VŠB-TECHNICAL UNIVERSITY OF OSTRAVA
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
DEPARTMENT OF COMPUTER SCIENCE

P H D T H E S I S

Study branch: COMPUTER SCIENCE

PhD Thesis

Parallel Methods for Mining Frequent Sequential Patterns

Author: **Quoc Bao Huynh**

Supervisor: Prof. RNDr. Václav Snášel, CSc.

Abstract

The explosive growth of data and the rapid progress of technology have led to a huge amount of data that is collected every day. In that data volume contains much valuable information. Data mining is the emerging field of applying statistical and artificial intelligence techniques to the problem of finding novel, useful and non-trivial patterns from large databases. It is the task of discovering interesting patterns from large amounts of data. This is achieved by determining both implicit and explicit unidentified patterns in data that can direct the process of decision making.

There are many data mining tasks, such as classification, clustering, association rule mining and sequential pattern mining. In that, sequential pattern mining is an important problem in data mining. It provides an effective way to analyze the sequence data. The goal of sequential pattern mining is to discover interesting, unexpected and useful patterns from sequence databases. This task is used in many wide applications such as financial data analysis of banks, retail industry, customer shopping history, goods transportation, consumption and services, telecommunication industry, biological data analysis, scientific applications, network intrusion detection, scientific research, etc. Different types of sequential pattern mining can be performed, they are sequential patterns, maximal sequential patterns, closed sequences, constraint based and time interval based sequential patterns.

Sequential pattern mining refers to the identification of frequent subsequences in sequence databases as patterns. In the last two decades, researchers have proposed many techniques and algorithms for extracting the frequent sequential patterns, in which the downward closure property plays a fundamental role. Sequential pattern is a sequence of itemsets that frequently occur in a specific order, where all items in the same itemsets are supposed to have the same transaction time value.

One of the challenges for sequential pattern mining is the computational costs beside that is the potentially huge number of extracted patterns. In this thesis, we present an overview of the work done for sequential pattern mining and develop parallel methods for mining frequent sequential patterns in sequence databases that can tackle emerging data processing workloads while coping with larger and larger scales.

Keyword: Data Mining, Frequent Pattern, Sequential Pattern, Parallel Mining, Dynamic Load Balancing, Multi-core processors, Sequence Database, Large Sequence Database.

Acknowledgments

First, I would like to express my sincere gratitude to my supervisors, Prof. Václav Snášel and Assoc. Prof. Bay Vo, for their continuous support, patience, motivation, enthusiasm and immense knowledge. Their guidance helped me in doing research, writing papers and writing this thesis. I also would like to thank them for all the activities they organized, such as coursework, which introduce me to the world of research.

In addition, I would like to thank the organizers, professors and secretaries of the sandwich program for allowing me to participate in the international network. Especially, I would like to give special thanks to Dr. Phan Dao for his advice and support while I studied in Vietnam and Czech Republic.

I would like to thank Ton Duc Thang University for their supporting me on during research time.

My sincere thanks also go to my colleagues Mr. Le Cung Tuong, Mr. Nguyen Anh, Mr. Nguyen Dinh Anh and Mr. Pham Phu Thanh Sang. They gave me much helpful advice and support on doing research during the past four years.

I would like to thank Assoc. Prof. Pavel Krömer for his support me to access the HPC server to implement some experiments.

I would like to thank all members of the committees of my state examination and thesis, for their insightful comments.

Finally, I sincerely thank my family who encouraged and motivated me and created favorable conditions during my PhD study.

List of Abbreviations

BIDE	*BI-Directional Extension*
CART	*Classification And Regression Trees*
ClaSP	*Closed Sequential Patterns*
CloFS-DBV	*Frequent Closed Sequences uses Dynamic Bit Vectors*
CloGen	*Closed-Generator*
CloSpan	*Closed Sequential pattern mining*
CM-SPADE	*Co-occurence MAP SPADE*
DAT	*Attribute of sequence such as the time stamp or space location*
CRM	*Customer Relationship Management*
DBV	*Dynamic bit vector*
DBV-SPM	*Dynamic bit vector Sequential pattern mining*
DDM	*Distributed Data Mining*
DFS	*Depth First Search*
DNA	*Deoxyribo Nucleic Acid*
EISP-Miner	*Enhanced Inter-Sequence Pattern Miner*
FITI	*First Intra Then Inter*
FP-array	*Frequent Pattern array*
FP-growth	*Frequent Pattern growth*
FP-tree	*Frequent Pattern tree*
FreeSpan	*Frequent pattern-projected Sequential PAttern mining*
FSP	*Frequent Sequential Patterns*
GapMis-OMP	*OpenMP-based version of GapMis*
GPU	*Graphics Processing Units*
GSP	*Generalized Sequential Pattern*
gSpan	*graph-based Substructure PAttern mining*
IAR Miner	*Inter-transaction Association Rule Miner*
ICMiner	*Inter-transaction Closed patterns Miner*
ID3	*Iterative Dichotomiser 3*
iISP-IC	*improved Inter-Sequence Pattern with Item Constraint mining*
IMSR_PreTree	*Improved mining sequential rules based on the prefix-tree*
ISP-IC	*Inter-Sequence Pattern with Item Constraint mining*
ISP-tree	*Inter-Sequence Pattern tree*
ITP-Miner	*Inter-Transaction Patterns Miner*

KDD	*Knowledge Discovery from Databases*
k-sequence	*Sequence with length k*
LP	*List of position*
LST	*Lexicographic Sequence Tree*
MNSR-PreTree	*Mining Non-redundant Sequential Rules based on the Prefix-tree*
MATI	*Multi-Core Apriori Transaction Identifiers*
minsup	*User-specified minimum support threshold*
OLAP	*Online Analytical Processing*
Par-CSP	*Parallel Closed Sequential Pattern*
pDBV-FCSP	*Parallel Dynamic Bit Vector Frequent Closed Sequential Patterns*
pDBV-SPM	*Parallel Dynamic Bit Vector Sequential Pattern Mining*
PFI-PrefixSpan	*Prefix-Frequent-Items PrefixSpan*
PGP-mc	*Parallel Gradual Pattern Extraction*
PIB-PRISM	*Parallel Independent Branch PRISM*
piISP-IC	*parallel iISP-IC algorithm*
PMFS-IB	*Parallel Mining Frequent Subgraphs with Independent Branch*
PMFS-SB	*Parallel Mining Frequent Subgraphs with Shared Branch*
PPL	*Parallel Patterns Library*
PrefixSpan	*Prefix-projected Sequential pattern mining*
PRISM	*PRIme-Encoding Based Sequence Mining*
PSP	*Prefix-tree for Sequential Pattern*
pSPADE	*Parallel Sequential Pattern Discovery using Equivalent classes*
SID	*Sequence identifier*
SIMD	*Single Instruction Multiple Data*
SMP	*Symmetric Multiprocessor*
SP	*Sequential Pattern*
SPADE	*Sequential Pattern Discovery using Equivalent classes*
SPAM	*Sequential PAttern Mining*
SPaMi-FTS	*Sequential Pattern Mining based on Frequent Two-Sequences*
SPM	*Sequential Pattern Mining*
SPMD	*Single Program Multiple Data*
SW	*Smith-Waterman*
TPL	*Task Parallel Library*
VLSI	*Very Large Scale Integration*

Annotation

I	$I = \{i_1, i_2, ..., i_m\}$	Set of items
X	$X = (I_1, I_2, ..., I_u)$, $I_k \subseteq I$, $1 \leq k \leq u$	Itemset
S	$S = \langle I_1 I_2 ... I_v \rangle$, $I_k \subseteq I$, $1 \leq k \leq v$	Sequence
TDB	$TDB = \{T_1, T_2, ..., T_n\}$, $T_i = \langle tid, X \rangle$, $1 \leq i \leq n$	Transaction database
SDB	$SDB = \{S_1, S_2, ..., S_n\}$, $S_i = \langle sid, S \rangle$, $1 \leq i \leq n$	Sequence database
$\sigma(p)$	$\sigma(p) = \lvert\{S_i \in SDB \mid p \subseteq S_i\}\rvert$	Minimum support (*minsup*)
Ψ	$\Psi = S_1[0] \cup S_2[d_2 - d_1] \cup \ldots \cup S_k[d_k - d_1]$, $k \geq 1$	Mega sequence
χ	$\chi = \{u_1, u_2, \ldots, u_k\}$	Set of constraint
α	$\alpha = \langle a_1\ a_2\ ...\ a_u \rangle$	Sequence α
β	$\beta = \langle b_1\ b_2\ ...\ b_v \rangle$	Sequence β
ξ	$\xi = \{\alpha \mid \alpha \in L \wedge \beta \in L: \alpha \subseteq \beta \wedge \sigma(\alpha) = \sigma(\beta)\}$, L is a completed set of frequent sequential patterns	Set of frequent closed sequential pattern
G	$G = \{2, 3, 5, 7, 11, 13, \ldots\}$	Set of prime numbers in ordered
$P(G)$	$P(G) \subseteq G$	Set of all subset of G
$\otimes S$	$\otimes S = s_1 \times s_2 \times ... \times s_n$, $s_i \in S$	Multiplication operator of S
$\otimes P(G)$	$\otimes P(G) = \{\otimes S: S \in P(G)\}$	All sets S in $P(G)$
$gcd(x, y)$	$gcd(x, y) = \prod_i p_i^{m_i}$	Greatest common divisor
$lcm(x, y)$	$lcm(x, y) = \frac{\lvert x \,.\, y \rvert}{\gcd(x,y)}$	Least common multiple

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the past few decades, the explosion of data science, the growth of technological advances in hardware and data storage have led to a growth in the size of databases. However, extracting useful information is extremely challenging because traditional data analysis methods cannot be used due to the massive size of databases.

Many techniques have been implemented in order to efficiently store and retrieve data from database systems. SQL and all other database query languages allow efficiently retrieving data, but they do not help users to automatically extract hidden knowledge. Artificial intelligence can help us in creating models and extracting information, but usually the proposed approaches do not scale well on a large set of data.

Data mining [42] is an important part of computer science, a main step in knowledge discovery in databases (KDD) [31] process, a research field that is focused on automatic extraction of hidden useful information from huge databases. It combines traditional database system techniques with information retrieval and artificial intelligence techniques (Figure 1.1). The data mining goal is to automatically identify correlations among data in order to extract useful and previously unknown information. On the other hand, recent progress in data mining research has led to the development of numerous efficient and scalable methods for mining interesting patterns and knowledge in large databases.

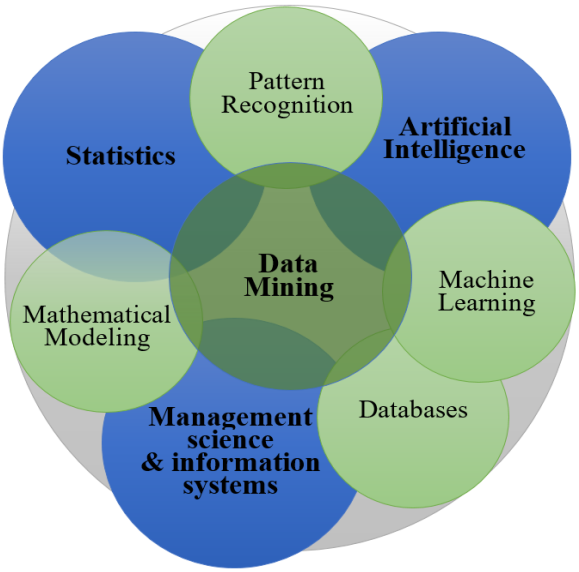


Figure 1.1. Data mining in hierarchy [31][1]

[1] http://frontender.com/blog/enablers/data-mining/

From a commercial viewpoint, there is the need of data mining because lots of data are being collected and warehoused (web data, e-commerce, bank transactions, etc.). The extracted information is frequently used to support marketing decisions. For example, results of a data mining tools can support a wide range of business intelligence applications such as customer profiling, targeted marketing and fraud detection. Hence, sales can use this information to implement more efficient strategies. Furthermore, from a scientific viewpoint, data are collected and stored at enormous speeds (sensors on satellites, telescopes, microarrays generating gene expression data, etc.) and traditional techniques are infeasible to treat them. Thus, data mining may help scientists in classifying and segmenting data, as well as in hypothesis formation.

Be it science, marketing, finance, health care, retail, or any other fields, the classical approach to data analysis relies fundamentally on one or more analysts becoming intimately familiar with the data and serving as an interface between the data, the users and products. For these applications, this form of manual probing of a data set is slow, expensive and highly subjective. In fact, as data volumes grow dramatically, this type of manual data analysis is becoming completely impractical in many domains [31].

1.1. Knowledge discovery process

Traditional methods of data analysis, based mainly on a person dealing directly with the data, do not scale to enormous databases. While database technology has provided us with the basic tools for the efficient storage and retrieval of large databases, 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 is the nontrivial process of identifying valid, novel, potentially useful and ultimately understandable patterns in data" [31]. 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.

Finding useful patterns in data has been given, including data mining, knowledge extraction, information discovery, information harvesting, data archeology and data processing. The term data mining has mostly been used by statisticians, data analysts and the management information systems communities. It has also gained popularity in the database field. The phrase KDD was coined to emphasize the knowledge of the product of a data-driven discovery.

KDD is the nontrivial process of identifying valid, novel, potentially useful and ultimately understandable patterns in data. Here, data are a set of facts and a pattern is an expression in some language describing a subset of the data or a model applicable to the subset. Hence, extracting a pattern also designates fitting a model to data, finding structure from data, or making any high-level description of a set of data.

The nontrivial term means that some search or inference is involved. Hence, it is not a straightforward computation of predefined quantities like computing the average value of a set of numbers. The discovered patterns should be valid on new data with some degree of certainty. The discovered patterns should also be novel and potentially useful to lead some benefit to the user. Finally, the patterns should be understandable, if not immediately, then after some post processing.

In order to uncover the interesting patterns hidden in large data, efficient and scalable knowledge extraction methods are needed. Knowledge is valuable only when it can be used effectively to improve the decision making process of an organization. The effective and efficient analysis of data from such different forms of data by integration of information retrieval, data mining and information network analysis technologies is a challenging task.

Traditionally, data were turned into knowledge by means of manual analysis. Since amount of data is growing at a phenomenal rate, this type of manual data analysis is becoming impractical in many domains. This motivates the need for efficient automated knowledge discovery processes. KDD is an organized process of identifying valid, novel, useful and understandable patterns from large and complex databases.

