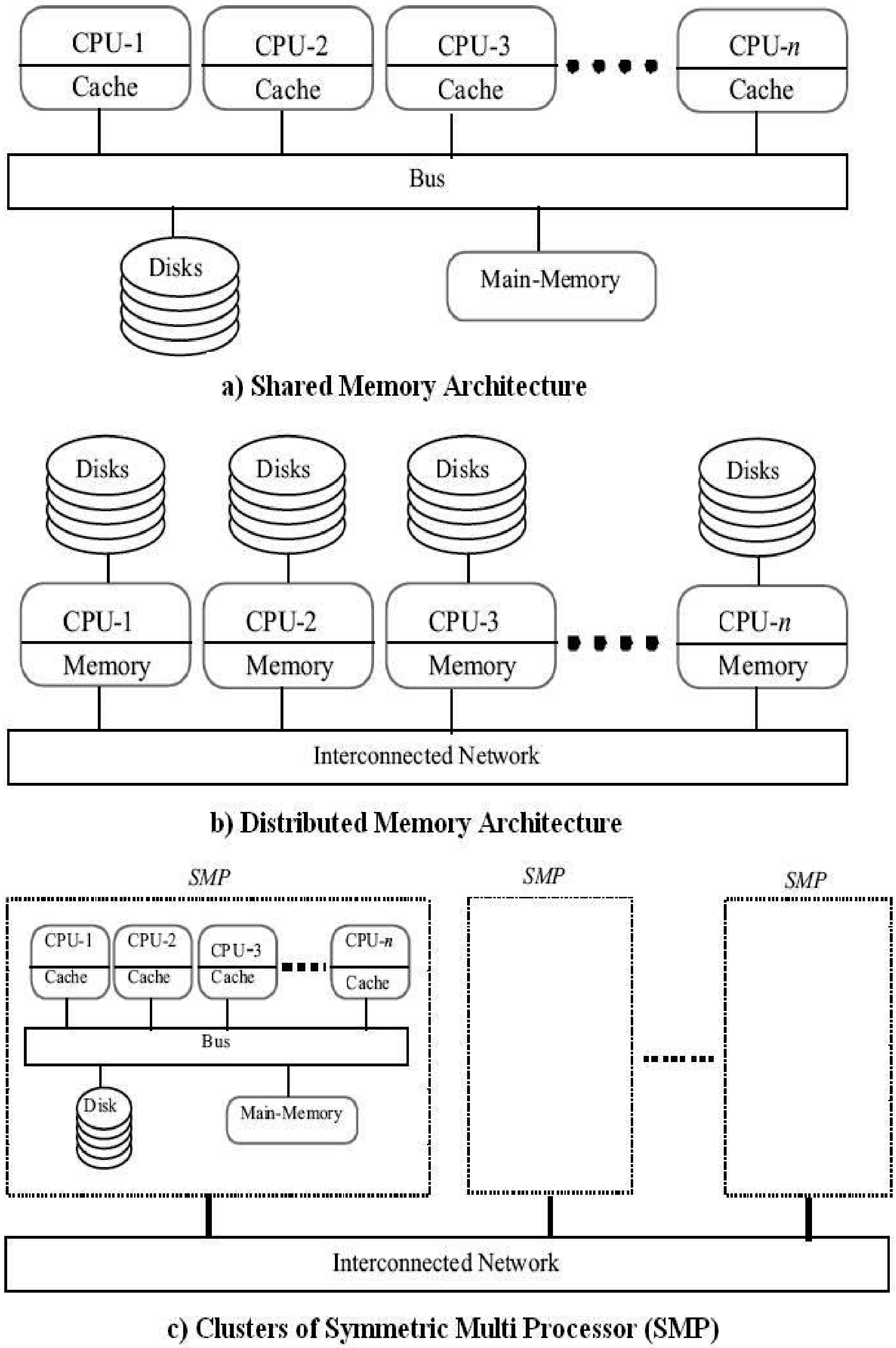


Figure 3.1: Three types of parallel architectures

From a data mining viewpoint, data parallelism has several main advantages over task parallelism. A lot of previously written serial code can be reused in a data parallel fashion. This simplifies programming and leads to a development time significantly smaller than one associated with task parallel programming. Data parallelism has a higher degree of machine architecture independence, in comparison with task parallelism. But problem of machine architecture dependence is not completely eliminated in data parallelism. In most applications, the amount of data can increase arbitrarily fast, while the number of lines of code typically increases at a much slower rate. To put it in simple terms, the more data is available, the more opportunity to exploit data parallelism. Despite the above advantages of data parallelism, it should be emphasized that the exploitation of task parallelism is also useful in data mining. To summarize, data parallelism addresses the problem of very large databases, whereas task parallelism addresses the problem of very large search spaces. Note that if a large enough number of processors are available, both types of parallelism can be used at the same time, which can greatly speedup the execution of data mining algorithms [44].

Parallel algorithms can further be classified as having a static or dynamic load balancing. Static load balancing refers to some heuristic cost functions. In contrast, dynamic load balancing refers to the environment where data is continuously moved from heavily loaded processors to less busy ones. In static load balancing, there is no subsequent data or computation movement to correct load imbalances which result from the dynamic nature of mining algorithms. Dynamic load balancing is especially important in multi-user environments with transient load and in heterogeneous platforms, which have different processors and network speeds [47]. The operating system can play an important role to allocate tasks to each processor. Once a processor is idle, a task in the queue will be immediately allocated to it.

Mining algorithms can differ in the way new candidates are generated for evaluation. Complete search approaches generate and test all valid candidates, while in heuristic approaches, at each step, it only examines a limited number of candidates.

Data layout can be row-wise or column-wise. In row-wise or horizontal database, there are some rows, also called record or tuple, and several columns, also called dimension or attribute. Other methods use a column-wise or vertical database, where they associated with each attribute a list of all transaction containing the item, and the corresponding attribute value in that transaction.

An easy way to discuss the many parallel and distributed mining methods is to describe them in terms of the computation and data partitioning methods used (Figure 3.2).

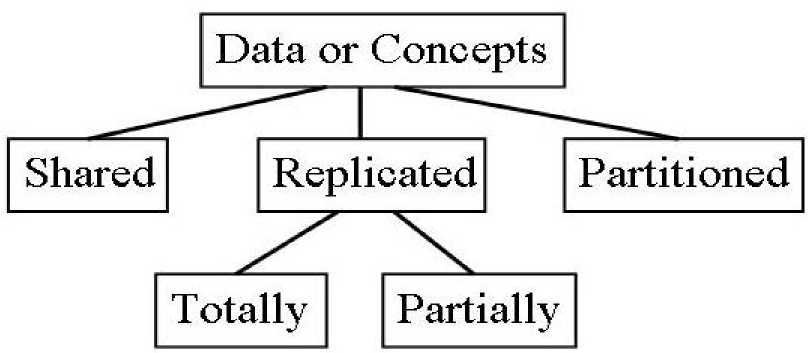


Figure 3.2: Concepts for parallel and distributed mining methods

For example, the database itself can be shared (in shared memory or shared disk architectures), partially or totally replicated, or partitioned (using round robin, hash, or range scheduling) among the available nodes (in distributed memory architectures). Similarly, the generated and evaluated candidate concepts in the different mining methods can be shared, replicated or partitioned. If they are shared then all processors evaluate a single copy of the candidate set. In the replicated approach the candidate concepts are replicated on each machine, and are first evaluated locally, before global results are obtained by merging them. Finally, in the partitioned approach, each processor generates and tests a disjoint candidate concept set [47].

3.2 Parallel Data Mining Algorithms

Traditional methods typically made the assumption that the data is memory resident. This assumption is no longer acceptable. Implementation of data mining algorithms in high performance parallel computing environment is becoming crucial for ensuring system scalability and interactivity as data continues to grow extremely in size and complexity. In fact, a fundamental challenge is to extend data mining to large datasets. There are many issues to implement data mining techniques with different parallel strategies, must be taken into account. These parallel approaches may be combined to form different hybrid parallelization strategies to improve both performance and accuracy.

There are different forms of parallelism in data mining depending on the context of the problem and appropriate parallelization strategy could depend upon the data mining algorithm that is being parallelized. According to [47] in Table 3.1 summarizes in list form where each parallel algorithm for each data mining tasks lies in the design space. Following sections discuss about different ways to exploit parallelism in data mining algorithms, briefly.

Algorithm	Base Algorithm	Machine	Parallelism	LoadBal	DB Layout	Concepts	Database
Association Rule Mining							
CD, PEAR, PDM, FDM, NPA	Apriori	DMM	Data	Static	Horizontal	Replicated	Partitioned
DD, SPA, IDD	Apriori	DMM	Task	Static	Horizontal	Partitioned	Partitioned
HD	Apriori	DMM	Hybrid	Hybrid	Horizontal	Hybrid	Partitioned
CCPD	Apriori	SMP	Data	Static	Horizontal	Shared	Partitioned
CandD, HPA, HPA-ELD	Apriori	DMM	Task	Static	Horizontal	Partitioned	Partially Replicated
PCCD	Apriori	SMP	Task	Static	Horizontal	Partitioned	Shared
APM	DIC	SMP	Task	Static	Horizontal	Shared	Partitioned
PPAR	Partition	DMM	Task	Static	Horizontal	Replicated	Partitioned
PE, PME, PC, PMC	Eclat, Clique	CLUMPS	Task	Static	Vertical	Partitioned	Partially Replicated
Sequence Mining							
NPSPM	GSP	DMM	Data	Static	Horizontal	Replicated	Partitioned
SPSPM	GSP	DMM	Task	Static	Horizontal	Partitioned	Partitioned
HPSPM	GSP	DMM	Task	Static	Horizontal	Partitioned	Partially Replicated
pSPADE	SPADE	SMP	Task	Dynamic	Vertical	Partitioned	Shared
D-MSDD	MSDD	DMM	Task	Static	Horizontal	Partitioned	Replicated
Decision Tree Classification							
SPRINT, SLIQ/R, SLIQ/D, ScalParC	SLIQ/SPRINT	DMM	Data	Static	Vertical	Replicated	Partitioned
DP-att, DP-rec, PDT	C4.5	DMM	Data	Static	Horizontal	Replicated	Partitioned
MWK	SPRINT	SMP	Data	Dynamic	Vertical	Shared	Shared
SUBTREE	SPRINT	SMP	Hybrid	Dynamic	Vertical	Partitioned	Partitioned
HTF	SPRINT	DMM	Hybrid	Dynamic	Vertical	Partitioned	Partitioned
pCLOUDS	CLOUDS	DMM	Hybrid	Dynamic	Horizontal	Partitioned	Partitioned
Clustering							
P-CLUSTER	K-Means	DMM	Data	Static	Horizontal	Replicated	Partitioned
MAFIA	-	DMM	Task	Static	Horizontal	Partitioned	Partitioned

Table 3.1: Design space for parallel mining algorithms

3.2.1 Classification

Data parallel approach used for parallel classification is based on attribute or record. In attribute data parallelism, attributes are divided equally among *N* processors so that each processor is responsible for *1/N* attributes. In this data parallelism, each processor is responsible for processing *1/N* fraction of each attribute list. Task parallel approach used for parallel classification is based on the parallelism that exists in different sub-trees. In the other words, one process or task is associated to each sub-tree of the decision tree that is built to represent a classification model.

Among other techniques for data classification, decision trees are the most popular ones. While there are couple of classification methods including neural network and genetic algorithms, decision trees are particularly suited to data mining, since they can be constructed quickly, and are simple and easy to understand [47].

A decision tree classifier is built in two phases: growing and pruning phase. The tree growth phase is computationally much more expensive than pruning, since data is scanned multiple times during the growth phase. Pruning requires access only to the fully grown decision tree. Therefore, most parallel version of classification algorithms based on decision tree focus on the parallelization of the growing phase [44].

A data parallel approach to growing a tree, splits the data into *P* partitions, grows a single tree on each partition, and then produces an ensemble of *P* trees. It should be noted that after generating a decision tree, in order to simplify the tree, some post processing steps might still be remained. These steps may require more computation time than the actual tree generation phase.