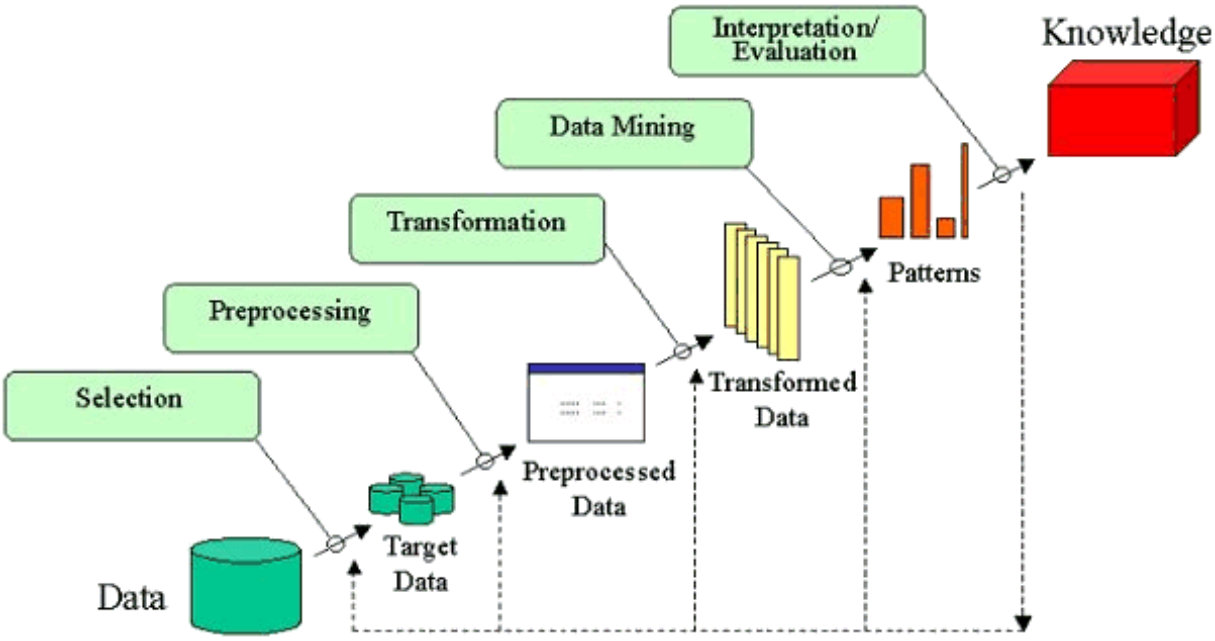


Figure 1.2. The knowledge discovery process[2]

The KDD process starts with the developing and understanding of the application domain, the relevant prior knowledge and the goals of the end-user. Then, a target of database is created by selecting from data sources or focusing on a subset of variables, or data samples, on which discovery is to be performed. After that, data are cleaned and preprocessed to transform the raw input data into a suitable format for subsequent analyzes. This step includes fusing data from multiple sources, cleaning data to remove noise or duplicated, collecting necessary information to model or account for noise, handling

[2] http://www2.cs.uregina.ca/~dbd/cs831/notes/kdd/1_kdd.html

missing data fields and accounting for time sequence information and known changes. Because of the many ways data can be collected and stored, data preprocessing is perhaps the most laborious and time-consuming step in the overall knowledge discovery process. Then, data are reduced and useful features are found to represent the data depending on the goal of the task. Furthermore, the appropriate data mining task and algorithms are selected and executed. At the end, as post-processing phase, mining patterns have to be interpreted and validated, thus consolidating the discovered knowledge.

KDD refers to the overall process of discovering useful knowledge from data which involve data preparation, search for patterns, knowledge evaluation and refinement, all repeated in multiple iterations, as shown in Figure 1.2. Data mining refers to a particular step in KDD process.

Data Selection: At this stage a target database relevant to the domain of application is selected or created from many different and heterogeneous data sources.

Data cleaning/Preprocessing: High quality data is a requirement for the KDD process to produce reliable knowledge. Hence this stage includes tasks such as missing value imputation, removal of noise or outliers, mapping of feature values onto appropriate domains and attribute discretization to improve the quality of data contained in the database.

Data reduction: In a high dimensional database all attributes are not equally significant. The task of data reduction is to identify predominant attributes and remove non-relevant ones from the database by preserving the knowledge involved in the database as much as possible. The process of attribute reduction will significantly reduce the complexity of the data mining operations and also increase the quality of the mined results.

Data mining: Data mining is the most important step in the knowledge discovery process. It is a process to extract hidden information from real world databases. Depending on the goals of the knowledge discovery process such as prediction or description, this step applies algorithms to extract the knowledge from the transformed data. Prediction is often referred to as supervised data mining while descriptive data mining includes the unsupervised and visualization aspects. Most data mining techniques are based on inductive learning where a model is constructed explicitly or implicitly by generalizing a sufficient number of training examples. This is usually called the generalization of knowledge and it is assumed that this trained model is applicable to future cases.

Interpretation and evaluation: The knowledge obtained as a result of performing a data mining task must be correctly interpreted and properly evaluated to ensure that the resulting information is meaningful and accurate. Also, it is to be presented to the users in an understandable manner. Generally visualization is the technique used to show the complex results of a data mining task in a simplified manner.

Knowledge presentation: To better understand and analyze the discovered knowledge of 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. If the data analyst is using some data mining algorithms and needs different data format, then it 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.

1.2. Data Mining

Information or patterns can be extracted from huge data repositories with data mining. The data can be analyzed with the purpose to find rules and patterns that describe its distinguished properties by using the data mining [2, 42] methods. A model representing the semantic structure of the database can be formed with the discovered information. This representation could further be used on new data for classification or prediction. Data mining is one of the main process of Knowledge Discovery in Databases.

Data mining is an extremely important step in the knowledge discovery process. It is the process of extracting interesting patterns from large amounts of data [42, 110], analyzes the data with the objective of finding rules and patterns to categorize the data [2]. Data mining also can solve problems by analyzing data already presented in databases. 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. It has wide application in all fields like market-basket analysis [2, 46], bioinformatics [27, 53], recommender systems [24, 52], web-mining [64, 84, 97], medicine [88, 116], weather prediction [78, 91], fraud detection [35], network intrusion detection [13, 19], smart health [54] and others. It can be performed on different type of data repositories including databases as well as transient data. These data repositories include 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, among several others. The challenges and techniques of data mining vary with the type of data repository [42]. Here we discuss briefly some of them.

- Flat files: Flat files are the most used data sources in data mining. This kind of data may be the simplest such as 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 Inmon [51], 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, where each dimension represents a set of attributes in the schema and each cell stores a value of some aggregate measures [30]. A data cube provides a multidimensional view of data and allows the pre-computation 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.

- Transaction 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 transaction 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 geography 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 database 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.

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, while 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 [42]. These two processes are sometimes referred to as supervised and unsupervised learning. Supervised learning has tagged data that are used to identify and categorize the future data. Unsupervised learning finds correlations without any external inputs other than the raw data to identify and categorize them. These techniques determine the type of patterns that are discovered from the data repositories. Among the various descriptive data mining tasks, frequent mining, association rule mining and sequential mining have been the most commonly employed tasks to facilitate

business development. The patterns that occur frequently in the database are called as frequent patterns. These patterns may include itemsets that appear together, correlated and frequent, such data when extracted from the transaction databases or other data repositories is called association rule mining [5, 46], for example, a customer buying soap and shampoo together. However, frequently occurring subsequence like a pattern where the customer purchase a butter followed by bread is a frequent sequential pattern.

Sequential pattern mining is similar to association rule mining, with an important difference of having the events being linked by time [2, 42, 92, 125]. Sequential patterns indicate the correlation between transactions while association rule represents intra transaction relationships [96, 102]. The mining results are based on the items being brought together frequently within the same transaction in case of the association rule mining [69]. While the results of sequential pattern mining are about which items are bought in a certain order, with those items coming from different transactions [2, 92]. Sequential patterns are generated by finding the correlation between transactions while association rule mining is based on intra transactions. To elaborate it further, association rule mining results are based on items that are purchased together frequently and belong to the same transaction [17]. However, results generated by sequential pattern mining are based on items that are brought in a certain order by the same customer and can be from different transactions.

Data mining has attracted a great deal of attention in the information industry and in society as a whole in recent years, due to the wide availability of huge amounts of data and the imminent need for turning such data into useful information and knowledge. The information and knowledge gained can be used for applications ranging from market analysis, fraud detection and customer retention, for production control and science exploration. This research study is based on sequential pattern mining and the next section discusses other important aspects of this data mining task.

1.3. Data Mining Tasks

The tasks of data mining are very diverse and distinct because there are many patterns in a large database. A data mining task can be considered as a kind of problem to be solved by a data mining algorithm. There are several data mining tasks and each task has its own requirements. In general, data mining task can be classified into two classes – descriptive and predictive (Figure 1.3). Descriptive mining tasks characterize the general properties of the data in the database. Predictive mining task perform inference on the current data in order to make predictions. This section reviews some of the basic data mining tasks.

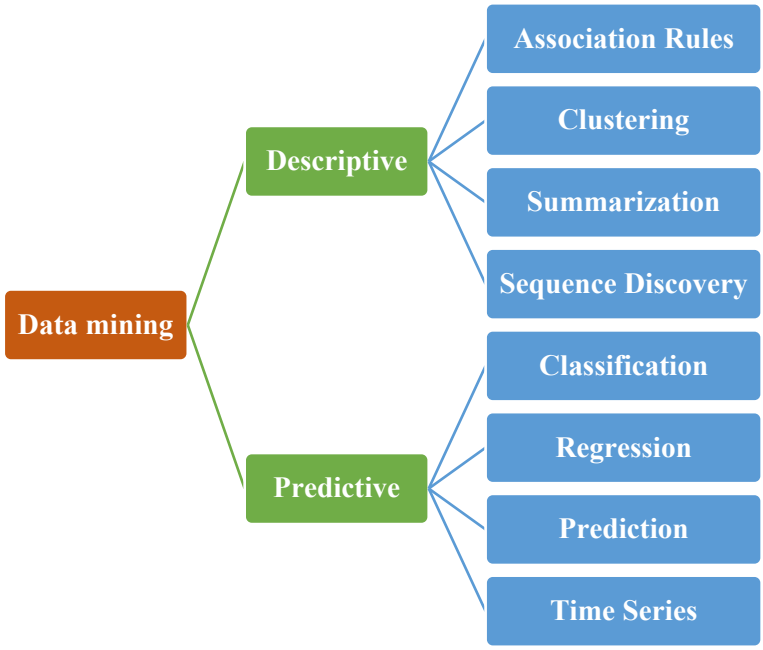


Figure 1.3. Data mining tasks

1.3.1. Clustering

Clustering is the identification of classes, also called clusters or groups, for a set of objects whose classes are unknown. The objected are so clustered that the intra class similarities are maximized and the interclass similarities are minimized based on some criteria defined on the attributes of objects. Once the clusters are decided, the objects are labeled with their corresponding clusters and common features of the objects in a cluster are summarized to form the class description.

For example, a bank may cluster its customers into several groups based on the similarities of their age, income, residence, etc. and the common characteristics of the customers in a group can be used to describe that group of customers. The clusters will help the bank to understand its customers better and thus provide more suitable products and customized services.

Clustering has many different methods and techniques. In the following we discuss the main clustering methods, briefly [12].

- Partitioning methods [56]: In partitioning methods the database 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 [8, 11, 67] 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 databases, partitioning based methods need to be extended.
- Hierarchical methods [81]: 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 [124] is well known hierarchical clustering algorithms.

- Density based methods [58]: 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 [112], OPTICS [7] and DENCLUE [50, 83] are the famous density based clustering algorithms.
- Grid based methods [26]: 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 [108] is a typical example of a grid based method.
- Model based methods [10]: 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 as COBWEB [61] or neural network approaches [90, 95].

1.3.2. Summarization

Summarization is the abstraction or generalization of data. A set of task-relevant data is summarized and abstracted, resulting a smaller set which gives a general overview of the data and usually with aggregation information. For example, the long distance calls of a costumer can be summarized into total-minutes, total-spending, total-calls, etc. Such high-level, summary information, instead of detailed calls is presented to the sales managers for customer analysis.

The summarization can go up to different abstraction levels and can be viewed from different angles. For example, the calling minutes and spending can be totaled along the calling period in weeks, months, quarters or years. Similarly, the calls can be summarized into in-state calls, state-to-state calls, Asia calls, Europe calls, etc., which can be further summarized into domestic calls and international calls. Different combinations of abstraction levels and dimensions reveal various kinds of patterns and regularities.

1.3.3. Association Rules

Mining association rules is the discover of togetherness or connection of objects. Such kind of togetherness or connection is termed as association rule. An association rule reveals the associative relationships among objects, i.e., the appearance of a set of objects in a database is strongly related to the appearance of another set of objects. For example, in a telecommunication database, an association rule that “call waiting” is associated with “call display”, denoted as “call waiting call display”, says if a customer subscribes to the “call waiting” service, he or she very likely also has “call display”.

The association rules can be useful for marketing, commodity management, advertising, etc. For example, a retail store may discover that people tend to buy soft drinks together with potato chips and then put the potato chips on sale to promote the sale of soft drinks.

1.3.4. Classification

Classification is the derivation of a function or model which determines the class of an object based on its attributes. A set of objects is given as the training set in which every object is represented by a vector of attributes along with its class. A classification function or model is constructed by analyzing the relationship between the attributes and the classes of the objects in the training set. Such a classification function or model can be used to classify future objects and develop a better understanding of the classes of the objects in the database.

For example, from a set of diagnosed patients, who serve as the training set, a classification model can be built, which concludes a patient's disease from his/her diagnostic data. The classification model can be used to diagnose a new patient's disease based on the patient's diagnostic data, such as age, sex, weight, temperature, blood pressure, etc.

To perform the classification, the target database is divided into two mutually exclusive and exhaustive sets called training set and test set. From the training data, by studying the relationship between the conditional attributes and the decision attributes, a model is derived to describe the concepts. The derived model can be presented in various forms such as prediction rules, decision trees, mathematical formula, or neural networks. The model is then used to predict the class of each data instance in the test set. The main goal of the classification task is to maximize the classification accuracy rate which is the ratio of the number of correct predictions to the total number of predictions in the test database.

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 [93]: The decision tree classification method is one of the most useful methods in data mining. Classifications based on trees are methodologies to classify data into discrete ones using the tree structured algorithms. The decision tree nodes are tested 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 [32], C4.5 [71] and CART [66] are used in classification to make predictions of current and future values of some attributes.

- Bayesian classifiers [99, 113]: Bayesian classification is based on Bayes' theorem. This method assumes that the effect of an attribute value of a given class is independent of the values of the other attributes.

- Rule based classifiers [73, 85]: 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.

1.3.5. Trend analysis

A lot of data available now are time series data that is accumulated over time. For example, a company's sales, a customer's credit card transaction and stock prices are all time series data. Such kind of data can be viewed as objects with an attribute time and the objects are the snapshots of entities with values that changes over time. It is interesting to find the patterns and regularities in the data evolutions along the dimension of time.

Trend analysis discovers interesting patterns in the evolution history of the objects. One topic in trend analysis is the identification of patterns in an object's evolution, such as up, down, peak, valley, etc. A model or function is constructed to simulate the behaviors of the object, which can be used to predict the future behaviors. For example, we can estimate this year's profit of a company from its last year's profit and the estimated annual increasing rate.

Another topic in trend analysis is the matching of the objects' changing trends, such as increasing streaks, decreasing streaks, etc. By comparing two or more objects' historical changing curves or tracks, similar and dissimilar trends can be discovered which will help us to understand the behaviors of the objects. For example, a company's sales and profit figures can be analyzed to find the disagreeing trends and search for the reasons behind such disagreements.

1.3.6. Rule Mining and Evaluation

Rule mining is a process to discover interesting relationships among data items in large databases. These relationships always represent the knowledge involved in the given database. This knowledge is expressed as implications in propositional logic. In data mining these rules are useful for analyzing and predicting the occurrence of an item based on the occurrences of other items in the transactions. Hence, these rules are commonly referred to as association rules. In an association rule the antecedent part represents an item or combination of items found in the given database and the consequent part is an item that is found in combination with the antecedent.

A challenging problem in rule generation is the assessment of derived rules based on the quality. The quality of a decision rule is measured in terms of its predictive accuracy, comprehensiveness and interestingness. The predictive accuracy of a decision rule generated from a training database is measured by applying the rule on a separate test database containing data instances that were not seen during training.

Most of the rule mining algorithms produce a large number of decision rules even for small databases. A large set of rules may give high predictive accuracy but it is not comprehensible in the sense that it is infeasible for the humans to manually inspect them to gain insight into the application domain. Hence, as a post processing operation to rule mining, efficient and effective methods are required to extract interesting, relevant and novel rules automatically. With this idea in mind, a new rule evaluation measure is introduced in this thesis, based on which an objective and automatic approach to rank and extract significant rules is proposed.

1.3.7. Attribute Reduction

The entire knowledge available in a high dimensional database is not always necessary to define various categories represented in the database. Though the machine learning and data mining techniques are suitable for handling data mining problems, they may not be effective for handling high dimensional data. When the dimension of the input data increases, the accuracy and efficiency of the results produced by the data mining operations decrease rapidly. This motivates the need for efficient automated feature selection processes in the area of data mining. Feature selection is a problem closely related to attribute reduction. It is a process which selects an optimal subset of features according to an objective function. The process of feature selection maps the original high dimensional feature space into a low dimensional space. The main objective of feature selection is to reduce the dimensionality of the database so as to improve mining performance such as speed of learning, predictive accuracy and comprehensibility of the mined results.

Different perspectives of feature selection include searching a feature subset followed by an evaluation criterion to test the optimality and feature ranking. Generally, a combinatorial nature of searching for an optimal subset is employed in most conventional feature selection methods. Due to the high computational cost involved in the searching process these methods are impractical in many cases. So modern feature selection approaches follow random search methods where an initial feature subset is selected as a starting point of search and features are added or removed iteratively until an optimal feature subset is obtained. Random search methods are also computationally expensive when dealing with high dimensional databases.

Another commonly used approach adopted in feature selection is to assign weights to the features based on some criteria and then rank individual features. After setting a suitable threshold value, the top ranked features are selected. Even though this method is efficient and easy to implement the selection of an appropriate threshold value is sometimes difficult. Another disadvantage of this method is that, at the time of weighting and ranking it does not consider the domain knowledge or the correlation between the features. These issues are handled in the attribute reduction part of this thesis. As a result, a domain dependent attribute weighting measure is proposed by exploiting the discrimination information involved in the conditional attributes to discriminate various objects of a decision table. Based on this measure a novel method is suggested to rank and select the significant features automatically.

1.3.8. Missing Value Handling

In a real world database, missing data have been one of the major factors affecting the quality of the data. Even a small percentage of missing data can cause serious problems with the analysis leading to wrong conclusions and imperfect knowledge. Fortunately, missing data imputation techniques can be used to improve the data quality. The conventional methods available to handle missing values focused on either by ignoring all cases with missing attribute values or by substituting plausible values for the missing items. Though these methods are easy to implement they have some serious drawbacks. Later, methods are suggested for knowledge acquisition directly from the original database with missing attribute values.

1.4. Data Mining Applications

Data mining is an important technology that has been integrated with many domains such as 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 use 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 applied for a longer time, 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 reacts 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 databases 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 the 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 patterns 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, a huge amount of data is 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.

1.5. Motivation

Mining sequential patterns in sequence databases have 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. 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, 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.

Although efficiency of mining the complete set of sequential patterns has 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 large and dense sequence databases. 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 databases require powerful computational resources. In fact, data mining algorithms working with very large databases 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 databases by analyzing them in parallel. This motivates our research and development parallel methods for mining frequent sequential patterns in large sequence databases. This thesis is especially concerned with improving the efficiency of parallel processing methods on multi-core processors architectures for enormous amounts of data for future parallel data mining systems.

1.6. Contributions

This section describes the contributions of this thesis to develop a model for large sequence database mining and parallel this model. Parallel methods for mining frequent sequential patterns are the main subject of this 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 is to efficient mining

sequential patterns in parallel environment, where the whole computation is broken up into tasks parallel. Two common ways to partition computation are task and data partitioning. We explore data parallel to extract patterns from sequence databases.

The contributions of this research as follow:

1. Presenting overview sequential pattern mining, parallel computing and data structure for representing compact sequence database.

2. Developing parallel methods for mining frequent sequential pattern that utilizes task parallel. Parallel approaches are propose for extracting sequential patterns in large sequence database, these methods have several advantages over task parallel. This simplifies programming and leads to a development time significantly smaller than the one associated with task parallel programming, because a lot of previously written serial code can be reused.

3. Developing a parallel method for mining frequent closed sequential pattern with dynamic load balancing. Parallel approach is propose for extracting closed sequential patterns in large sequence database that solves the load balance issues of the workload between processors with a dynamic mechanism that re-distributes the work when some processes are out of work to minimize the idle CPU time.

4. Proposing three novel methods for mining inter-sequence patterns with item constraints. In that, apply parallel processing to find patterns led to improving the performance. In our research, we demonstrated that algorithms have good performance and a good work loading as well. In fact, it tries to optimize local mining at processors while minimizing the data transfer among them.

1.7. 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: Background. In this chapter introduces some basic concepts and models of sequential pattern mining, theoretically, we show the kinds of sequential patterns and some properties of them. We presented overview some traditional frequent pattern mining framework, parallel computing and a compact data structure.

Chapter 3: Parallel sequential pattern mining. This chapter introduces efficiency parallel approaches for mining frequent sequential pattern, mining inter-sequence patterns with item constraints. We presented the experimental results of our proposed methods for proven the efficiency.

Chapter 4: Parallel closed sequential pattern mining. We introduce an approach for mining frequent closed sequential pattern on large database. We show that there are rarely parallel methods using multi-core processors for closed sequential pattern deal with dynamic load balancing on large sequence databases and it can be one of the most important motivations and advantages of our approach.

Chapter 5: Conclusion and future works. Summarizing the whole thesis and giving directions for future works.

1.8. Summary

In this chapter, we provide an overview of the knowledge discovery process, data mining and its basic approaches. In addition, a brief introduction of data mining tasks and data mining applications. The motivations and contributions are primarily content presented in this chapter, includes:

- Presenting a data structure for representing compact sequence database.
- Developing parallel methods for mining frequent sequential pattern that utilizing task parallel.
- Developing a parallel method for mining frequent closed sequential pattern with dynamic load balancing.
- Proposing three novel methods for mining inter-sequence patterns with item constraints.

Chapter 2

Background

In this chapter, we introduce some basic concepts of sequential pattern mining and overview some traditional frequent pattern mining methods, which include sequence mining and closed pattern mining. Then we introduce the parallel computing, multi-core processors architecture, review some parallel sequential patterns and present a compact data structure to store database for sequential pattern mining.

2.1. Preliminaries

Definition 2.1 (itemset). Let $I = \{i_1, i_2, ..., i_m\}$ be a set of m distinct attributes, also called *items*. An itemset $X = (I_1, I_2, ..., I_u)$, such that $I_k \subseteq I$ $(1 \leq k \leq u)$ is a non-empty unordered collection of items. Each itemset is given in brackets; for itemsets that contain only a single item, the brackets are omitted. For example, the itemset (ABC) represents the sets of items A, B, C and itemset (B) can be written as B.

Definition 2.2 (transaction database). A transaction database is a set of transactions, denoted $TDB = \{T_1, T_2, ..., T_n\}$, where n is the number of transaction in TDB, each transaction $T_i = \langle tid, X \rangle$ $(1 \leq i \leq n)$ consists of a set of items where *tid* is the transaction identifier and X is an itemset.

Definition 2.3 (itemset support). The absolute support of an itemset $X \subseteq I$ in TDB, denoted $\sigma(X)$, is the number of transactions in TDB which contain X. The relative support of X in TDB is the probability that X occurs in a transaction or in other words, the support of X divided by the total number of transactions in the database. An itemset is frequent if its support is not less than an user specified threshold called minimum support (denoted *minsup*).

Table 2.1. An example transaction database

tid	Items
T_1	⟨*bread, milk, cheese*⟩
T_2	⟨*cheese, butter, cereal*⟩
T_3	⟨*milk, cheese, cereal*⟩
T_4	⟨*bread, milk, cereal*⟩

For example, assume $I = \{bread, milk, cheese, butter, cereal\}$, $TDB = \{T_1, T_2, T_3, T_4\}$ as Table 2.1, where $T_1 = \langle bread, milk, cheese \rangle$, $T_2 = \langle cheese, butter, cereal \rangle$, $T_3 = \langle milk, cheese, cereal \rangle$, $T_4 = \langle bread, milk, cereal \rangle$ and $minsup = 2 = 50\%$. The absolute support of cheese is $\sigma(cheese) = 3$ since three transactions, namely T_1, T_2 and T_3 contain *cheese* and the relative support is 75% > 50% so *cheese* is a frequent itemset, *butter* is infrequent because $\sigma(butter) = 1 = 25\% < minsup$.

Definition 2.4 (sequence database). A sequence database is a set of sequences, denoted $SDB = \{S_1, S_2, ..., S_n\}$, where n is the number of sequences in SDB and each sequence S_i $(1 \leq i \leq n)$ has the form $\langle sid, S \rangle$,

where each *sid* is a unique sequence identifier of sequence and $S = \langle I_1\ I_2 \ldots\ I_v\rangle$, such that $I_j \subseteq I\ (1 \le j \le v)$ is a ordered list of event or itemsets.

Definition 2.5 (sequence). A sequence $\alpha = \langle a_1\ a_2 \ldots\ a_u\rangle$ is an ordered list of itemsets, where itemset a_1 occurs before a_2, which occurs before a_3 and so on. A sequence element a_i is an itemset and u (the size of a sequence) is the number of itemsets (or elements) in the sequence. An item can occur at most once in an itemset of a sequence, but can occur multiple times in different itemsets of a sequence. For example, Table 2.2 have four sequences having the *sids* are S_1, S_2, S_3 and S_4. The first sequence $S_1 = \langle CAA(AC)\rangle$ contains four itemsets. It indicates that item *C*, were followed by *A*, then *A* and lastly are items *A* and *C* occurred at the same time.

Definition 2.6 (length of sequence). Given a sequence $\alpha = \langle a_1\ a_2 \ldots\ a_u\rangle$, the length of a sequence is the total number of items in the sequence, denoted as $k = \sum_j |a_j|$. A sequence with length *k* is called a *k*-sequence. For example, the sequence $\langle B(AC)\rangle$ is a 3-sequence of size 2.

Definition 2.7 (extending the length of a sequence). Given a sequence $\alpha = \langle a_1, a_2, \ldots, a_u\rangle$ and an item e_k. Sequence β is called the extending length of the sequence α with the item e_k if and only if $\beta = \langle a_1, a_2, \ldots, a_u, a_{u+1}\rangle$ and $a_{u+1} = e_k$.

Definition 2.8 (prefix). A sequence $\alpha = \langle a_1, a_2, \ldots, a_u\rangle$ is called a prefix of a sequence $\beta = \langle b_1, b_2, \ldots, b_v\rangle$ if and only if $\forall u < v$ then $a_1 = b_1$, $a_2 = b_2, \ldots, a_u = b_v$.

Definition 2.9 (pattern). A pattern is a subsequence of ordered itemsets, each itemset in a pattern is called an element. For example, the sequence $\langle B(AC)\rangle$ is a subsequence of $\langle (AB)E(ACD)\rangle$ so that it is a pattern.

Table 2.2. Sequence database

sid	Sequence
S_1	$\langle CAA(AC)\rangle$
S_2	$\langle AB(ABC)B\rangle$
S_3	$\langle A(BC)ABCE\rangle$
S_4	$\langle AB(BC)AD\rangle$

Definition 2.10 (subsequence and super sequence). A sequence $\beta = \langle b_1\ b_2 \ldots\ b_v\rangle$ is called a subsequence of sequence $\alpha = \langle a_1\ a_2 \ldots\ a_u\rangle$ and α is a super sequence of β, denoted as $\beta \subseteq \alpha$, if there exists integers $1 \le j_1 < j_2 < \ldots < j_u \le v$ such that $b_k \subseteq a_{jk}$, $1 \le k \le u$. For example, the sequence $\langle B(AC)\rangle$ is a subsequence of $\langle (AB)E(ACD)\rangle$, but $\langle (AB)E\rangle$ is not a subsequence of $\langle ABE\rangle$.

Definition 2.11 (support). The absolute support of a sequence α in *SDB* is defined as the total number of sequences in *SDB* that contain α, denoted as $\sigma(\alpha) = |\{S_i \in SDB \mid \alpha \subseteq S_i\}|$. The relative support of α is given as the fraction of sequences that contain α in a database. Absolute and relative supports are sometimes used interchangeably.

Definition 2.12 (horizontal database format). A sequence database in the horizontal format is database where each row is a transaction in the form ⟨*sid-itemset*⟩, where *sid* is a sequence ID and itemset is a set of items that appear in that sequence, Table 2.2 shows an horizontal sequence database.

Definition 2.13 (vertical database format). A sequence database in vertical format is a database where each row has a transaction in the form ⟨*item*, *sid*⟩, where *sid* is a set of sequence IDs containing item. Table 2.3 shows the vertical representation of the database of Table 2.2.

Table 2.3. The vertical representation of database shown in Table 2.2

	A	***B***	***C***	***D***	***E***
S_1	2, 3, 4		1, 4		
S_2	1, 3	2, 3, 4	3		
S_3	1, 3	2, 4	2, 5		6
S_4	1, 4	2, 3	3	5	

Definition 2.14 (sequential pattern mining). Given a *SDB* and a user-specified threshold, called the minimum support (denoted *minsup*), a sequence α is said to be a frequent sequential pattern if it occurs more than *minsup* times, that is $\sigma(\alpha) \geq minsup$.

Definition 2.15 (maximal sequential pattern). A sequential pattern is *maximal* if it is not a subsequence of any other sequential patterns.

Definition 2.16 (closed sequential pattern). A sequential pattern is *closed* if it is not a subsequence of any other sequential patterns with the same support.

Definition 2.17 (*s*-extension). The *s*-extension of a sequential pattern $\langle a_1, a_2, \ldots, a_u \rangle$ by item I_x is defined $\langle a_1, a_2, \ldots, a_n, I_x \rangle$.

Definition 2.18 (*i*-extension). The *i*-extension of a sequential pattern $\langle a_1, a_2, \ldots, a_u \rangle$ by item I_x is defined $\langle a_1, a_2, \ldots, a_u \cup I_x \rangle$.