In parallel decision tree construction the concepts, shown in Figure 3.2, are the possible split points for all attributes within a node of the expanding tree.

3.2.2 Clustering

Parallelism in clustering algorithms can be exploited both in the clustering strategy and in the computation of the similarity/distance among the objects. Often in parallel clustering and in similarity/distance calculation phase, each processor focuses on the portion of data assigned to it. This phase can be executed asynchronously and in parallel for each data portion.

For example, in partitioning methods processors can work on disjoint region of the search space using the whole dataset. The K-Mean algorithm is a popular clustering method and parallelization of this method received a lot of attention in the past. In hierarchical methods each processor can find nearest neighbor clusters by computing the similarity among its cluster and the others and merge clusters with global shortest distance between two clusters.

In clustering techniques, similarity/distance computation can be good concepts to parallelize and based on Figure 3.2 we can classify these techniques.

3.2.3 Association Rules

The *Apriori* based algorithms are the most known algorithm for association rules discovery and based on this property, several parallel implementations have been proposed. *Apriori* based algorithms are an iterative algorithm that for counting itemsets uses a complete, bottom-up search, and horizontal dataset. There are some methods that utilize a vertical database and structural proportion of frequent itemsets to facilitate fast discovery.

A well known paradigm for parallel association rules mining is to partition the database and candidate itemsets replicated. Using *Apriori* property, each processor computes the frequency of the candidate set in its local partition. There are two steps to construct frequent itemsets, creation of candidate frequent itemsets and counting them. Parallel approach can be used in both steps.

We can partition datasets among processors but candidate itemsets *I* are replicated on each processor. Each process p counts in parallel the partial support S_p of the global itemsets on its local partition of the dataset. At the end of this phase the global support S_g is obtained by collecting all local supports S_p [48].

Based on Figure 3.2, frequent itemsets are main concept in association rules mining algorithms which can shared, partially replicated, or totally replicated.

3.2.4 Sequential Patterns

The set of all frequent sequences is a superset of the set of frequent itemsets. Hence, sequence search is much more complex and challenging than itemsets search, thereby we need fast parallel algorithms [47].

There are two main streams in parallel sequential patterns. One of them is based on *GSP* (*Generalized Sequential Patterns*), and the other is based on *SPADE*. The parallelized versions of the former were implemented in distributed memory architecture and the latter run on shared memory machines.

3.3 Challenges and Future Researches

The main challenge of parallel and distributed algorithms face today include communication overhead minimization, synchronization, workload balancing, data decomposition, efficient memory usage, and disk I/O minimization. Various algorithms have been proposed that explore trade-offs between these issues [49]. Databases continue to increase in size and dimension. Current methods are able to handle data in the gigabyte range and a few thousand dimensions. Even a single scan for these types of databases is very expensive. Data type is another challenge for data mining research in which has focused on simple structure such as tabular data. Therefore, an important challenge is to develop parallel data mining for unstructured and semi-structured data with complex data types. Most of parallel data mining algorithms use static load balancing while one of problems is sensitivity to data skew. Databases in real world are often heterogeneous and dynamic load balancing is crucial in this environment with computers ranging from ordinary workstation to supercomputers. Lake of incremental methods and multi-relational mining are other important challenges and researches issues in parallel data mining.

Chapter 4

Sequential Patterns Mining

Sequential pattern mining finds sets of data items that occur together frequently in some sequences. Sequential patterns indicate the correlation between transactions. Sequential pattern mining, which extracts frequent subsequences from a sequence database, has attracted a great deal of interest during the recent surge in data mining research because it is the basis of many applications.

The diversity of the applications may not be possible to apply a single sequential pattern model to all these problems. Each application may require a unique model and solution. In this chapter, we provide a brief overview of models of sequential patterns represented in [50] and theoretically we describe kinds of sequential patterns and some properties of them. These models fall into three classes are called periodic pattern, statistically pattern, and approximate pattern. We also have illustrated these models in our published paper [11].

4.1 Introduction

It is obvious that time stamp is an important attribute of each dataset, and it can give us more accurate and useful information and rules. A database consists of sequences of values or events that change with time are called a time series database. This type of database is widely used to store historical data in a diversity of areas. One of the data mining techniques which have been designed for mining time series data is sequential pattern mining. Sequential pattern mining is trying to find the relationships between occurrences of sequential events by looking for any specific order of the occurrences. In the other words, sequential pattern mining is aiming at finding the frequently occurred sequences to describe the data or predict future data or mining periodical patterns [2][51].

Sequences are an important kind of data which occur frequently in many fields such as medical, business, financial, customer behavior, education, security, and other applications. In these applications, the analysis of the data needs to be carried out in different ways to satisfy different application requirements, and it needs to be carried out in an efficient manner [50].

To gain a better understanding of sequential pattern mining problem, let's start by looking at an example. From a shopping store database, we can find frequent sequential purchasing patterns, for example "70% customers who bought the TV typically bought the DVD player and then bought the memory card with certain time gap." It is conceivable that achieving this pattern has great impact to better advertisement and better management of shopping store.

Interestingness measures play an important role in data mining methods, regardless of the kind of patterns being mined. These measures are intended for selecting and ranking patterns according to their potential interest to the user. Choosing interestingness parameters that reflect real human interest remain an open issue.

Sequential pattern mining methods often use the support, which is the criterion to evaluate frequency but this parameter is not efficient to discover some patterns.

Due to the huge increase in data volume and also quite large search space, efficient solutions for finding patterns in sequence data are nowadays very important. For this reason, using some pruning techniques are necessary to optimize the algorithms. They aim to reduce the number of candidate sequence in early stage by utilizing current information.

Regardless of a great diversity of models for sequential pattern mining, according to [52] the problem of mining sequential pattern can be partitioned into three categories: periodic patterns, statistically patterns, and approximate patterns.

The remaining of this chapter is organized as follows. Sequence mining and related works describe in section 4.2. In section 4.3 we provide the most popular model, periodic patterns. Statistical significant patterns model introduces in section 4.4. Section 4.5 and 4.6 present the flexible model for approximate and multidimensional sequential patterns, respectively.

4.2 Sequence Mining and Related Works

The problem of mining frequent patterns in a set of data sequences together with a few mining algorithms was first introduced in [2]. They also presented three algorithms for solving this problem. The sequence mining task is to discover a sequence of attributes, shared across time among a large number of objects in a given database. Many studies have been contributed to the efficient mining of sequential patterns in the literature, most of which was focused on developing efficient algorithms for finding all sequential patterns such as AprioriAll [2], GSP [3], SPADE [5], PrefixSpan [53] and so on. In addition, enormous size of available databases and possibly large number of mined sequential patterns demand efficient and scalable parallel algorithms. While parallel association mining has attracted wide attention there has been relatively less work on parallel mining of sequential patterns [54]. Several parallel algorithms such as HPSPM [55], pSPADE [3], TPF [56] were proposed and have good performance [47]. Moreover, recent research on sequential pattern mining has progressed closed sequential pattern mining, which can greatly reduce the number of frequent subsequences and improve the efficiency. Yet, the above works all assume that the database is static, and a database updates requires rediscovering all the patterns by scanning the entire old and new database. Then, there is a need for efficient algorithms to update, maintain and manage the information discovered. Some incremental mining algorithms of sequential patterns were proposed and perform significantly better than the original approach of mining the whole updated database from scratch. Other works contribute on an extension of the problem of sequential pattern mining like constraint based in sequential pattern mining, mining cyclically repeated patterns, approximate mining of consensus sequential patterns, mining top-*k* closed sequential patterns, mining frequent max sequential patterns, mining long sequential patterns in a noisy environment, mining hybrid sequential patterns and sequential rules, etc. Mining multidimensional sequential pattern is also one of the central topics in sequence mining as showed by the research efforts produced in recent years [7][57][58][59].

4.3 Periodic Patterns

This model of pattern is quite rigid and it fails to find patterns whose occurrences are asynchronous [60]. Periodicity detection on time series database is an important data mining task and has broad applications. For example, "The gold price increases every weekend" is a periodic pattern. As mentioned above, this model is often too restrictive since we may fail to detect some interesting pattern if some of its occurrences are misaligned due to inserted noise events. A pattern can be partially filled to enable a more flexible model. For instance, pattern length three *(I_1,*,*)* is a partial pattern showing that the first symbol must be I_1. The system behavior may change over time and some patterns may not be present all the time.

Two parameters, namely *min_rep* and *max_dist*, are used to specify the minimum number of occurrences that is required within each subsequences and the maximum disturbance between any two successive subsequences. The rationale behind this is that a pattern needs to repeat itself at least a certain number of times to demonstrate its significance and periodicity [61].

Couple of algorithms in this area focused on patterns for some pre-specified period length and several models can discover all periodic patterns regardless of the period length. Notice that period usually is apart of what we would like to mine from data. Let us look at an example to explain some definitions.

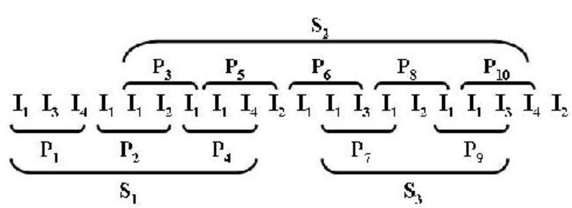


Figure 4.1: Example of symbol sequence

Figure 4.1 shows matches of pattern $(I_1,*,*)$ that is a *1*-pattern of period *3*. In this Figure, $P_1, P_2, \ldots, P_{10}$ are ten matches of pattern $(I_1,*,*)$ and S_1, S_2, and S_3 are three segments which each one forms a list of *N* consecutive matches of $(I_1,*,*)$. For example, segment S_2 is consist of *5* successive matches of this pattern. S_1 and S_3 are disjoint segments and S_1 and S_2 are overlapped segments. In many applications, overlapped segments often are not considered.

If the value of *min_rep* is set to *3*, then S_1 and S_2 qualify as valid segments and S_3 is not a valid segment. Two or more than two contiguous and disjoint segments can construct a valid subsequence provided that distance between any two successive valid segments does not exceed the parameter *max_dist*. For example, if the value of *min_rep* and *max_dist* are set to *2* and *3* respectively, both S_1 and S_3 recognized as valid subsequences whose overall number of repetition are *5*.

Given a sequence *S*, the parameters *min_rep* and *max_dist* and the maximum period length L_{max}, in three phases we can discover the valid subsequences that have the most repetitions for each valid pattern whose period length does not exceed L_{max}. The following three phases outline algorithm for mining periodic patterns in brief [52].

- The First Phase: For each symbol *I*, we examine the distance between any two occurrences of *I* and then for each period *l*, the set of symbols whose number of times are at least *min_rep* are sent to the next phase. Since there are a huge number of candidates, a pruning method is needed to reduce it.
- The Second Phase: In this phase, the single patterns (*1*-pattern) are generated. For each period *l* and each symbol *I* a candidate pattern *(I,*,*,…,*)* is formed that number of symbol * is (*l*-1).
- The Third Phase: After discovering the single patterns in the second phase, *i*-patterns are generated from the set of valid *(i-1)*-patterns and then these patterns are validated. In this phase, we can apply some heuristics. For example, it is obvious that if a pattern is valid, then all of its

generalizations are valid. For example pattern *(I_1,I_2,*)* is a generalization of pattern *(I_1,I_2,I_3)*.

When parameters are not set properly, noise may be qualified as a pattern. The parameter *max_dist* is employed to recognize the noises between segments of perfect repetition of a pattern.

4.4 Statistically Significant Patterns

The support and confidence are the most popular measurements for sequential patterns. The support evaluates frequencies of the patterns and the confidence evaluates frequencies of patterns in the case that sub-patterns are given. These parameters are meaningful and important for some applications. However, in other applications, the number of occurrences (support) may not always represent the significance of a pattern. Sometimes, a large number of occurrences of an expected frequent pattern may not be as interesting as few occurrences of an expected rare pattern. This pattern called surprising pattern instead of frequent pattern. The information gain metric which is widely used in the information theory field, is useful to evaluate the degree of surprise of the pattern [62]. We want to find set of patterns that have information gain higher than minimum information gain threshold. Experiments show that the support threshold has to be set very low to discover a small number of patterns with high information gain.