Definition 2.19 (parameters for standard databases). Synthetic databases were generated using the IBM synthetic data generator to mimic transactions in a retail environment. The synthetic data generation program used the following parameters. (Notes: parameters in Table 2.4 is different with Anotation table)

Table 2.4. Parameters for synthetic databases

C	Average number of items per sequences
T	Average number of items per itemset
S	Average number of itemsets in maximal sequences
I	Average number of items in maximal sequences
N	Number of distinct items
D	Number of sequences

Property 2.1 (Apriori Property) All non-empty subsets of a frequent itemset must also be frequent, any superset of some infrequent itemset cannot be frequent. An itemset is said "*frequent pattern*" if its frequency is no less than a given minimum support threshold. The property can be explained as follows. Assuming X and Y are two patterns, $\sigma(X) \leq \sigma(Y)$ if $X \subseteq Y$. For example, assuming $\langle A, B, C\rangle$ is frequent, all of its sub-itemsets such as $\langle A, C\rangle$ are also frequent. If $\langle D, E\rangle$ is infrequent, its supersets such as $\langle A, D, E\rangle$ and $\langle D, E, F\rangle$ are infrequent.

Proposition 2.1 (pruning a prefix). Considering the prefix $\alpha_p = \langle a_1 a_2 \ldots a_u\rangle$. If there exists an item a before the starting position of prefix α_p in each of the transactions containing α_p in *SDB*, the extension can be pruned by prefix α_p. For example, considering *SDB* in Table 2.2, there is no need to extend prefix $\langle B\rangle$ because there exists a sequence $\langle A\rangle$ that occurs before $\langle B\rangle$ in each transaction that contains prefix $\langle B\rangle$. If we extend prefix $\langle B\rangle$, the obtained results will be absorbed due to the extension of prefix $\langle B\rangle$ already containing $\langle B\rangle$ and having the same support.

Proposition 2.2 (checking sequence closure). If there exists a sequence β that is a forward-extension or backward-extension of sequence α, sequence α is not closed and α can be safely absorbed by β. Considering the above example, supposed that $\alpha = \langle CC\rangle$:2 and $\beta = \langle CAC\rangle$:2, then $\langle CC\rangle$:2 will be absorbed by $\langle CAC\rangle$ because $\langle CC\rangle \subseteq \langle CAC\rangle$ and $\sigma(\langle CC\rangle) = \sigma(\langle CAC\rangle) = 2$.

2.2. Sequential pattern mining

The sequential pattern mining problem was first introduced by Agrawal and Srikant in 1995 [2] based on their study of customer purchase sequences, as follows: "*Given a set of sequences, where each sequence consists of a list of events (or elements) and each event consists of a set of items and given a user-specified minimum support threshold of minsup, sequential pattern mining finds all frequent subsequences, that is, the subsequences whose occurrence frequency in the set of sequences is no less than minsup*."

Sequential pattern mining (SPM) is an important task in data mining with computationally challenging because such mining may generate and/or test a combinatorically explosive number of intermediate subsequences and is applied in many domains, such as analysis of buying behavior of customers [2], analyzing web access [84, 97], weather prediction [78, 91], network anomaly detection [13, 19, 127], DNA sequence analysis [114, 126], or genetic structure [129], etc. The input data, called a sequence database, are a set of sequences. Each sequence is an ordered list of events (or elements), each of which contains a set of items. The problem is to find all frequent subsequences which satisfy the user-specified minimum support threshold. For example, when customers buy a computer, they will buy from potential together with the projector or when they will buy more RAM, HDD. Sales can use this information to analyze customer behavior, to understand the concerns as well as the satisfaction of our customers, anticipate and satisfy customer needs in the future.

It is challenging to discover all frequent subsequences in huge databases as the search space is extremely large. The factors that make sequential pattern mining difficult and time consuming are as follows:

- The pattern can be formed using single item as well as itemsets.

- The details regarding the number of itemsets in a pattern or the number of items in an itemset is not known beforehand.

- The permutation and combination of all possible items in the database are used to form patterns.

Recent developments have made progress in two directions: (1) efficient methods for mining the full set of sequential patterns and (2) efficient methods for mining only the set of closed sequential patterns, where a sequential pattern α is closed if there exists no sequential pattern β where β is a proper super sequence of α and β has the same support as α. Because all of the subsequences of a frequent sequence are also frequent, mining the set of closed sequential patterns may avoid the generation of unnecessary subsequences and thus lead to more compact results as well as more efficient methods than mining the full set. This section will examine methods for mining the full set and then study how they can be extended for mining the closed set.

There are many efficient algorithms for mining the full set of frequent sequential patterns (FSP), including GSP [96], AprioriAll [2], PrefixSpan [79], SPADE [121], PRISM [37], CM-SPADE [34] and SPaMi-FTS [44].

All approaches either directly or indirectly explore the Apriori property, stated as follows: *every nonempty subsequence of a sequential pattern is a sequential pattern*. The Apriori property is anti-monotonic (or downward-closed) in that, if a sequence is infrequent, all of its super sequences will be infrequent. Use of this property to prune the search space can help make the discovery of sequential patterns more efficient.

The general idea of all existing methods is to start with general (short) sequences and then extend them towards specific (long) ones. Existing methods can be divided into the following three main types as below.

(1) Horizontal methods. Databases are organized in the horizontal format, where each row has a transaction in the form ⟨*sid*-*itemset*⟩, where *sid* is a sequence (customer) ID and *itemset* is a set of items that appear in that sequence. The Generalized Sequential Pattern (GSP) [96], AprioriAll [2] and Prefix-tree for Sequential Pattern (PSP) [70] algorithms are examples of horizontal methods.

(2) Vertical methods. Databases are organized in the vertical format, where each row has a transaction in the form ⟨*item*, *sid*⟩, where *sid* is a set of sequence IDs containing *item*. In this layout, the support of a sequence is the size of the set of *sids*. Represented for this layout are SPADE [121], SPAM [9], PRISM [37] and PIB-PRISM [47].

(3) Projection methods. Projection methods are hybrid methods that combine horizontal and vertical approaches. Given any prefix sequence *P*, the main idea is to project the horizontal databases based on the methodology of pattern-growth mining of frequent patterns in transaction databases developed in the FP-growth algorithm [41]. Hence, the projected database contains only those sequences that have *P*. The frequency of extensions of *P* can be directly counted in the projection database. PrefixSpan [79], an extension of FreeSpan [40, 79], is an example. The prefix tree

architecture is efficient for organizing and storing candidate sequences it was represented by IMSR_PreTree [105], MNSR-PreTree [80] and PFI-PrefixSpan [43].

In this section, we briefly introduce some algorithms for sequential pattern mining.

2.2.1. AprioriAll

AprioriAll [2] is believed to be the first algorithm solving sequential pattern mining. First, it finds all frequent 1-patterns whose support values satisfy a user-defined minimum support. Then, it initializes and maintains two types of list containers, namely the candidate lists and the frequent pattern lists. For every (k+1)-candidate constructed by joining two frequent k-patterns, the support needs to be scanned from the original database. The process repeats until no further patterns can be found.

2.2.2. GSP

GSP (Generalized Sequential Patterns) algorithm [18, 96] is a very popular method of sequential pattern mining. It is an extension of the Apriori algorithm [2] for sequence mining. The main structure is similar to AprioriAll and the details are as follows. First, it scans the database to obtain the frequent 1-sequences. Then it generates the next level candidates by joining the previous level frequent sequences, the same as AprioriAll. The differences are in the candidate generation and candidate support counting. In the candidate generation stage, they use a mechanism to prune the unpromising candidates. Thus in the same level, the number of candidates is no more than that of AprioriAll. In the support counting stage, a hash-tree data structure is used to reduce the number of candidates to be checked. The representation of the database is transformed to efficiently determine whether a specific candidate is contained in the database.

2.2.3. SPAM

SPAM (Sequential Pattern Mining) algorithm [9] is a depth-first algorithm that integrates the ideas of GSP [96], SPADE [121] and FreeSpan [40]. A group of novel concepts such as the sequence-extension step, itemset-extension step and the lexicographical tree are firstly introduced. Similar to FreeSpan, SPAM uses a depth-first strategy to traverse the lexicographical tree to extract the complete set of frequent sequential patterns. More importantly, SPAM encodes the *id-list* from SPADE to a vertical bitmap data structure and puts them in the memory so that the "joining" operation between two *id-lists* is extremely fast. That is the key reason why SPAM outperforms any of the previous algorithms.

2.2.4. SPADE

SPADE (Sequential Pattern Discovery using Equivalent classes) algorithm [121] is also a level-wise sequential pattern mining algorithm that uses a vertical data format. The sequential database is converted into a vertical *id-list* database format and each *id* is associated with corresponding items and the time stamp. This algorithm aims to find frequent sequences using efficient lattice search techniques and simple joins. All the sequences are discovered with only three passes over the database. The frequent items are discovered in first scan, the frequent sequences of length 2 are searched in the second scan and the last scan associates a table of the sequences *id* and itemsets *id* in the database to the corresponding frequent sequences of length 2. The method forms the enumeration of the candidate sequences, based on their common prefixes. The key difference between SPADE with AprioriAll [2] and GSP [96] is that

SPADE avoids scanning the original database or a representation. Instead, SPADE builds an *id-list* for each candidate. The support count of the candidate can be easily calculated from its *id-list*, which greatly reduces the cost of scanning.

2.2.5. FreeSpan

FreeSpan (Frequent pattern-projected Sequential pattern mining) algorithm [40] is the first projection-based depth first algorithm. Similar to the previous algorithms, FreeSpan scans the database once to obtain the frequent 1-sequences and put them in the *f-list*. Then it constructs a matrix called *s-matrix* which contains the 2-sequences and their supports generated from the *f-list* and the infrequent ones are filtered. Each sequential pattern in the *s-matrix* corresponds to a projected database that all the sequences contain the sequential pattern itself. The next step is to construct level-2-sequences from the *s-matrix* and find annotations for repeating items and projected databases in order to discard the matrix and generate level-3 projected databases. The process repeats until no candidates can be generated.

2.2.6. PrefixSpan

PrefixSpan (Prefix-projected Sequential pattern mining) algorithm [79] is an extends of the pattern-growth approach for frequent pattern mining and the first algorithm that does not generate a candidate. As an enhanced algorithm of FreeSpan [40] is the first algorithm proposed that considers the projection method for SPM. PrefixSpan uses the "prefix" of the sequence to project the database. Then it scans the projected database for the items to be concatenated to the prefix and counts the support for each item. The infrequent concatenation items will be discarded and frequent items will be retained. Lastly, for each frequent concatenation item, a new prefix and its corresponding smaller projected database can be constructed. The process continues until no more frequent concatenation items can be scanned.

2.2.7. Limitations of sequential pattern mining

A major challenge in frequent patterns mining from large database is the fact that the large number of patterns is usually generated and many of them are redundant. This happens especially when the minimum support threshold is low. This is because if a pattern is frequent, all of its sub-patterns are frequent as well. A very long pattern will contain an exponential number of smaller, frequent sub-patterns, which makes the number of patterns grows explosively. On the other hand, truly valuable patterns in which users might be interested might be flooded in hundreds of thousands of similar patterns.

Although there is an improvement in the efficiency of mining completed set of sequential patterns, but still in many cases, sequential pattern mining faces strong challenges (Figure 2.1). The limitations of SPM algorithms can be summarized as below:

- **Efficiency:** A huge set of sequential patterns are generated especially if the database is large. Generally, only a small subset of such patterns is of an interest for the user.

- **Understandability and Usefulness:** It is difficult to understand and hard to use the large number of mined sequential patterns for a real world application.

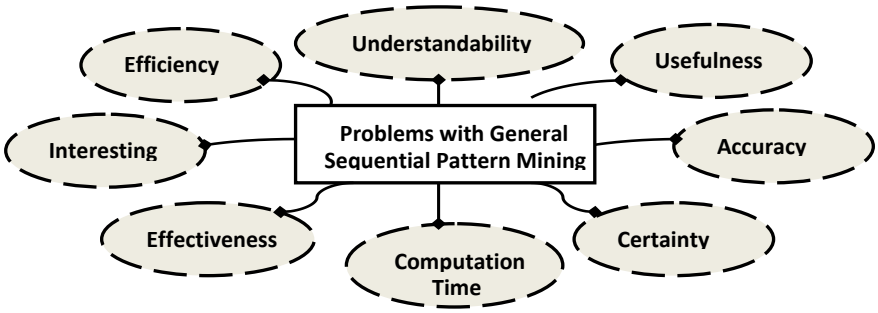


Figure 2.1. Challenges of sequential pattern mining algorithms

- **Effectiveness:** In addition, the algorithms may reduce the number of patterns but unimportant patterns are still found in the result patterns. It would be better if the unimportant patterns could be pruned first, resulting in fewer but important patterns after mining.

- **Certainty:** Although efficient algorithms have been developed to tackle these problems, the performance of the algorithms dramatically degrades in case of mining long sequential patterns in dense databases or using low minimum supports.

- **Interestingness:** In general, SPM considers subsequences and items in a sequence uniformly. The data elements that have different importance in real world applications are not differentiated and thus general sequential pattern mining is limited in finding more interesting characteristics embedded in the real world applications. There are several applications in which specific sequences are more important or have more priority than other sequences.

- **Computation Time:** Mining a large amount of sequential patterns from large sequence databases is a computationally expensive task. If we can focus on only those sequential patterns that are interesting to users, we may be able to save a lot of computation cost [96].

- **Accuracy:** To ensure that the results of sequential pattern mining are completed and accurate is a challenge.

2.3. Frequent closed sequential pattern mining

Frequent closed sequential pattern mining is slightly different from frequent sequential pattern mining. The definition is as follows. Given a *SDB* and a minimum support σ, assume L contains all the frequent sequential patterns in *SDB* that satisfies σ. The set of frequent closed sequential pattern ξ is defined as $\xi = \{\alpha | \alpha \in L \ \wedge \beta \in L$ such that $\alpha \subseteq \beta \wedge \sigma(\alpha) = \sigma(\beta)\}$.

Because mining the completed set of frequent subsequences can generate a huge number of sequential patterns, an interesting alternative is to mine frequent closed subsequences only, that is, those containing no super sequence with the same support. Frequent closed sequential patterns (FCSP) mining can produce a significantly less number of sequences than the full set of sequential patterns. Note that the full set of frequent subsequences, together with their supports, can easily be derived from the closed subsequences. Thus, closed subsequences have the same expressive power as the corresponding full set of subsequences. Because of their compactness, they may also be quicker to find.

Many algorithms have been proposed for mining frequent closed sequential patterns to reduce the required storage space and runtime. Popular algorithms for frequent closed sequential pattern mining are CloSpan [117], ClaSP [36], BIDE [109, 111] and CloFS-DBV [101]. CloSpan uses a maintain-and-test pattern method and combines a hash-index structure with a tree structure for storing sequences. This algorithm prunes the patterns using techniques such as Common Prefix and Backward Sub-Pattern to reduce the search space. The ClaSP algorithm uses a vertical database format strategy and a heuristic to prune non-closed sequences. This algorithm maintains previous candidates to test the closure of sequences and remove them later. The maintenance of candidates increases memory consumption and the number of test candidates increases with the number of generated frequent closed sequences. Most of these algorithms maintain mined frequent itemsets in order to test frequent closed sequences, which requires a lot of memory.