Note that surprising patterns are anti-monotonic. It means we can not take advantage of standard pruning techniques such as *Apriori* property. For example, the pattern *(I_1,I_2)* may have enough information gain while neither *(I_1,*)* nor *(*,I_2)* does.

Given a pattern $P=(I_1,I_2,\ldots,I_n)$ and an information gain threshold *min_gain*, the goal is to discover all patterns whose information gain in the sequence *S* exceed the *min_gain* value. Similar to other parameters in data mining algorithms, the appropriate value of the *min_gain* is application dependent and may be defined by a domain expert. There are some heuristics and methods that user can set the value of this threshold.

Information gain of pattern *P* is defined as follows:

$Info\_gain(P)=Info(P)*Support(P)$

$Info(P)=Info(I_1)+ Info(I_2)+\ldots+ Info(I_l)$

$Info(I_k) = -\log_{|I|}^{prob(I_k)}$

Where $prob(I_k)$ is probability that symbol I_k occurs and $|I|$ is number of events in *S*.

For example, the sequence in Figure 4.1 contains 4 different events I_1, I_2, I_3, and I_4. Their probabilities and information gains are shown in Table 4.1.

Table 4.1: Probability of occurrence and information gain

Event	Probability	Information	Information gain
I_1	10/20=0.50	$-\log_4(0.50)=0.50$	0.50*10=5.0
I_2	4/20=0.20	$-\log_4(0.20)=1.16$	1.16*4=4.64
I_3	3/20=0.15	$-\log_4(0.15)=1.37$	1.37*3=4.11
I_4	3/20=0.15	$-\log_4(0.15)=1.37$	1.37*3=4.11

The main strategy to tackle the problem of mining statistically significant patterns is a recursive method which at the *i*th level of recursion, the patterns with *i* events are examined.

In some applications, users may want to find the *k* most surprising patterns. On the other words, users may want to discover patterns that their information gain is greater than a threshold. However, the major limitation of information gain value is that it does not recognize location of the occurrences of the patterns. Let's take a look at two sequences S_1 and S_2 in Figure 4.2.

$$S_1 = I_1\ I_2\ I_3\ I_2\ I_1\ I_2\ I_4\ I_3\ I_1\ I_2\ I_2\ I_4$$

$$S_2 = I_2\ I_3\ I_4\ I_2\ I_1\ I_2\ I_1\ I_2\ I_1\ I_2\ I_4\ I_3$$

Figure 4.2: Two examples of symbol sequences

The pattern (I_1,I_2) has the same information gain in the two sequences, it scatter in S_1 but repeats consecutive in S_2. As a result, it is beneficial if we can specify a parameter to evaluate this situation.

4.5 Approximate Patterns

Noisy data are commonplace properties of large real world databases. Noise is a random error or variance in a measured variable and there are many possible reasons for noisy data [1]. In many applications, due to the presence of this noise may prevent an occurrence of a pattern and can not be recognized. In addition, a large pattern is much more vulnerable to distortion caused by noise. In these conditions, it is necessary to allow some flexibility in pattern matching. Two previous models only take into account exact match of the pattern in data. An approximate pattern is defined as a sequence of symbols which appears more than a threshold under certain approximation types in a data sequence.

For solving the problem of finding approximate patterns, the concept of compatibility matrix is introduced [52]. This matrix provides a probabilistic connection from observed values to the true values. Based on the compatibility matrix, real support of a pattern can be computed. Figure 4.3 gives an example of the compatibility matrix.

True value	Observed value			
	I_1	I_2	I_3	I_4
I_1	0.80	0.15	0.00	0.05
I_2	0.10	0.70	0.10	0.10
I_3	0.0	0.00	0.90	0.10
I_4	0.10	0.15	0.00	0.75

Figure 4.3: An example of compatibility matrix

For example, an observed I_4 corresponds to a true occurrence of I_1 ,I_2 ,I_3 , and I_4 with probability $C(I_1,I_4)=0.05$, $C(I_2,I_4)=0.10$, $C(I_3,I_4)=0.10$, and $C(I_4,I_4)=0.75$,respectively. Compatibility matrix usually is given by some domain expert but there are some ways to obtain and justify the value of each entry in the matrix so that even with a certain degree of error contained in matrix, sequential pattern mining algorithm can still produce results of reasonable quality.

A new metric, namely match is defined to quantify the significance of a pattern. The combined effect of support and match may need to scan the entire sequence database many times. Similar to other data mining methods, we can use sampling based algorithms to tackle this problem. Consequently, the number of scans through the entire database is minimized.

Given a pattern $P=(I_1,I_2,\ldots,I_{lp})$ and a symbol sequence $S=(I'_1,I'_2,\ldots,I'_{ls})$ where $ls \geq lp$, match of P in S (denoted by $M(P,S)$) is defined as the maximal conditional probability P in every distinct subsequence of length lp in S. Following formula show how to compute match provided that each observed symbol is generated independently.

$M(P,S)=max_{s \in S} M(P,s)$ if $ls>lp$

$M(P,s)=prob(P|s)=\Pi_{1 \leq i \leq lp} C(I_i,I'_i)$ if $ls=lp$

For example, the process of calculating the match of two pattern $P_1=I_2I_1I_3$ and $P_2=I_2I_3$ in the sequence $S=I_1I_2I_3$ based on compatibility matrix values in Figure 4.3 is shown as follows:

$M(P_1,S)=prob(P_1|S)= \Pi_{1\le i\le 3}C(I_i,I'_i)= C(I_2,I_1)* C(I_1,I_2)* C(I_3,I_3)=0.1*0.15*0.9=0.0135$

Because of there exist *3* distinct subsequences of length *2* in *S* (I_1I_2 ,I_1I_3 ,and I_2I_3) the match of P_2 in S is:

$M(P_2,S)=max_{s\in S}M(P_2,s)=max\{M(P_2,I_1I_2) , M(P_2,I_1I_3) , M(P_2,I_2I_3) \}$

$M(P_2,I_1I_2)= M(I_2I_3 ,I_1I_2)=C(I_2,I_1)*C(I_3,I_2)=0.1*0=0$

$M(P_2,I_1I_3)= M(I_2I_3 ,I_1I_3)=C(I_2,I_1)*C(I_3,I_3)=0.1*0.9=0.09$

$M(P_2,I_2I_3)= M(I_2I_3 ,I_2I_3)=C(I_2,I_2)*C(I_3,I_3)=0.7*0.9=0.63$

$M(P_2,S)=max\{0 , 0.09 , 0.63\}=0.63$

User identifies a minimum match threshold *min_match* to qualify significant patterns. In this model, as the pattern length increases, the match decreases at a much slower pace than the support. For this reason, the number of generated frequent patterns is usually larger than that using the support [63].

4.6 Multidimensional Sequential Patterns

Sequence identifiers (representing individual customers, for example) and sequence items (such as products bought) are often associated with additional pieces of information. Sequential pattern mining should take advantages of such additional information to discover interesting patterns in multidimensional, multilevel information space.

Mining multiple dimensional sequential patterns can give us more informative and useful patterns. Take customer shopping transactions, for instance. In a sequence database for such data, the additional information associated with sequence IDs could include customer age, address, group, and profession. Information associated with items could include item category, brand, model type, model number, place manufactured, and manufacture date. Mining multidimensional sequential patterns is the discovery of interesting patterns in such a broad dimensional space, at different levels of detail [1]. Multidimensional sequential pattern mining is first introduced in [57] and then several algorithms were proposed, such as [7][58][59].

Chapter 5

Proposed Parallel Method

Theoretically, mining the complete set of sequential patterns is #P-complete [50]. #P-Complete problems are the enumeration problems which might be intractable even if P=NP. Besides mining sequential patterns in a single dimension, mining multidimensional sequential patterns can give us more informative and useful patterns [60]. In a multidimensional sequence each event depends on more than one dimension. Due to the huge increase in data volume and also quite large search space, efficient algorithms for finding patterns in multidimensional sequence data are nowadays very important. Although efficient algorithms have been proposed but mining so many patterns from large sequence datasets is inherently a computationally expensive task. To address this, it is necessary to study scalable parallel implementations of sequence mining algorithms.

In this chapter, we first consider previous works in which mine patterns from multidimensional sequence database. Then we move on the main body of thesis and describe a model for multidimensional sequence and represent a scalable parallel algorithm based on data parallelism. This algorithm follows the level-wise approach and all participating processors generate candidate sequences and count their supports independently. The main target is usually to reduce the execution time as much as possible by using multiple processors (workers). The terms processor and worker are used interchangeably throughout of this chapter.

5.1 Related Works

While many studies have contributed to find sequential patterns from sequence dataset, there has been relatively less works on mining patterns from multidimensional sequence dataset.

According to [57] a multidimensional sequence database is defined over the schema $(PK, A_1, \ldots, A_n, S)$, where PK is a key, A_i $(1 \leq i \leq n)$ stands for the dimensions and S is in the domain of sequences. A multidimensional sequence takes the form of $(a_1, \ldots, a_n, s)$, where a_i is in $\{A_i, *\}$ for $1 \leq i \leq n$, and s is a sequence. In this paper some methods are proposed that can be classified into two groups: (1) integration of efficient sequential pattern mining and multidimensional analysis methods, and (2) embedding multidimensional information into sequences and using sequential pattern mining method to find patterns. The sequences found by this approach do not contain several dimensions because the time only can be combined with specific dimensions. Consequently, some sequences are not generated.

In [59], two algorithms have been developed for mining sequential patterns from multidimensional sequence data in the framework of web usage mining. There are some main drawbacks in the work as pointed out by [7]. Three dimensions (page, session, day) consider and these dimensions belong to a single hierarchical dimension. The generated sequences explain correlation between objects over time by considering only one dimension which corresponds to the web pages.

In [7], a definition for multidimensional sequential patterns has been proposed that is close to our model. The authors claim that they aim at considering more than one dimension. This proposition can be extended for approximate values on quantitative dimensions.

5.2 Basic Definitions

There is a rich variety of sequence terminology but this section defines the concepts sequence, subsequence, sequence pattern, subset, segment, and support; it also discusses major characteristics of our model, multidimensional sequence dataset. Some of the definitions are generic, because different applications have variety properties.

The main table *M* has tuples $<D,Att_1,Att_2,\ldots,Att_n>$, where *D* is an attribute whose its domain is totally ordered and other attributes divided into two groups: analysis and relevant attributes. Notice that irrelevant attributes removed in preprocessing phase. The schema $<D,A_1,\ldots,A_n, R_1,\ldots,R_m>$ is a multidimensional sequence database, where A_i $(1 \le i \le n)$ are analysis and R_j $(1 \le j \le m)$ are relevant dimensions. The schema is partitioned into subsets according distinct values of relevant attributes and sequence support computed by number of partitions in which keep corresponding sequence. A subset table *T* is a set of tuples $<D,A_1,A_2,\ldots,A_n>$, where *D* that stated before is an attribute whose its domain is totally ordered, and A_i $(1 \le i \le n)$ are analysis attributes. A sequence *S* is denoted by an ordered list $<t_1,t_2,\ldots,t_k>$, where t_i is a tuple, i.e., $D(t_1) \le D(t_2) \le \ldots \le D(t_k)$ for $1 \le i \le k$ and $D(t_i)$=value of *D* tuple t_i. Every tuple has *n* analysis attributes along with an ordered value. A set of analysis attribute values can occur at most once in a same value of *D*, but can occur multiple times in different values of *D* attribute.

The number of distinct values of D in a sequence is the length of that sequence. If the length of S is k, then we call it a k-sequence and in similar way k-pattern refers to a pattern with length k.

A sequence $S_1=<a_1,a_2,...,a_n>$ is called a subsequence of another sequence $S_2=<b_1,b_2,...,b_m>$ and S_2 a super-sequence S_1, if there exist integers $1\leq j_1\leq j_2\leq\ldots\leq j_n\leq m$ such that $a_1=b_{j1}$, $a_2=b_{j2}$,…, and $a_n=b_{jn}$.

Given a minimum support threshold *min_support*, a multidimensional sequence S is called a multidimensional pattern if and only if support(S)≥*min_support*. The problem of sequential pattern mining is to find the complete set of frequent sequential patterns satisfying a minimum support in the sequence database.

In this framework, $<(t_1, t_2),(t_3),(t_4, t_5, t_1)>$ is a multidimensional 3-sequence. Brackets and commas may be added to make multidimensional sequences more readable. All tuples in a bracket are assumed to occur at the same time. In first bracket, due to tuples t_1 and t_2 have the same value of D; both of them appear in one bracket (D(t_1)=D(t_2)). This way a sequence is an ordered list of brackets in which ordered according to their associated time in a sequence. Number of brackets show length of sequence. Therefore, above sequence is a sequence with length 3 (3-sequence). It is obvious that each tuple can be repeated in different brackets (time), such as tuple t_1, which appears in first and third bracket. On the other side, the ordering between tuples in one bracket is not important. According above definition, multidimensional sequence $<(t_2),(t_3)>$ is a subsequence example of sequence $<(t_1, t_2),(t_3),(t_4, t_5, t_1)>$ with length 2.

A segment is the largest subsequence in which all of its tuples belong to a same time value. As a result, every sequence with length n has n segments. Given a sequence $S=<x_1x_2...x_n>$ with length n, where each x_i ($1\leq i\leq n$) consists of one or some tuples with time value i. Each x_i is a segment. For example, in 3-sequence $S=<(t_1, t_2),(t_3),(t_4, t_5, t_1)>$, there are three segments $<(t_1, t_2)>$, $<(t_4, t_5, t_1)>$, and $<(t_3)>$.

To gain a better understanding of framework, let's start by looking at an example. The main table *M* is defined over five dimensions as shown in Figure 5.1. The *D* is time dimension, *A1* and *A2* are analysis dimensions and *R1* and *R2* are relevant dimensions. According to distinct values of combination of *R1* and *R2*, table *M* partitioned into two subsets (Figure 5.2). As can be seen from Figure 5.2a and 5.2b, partition lengths are two and three, respectively.

D	A_1	A_2	R_1	R_2
1	a_1	a_2	r_1	r_2
1	a_1	a_2	r_3	r_4
1	a_5	a_6	r_3	r_4
2	a_3	a_4	r_1	r_2
2	a_5	a_6	r_3	r_4
3	a_3	a_4	r_3	r_4

Figure 5.1: Example of a table with five dimensions

D	A_1	A_2
1	a_1	a_2
2	a_3	a_4

(a)

D	A_1	A_2
1	a_1	a_2
1	a_5	a_6
2	a_5	a_6
3	a_3	a_4

(b)

Figure 5.2: Partitioned subsets based on relevant attributes

As mentioned previously, a sequence is called *k*-sequence if there exist *k* subsequences which be selected from different time dimension. For example in Figure 5.2b, $<\{(a_1,a_2),(a_5,a_6)\},\{(a_3,a_4)\}>$ is a *2*-sequence. We will see using this concept can be helpful to distribute subsets among processors or workers.

Let us consider a real example of taking the course by students. Suppose we have a relation *Marks(id, course, score, year/semester)* in which we want to record for each student their identify number, course name, score, and year/semester. Each tuple *(d,c,s,y)* indicate that student with identify number *d* has taken course *c* in semester *y* and obtained score *s*. As you can suggested, we set *id* as relevant attribute, *year/semester* as ordered attribute, and other columns *(course* and *score)* as analysis attributes. According to *id* column values, main table *Marks* partition into subsets that each one holds information about specific student. Finding the patterns may help to capture some ordering of students' course taking. Following patterns are several useful and important pattern results.

- More than 60 percent of students fail Programming course.
- While Operating System is not prerequisite of Database but at least 80 percentages of students take it before Database.
- Students who take Data Mining always observe following ordering: Programming → Data Structure → Database → Data Mining

5.3 STA: Primary Parallel Algorithm

In this section, we will first describe a standard algorithm (STA) for mining multidimensional sequence patterns within the entire sequence data and later in the next section, we will improve the algorithm (IMP). We use data parallelism such that subsets are distributed among the workers and each worker performs same operation on it.

Let us first analyze search space before discussing about main method. Given a subset with length c (c-sequence), and suppose that each element of vector d_i for $1 \le i \le c$ shows number of tuples with equal time dimension. Number of sequences with length $1, 2, \ldots, c$ for this subset computed as follows:

Number of sequences with length 1: $\sum_{i=1}^{c}(2^{d_i}-1)$

Number of sequences with length 2: $\sum_{i=1}^{c-1}(2^{d_i}-1)\sum_{j=i+1}^{c}(2^{d_j}-1)$

Number of sequences with length 3: $\sum_{i=1}^{c-2}(2^{d_i}-1)\sum_{j=i+1}^{c-1}(2^{d_j}-1)\sum_{k=j+1}^{c}(2^{d_k}-1)$

Number of sequences with length c: $\prod_{i=1}^{c}(2^{d_i}-1)$

If the value of c set to 4 and vector d_i ($1 \le i \le 4$) set to $[3,4,5,3]$, then following equations calculate number of i-sequences for $1 \le i \le 4$.

1-Sequence: $\sum_{i=1}^{4}(2^{d_i}-1) = 7+15+31+7 = 60$

2-Sequence: $\sum_{i=1}^{3}(2^{d_i}-1)\sum_{j=i+1}^{4}(2^{d_j}-1) = 105+217+49+465+105+217 = 1158$

3-Sequence: $\sum_{i=1}^{2}(2^{d_i}-1)\sum_{j=i+1}^{3}(2^{d_j}-1)\sum_{k=j+1}^{4}(2^{d_k}-1) = 3255+735+1519+3255 = 8764$

4-Sequence: $\prod_{i=1}^{4}(2^{d_i}-1) = 7*15*31*7 = 22785$

It is clear that, we should generate and test *32767*(60+1158+8764+22785) sequences only for one subset. Assume that the relevant attributes have *100* different values. Even in this situation which every parameter set to small value, we encounter with *$3*10^6$* candidates approximately. This example is an extremely simplified view of the situation but it is enough to demonstrate that serial algorithms are not feasible.

As you observe, search space is quite large and the serial algorithms are not scalable. In parallel environment, each worker scans datasets to find patterns with length one (*1*-patterns) according to the predefined support threshold. Next steps generate the sequences by joining the patterns in the previous step. Datasets is scanned to check the support of candidates. We can use *Apriori* property to develop breadth first search algorithms to find sequence patterns. The main idea is that, if a sequence is not pattern, we do not looking for any super-sequence of it.

Schematic diagram of primary proposed method shows in Figure 5.3. It is obvious that phases 1 and 2 have been repeated once and phase 3 repeats until there is no sequence to be mined.

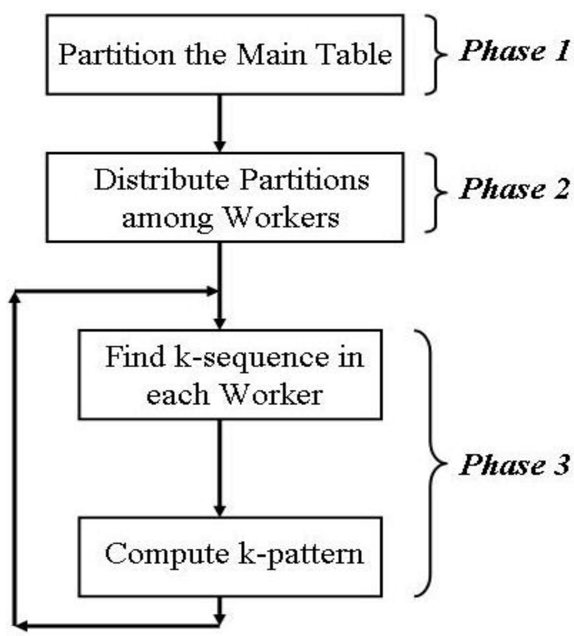


Figure 5.3: Schematic diagram of proposed algorithm

Our algorithm conducts a level-by-level candidate generation and test pruning following the *Apriori* property in workers independently. At each level, only potentially frequent candidates are generated and tested. Although this model is much more scalable than serial performance, the computation could be costly due to the absence of more powerful pruning rules. For this reason, after discussing the main three phases of our parallel algorithm in following subsections, we present two modification rules to improve STA algorithm in subsequent section.

5.3.1 Partition the Main Table

In first phase of algorithm, one worker (processor) known as coordinator, partitions table *T* into *P* sequence datasets (subsets). This phase can be as a data transformation stage to other phases. Data transformation is one of preprocessing task that is concerned with mapping data from one form to another. For our proposed algorithm, this process does in the three following steps:

I. The main table should be partitioned into subsets according to current values of relevant attributes. Because every subset has the same relevant attribute value, we discard relevant attribute columns.

II. We assign a simple code to each combination of current analysis attribute values. It is easier and faster for algorithm to works with smaller dataset.

III. Merge tuples with equal time value into identical segment.

Let us examine how to use above three steps to partition main table into several sequence dataset. The method is illustrated in Figure 5.4 over table in which have ten tuples, two analysis attributes, and one relevant attribute. The table consists of two distinct values for relevant attribute (*R1* and *R2*), for this reason, it partition into two subsets with length two and three, respectively.

D	A_1	A_2	R
1	V1	V2	R2
1	V2	V1	R2
1	V2	V2	R2
2	V1	V1	R1
2	V1	V2	R1
2	V2	V1	R1
2	V1	V1	R2
3	V1	V1	R2
3	V2	V2	R2
3	V1	V2	R1

D	A_1	A_2
2	V1	V1
2	V1	V2
2	V2	V1
3	V1	V2

D	A_1	A_2
1	V1	V2
1	V2	V1
1	V2	V2
2	V1	V1
3	V1	V1
3	V2	V2

D	A
2	a
2	b
2	c
3	b

D	A
1	b
1	c
1	d
2	a
3	a
3	d

D	A
2	abc
3	b

D	A
1	bcd
2	a
3	ad

Figure 5.4: Partition a table into two sequence dataset

It is time to assign a simple code to every combination of analysis attributes. As you see, for example we assign letter *a* for composition (*v1,v1*) and letter *c* for (*v2,v1*). Finally, we Merge tuples with the same time value into same segment.

One technique which may reduce the running time to partition is using index data structure instead of performing physical partitioning. Thus, it is efficient in terms of both executing time and space.

5.3.2 Subsets Distribution

In this phase, coordinator distributes *m* subsets among *n* workers regardless of length of subsets. This way, $subset_i$ is assigned to worker $P_{i \bmod n}$, for $i=0,\ldots,m-1$. Thus, worker P_j owns the subsets $j, j+n, \ldots, j+n.(\lceil m/n \rceil - 1)$ for $j<m \bmod n$ and $j, j+n, \ldots, j+n.(\lceil m/n \rceil - 2)$ for $(m \bmod n) \leq j<n$.

For example when we set $n=4$ and $m=10$ the subsets distribution results are such that P_j for $j<2=(10 \bmod 4)$ owns the subsets $j, j+4, j+4*(3-1)$ and P_j for $2\leq j<4$ owns the subset $j, j+4*(3-2)$.

Following we can see in this example what subset goes to which processor:

P_0: owns *subset*$_0$, *subset*$_4$, *subset*$_8$

P_1: owns *subset*$_1$, *subset*$_5$, *subset*$_9$

P_2: owns *subset*$_2$, *subset*$_6$

P_3: owns *subset*$_3$, *subset*$_7$

5.3.3 Patterns Mining

In iteration *k*, every worker mines all *k*-sequence and computes its support (local support) and then sends to coordinator. Coordinator collects results and discards some sequences in which its support is less than *min_support* and sends *k*-patterns to workers (as shown in Figure 5.5).