BIDE does not need to keep track of any single historical frequent closed sequences (or candidates) for a new pattern's closure checking. It uses bi-directional extension techniques to examine frequent closed patterns as candidates before extending a sequence. Moreover, the algorithm uses a BackScan process to determine candidates that cannot be extended to reduce mining time and pseudo projection techniques to reduce database storage space and is efficient for low support thresholds. However, this method has to project and scan databases many times for each prefix, making it inefficient. In addition, CloFS-DBV [101] algorithm uses dynamic bit vector (DBV) structure combined with location information in the structure of the transaction CloFS-DBVPattern to mine frequent closed sequences. CloFS-DBV early prunes of prefix sequences and checks the backward extension and forward extension quickly based on CloFS-DBVPattern structure. For each transaction, it just considers the start position or the last position of the sequence. Therefore, if the sequence has N transactions, the CloFS-DBV takes only N operations to check each candidate. In contrast, BIDE algorithm that is more efficient than CloSpan algorithm in almost all the cases use a local database to check backward extension and use a projected local database to check forward extension. Let k be the sequence length and N be the number transaction of sequence. Thus, BIDE requires $k \times N$ operations to check each candidate.

2.4. Parallel computing

Data mining is often 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 database. 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 databases. 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, minimize communication and data transfer to avoid wasting resources [23].

With the availability of huge databases 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 become essential.

Parallel computing is a form of computation in which many calculations are carried out simultaneously [3]. It is based on the principle that large problems can often be divided into smaller ones, which are then solved concurrently. Some of the more commonly used terms associated with parallel computing are listed below [115].

Speedup: It is one of the simplest and most widely used indicators for a parallel program's performance. It is defined as: $s_n = \frac{t_s}{t_p}$, where t_s is the execution time using only one processor and t_p is the execution time using n processors. The maximum speedup that can be reached is linear speedup.

Scale up: It captures how well the parallel algorithm handles larger databases when more processors are available. Scale up study measures execution times by keeping the problem size per processor fixed while increasing the number of processors.

Shared memory system: Multiple processors can operate independently but share the same memory resources. Thus, data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs. However, adding more CPUs can geometrically increase traffic on the shared memory-CPU path.

Distributed memory system: Each processor has its own local memory, a communication network is used to connect the inter-processor memory. Increasing the number of processors can increase size of memory proportionately. Since the concept of cache coherency and global address do not apply in a distributed memory system, the programmer is responsible for many of the details associated with data communication among processors.

Currently, hardware architects are no longer using the additional transistors afforded by Moore's Law to make sequential processors faster, but instead to put more processing cores on a single chip, leaded to so-called chip multiprocessors [94] or simply multi-core processors which lead to an increasing interest in parallel computing strategies. Some researches take full advantage of these architectures in order to improve performance of data mining tasks.

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, workload unbalancing and consolidation. In parallel environment, there is a startup cost associated with initiating each process. In this situation, often one process for communication with others may be forced to wait. System resources are limited, 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 [55, 123]:

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

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. 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.

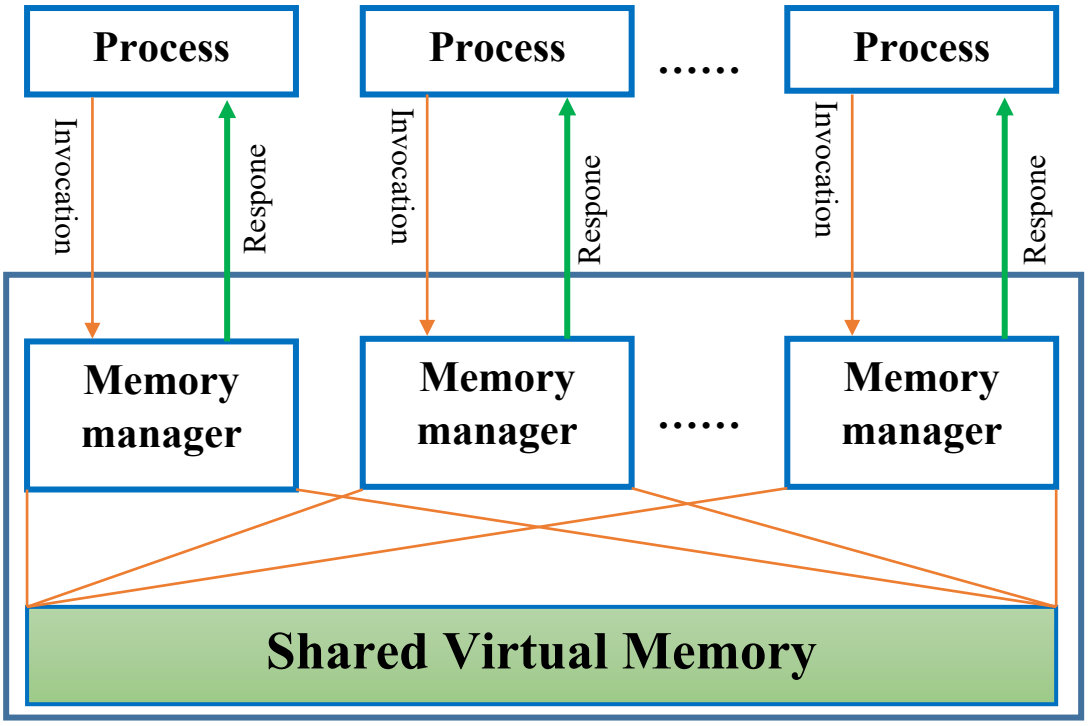


Figure 2.2. Distributed shared memory[3]

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. 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 that 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.

An easy way to discuss the many parallels and distributed mining methods is to describe them in terms of the computation and data partitioning methods used (Figure 2.3). 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.

[3] https://en.wikipedia.org/wiki/Distributed_shared_memory

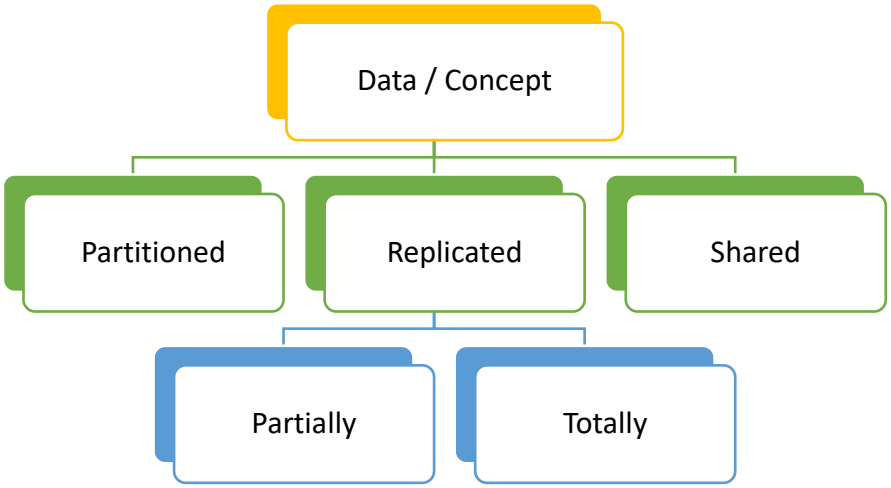


Figure 2.3. Concepts for parallel and distributed mining methods

Recently, there is a distributed programming model called MapReduce [62, 82] raise researchers and developers concerns. MapReduce is developed by Google, intended for processing massive amounts of data in large clusters. It is an implemented as two functions, Map which applies a function to all the members of a collection and returns a list of results based on that processing and Reduce, which collates and resolves the results from two or more Maps executed in parallel by multiple threads, processors, or stand-alone systems. Both Map and Reduce may run in parallel, though not necessarily in the same system at the same time.

The recent improvements of Graphics Processing Units (GPU) offer a powerful processing platform for both graphics and non-graphics applications [77]. GPU has a parallel "multi-core" architecture, each core capable of running thousands of threads simultaneously - if an application is suited to this kind of an architecture, the GPU can offer large performance benefits. However, a typical computation running on the GPU must express thousands of threads in order to effectively use the hardware capabilities. Therefore, finding large scale parallel is important for programming on GPU. The introduction of the NVIDIA CUDA, through a C-based API, an easy way to take advantage of the high performance of GPUs for parallel computing.

2.5. Multi-core processors architecture

The computers with single processors which has limited the performance and efficiency of the computers. The classic way of overcoming the performance issue was to use bigger processors for executing the data with higher speed. Big processor did improve the performance to certain extent but these processors consumed a lot of power which started over heating the internal circuits. To achieve the efficiency and the speed simultaneously the CPU architectures developed multi-core processors units in which two or more processors were used to execute the task. The multi-core technology offered better response-time while running big applications, better power management and faster execution time.

Multiple cores are embedded in modern CPU, using non trivial cache hierarchies to access the main memory and equipped with a large set of SIMD instructions. In multi-core processors architecture, a processor includes two or more independent cores in the same physical package [6, 94, 104], with each

processing unit having separate memory and shared main memory. An example of a quad-core processor is shown in Figure 2.4. Multi-core processors allow multiple tasks or processes to be executed simultaneously to increase performance.

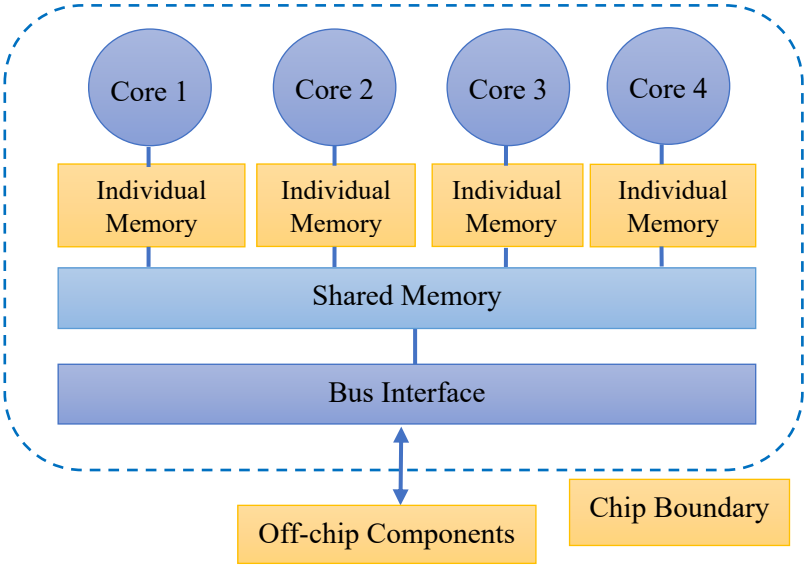


Figure 2.4. Diagram of quad-core processor architecture[4]

The main advantage of multi-core processors that signals between different CPUs travel shorter distances and therefore those signals degrade less. These higher quality signals allow more data to be sent in a given time period since individual signals can be shorter and do not need to be repeated as often. The performance improved largest while running CPU intensive processes.

In physically, the multi-core CPU designs require much less Printed Circuit Board (PCB) space than multi-chip symmetric multiprocessing (SMP) designs. Also, a dual-core processor uses slightly less power than two coupled single-core processors, principally because of the decreased power required to drive signals external to the chip. Furthermore, the cores share some circuitry, like the L2 cache and the interface to the front side bus (FSB). In terms of technologies, multi-core design can make use of proven CPU core library designs and produce a product with lower risk of design error than devising a new wider core design.

Multi-core processors are an improvement over the deficiencies of single-core processors, as processor speed increases, the amount of heat produced and the amount of power consumed by the processor increase with it. Multi-core processors can perform more works within a given amount of time by dividing the work between two or more cores while reducing the power consumption and dissipate the heat [22]. Due to the computing power and cost-to-performance effectiveness, the multi-core processors are widely used in modern computers. Applications of multi-core processors are to speed up the work of operating systems and to support multithreading.

One basic difference between single processor and multi-core processors is that a single processor has a unique L1 cache along with a L2 cache whereas each independent processor in a multi-core system

[4] Understanding Multiprocessing and Multiple Cores

has a common shared L2 cache in addition to an individual L1 cache. Therefore, number of caches and memories required become less than if single core processor is used for the equal number of jobs that need to be performed.

The overall performance of a microprocessor is determined by interconnect characteristics. Several interconnection mechanisms have been proposed in [16, 57] to speed up the interconnect performance. Among them, a shared bus fabric (SBF) is most commonly used. A shared bus fabric is a high speed link which can communicate data between processors, caches, I/O and memory in a multi-processor system. The effectiveness of such an approach depends on the probability that an L2 miss is serviced on a local cache.

The interaction between cores in multi-core CPU's can be implemented by various mechanisms, affecting overall CPU performance due to shared workloads between cores. The most efficient way to allow on chip interprocess communication (IPC) is through the use of shared memory. This is a very important feature for efficient hardware utilization of multi-core systems. Shared memory IPC is more efficient because data does not need to be copied from one process memory space to another. Instead, a memory space that is shared between the communicating processes is created. By this way, IPC that uses shared memory avoids many costly memory operations.

Task scheduling for multi-core processors is a key factor on the performance of the multi-core processors. Most traditional task scheduling strategies focused on single-core processor and they cannot work well for multi-core processors system. With the widespread use of multi-core processors, designing an effective task scheduling strategy has been a hot issue.

The advent of multi-core processors has refocused attention on the cognitive challenges of parallel programming. As single core performance reaches a max threshold, programs do not automatically increase in performance when run on new hardware. Instead, programs must run in parallel if they are to make use of new hardware resources. The parallel required to exploit new hardware fully is growing exponentially with each generation.

Multi-core processors also gave developer an opportunity to parallel programming to execute the task in parallel. Parallel programming is used to execute a task by distributing it in smaller instructions and executing them on different cores. By using parallel programming, the complex tasks that are carried out in a multi-core environment can be executed with higher efficiency and performance.

Programming for multi-core processors poses new challenges. The main objective of multi-core processors architecture is the extraction of higher performance from multi-cores which depend upon an efficient parallel programming mechanism and its implementation.

2.6. Parallel programming

Parallel programming is difficult because it adds synchronization as a new problem area to be dealt with. Synchronization defects such as race conditions, deadlocks and livelocks are known to be difficult to detect [4].

Although previous work in algorithms [39], database systems [72] and high performance computing [38, 68] including dealt with these problems, the novel challenge now is to develop general purpose

engineering approaches for assisting ordinary programmers with the creation of potentially large, parallel applications. These approaches must not only target algorithmic levels, but also higher abstraction levels such as design patterns [14]. Strategic engineering aspects on how to approach parallelization on different abstraction levels are not well-developed. Other approaches with different underlying paradigms, such as transactional memory [89] or stream programs [98] are active areas of research.

Multi-core programming is a form of parallel programming in which the code takes advantage of the multiple execution cores to run many instructions in parallel at the same time. Multi-core and multi-processor computers offer more than one processing core in a single machine. Hence, the goal is to do more in less time by distributing the work to be done in the available cores.