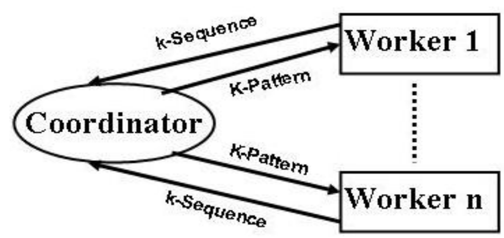


Figure 5.5: Schematic diagram of the third phase

While the general procedure of this phase is simple, the new sequence generation is non trivial. As mentioned previously, every worker scans subsets once to find all *1*-sequence and computes local support and then sends to coordinator. Coordinator collects results and sends back *1*-patterns to workers. After generating *1*-patterns, the set of candidate *k*-sequences (for $k>1$) are generated by joining the *(k-1)*-patterns found in the previous step. New scans of each subset collect the support for candidates and then send to coordinator to find the new set of patterns. Two patterns with length *k* can be joined if (i) patterns have equal length and (ii) value of time dimension *k-1* segments of two patterns are identical.

We assume that $S_x=<x_1x_2...x_{k-1}x_k>$ and $S_y=<y_1y_2...y_{k-1}y_{k+1}>$ are two k-patterns ($k>1$), where each x_i ($1\leq i\leq k$) and y_j ($1\leq j\leq k+1,\ j\#k$) are a tuple or consists of some tuples with identical value of time. Without loss of generality, we assume that subscripts show time dimension value for segments. As you know, both patterns have equal length (k) and based on subscripts of each segment, values of time $k-1$ segments of two patterns are same. The conditions satisfied and join of S_x and S_y a new $(k+1)$-sequence is generated: $S_{xy}=<(x_1y_1)(x_2y_2)...(x_{k-1}y_{k-1})(x_k)(y_{k+1})>$. While in each segment the ordering of tuples is not important ($x_iy_i=y_ix_i$), duplication of tuples is not allowed. In other words, when segments x_i and y_i are merged, every tuple appears once and repetitions are removed. Figure 5.6 illustrates all steps of joining two patterns S_1 and S_2. In each segment, subscript shows time dimension value for corresponding segment.

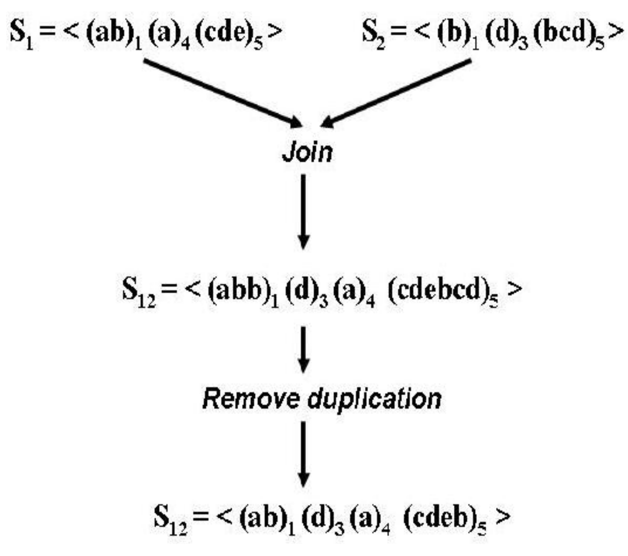


Figure 5.6: Joining two patterns

5.4 IMP: Improved STA Algorithm

The STA algorithm can be improved further. The number of candidate patterns of a certain level is typically an exponential function of number of discovered lower level. Consequently, coordinator became a bottleneck. Message passing between workers and coordinator can be heavy especially when number of patterns and their length grow up. Besides, we usually have no good load balance and the more workers are idle. Since the time cost of the passes over the datasets plays an important role to performance of algorithm, we made two following rules to improve STA algorithm.

5.4.1 Rule 1: Distribution

Using a suitable data distribution can be useful to obtain good load balancing. This simplifies the parallel programming and supports a good performance. The load imbalance is generally defined as:

Load Imbalance$=Max\ (RT_1,\ldots,RT_p)/Min(RT_1,\ldots,RT_p)$, Where RT_i is runtime on worker or processor *i*. Ideally, this value is very nearly to 1. The larger the load imbalance, the more processors are idle. The main goal of this improvement rule is to assign the subsets such that a good load balancing results.

In improved algorithm (IMP), we present a cyclic data distribution. Cyclic distributions are one of the most useful distributions for increasing the performance of embarrassingly parallel computations. The set of processors is denoted as $P=\{P_0,P_1,\ldots,P_{n-1}\}$ and the subsets is denoted as $S=\{S_0,S_1,\ldots,S_{m-1}\}$, where S_i is a subset, i.e., Length(S_0)≤Length(S_1)≤… ≤Length(S_{m-1}) for $0\leq i\leq m-1$. We assign subsets to processors in round robin two ways so that subset S_i is assigned to processor $P_{(i\ mod\ 2n)}$ for $(i\ mod\ 2n)<n$, and processor $P_{(2n-1)-(i\ mod\ 2n)}$

when *(i mod 2n)≥n.* Figure 5.7 shows how to distribute *m* subsets among *n* processors.

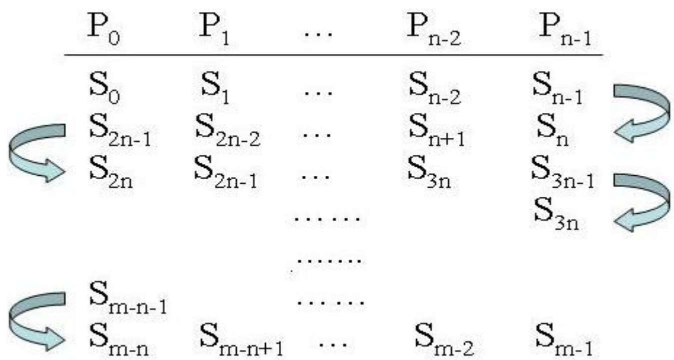


Figure 5.7: Distribution of *m* subsets among *n* processors

For the example *n=4* and *m=18* the subsets distribution

P_0: owns $subset_0, subset_7, subset_8, subset_{15}, subset_{16}$

P_1: owns $subset_1, subset_6, subset_9, subset_{14}, subset_{17}$

P_2: owns $subset_2, subset_5, subset_{10}, subset_{13}$

P_3: owns $subset_3, subset_4, subset_{11}, subset_{12}$

Results, where $subset_j$ for *(j mod 8)<4* belong to processor $P_{(j \bmod 8)}$, and send to processor $P_{7-(j \bmod 8)}$ when *(j mod 8)≥4*.

This distribution is only based on length of subsets. We know that each subset with length *k* usually has cardinality (number of tuples) more than *k*. If we can use length of subset along with cardinality to distribute subsets, we will achieve more performance, surly.

5.4.2 Rule 2: Efficient Pattern Mining

In algorithm STA, the check process is quite straightforward; by scanning the data the support counts of these candidate sequences can be obtained. By comparing them with the support threshold we can get those frequent sequential patterns.

In IMP approach, our strategy first constructs the *2*-sequences and *3*-sequences and then mine whole of *2*-patterns and *3*-patterns, similar to STA algorithm. After this, each worker generate sequence with length larger than three and discover patterns, independently. Every worker uses all of patterns with length two and three to generate new patterns. The algorithm will iteratively generates new candidate *(k+1)*-sequence using the frequent sequence with length *k* (*k*-pattern) found in the previous iteration.

Given sequence $S_x=<x_1x_2...x_{k-1}x_{k1}>$ and $S_y=<y_1y_2...y_{k-1}y_{k2}>$ are two *k*-patterns ($k>1$), where each x_i and y_i (i *is in* $\{1,2,...,k\text{-}1,k1,k2\}$) are a tuple or consists of some tuples with identical value of time, and subscripts show time value. Thus, there are *(k-1)* segments in which have equal time value. New sequence *S* generates depends on ordering between *k1* and *k2*.

$$S=\begin{cases} S_{xy}=<(x_1y_1)(x_2y_2)...(x_{k-1}y_{k-1})(x_{k1})(y_{k2})> & if\ \ k1<k2 \\ S_{yx}=<(x_1y_1)(x_2y_2)...(x_{k-1}y_{k-1})(y_{k2})(x_{k1})> & if\ \ k1>k2 \end{cases}$$

According to second improvement rule, the new sequence S_{xy} (or S_{yx}) is a pattern with length *(k-1)* if two following conditions are satisfied.

(1) The *2*-sequence $<(x_{k1})(y_{k2})>$ (or $<(y_{k2})(x_{k1})>$) is pattern.

(2) The *3*-sequences $<(x_iy_i)(x_{k1})(y_{k2})>$ (or $(x_iy_i)(y_{k2})(x_{k1})$), where $1\leq i\leq k\text{-}1$ are pattern.

For example, consider the case where $S_x=<(ab)_1,(a)_4,(cde)_5>$ and $S_y=<(b)_1,(d)_3,(bcd)_5>$. New sequence generates $S=<(ab)_1,(d)_3,(a)_4,(cdeb)_5>$. While STA algorithm scan all of subsets, and coordinator collects their support value to check *S* is pattern or not, in IMP algorithm according to second rule, we must test *2*-sequence $<(d)_3,(a)_4>$ and two *3*-sequences $<(ab)_1,(d)_3,(a)_4>$ and

$<(d)_3,(a)_4,(cdeb)_5>$ are patterns or not. If so, *S* is pattern otherwise *S* does not satisfy the rules and is not a pattern.

In modified algorithm IMP, each worker keeps two tables consist of *2*-patterns and *3*-patterns. This way, algorithm must check at most *k-1* sequences to test one *k*-sequence may be pattern or not. Consequently, every worker is able to check whether new sequence is pattern or not, independently. In other words, after mining patterns with length two and three, communication between coordinator and workers will be terminated.

Theoretically, the all of identified patterns using this rule are not guaranteed to be valid because we only consider some subsequence of candidate and do not count support of corresponding patterns. Number of extra patterns is not high and in near all our experiments, number of patterns is equal. In other words, there are no any extra patterns when we have sequences with length less than *15*. In fact, using this rule is made to generate further patterns but we will see that number of extra generated patterns increase slowly when length of patterns grows. Experiments in the next section demonstrate the power of these two modification rules.

5.5 Experimental Results

All algorithms were implemented in MATLAB with Parallel Computing Toolbox. We briefly describe about MATLAB Parallel Computing Toolbox in Appendix A. Our machine environment consists of a network of 8 computers (2.6 GHz CPU) with 2 GB memory running Windows XP operating system. In this section, after explaining how to generate synthetic datasets and what real datasets are selected, depict different plots and discuss regarding results.

5.5.1 Sequence Datasets

To gain insight on how algorithms behave under various setting, we choose two most common used resources UCI Repository and IBM Data Generator.

The UCI Machine Learning Repository is a collection of real databases that are widely used by the researchers for the empirical analysis of algorithms [64]. For evaluation of our algorithms in practical experiment, we selected four datasets from UCI Repository. The training datasets were chosen to include a wide range of domains and are summarized in Table 5.1. Datasets are used with some modifications because we must adjust the datasets to suit our multidimensional sequence model. For example, the chosen dataset may be contained of attributes with missing values or as mentioned previously we need at least one dimension whose its domain is totally ordered.

Table 5.1: Real dataset descriptions for test

Name of Dataset	Data Types	Number of Instances	Number of Attributes
Adult	Multivariate	48842	14
Localization Data for Person Activity	Univariate, Sequential, Time series	184860	8
Spoken Arabic Digit	Multivariate, Time series	8800	13
Thyroid Disease	Multivariate, Domain Theory	7200	21

We also used IBM Synthetic Data Generator with minor modification. Table 5.2 summarizes the main parameters of the datasets. For the purpose of evaluation of the algorithms, we use some synthetically generated sequences, based on parameters shown in Table 5.2. We provide an example of dataset and values of parameters in Appendix B.

Table 5.2: The Parameters of the Datasets Generator

Notation	Meaning	Default Setting
N_{seq}	Number of subsets (sequences)	100,000
N_{tra}	Average number of time values per subset	10
L_{tra}	Average number of tuples with identical time value	2.5
N_{tup}	Number of different tuples	10,000
R	Repetition level for tuples	0

5.5.2 Discussion

We ran many experiments with various datasets. We obtain results over several criteria including minimum support, size of dataset, number of workers, number of sequence (pattern), length of sequence (pattern), and maximum length of sequence (pattern). We report and plot the results on some selected datasets here.

Now, let us examine the effect of above criteria on our approaches. Figure 5.8 plots the effect of minimum support over number of patterns to be mined. The first observation is that, as the minimum support was increased, number of patterns reduced quickly. This shows why it is hard to find patterns when minimum support is low. It is also clear that maximum length of patterns grow up when value of minimum support reduced (Figure 5.9).

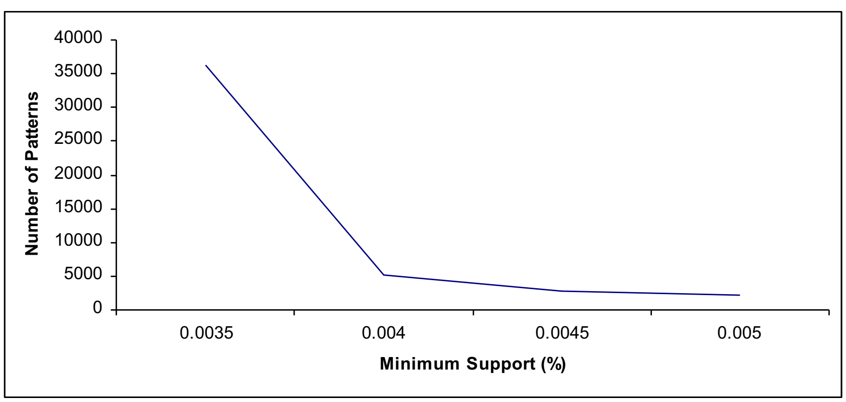


Figure 5.8: Effect of minimum support value over number of patterns

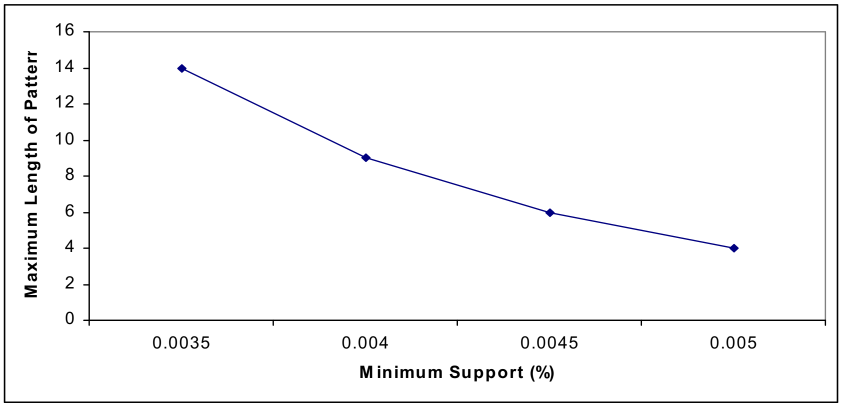


Figure 5.9: Effect of minimum support threshold over length of patterns

We have also studied the effect of changing minimum support on the STA algorithm performance. We used 1, 2, 4, and 8 workers, and support threshold ranges from 0.0035% to 0.0050%.

Figure 5.10 shows execution time over minimum support in 4 statuses. It can be seen that STA algorithm scales well. As expected, speed up depends on minimum support value. When we set a lower value for minimum support, we encourage too many of patterns. In other words, when we decrease the value of minimum support, length of patterns to be mined grow. As a result, STA algorithm has to deal with longer patterns and data transfer between coordinator and workers extremely grow up.

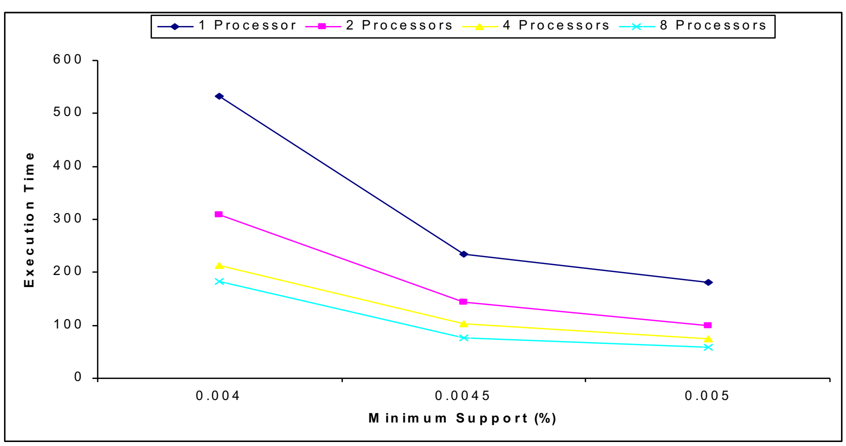


Figure 5.10: Effect of minimum support over execution time

The coordinator becomes a bottleneck and major cost is counting support value for sequences in which send to coordinator. Even in this case (Figure 5.10), STA speed up is obvious and scales almost linearly. But it seems using more than 4 processors is not as well as less than that. We will see it can be improved more by IMP algorithm.

Figure 5.11 compares subsets distribution of STA and IMP approaches when the size of dataset (number of transaction) changes. The number of workers is set to 4. After distribution of subsets among 4 workers, we add length of subsets or sequences in which assign to each worker and then calculate standard deviation. As the dataset becomes longer, standard deviation of STA algorithm grows up while there is no considerable variation in standard deviation of IMP algorithm. We observe that the work balancing of IMP in terms of length of sequences is better than STA. The above analysis indicates that IMP is effective and scalable in mining large database.

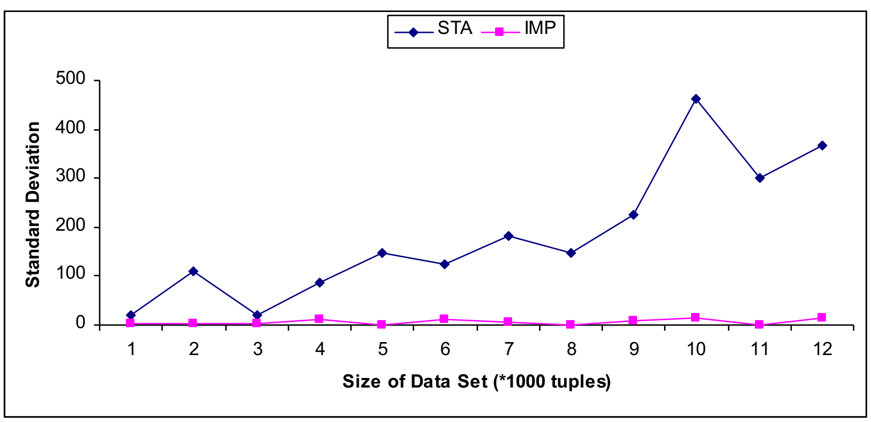
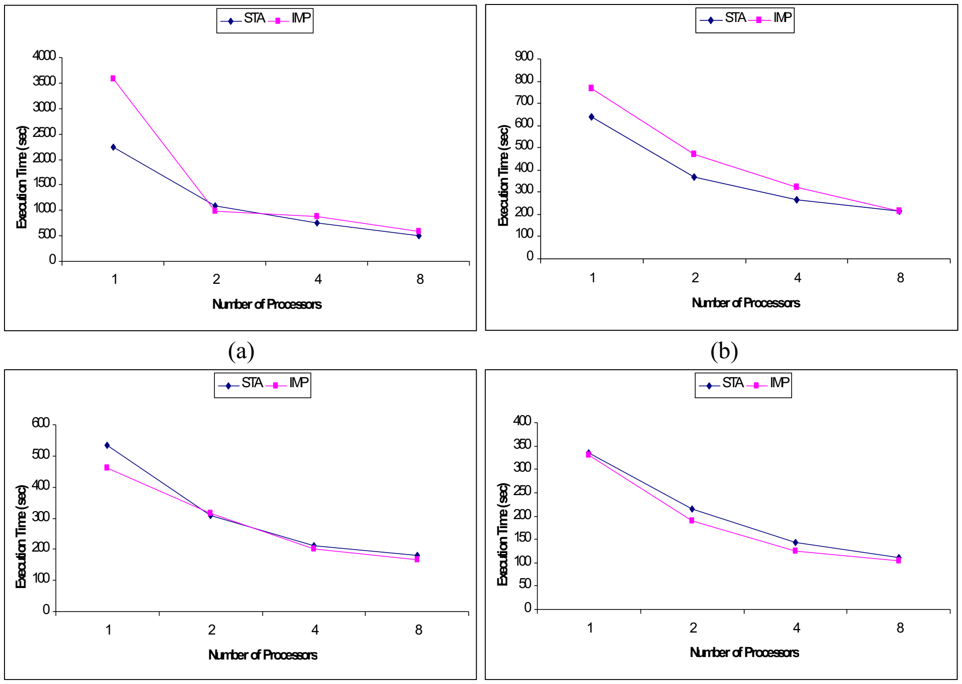


Figure 5.11: Comparison of subsets distribution



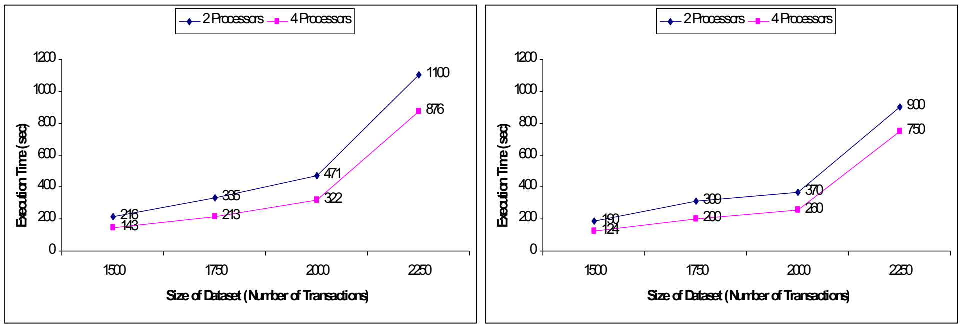
(a) (b)

(c) (d)

Figure 5.12: Execution time and speedup for generated datasets

Figure 5.12 compares the behavior of the two approaches when the database size changes. The support threshold is set to 0.004 and the database size ranges from 1500 to 2250 increased by 250. Figure 5.12a shows results when database size is 1500, Figure 5.12b for 1750, Figure 5.12c for 2000, and Figure 5.12d for 2250.

In these range of dataset, our experimental results show that both methods have almost identical performance. Number of patterns that generated by algorithms is equal and with these size of datasets, minimum and maximum length of generated patterns are 6 and 10, respectively. But when we notice the plots consecutive, it is clear that as the size of database increases, IMP algorithm is getting better performance than STA. As mentioned previously, this is because when database grow up, calculating total support of the subsets by coordinator is a bottleneck.



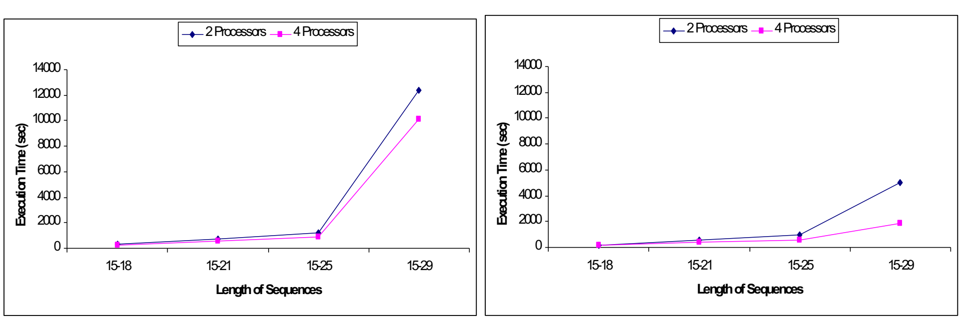
(a) (b)

Figure 5.13: Execution time over database size

From another point of view, Figure 5.13 plot execution time over dataset size for two approaches separately. The database size changes from 1500 to 2250 and the support threshold set to 0.004%.