The multi-threading model was not designed to help developers tackle the multi-core revolution. In fact, creating a new thread requires a lot of processor instructions and can introduce a lot of overhead for each algorithm that has to be split into parallelized threads. Because this multi-threading model is too complex to handle the multi-core revolution, it is known as heavyweight concurrency.

Multi-threading requires adding too many lines of code to handle potential problems because of its lack of support of multi-threaded access at the framework level and it makes the code complex to understand. The aforementioned problems associated with the multi-threading model and the increasing number of logical cores offered in modern microprocessors motivated the creation of new models to allow creating parallelized sections of code.

The new model is known as lightweight concurrency because it reduces the overall overhead needed to create and execute code in different logical cores. The lightweight concurrency model takes into account the new micro-architectures in which many logical cores are supported by some physical cores. This model is not just about scheduling work in different logical cores. It also adds support for multi-threaded access at the framework level and it makes the code much simpler to understand.

Two main approaches were used for parallelization are task parallel and data parallel.

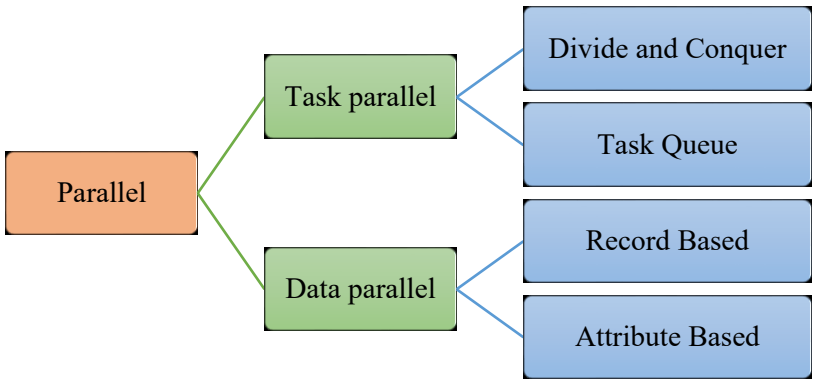


Figure 2.5. Parallel approaches

With task parallel, each processor has or needs access to the entire database and multiple operations are concurrently executed. Task parallel algorithms assign portions of the search space to separate

processors. The task parallel approaches can again be divided in two groups. The first group is based on a divide-and-conquer strategy that divides the search space and assigns each partition to a specific processor. The second group is based on a task queue that dynamically assigns small portions of the search space to a processor whenever it becomes available.

A task parallel implementation of search tree induction will form tasks associated with branches of the tree. A divide and conquer approach seems a natural reflection of the recursive nature of search trees. Task parallel implementation of search trees that seems to work well due to the independent nature of the subtasks are presented in [123]. However, the task parallel implementations suffer from load balancing problems caused by uneven distributions of records between branches. The success of a task parallel implementation of search trees seems to be highly dependent on the structure of the database.

The second approaches, called data parallel, distribute the data set over the available processors, the database is partitioned among the processors and the same operation is executed on multiple partitions at the same time. Data parallel approaches come in two flavors. A partitioning based on records will assign non-overlapping sets of records to each of the processors. Alternatively, a partitioning of attributes will assign sets of attributes to each of the processors.

Attribute based approaches are based on the observation that many algorithms can be expressed in terms of primitives that consider every attribute in turn. If attributes are distributed over multiple processors, these primitives may be executed in parallel. These approaches work well if a good load balancing can be achieved by a careful distribution of attributes. Unfortunately, the number of attributes usually is in the same order of magnitude as the number of processors, which makes an uneven distribution of attributes more likely. Also, the set of attributes that is considered during the course of the algorithm may differ. Many data mining primitives consider multiple attributes at once, which may require duplication of data in order to minimize communication between processors.

Record based partitioning is based on the notion that most primitives common in data mining that consider all records can be executed in parallel if the records are distributed over the processors. Record-based approaches are dependent on the fact that every record has an equal probability of being considered during the execution of a primitive. For a random distribution of records this is the case. The number of records is usually several orders of magnitude larger than the number of processors. This will facilitate an equal distribution of the data over the available processors.

From a data mining viewpoint, data parallel has several advantages over task parallel. In 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 parallel has a higher degree of machine architecture independence, in comparison with task parallel. But problem of machine architecture dependence is not completely eliminated in data parallel. 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 parallel. Despite the above advantages of data parallel, it should be emphasized that the exploitation of task parallel is also useful in data mining. To summarize, data parallel addresses the problem of very large databases, whereas task parallel addresses the problem of very large search spaces. Note that if a large enough number of processors are available, both types of

parallel can be used at the same time, which can greatly speed up the execution of data mining algorithms [1].

2.7. Task parallel programming

Traditional task parallel programming models try to abstract over threads and offer a higher-level abstraction for expressing parallel. Cilk [15] is a task based, shared memory, programming model that allows the programmer to specify recursively spawned tasks, which are efficiently scheduled on threads using continuations.

With .NET framework 4.0, task is part of the Task Parallel Library (TPL), which is a new collection of very, very useful classes aimed at not only making parallel programming easier to read, but also offers lighter weight objects when compared to the classic threading alternatives. The TPL dynamically scales the degree of parallel to most efficiently use all the processors that are available. In addition, the TPL assists in the partitioning of work and the scheduling of tasks in the .NET thread pool. The library provides cancellation support, state management and other services.

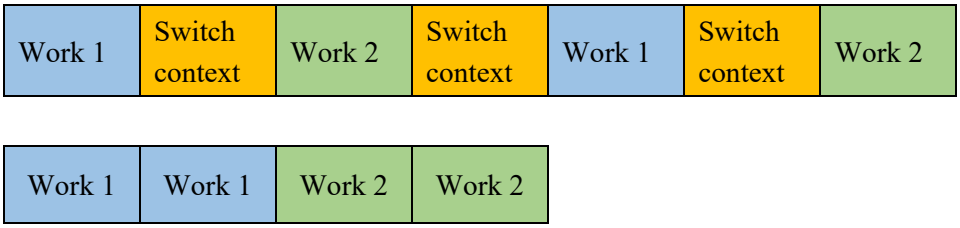


Figure 2.6. Switching between pieces of work

One of the first advantages of using task is that it becomes easier to guarantee that you are going to maximize the performance of applications on any given system. For example, if we are going to fire off multiple threads that are all going to be doing heavy CPU bound work. On a single core machine we are likely to cause the work to take significantly longer. It is clear, threading has overhead and if you are trying to execute more CPU bound threads on a machine than you have available cores for them to run, then you can possibly run into problems. Each time the CPU has to switch from thread to thread, there is a bit of overhead and if you have many threads running at once, then this switching can happen quite often, causing the work to take longer than if it had just been executed synchronously.

If we are not switching between pieces of work, then we do not have the context switches between threads as shown in Figure 2.6. So, the total cumulative time to process in that manner is much longer, even though the same amount of work was done. If these are being processed by two different cores, then we could simply execute them on two cores and the two sets of work would get executed simultaneously, providing the highest possible efficiency.

Tasks allow programmers to express the maximum parallel in an algorithm without hard-wiring such parallel into threads and making assumptions about the parallel available in hardware. The compiler and runtime system can then extract parallel as needed by mapping tasks to available cores, automatically detect dependencies, perform synchronization and even produce deterministic parallel executions [103]. Task parallel programming models offer a much better abstraction for checkpointing

an application, as tasks are atomic units of work that do not communicate or synchronize with other tasks and have a fixed set of inputs and outputs.

2.8. Parallel sequential pattern mining

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 databases. 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 parallel 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. The main challenges of parallel and distributed algorithms face today include communication overhead minimization, synchronization, workload balancing, data decomposition, efficient memory usage and disk I/O minimization.

Many FSP and FCSP mining algorithms are implemented based on a sequential strategy and single-task processing. This means that a task must be completed before the next one can be started. Hence, these methods are time-consuming for large databases. To improve performance, some researchers have applied parallel computing approaches to speed up the processing such as pSPADE algorithm [122] is a parallel sequential pattern mining algorithm was proposed for fast discovering the set of all frequent subsequences on computers with shared memory, which is based on SPADE algorithm [121]. pSPADE is the first algorithm for shared memory systems, pSPADE treats the pattern space as a tree of independent suffix-based classes. Each class can be solved independently on each processor requiring no synchronization.

Many algorithms have been proposed for mining frequent closed sequential patterns to reduce the required storage space and runtime. Popular algorithms for closed sequential pattern mining are CloSpan [117], ClaSP [36], BIDE [109, 111] and CloFS-DBV [101].

Par-CSP [25] and Par-ClosP [128] algorithms were proposed for a distributed memory system, it partitions the work among the processors based on the divide-and-conquer property so that the overhead of inter-processor communication is minimized. The Par-CSP algorithm is a method for mining closed sequential patterns on computers with distributed memory, it is an extended from BIDE [109]. To efficiently parallelize BIDE, the Par-CSP uses the divide-and-conquer strategy to minimize inter-processor communications. A method called dynamic scheduling is used to reduce processor idle time and the authors devise a technique called selective sampling to estimate the relative mining time of the subtasks and achieve load balancing. BIDE-MR algorithm [120] is a parallel implementation of BIDE algorithm on MapReduce. It iteratively assigns the tasks of closure checking and pruning to different nodes in cluster.

In addition, multi-core processors [6, 39, 104] allow for multiple tasks to be executed in parallel to enhance performance. Based on the multi-core processors architecture, some data mining researches

developed parallel data mining algorithms to speed up the execution process, thereby reducing computational cost and improving the efficiency of systems. Such as a cache-conscious FP-array and a mechanism for parallel data lock-free dataset tiling [65] for mining frequent itemsets on multi-core computers. Multi-Core Apriori Transaction Identifiers (MATI) algorithm [119] proposed a strategy for effective load balancing to reduce the number of duplicate candidates generated. In addition, parallel mining has been applied to closed itemset mining [87, 119], correlated pattern mining [21], class association rule mining [76], frequent subgraph mining [48], generic pattern mining [74] and frequent sequential pattern mining [47, 49].

Other methods based on multi-core architectures includes PGP-mc, which uses parallel gradual pattern extraction [59], GapMis-OMP, which is a tool for pairwise short-read alignment [33], SW (Smith-Waterman), which compares sequence lengths [86], PIB-PRISM and pDBV-SPM, which mining frequent sequential pattern on multi-core processors [47, 49].

2.9. Bit vectors

A bit vector is an array of bits (also known as a bitmap) that compactly stores bits. It can be used to implement a simple set data structure. Since there are only two possible values, they can be stored in one bit. A bit vector is built to represent the positions of an item X appearing in a sequence α. If the i-th itemset in α contains item X, then the i-th bit in the bit vector is 1, otherwise, it is 0.

For example, Table 2.5 shows a 24-byte bit vector. Table 2.6 is a mapping of *SDB* in Table 2.2 to a bit vector. Item A appears in all sequences so its bit vector is 1111, item E appears only in sequense S_3 so its bit vector is 0010 and so on. The length of a bit vector is the number of sequences in the database.

Bit vectors have been widely used for mining frequent itemsets, with good results obtained. However, the sizes of bit vectors for itemsets are always the same, i.e., equal to the number of transactions in the database. When the number of transactions is large, it is difficult to store them all in main memory. In addition, a lot of memory and time are needed for computing the intersection among bit vectors. There are usually many ‘0’ bits in a bit vector. The bit vector of an itemset with many ‘0’ bits can be shortened to reduce storage space and computation time. DBVs were thus proposed [107] to reduce memory usage and computation time. A DBV represents a bit vector in bytes after removing the ‘0’ bits at the front and end of the vector. Itemsets with many ‘0’ bits are thus more efficiently stored in a DBV, different itemsets can have different vector lengths.

2.10. Dynamic bit vectors

Some sequential pattern mining (SPM) algorithms based on a vertical data format have proven to be more efficient than those based on a horizontal data format. These algorithms scan the database only once and quickly calculate the support of patterns. However, they require a lot of memory to store additional information. The main drawback of the bit vector structure is its fixed size, which depends on the number of sequences in a sequence database. During sequence extension, ‘0’ bits thus appear often, which increases the required memory and processing time. The DBV data structure is used to overcome this problem.

2.10.1.Dynamic bit vector data structure

The DBV structure is used to store transactions in a vertical format. The supports can be quickly calculated by counting the number of '1' bits.

A DBV consists of two parts:

- *Start bit*: the position of the first appearance of a '1' bit.
- *Bit vector*: a sequence of bits starting from the first non-zero byte to the last non-zero byte.

Suppose that there are 24 sequences in a sequence database. An item *i* exists in sequences 5, 6, 9, 11 and 13. The bit vector for item *i* needs 24 bytes, as shown in Table 2.5. Using the DBV structure, only 11 bytes are required (9 bytes for the bit vector and 2 bytes for position). Let *X* and *Y* be two bit vectors, v_1 and v_2 are the probabilities of '1' bits in bit vectors *X* and *Y*, respectively. Let *h* be the probability of '0' bits after joining *X* and *Y* to get *XY* via sequence extension. Therefore, the probability of '1' bits in bit vector *XY* is $\min(v_1, v_2) - h$, where $\min(v_1, v_2)$ is the minimum value of v_1 and v_2. The gap between v_1 and v_2 increases quickly after several sequence extensions.

Table 2.5. Example of 24-byte bit vector

0	0	0	0	1	1	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0

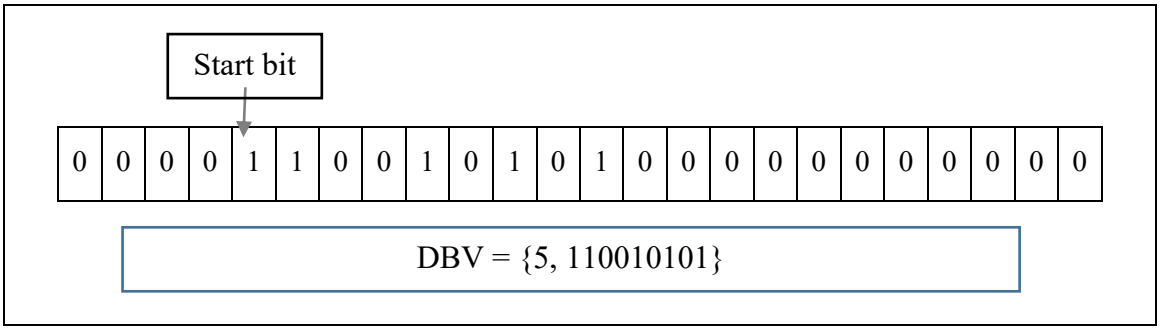
In Table 2.5, the first non-zero byte appears at position 5. The conversion of the bit vector to a DBV is shown in Table 2.7.

Table 2.6. Conversion of SDB in Table 2.2 to DBV format

Item	*sid*	*Bit encoding*		*Start bit*	*Bit vector*	*Value*
A	1, 2, 3, 4	1111	*Conversion to DBV*	1	1111	15
B	2, 3, 4	0111		2	111	7
C	1, 2, 3, 4	1111		1	1111	15
D	4	0001		4	1	1
E	3	0010		3	1	1

Consider *SDB* in Table 2.2. Item *A* exists in sequences S_1, S_2, S_3 and S_4, the start bit is 1, the bit vector is 1111 and thus $\sigma(A)$ is 4 because the bit vector has four '1' bits. Item *B* exists in sequences S_2, S_3 and S_4, the start bit is 2, the bit vector is 0111 and thus $\sigma(B)$ is 3 because the bit vector has three '1' bits. Similarly, $\sigma(C)$ is 4, $\sigma(D)$ and $\sigma(E)$ is 1. Table 2.6 shows the conversion of *SDB* in Table 2.2 to the DBV format.