Figure 5.13a shows the performance of STA algorithm when it runs with 2 and 4 processors. Figure 5.13b shows the scalability of IMP algorithm since execution time grows almost linearly when the size of database (number of transactions) increases. We observe that the performance of IMP in terms of scalability and runtime is better than STA.

One of the most important criteria to consider two approaches is length of sequences. We test the effect of this parameter over execution time. The results are shown in Figure 5.14a and 5.14b for STA and IMP algorithms, respectively. The length of sequences rang from 15 to 29, database size set to 2000 transactions and minimum support value is 0.004%.



(a) (b)

Figure 5.14: Effect of length of sequences on execution time

As expected IMP algorithm is faster when length of sequences grows. This is because STA has to deal with longer sequences and patterns when dataset contains of longer patterns. While the algorithm IMP rather than scan the datasets many times for longer patterns, it only use a dataset with all of *2*-patterns (patterns with length 2) and *3*-patterns in which every worker keep it.

As stated before, theoretically the number of generated patterns with algorithm IMP is usually much than that with algorithm STA. In near all our experiments, number of patterns is equal. Figure 5.15 shows number of patterns when length of patterns changes. We use real datasets and generate some synthetic datasets with size 2000, and minimum support changes from 0.0035 to 0.005.

As the length of patterns become larger, there could be many generated patterns by IMP algorithm which are not patterns but it grows slowly. When length of patterns is between 1 and 12, number of patterns generated by two algorithms is same. Number of patterns goes from 36232 to 36401 when length of patterns is between 13 and 16. The results in Figure 5.15 show that difference of patterns number for sequences with length 16 is only 169 patterns. While in real multidimensional dataset we rarely found patterns with length more than 16.

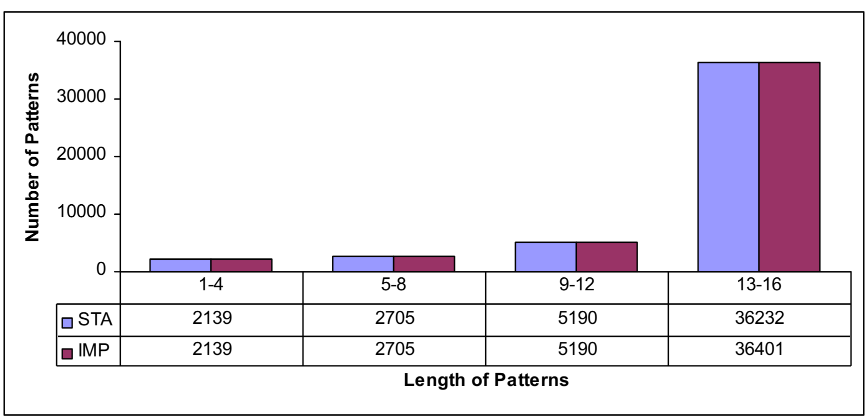


Figure 5.15: Number of generated patterns by two approaches

Our experimental results indicate that both methods STA and IMP have a similar behavior when length of patterns is low. As the length of patterns and database size grow up, IMP scales better than STA algorithm but number of patterns generated by IMP is usually much than algorithm STA. In fact, IMP approach try to use parallel techniques for optimizing the local mining at a worker, and use distributed techniques for construction global patterns or model, while minimizing the communication and data transfers. Consequently, while the primary proposed parallel algorithm scales well, the IMP algorithm is faster than it.

Chapter 6

Conclusion and Future Works

This chapter summarizes the main contributions of this dissertation, and identifies future directions to extend this work.

6.1 Summary

Multidimensional mining has been attracting attention in recent researches into data mining. Very large search space and data volume have made many problems for serial algorithms to mine sequential patterns. In order to effectively mine, efficient parallel algorithm is necessary. In addition, parallelism often can not be fully used because of communication and processing limits.

In the main part of this dissertation, we first theoretically utilize a multidimensional sequence model and then use SPMD (Single Program Multiple Data) strategy to parallelize multidimensional sequential pattern mining. According to this method a set of processors execute in parallel the same algorithm on different partitions of a dataset. From a data mining viewpoint, our approach has several main advantages over task parallelism. This simplifies programming and leads to a development time significantly smaller than one associated with task parallel programming, because a lot of previously written serial code can be reused. It has also a higher degree of machine architecture independence, in comparison with task parallelism. In most applications, the amount of data can increase arbitrarily fast, while the number of lines of code typically increases at a much slower rate. To put it in simple terms, the more data is available, the more opportunity to exploit data parallelism.

The main goal of the algorithm is balanced workload among the processors and good scalability. We use two modification rules to improve the main parallel algorithm. Using a suitable distribution of dataset and an approximate way to find patterns led to have good performance for improved algorithm. In other words, an attractive property of the improved algorithm is good behavior and scalability when length of patterns grows up while main parallel algorithm faced to very high computation time in this situation.

We have implemented our parallel algorithm using MATLAB Parallel Computing Toolbox and several datasets on a network with 8 workstations. In summary our approach can greatly reduce the number of scans through the sequence dataset by only examining a small data of patterns with length two and three and a good work loading as well. In fact, it tries to use parallel techniques for optimizing the local mining at a worker, and uses distributed techniques for construction global patterns or model, while minimizing the amount of results communicated.

Our experimental results indicate that primary and improved algorithms have a similar behavior when length of patterns is low. As the length of patterns and database size grow up, improved algorithm scales better and faster than primary parallel algorithm but number of patterns generated by improved approach is usually much than main parallel algorithm. Consequently, while the primary proposed parallel algorithm scales well, the IMP algorithm is faster than it.

6.2 Suggestions for Future Works

It is easy to see that the first scan through the dataset to *1*-sequence construction takes so much computation. Using some techniques such as sampling can be efficient for mining *1*-sequence and reduce time for next steps. It is obvious that using a suitable data structure can improve our approach. Besides, if we can use length of subset along with cardinality to distribute subsets, we will achieve more performance, surly.

The future research issues in multidimensional sequential pattern mining can be: multidimensional sequential pattern mining with constraints, interactive multidimensional sequential pattern mining or multidimensional sequential pattern mining integrated with taxonomies and hierarchies.

Bibliography

[1] J. Han and M. Kamber. *Data Mining Concepts and Techniques.* Morgan Kaufmann, Edition 2, 2006.

[2] R. Agrawal and R. Srikant. *Mining Sequential Patterns.* International Conference on Data Engineering (ICDE'95), pp 3–14, Taipei, Taiwan, March 1995.

[3] R. Srikant and R. Agrawal. *Mining Sequential Patterns: Generalizations and Performance improvements.* 5th International Conference on Extending Database Technology (EDBT'96), pp 3–17, Avignon, France, March 1996.

[4] M. J. Zaki and C. J. Hsiao. *CHARM: An Efficient Algorithm for Closed Itemset Mining.* SIAM International Conference on Data Mining (SDM'02), pp 457–473, Arlington, VA, April 2002.

[5] M. J. Zaki. *SPADE: An Efficient Algorithm for Mining Frequent Sequences.* Machine Learning, Vol. 40, pp 31–60, 2001.

[6] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M. C. Hsu. *Mining Sequential Patterns by Pattern-growth: The Prefixspan Approach.* IEEE Transaction on Knowledge and Data Engineering, Vol. 16, pp 1424–1440, 2004.

[7] M. Plantevit, Y. W. Choong, A. Laurent, D. Laurent, and M. Teisseire. *M2SP: Mining Sequential Patterns Among Several Dimensions.* Principles of Knowledge Discovery in Databases, Vol. 3721, pp 205–216, 2005.

[8] M. Esmaeili and M. Tarafdar. *Application and Limitation of Parallel Modeling in Multidimensional Sequential Pattern.* International Conference on Electrical, Computer, Electronics and Communication Engineering, pp 199–202, Paris, France, October 2010.

[9] M. Esmaeili and M. Tarafdar. *Sequential Pattern Mining from Multidimensional Sequence Data in Parallel.* International Journal of Computer Theory and Engineering (IJCTE), Vol. 2, Issue 5, pp 733–738, October 2010.

[10] M. Esmaeili and G. Fazekas. *An Experience in Using MATLAB Parallel Computing Toolbox.* Technical Report Published in University of Debrecen, Faculty of Informatics, No 2011/3 (390), 2011

[11] M. Esmaeili and G. Fazekas. *Finding Sequential Pattern from Large Sequence Data.* International Journal of Computer Science Issues (IJCSI), Vol. 7, Issue 1, pp 43–46, January 2010.

[12] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. *From Data Mining to Knowledge Discovery: An Overview.* In Advances in Knowledge Discovery and Data Mining, pp 1–34, 1996.

[13] W. H. Inmon. *Building the Data Warehouse.* John Wiley & Sons, 1996.

[14] R. Elmasri and S. Navathe. *Fundamentals of Database Systems.* Addison Wesley, Edition 5, 2005.

[15] J. Liu, Y. Pan, K. Wang, and J. Han. *Mining Frequent Itemsets by Opportunistic Projection.* International Conference on Knowledge Discovery in Databases (KDD'02), pp 239–248, Edmonton, Canada, July 2002.

[16] M. J. Zaki. *Scalable Algorithms for Association Mining.* IEEE Transaction on Knowledge and Data Engineering, Vol. 12, pp 372–390, 2000.

[17] S. P. Lloyd. *Least Squares Quantization in PCM.* IEEE Transaction Information Theory, Vol. 28, pp 128–137, 1982.

[18] J. Mac Queen. *Some Methods for Classification and Analysis of Multivariate Observations.* Proceeding 5^{th} Berkeley Symposium Mathematics, Statistics, and Probability, Vol. 1, pp 281–297, 1967.

[19] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley & Sons, 1990.

[20] C. P. Green and J. Carroll. *K-modes Clustering.* J. Classification, Vol. 18, pp 35–55, 2001.

[21] P. Bradley, U. Fayyad, and C. Reina. *Scaling Clustering Algorithms to Large Databases.* International Conference on Knowledge Discovery and Data Mining (KDD'98), pp 9–15, New York, NY, August 1998.

[22] W. H. E. Day and H. Edelsbrunner. *Efficient Algorithms for Agglomerative Hierarchical Clustering Methods.* J. Classification, Vol. 1, pp 7–24, 1984.

[23] T. Zhang, R. Ramakrishnan, and M. Livny. *BIRCH: An Efficient Data Clustering Method for Very Large Databases.* ACM-SIGMOD International

Conference on Management of Data (SIGMOD'96), pp 103–114, Montreal, Canada, June 1996.

[24] S. Guha, R. Rastogi, and K. Shim. *Cure: An Efficient Clustering Algorithm for Large Databases.* ACM-SIGMOD International Conference on Management of Data (SIGMOD'98), pp 73–84, Seattle, WA, June 1998.

[25] G. Karypis, E. H. Han, and V. Kumar. *CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling.* COMPUTER, Vol. 32, pp 68–75, 1999.

[26] M. Ester, H. P. Kriegel, J. Sander, and X. Xu. *A Density-based Algorithm for Discovering Clusters in Large Spatial Databases.* International Conference on Knowledge Discovery and Data Mining (KDD'96), pp 226–231, Portland, OR, August 1996.

[27] M. Ankerst, M. Breunig, H. P. Kriegel, and J. Sander. *OPTICS: Ordering Points to Identify the Clustering Structure.* ACM-SIGMOD International Conference on Management of Data (SIGMOD'99), pp 49–60, Philadelphia, PA, June 1999.

[28] W. Wang, J. Yang, and R. Muntz. *STING: A Statistical Information Grid Approach to Spatial Data Mining.* International Conference on Very Large Data Bases (VLDB'97), pp 186–195, Athens, Greece, August 1997.

[29] A. Dempster, N. Laird, and D. Rubin. *Maximum Likelihood from Incomplete Data via the EM Algorithm.* J. Royal Statistical Society, Vol. 39, pp 1–38, 1977.

[30] P. K. Chan and S. J. Stolfo. *Experiments on Multistrategy Learning by Metalearning.* 2nd International Conference on Information and Knowledge Management, pp 314–323, 1993.

[31] S. L. Lauritzen. *The EM Algorithm for Graphical Association Models with Missing Data.* Computational Statistics and Data Analysis, Vol. 19, pp 191–201, 1995.

[32] G. E. Hinton and T. J. Sejnowski. *Unsupervised Learning: Foundation of Neural Network Computation.* MIT Press, 1999.

[33] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[34] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees.* Wadsworth International Group, 1984.

[35] J. R. Quinlan. *Induction of Decision Trees.* Machine Learning, Vol. 1, pp 81–106, 1986.

[36] L. A. Breslow and D. W. Aha. *Simplifying Decision Trees: A Survey.* Knowledge Engineering Review, Vol. 12, pp 1–40, 1997.

[37] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification.* John Wiley and Sons, Edition 2, 2001.

[38] S. M. Weiss and C. A. Kulikowski. *Computer Systems that Learn Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems.* Morgan Kaufmann, 1991.

[39] T. M. Mitchell. *Machine Learning.* McGraw Hill, 1997.

[40] J. Hong, I. Mozetic, and R. S. Michalski. *AQ15: Incremental Learning of Attribute-based Descriptions from Examples, the Method and User's Guide.* In

Report ISG 85–5, UIUCDCSF–86–949, Department of Computer Science, University of Illinois at Urbana–Champaign, 1986.

[41] P. Clark and T. Niblett. *The CN2 Induction Algorithm.* Machine Learning, Vol. 3, pp 261–283, 1989.

[42] P. Smyth and R. M. Goodman. *An Information Theoretic Approach to Rule Induction.* IEEE Transaction on Knowledge and Data Engineering, Vol. 4, pp 301–316, 1992.

[43] W. Cohen. *Fast Effective Rule Induction.* International Conference on Machine Learning (ICML'95), pp 115–123, Tahoe City, CA, July 1995.

[44] A. Abbass Hussein, A. Sarker Ruhul, and S. Newton Charles. *Data Mining: A Heuristic Approach.* Idea Group Publishing, pp 261–289, 2002.

[45] J. Wang. *Data Mining: Opportunities and Challenges.* Idea Group Publishing, pp 106–141, 2003.

[46] G. Lee, W. Yang, and J. M. Lee. *A Parallel Algorithm for Mining Multiple Partial Periodic Patterns.* Information Sciences 176, pp 3591–3609, 2006.

[47] M. J. Zaki and C. T. Ho. *Large Scale Parallel Data Mining.* Lecture Notes in Computer Science. Lecture Notes in Artificial Intelligence, Vol. 1759, Springer, 2000.

[48] O. Maimon and L. Rokach. *Data Mining and Knowledge Discovery Handbook*. Springer, pp 1017–1039, 2005.

[49] M. J. Zaki. *Parallel and Distributed Association Mining: A Survey.* IEEE Concurrency, Special Issue on Parallel Mechanism for Data Mining, Vol. 7, Issue 4, pp 14–25, 1999.

[50] G. Dong and J. Pei. *Sequence data mining.* Springer, 2007.

[51] R. Agrawal and R. Srikant. *Fast Algorithms for Mining Association Rules.* 20th International Conference Very Large Data Bases (VLDB), Morgan Kaufmann, pp 487–499, 1994.

[52] W. Wang and J. Yang. *Mining Sequential Patterns from Large Data Sets.* Springer, 2005.

[53] J. Pei, J. Han, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. *PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth.* International Conference on Data Engineering, pp 215–224, 2001.

[54] W. Jinlin, Chen Xi, Z. Kefa, and W. Wei. *Parallel Research of Sequential Pattern Data Mining Algorithm.* International Conference on Computer Science and Software Engineering, 2008.

[55] T. Shintani and M. Kitsuregawa. *Mining Algorithms for Sequential Patterns in Parallel: Hash based Approach.* 2nd Pacific Asia Conference on Knowledge Discovery and Data mining, pp 283–294, 1998.

[56] V. Guralnik, N. Garg, and G. Karypis. *Parallel Tree Projection Algorithm for Sequence Mining.* 7th European Conference on Parallel Computing, pp 310–320, Springer, 2001.

[57] H. Pinto, J. Han, J. J. Pei, K. Wang, Q. Chen, and U. Dayal. *Multidimensional Sequential Pattern Mining.* Conference on Information and Knowledge Management, pp 81–88, 2001.

[58] S. de Amo, D. A. Furtado, A. Giacometti, and D. Laurent. *An Apriori based Approach for First order Temporal Pattern Mining.* In Symposium Brasileiro de Bancos de Dados, 2004.

[59] C. C. Yu and Y. L. Chen. *Mining Sequential Patterns from Multidimensional Sequence Data.* IEEE Transactions on Knowledge and Data Engineering archive, Vol. 17, Issue 1, pp 136–140, 2005.

[60] Q. Zhao and S. S. Bhowmick. *Sequential Pattern Mining: A Survey*. Technical Report, CAIS, Nanyang Technological University, No. 2003118, Singapore, 2003.

[61] J. Yang, W. Wang, P. S. Yu, and J. Han. *Mining Asynchronous Periodic Patterns in Time Series Data.* IEEE Transaction on Knowledge and Data Engineering, Vol. 15, Issue 3, pp 613–628, 2003.

[62] J. Yang, W. Wang, P. S. Yu, and J. Han. *Infominet: Mining Surprising Periodic Patterns.* 7th ACM International Conference on Knowledge Discovery and Data Mining (KDD), pp 395–400, 2001.

[63] J. Yang, W. Wang, P. S. Yu, and J. Han. *Mining Long Sequential Patterns in a Noisy Environment.* ACM SIGMOD International Conference on Management of Data, ACM Press, pp 406–417, 2002.

[64] C. L. Blake and C. J. Merz. *UCI Repository of Machine Learning Databases.* Irvine, CA: University of California, Department of Information and Computer Science, 2011. [http://archive.ics.uci.edu/ml/datasets.html]

Appendix A

MATLAB & Parallel Computing Toolbox

This appendix briefly provides information for programming with Parallel Computing Toolbox and MATLAB Distributed Computing Server software. Parallel Computing Toolbox software and MATLAB Distributed Computing Server software enable us to coordinate and execute independent MATLAB operations simultaneously on a cluster of computers, speeding up execution of large MATLAB jobs. The job manager is the part of the engine that coordinates the execution of jobs and the evaluation of their tasks. The job manager distributes the tasks for evaluation to the server's individual MATLAB sessions called workers (Figure A.1).

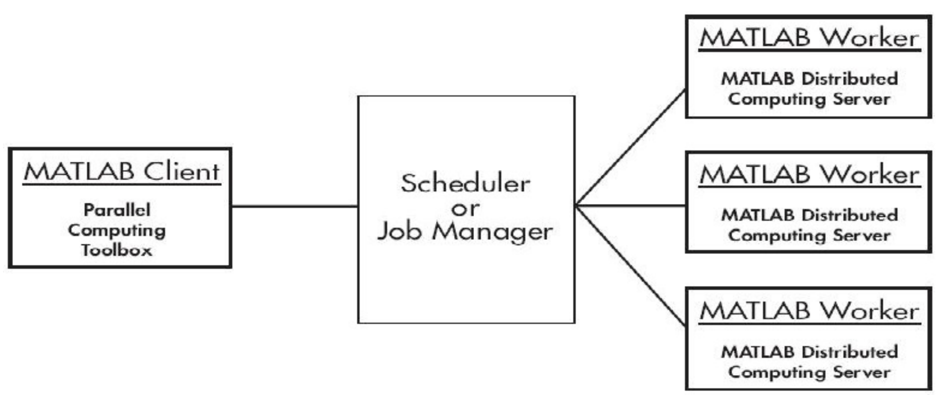


Figure A.1: Basic parallel computing configuration

Parallel Computing Toolbox software allows us to offload work from one MATLAB® session (the client) to other MATLAB sessions, called workers. We can use multiple workers to take advantage of parallel processing. Parallel Computing Toolbox software allows us to run as many as eight MATLAB workers on our local machine. MATLAB Distributed Computing Server software allows us to run as many MATLAB workers on a remote cluster of computers as our licensing allows.

The job manager can be run on any machine on the network. The job manager runs jobs in the order in which they are submitted, unless any jobs in its queue are promoted, demoted, canceled, or destroyed. Each worker is given a task from the running job by the job manager, executes the task, returns the result to the job manager, and then is given another task. Figure A.2 shows interaction of parallel computing sessions.

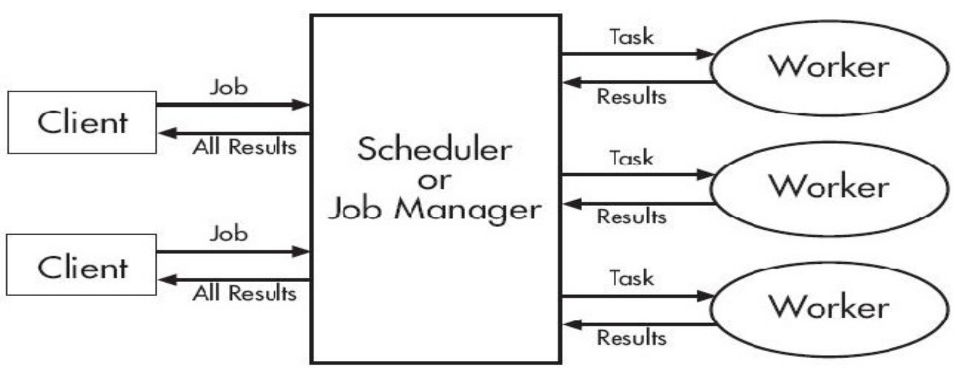


Figure A.2: Interaction of parallel computing sessions

Appendix B

IBM Dataset Generator

IBM Synthetic Data Generator is a most common used program to generate dataset for Associations and Sequential Patterns Mining. We used it with minor modification in parameters meanings. In *ASCII* output, each line contains a subset (sequence). The first number is the length of subset or sequence. After that, each tuple is displayed, first by the number of tuples in the same value of time (*D* value), then followed by the tuples in that time, in increasing order. Apart from the data file, this program also generates a pattern file including a description of the data and statistic information about patterns.

Figure B.1 shows a simple example of sequence dataset with *10* subsets and Figure B.2 shows its pattern file.

In our approach first line is a *4*-sequence like *<1584, 414, (1187, 1730), 347>*. The current pattern file has been written without any modification and Figure B.2 shows parameters in which set to generate corresponding dataset.

4 *1 1584 1 414 2 1187 1730 1 347*
6 *1 1813 3 376 1553 1776 3 261 337 1591 3 829 1398 1500 2 158 347 2 751 1941*
1 *2 54 960*
9 *3 158 302 1471 3 1193 1488 1789 1 1223 4 196 1021 1293 1609 3 128 365 1804 3 128*
223 548 2 827 1074 3 489 1296 1656 5 146 686 1201 1458 1854
2 *2 1279 1539 1 532*
6 *1 28 2 727 1887 1 1175 1 294 1 1027 1 779*
8 *3 692 982 1315 3 1067 1536 1566 3 277 1127 1943 3 737 792 1548 3 539 859 1170*
7 185 564 1403 1601 1724 1825 1959 2 149 1415 2 134 1221
6 *2 1474 1566 1 1128 2 950 1868 1 1982 2 1368 1568 2 629 1775*
5 *2 158 1240 1 1173 1 728 1 539 1 427*
9 *1 495 2 433 1122 4 91 457 1230 1962 4 1344 1594 1814 1953 1 253 1 400 1 512*
3 1043 1557 1866 6 467 968 1469 1710 1855 1892

Figure B.1: An example with 10 transactions (subsets)

Number of customers in database = 10
Average sequence length = 10
Average transaction length = 2.5
Number of items = 2000
Repetition-level = 0
Variation in repetition-level = 0.1
Large Itemsets:
- Number of patterns = 25000
- Average length of pattern = 1.25
- Correlation between consecutive patterns = 0.25
- Average confidence in a rule = 0.75
- Variation in the confidence = 0.1

Large Sequences:
- Number of patterns = 5000
- Average length of pattern = 4
- Correlation between consecutive patterns = 0.25
- Average confidence in a rule = 0.75
- Variation in the confidence = 0.1

Figure B.2: Parameter values for generating a dataset