Table 2.7. Conversion of bit vector to DBV



We use a byte to represent a block of 8 bits to speed up the counting. For example, a block of 8 bits, 00001111, is represented by number 15 in byte. In Table 2.6, value of item *A* is 15 (1111) and that of item *B* is 7 (0111). The intersection of 15 & 7 is equal to 1111 & 0111. Using DBV helps to reduce memory storage and computational cost in bitwise AND operator.

2.10.2.DBV-Pattern data structure

The DBV-Pattern data structure combines the DBV structure with a representation of a sequence. Each DBV-Pattern consists of three parts, namely *S*, *BitArray* and *LP*, where:

- *S* is a sequence.
- *BitArray* is a DBV.
- *LP* is list of positions of the occurrence in the sequence of each transaction. List positions are represented in the form *startPos*:{*list position*}, where *startPos* is the first appearance of the sequence in each transaction.

In *SDB*, item *A* exists in sequences S_1, S_2, S_3 and S_4. In sequence S_1, item *A* appears at positions {2, 3, 4}. The starting position is 2, so *LP* is 2:{2, 3, 4}. In sequence S_2, item *A* appears at positions 1 and 3. The starting position is 1, so *LP* is 1:{1, 3}. Similarly, *LP* of item *A* is 1:{1, 3} in sequence S_3 and 1:{1, 4} in sequence S_4. Table 2.8 presents the DBV-Pattern for item *A* in Table 2.2.

Table 2.8. DBV-Pattern for item A

Sequence	⟨***A***⟩			
Start bit	1			
Value	15			
Index	4	3	2	1
List Positions	1:{1, 4}	1:{1, 3}	1:{1, 3}	2:{2, 3, 4}

2.10.3.DBV-Tree

The DBV-Tree is an extension of the prefix-tree with the top level labeled NULL. Each node in a DBV-Tree is a DBV-Pattern. Each node *X* at level *k* in the tree can be extended via sequence extension or itemset extension by adding one item to get a child node *X* at level *k*+1. A new node *XY* is created by joining nodes *X* and *Y*, which must have the same length and the same |*X*| - 1 prefix item. Typical

algorithms for building a prefix-tree are IMSR_PreTree [105] and MNSR_PreTree [80]. Figure 2.7 shows candidates for *SDB* in Table 2.2.

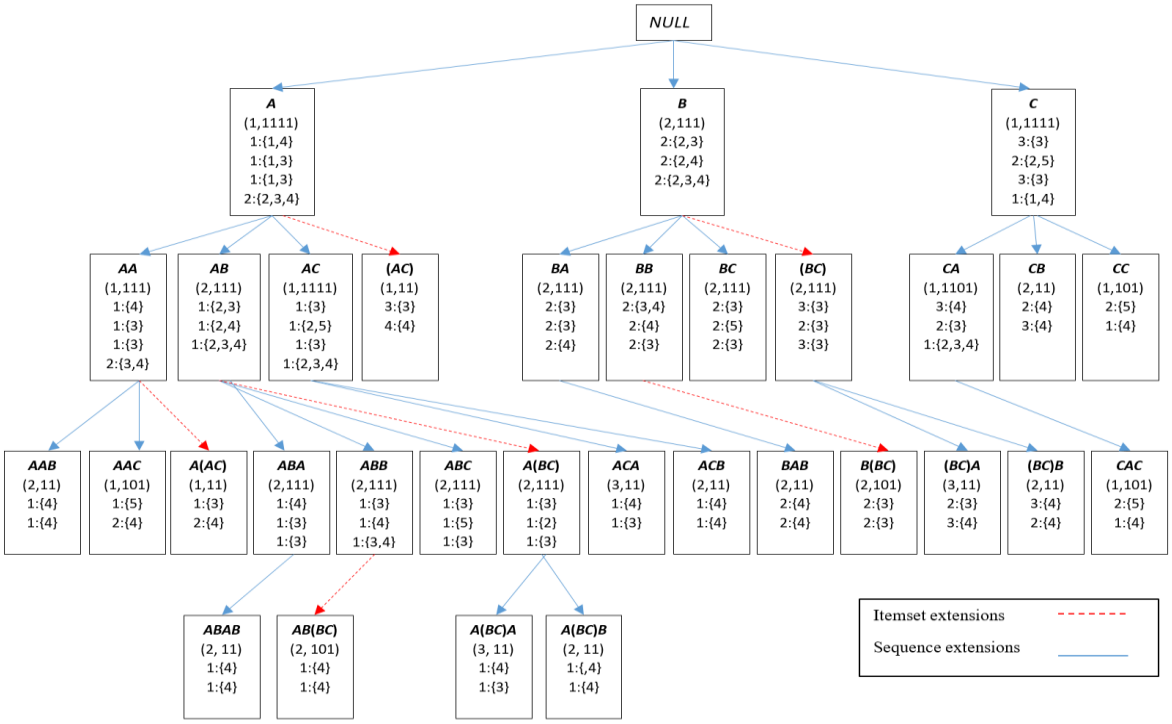


Figure 2.7. DBV-Pattern tree for SDB in Table 2.2 with minsup = 50%.

2.11. Summary

In this chapter, we presented the basic concepts of sequential pattern mining and overviewed some traditional frequent pattern mining methods, which contains frequent sequential pattern mining and frequent closed sequential pattern mining. Then we introduced the parallel computing, multi-core processors architecture, parallel programming and reviewed some parallel frequent sequential patterns methods. Specially, a novel compact data structure was presented to store database for sequential pattern mining.

Chapter 3

Parallel methods for mining sequential patterns

In this chapter, we propose parallel methods for sequential pattern mining (SPM) on large sequence database using multi-core processors architecture. This section also describes models for compact sequence database and scalable parallel algorithms based on data parallel. These algorithms follow 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 multi-core processors.

3.1. Introduction

Sequential pattern mining is to find a set of patterns that occur frequently together in some sequences. The general idea of all existing methods is to start with general sequences and then extend them towards specific ones.

Many sequential pattern mining algorithms are implemented based on a sequential strategy and single-task processing. Hence, these methods are time-consuming for large databases, especially large and dense databases. To improve performance, some researchers have applied parallel computing approaches to speed up the processing. Such as pSPADE algorithm [122], which is based on SPADE algorithm [121], pSPADE is the first algorithm for shared memory systems, pSPADE treats the pattern space as a tree of independent suffix-based classes. Each class can be solved independently on each processor requiring no synchronization. Par-CSP algorithm [25] was proposed for a distributed memory system, it partitions the work among the processors based on the divide-and-conquer property so that the overhead of inter-processor communication is minimized. In addition, multi-core processors [6, 94, 104] allow for multiple tasks to be executed in parallel to enhance performance.

Some data mining researchers have developed parallel algorithms for multi-core processors architectures such as cache-conscious FP-array and a mechanism for parallel data lock-free database tiling [65] and MATI algorithm [119] was proposed a strategy for effective load balancing to reduce the number of duplicate candidates generated. Parallel mining has been applied for mining closed frequent itemsets [119], correlated pattern mining [21], generic pattern mining [74], class association rule mining [76] and frequent graph mining [48]. Existing parallel algorithms only considered to distribute memory systems based on data parallel. This caused a high load imbalance and a higher synchronization happening at the end of each step. In this thesis, we utilize multi-core systems to solve these problems. In addition, parallel mining process invokes the Parallel Patterns Library (PPL) in .NET framework 4.0 which allows a simultaneous access to the whole available cores on modern computers. In this chapter, some efficient approaches for sequential pattern mining in parallel are presented.

3.2. Parallel Independent Branch PRISM (PIB-PRISM)

3.2.1. Prime block encoding

The PRISM algorithm was proposed as an effective approach for frequent sequence mining via prime block encoding [37]. It is briefly introduced as follows.

An integer p is a prime integer if $p > 1$ and the only positive divisors of p are 1 and p. Let $p_1, p_2, \ldots, p_r$ be the distinct prime factors of n, arranged in order, so that $p_1 < p_2 < \ldots < p_r$. All repeated factors can be collected together and expressed using exponents, so that $n = p_1^{m_1} p_2^{m_2} \ldots p_r^{m_r}$, where each m_i is a positive integer, called the multiplicity of p_i and this factorization of n is called the standard form of n.

Given two integers $x = \prod_{i=1}^{r_x} p_{ix}^{m_{ix}}$ and $y = \prod_{i=1}^{r_y} p_{iy}^{m_{iy}}$ in their standard forms, the greatest common divisor of the two numbers is given as $gcd(x, y) = \prod_i p_i^{m_i}$, where $p_i = p_{jx} = p_{ky}$ is a factor common to both x and y and $m_i = \min(m_{jx}, m_{ky})$, with $1 \le j \le r_x$, $1 \le k \le r_y$.

Given a set G of items is arranged, let $P(G)$ is a set of all subset of G and indexed by the set $\{1, 2, \ldots, |G|\}$. Considering $S \in P(G)$, any subset S can be represented as a bit vector contains n bit (denoted by S^B). $S^B = 1$ if the i-th element of G is in S, vice versa $S^B = 0$. Given a set S has n items, $S \in P(G)$, we define *multiplication operator* of S denoted is $\otimes S = s_1 \times s_2 \times \ldots \times s_n$, with $s_i \in S$. If $S = \varnothing$, define $\otimes S = 1$. Meanwhile, $\otimes P(G) = \{\otimes S: S \in P(G)\}$ is a set obtained when applying multiplication operator $\otimes$ on all sets S in $P(G)$. In this case, G also is a generator of $\otimes P(G)$ under the multiplication operator. A set G is a square-free generator if each. In case a generator G consists of only prime intergers, it was called a prime generator.

Theorem 3.1 [37]. A set P is a square-free semi-group with operator $\otimes$ iff it has a square-free prime generator G. In other words, P is a square-free semi-group iff $P = \otimes P(G)$.

Theorem 3.2 [37]. Let $\otimes P(G)$ be a square-free semi-group with prime generator G and let $X, Y \in \otimes P(G)$ be two distinct elements, then $gcd(X, Y) = \otimes(S_X \cap S_Y)$ and $lcm(X, Y) = \otimes(S_X \cup S_Y)$, where $X = \otimes S_X$ and $Y = \otimes S_Y$ and $S_X, S_Y \in P(G)$ are the prime factors of X and Y, respectively.

Given a set G of prime numbers sorted in increasing order, assume that N is a multiple of $|G|$ and B be a bit vector of length N. Then B can be partitioned into $m = \frac{N}{|G|}$ continuous blocks. Each block B_i includes the segment from $B[(i - 1) \times |G| + 1]$ to $B[i \times |G|]$, $1 \le i \le m$. Let $B_i[j]$ be the j-th bit in block B_i and $G[j]$ be the j-th prime number in G. The value of B_i with respect to G, denoted $v(B_i, G)$, is defined as $\otimes\{G[j]^{Bi[j]}\}$. The prime block encoding of a bit vector B with respect to a base prime set G is denoted $v(B, G)$, which is the set of all $v(B_i, G)$, $1 \le i \le m$. We can write $v(B_i, G)$ as $v(B_i)$ and $v(B, G)$ as $v(B)$ for simplicity. Note that a bit vector with all zeros is encoded as 1.

Given a bit vector $A = A_1 \times A_2 \times \ldots \times A_m$, where A_i is an array that contains $|G|$ bits, let f_A be the position of the first "1" bit in A. Then, the j-th bit in a mask operator $(A)^{\triangleright}$ is defined as:

$$(A)^{\triangleright}[j] = \begin{cases} 0, & \text{for } j \le f_A \\ 1, & \text{for } j > f_A \end{cases}$$

Each item can appear in different sequences and in different positions of a sequence. Therefore, two encoding mechanisms are executed for each item, including ID encoding of sequence data and position encoding of items in each sequence.

***Step 1.* Position block encoding**

A bit vector, named BIT, is built to represent the positions of an item I_x appearing in a sequence S. If the i-th itemset in S contains item I_x, then BIT[i] = 1, otherwise, BIT[i] = 0. Extra "0" bits are appended to the bit vector such that the vector size is a multiple of $|G|$. For example, Table 3.1 shows an example of 5 sequences with set of items $I = \{A, B, C\}$.

Table 3.1. Example of a sequence database

sid	sequences
S_1	$\langle(AB)BB(AB)B(AC)\rangle$
S_2	$\langle(AB)(BC)(BC)\rangle$
S_3	$\langle B(AB)\rangle$
S_4	$\langle BB(BC)\rangle$
S_5	$\langle(AB)(AB)(AB)A(BC)\rangle$

In the first sequence, item A appears in positions 1, 4 and 6, so the bit encoding of A is 100101. Assume $G = \{2, 3, 5, 7\}$. Because $|G| = 4 = 2^2$, two extra "0" bits are added and the bit encoding becomes 10010100. $v(1001)$ and $v(0100)$ are then calculated as 14 and 3, respectively. Figure 3.1 shows all the encoded positions of item A in the given five sequences. The same procedure is repeated for the other two items, B and C.

sid	Bit encoded pos	Prime encoded pos
S_1	1001 0100	{14, 3}
S_2	1000	{2}
S_3	0100	{3}
S_4	0000	{1}
S_5	1111 000	{210, 1}

Figure 3.1. Bit encoded position block of item A

Step 2. Sequence block encoding

Another bit vector can be used to represent the sequences in which an item appears. In the example above, since item A appears in all sequences except for the 4th sequence, the bit vector, 11101, is then used to represent the case. The bit vector is then encoded into a prime block based on the given prime set G. Again, extra "0" bits are appended to the bit vector such that the vector size is a multiple of $|G|$. Thus, three bits of "0" are added and the resulting bit vector of A is 11101000. The prime coding $v(A)$ generated from the bit vector for item A is $v(1110)$ $v(1000)$, which is {30, 2}. The results for all the three items are shown in Figure 3.2 and Figure 3.3 shows the final prime encoding, including sequence and position encoding.

Item	Bit-encoded sid	Prime-Encoded sid	support
⟨*A*⟩	1110 1000	{30, 2}	4
⟨*B*⟩	1111 1000	{210, 2}	5
⟨*C*⟩	1101 1000	{42, 2}	4

Figure 3.2. Bit-encoded sid for items

Item	Sequence block	Position block
⟨*A*⟩	{30, 2}	{14, 3}, {2}, {3}, {1}, {210, 1}
⟨*B*⟩	{210, 2}	{210, 2}, {30}, {6}, {30}, {30, 2}
⟨*C*⟩	{42, 2}	{1, 3}, {15}, {1}, {5}, {1, 2}

Figure 3.3. Full primal block

Compression of prime block encoding

A sequential pattern (SP) appears in only some sequences of a database and in each sequence only occurs in some positions of a sequence. Therefore, a bit vector usually includes many bits of zero. A block with all bits being zero is called an empty block. To reduce size, PRISM retains only non-empty blocks after prime encoding. Hence, it keeps an index with each sequence block to indicate which non-empty position blocks correspond to a given sequence block. Figure 3.4 shows the compact prime block encoding for item *A*. The first sequence block is 30, with a factor-cardinality of $||30||_G = 3$, meaning that there are 3 valid (non-empty position blocks) sequences in this block. For each of these, the offsets are stored in the position blocks.

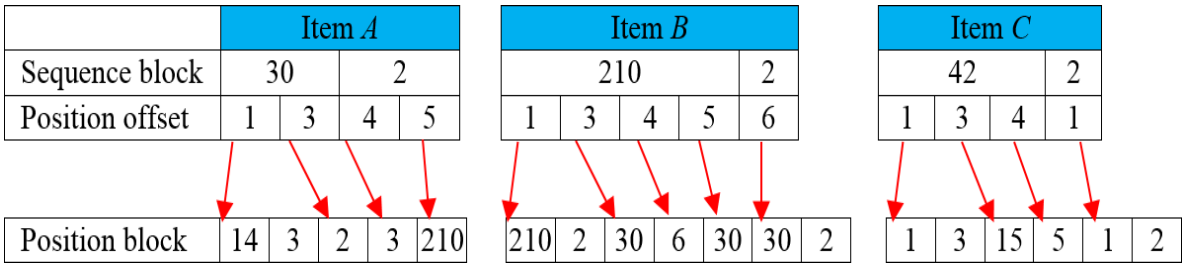


Figure 3.4. Compact encoded blocks

Support counting via prime block join

Consider a sequence α, with $P(S_\alpha, P_\alpha)$ denoting its prime encoding, where S_α is the set of all encoded sequence blocks α and P_α is the set of all encoded position blocks for α. The support of sequence α can be directly determined from the prime block encoding given by $\sigma(\alpha) = \sum_{v_i \in S_\alpha} \| v_i \|_G$. For example, $\alpha = \langle A \rangle$ and $v(\langle A \rangle) = \{30, 2\}$. We have $\sigma(\alpha) = ||30||_G + ||2||_G = 3 + 1 = 4$.

3.2.2. Parallel Independent Branch PRISM

To improve the efficiency and reduce the processing time of SPM method, this approach presents a parallel model based on PRISM theory and multi-core processors architecture.

We propose Parallel Independent Branch PRISM (PIB-PRISM) algorithm. PIB-PRISM is implemented based on the parallel model in .NET Framework 4.0, using tasks instead of threads. The algorithm using tasks has advantages over using threads [20, 45, 60]. First, tasks require less memory than threads do. Second, a thread runs on only one core, whereas a task can run on multiple cores. Finally, threads require more processing time than tasks do because the operating system needs to allocate data structures for threads, as initialization and destruction and also perform the context switch among threads.

In .NET framework 4.0 was enhanced with higher-level constructs. A new concept is introduced – Task, it is part of TPL, which is a new collection of classes that focus on parallel programming as well as offering lightweight object compared to traditional threading system. Using task has many benefits - not only are tasks more efficient, but they also abstract away from the underlying hardware and the OS specific thread scheduler. One other great feature about TPL is that it aims to use each core of CPU so no core will be idle.

The primary advantage of PIB-PRISM strategy is that each task is assigned to searching branches of the search tree and processed independently from other tasks without any synchronization. Our implementation can solve the high computational cost problem. The parallel mining process for finding FSPs from a task is shown as follows.

- ***Step 1***. Construct the search tree from the set of frequent 1-itemset sequences.
- ***Step 2***. Assign each branch of the tree to a processor and mine FSPs independently.

In PIB-PRISM, a database must be preprocessed into the vertical data format. This step is performed only once. After preprocessing, the database is called a transformed database *D*' and has a structure as shown in Figure 3.5, the first line indicates the number of transactions in the database, the second line includes an item and its support count of a task and the next lines (called support lines) are in the form of *sid* and *positions*, which indicate the item appeared in the positions of the sequence *sid*.

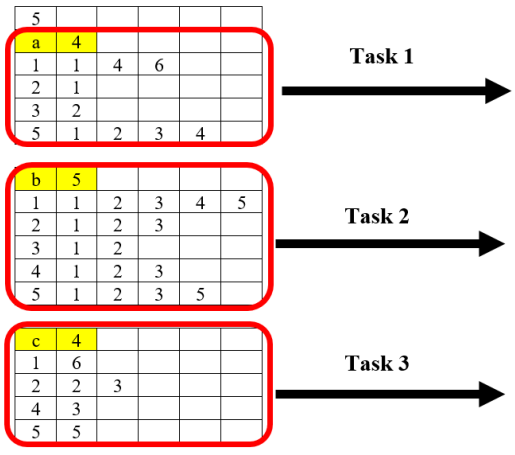


Figure 3.5. An example of a transformed database from Table 3.1

The pseudo code of the PIB-PRISM strategy is shown in Figure 3.6.

```
Procedure PIB-PRISM(D', minsup)
Input Sequence database D', minsup
Output: All FSPs satisfying minsup
1   Begin
2       Let dbpat = NULL
3       pGenarate-SPs(D', minsup)
4       For each item i in dpbat.Pats do
5           Create new task t_i
6           t_i call extendTree(dbpat.Pats[i], dbpat)
7       End for
8       For each t_i in the list of created tasks do
9           collect the set of patterns (Ps) returned by each task
10      totalPs = totalPs ∪ Ps
11  End
```

Figure 3.6. The PIB-PRISM strategy

In the PIB-PRISM strategy, the root of the tree was initialized NULL. The **pGenarate-SPs** procedure is called to identify all frequent 1-sequences from the database *D'* (line 2). Next, this procedure creates a new task corresponding to each pattern (line 5) and each task executes the procedure of **extendTree** (line 6) to extend itemsets and sequences. Tasks run parallel together and each task is then performed independently on a processor core to generate a partial set of FSPs. The final set of FSPs is then the union of the partial results. The procedure of **pGenerate-SPs** is stated in Figure 3.7.

```
Procedure pGenerate-SPs(D', minsup)
1   Begin
2   Create n tasks, with n = ⌈I/P⌉
    where I is the number of single items in D' and P is the number of processor cores
3   For each item i in I do
4       Assign item i for tasks[i]
5       Generate-SP(i)
6   End for
7   End
```

Figure 3.7. Procedure pGenerates-SPs

The **pGenerate-SPs** procedure creates *n* tasks (line 2) and assigns these tasks to cores for executing the **Generate-SP** procedure (line 5) based on PRISM to find frequent 1-sequences. The procedure **Generate-SP** checks support count for each item, the items that satisfy *minsup* threshold will be encoded based on PRISM.

3.2.3. Experimental results

Experiments were conducted to evaluate the proposed parallel algorithm. The experiments were performed on a personal computer with an Intel Core i7-4790 3.6-GHz CPU with 8 cores, 3 MB of L3 cache and 8 GB of RAM, running Windows 7, the algorithm was implemented using .Net Framework 4.0. The information about the databases are shown in Table 3.2.

Table 3.2. Databases used in experiments

Database	No. of sequences	No. of items
C6T5N1kD1k	1,000	1,000
C6T5N1kD10k	10,000	1,000

Experiments were then conducted to compare the execution times of the two algorithms for various *minsup* values (0.5-0.8%). The results are shown in Figure 3.8 and Figure 3.9 for the two databases, respectively.

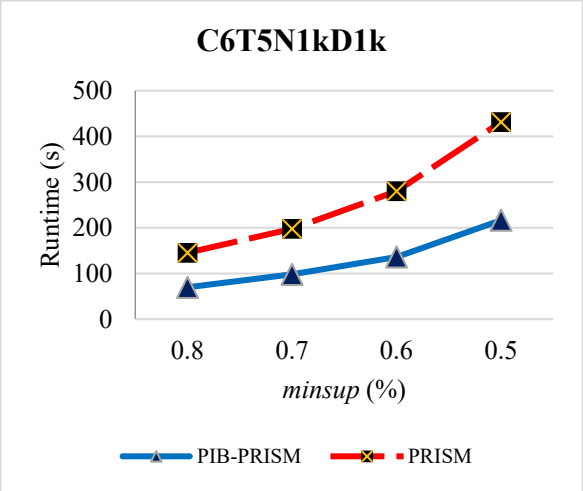


Figure 3.8. Runtime on C6T5N1kD1k

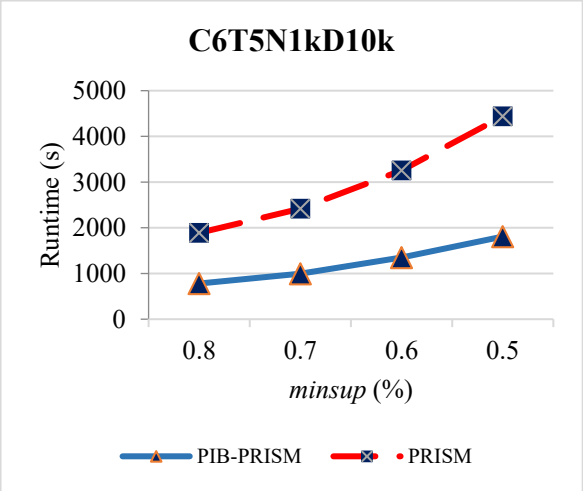


Figure 3.9. Runtime on C6T5N1kD10k

In this section, we compare the execution time of PIB-PRISM and PRISM for *C6T5N1kD1k* and *C6T5N1kD10k* database with various settings. When the threshold changes from 0.5% to 0.8%, we easily found that the runtime of the PIB-PRISM is faster than PRISM for all two databases and nearly twice. For example, consider *C6T5N1kD1k* database with *minsup* = 0.8%. The runtime of PRISM and PIB-PRISM were 145.454 and 69.803 seconds. When *minsup* decreased, the number of patterns increased so that execute times also increased. With *minsup* = 0.5%, the runtime of PRISM and PIB-PRISM were 431.34 and 216.58 seconds, respectively. In a similar way for *C6T5N1kD10k* database.

PIB-PRISM had a lower runtime than its counterpart because parallel processing divided the tasks to independent branches without synchronize. The experimental results show that the proposed algorithm significantly improved execution time, compared to the original algorithm for the two synthetic databases. When decreasing the *minsup* values, there were more FSPs obtained and thus the runtime increased.

3.3. Parallel Dynamic Bit Vector Sequential Pattern Mining (pDBV-SPM)

This section proposes a parallel approach called Parallel Dynamic Bit Vector Sequential Pattern Mining (pDBV-SPM) based on a multi-core processors architecture and the dynamic bit vector (DBV) data structure which was described in Chapter 2. pDBV-SPM overcomes the drawbacks of the PIB-PRISM and further reduces the computational cost of mining. The main contributions of pDBV-SPM are as follows:

1. Support values are fast determined and storage space is reduced by using the DBV data structure.
2. Quick mining is achieved based on a multi-core processors architecture.
3. Load balancing of the workload between processors is used.

3.3.1. Parallel Dynamic Bit Vector Sequential Pattern Mining

This section describes the proposed pDBV-SPM algorithm, which uses the DBV structure combined with location information in the structure of the transaction DBV-Pattern to mine FSPs based on a multi-core processors architecture.

Definition 3.1 (substring). Let α be a sequence. Then, $sub_{i,j}(\alpha)$ $(i \le j)$ is defined as a substring of length $(j-i+1)$ from position i to position j of α. For example, $sub_{2,4}\langle(ABCBD)\rangle$ is $\langle BCB\rangle$ and $sub_{1,3}\langle(ABCBD)\rangle$ is $\langle ABC\rangle$.

Definition 3.2 (extending a sequence from 1-sequence). Let α and β be two frequent 1-sequences. $\{t_\alpha, p_\alpha\}$ and $\{t_\beta, p_\beta\}$ are the transaction and position of sequences α and β, respectively. There are two forms of sequence extension.

Itemset extension: $\langle(\alpha\beta)\rangle\{t_\beta, p_\beta\}$, if $(\alpha \le \beta) \wedge (t_\alpha = t_\beta) \wedge (p_\alpha = p_\beta)$ (1)

Sequence extension: $\langle\alpha\beta\rangle\{t_\beta, p_\beta\}$, if $(t_\alpha = t_\beta) \wedge (p_\alpha < p_\beta)$ (2)

For example, considering a *SDB* in Table 2.2, $\langle A\rangle$ and $\langle B\rangle$ are two frequent 1-sequences. Itemset extension yields $\langle(AB)\rangle$ because $\langle A\rangle \le \langle B\rangle$ and $(t_A = t_B)$ is 2 (sequence S_2) and $(p_A = p_B)$ is 3 (position 3 in sequence S_2). Sequence extension yields $\langle AB\rangle$ because $(t_A = t_B)$ is 2 and the position of sequence $\langle A\rangle$ is 1, which is smaller than all positions of sequence $\langle B\rangle$, positions of $\langle B\rangle$ are $\{2, 3, 4\}$.

Definition 3.3 (extending a sequence from k-sequence). Let α and β be two frequent k-sequences $(k \ge 1)$, $u = sub_{k,k}(\alpha)$ and $v = sub_{k,k}(\beta)$. $\{t_\alpha, p_\alpha\}$ and $\{t_\beta, p_\beta\}$ are the transaction and position of sequences α and β, respectively. There are two forms of sequence extension.

Itemset extension:

$$\alpha + \beta_i = sub_{1,k-1}(\alpha)(uv)\{t_\alpha, p_\alpha\}, \text{ if } (u \le v) \wedge (t_\alpha = t_\beta) \wedge (p_\alpha = p_\beta) \wedge (sub_{1,k-1}(\alpha) = sub_{1,k-1}(\beta)) \quad (3)$$

Sequence extension:

$$\alpha + \beta_s = \alpha v\{t_\alpha, p_\alpha\}, \text{ if } (t_\alpha = t_\beta) \wedge (p_\alpha < p_\beta) \wedge (sub_{1,k-1}(\alpha) = sub_{1,k-1}(\beta)) \quad (4)$$

For example, ⟨*AB*⟩ and ⟨*AC*⟩ are two frequent 2-sequences. Itemset extension yields ⟨*A*(*BC*)⟩ and sequence extension yields ⟨*ABA*⟩, ⟨*ABB*⟩, ⟨*ABC*⟩ because ⟨*AB*⟩ and ⟨*AC*⟩ have the same prefix ⟨*A*⟩.

Definition 3.4. Let $\alpha = \langle a_1\ a_2 ...\ a_u \rangle$. An item a_k can be added to a pattern extension of α in one of three positions.

$$\beta = \langle a_1 a_2 ...\ a_u a_k \rangle \wedge (\sigma(\beta) = \sigma(\alpha)) \tag{5}$$

$$\exists i\ (1 \le i < n) \text{ such that } \beta = \langle a_1 a_2 ...\ a_i a_k ... a_u \rangle \wedge (\sigma(\beta) = \sigma(\alpha)) \tag{6}$$

$$\beta = \langle a_k a_1 a_2 ... a_u \rangle \wedge (\sigma(\beta) = \sigma(\alpha)) \tag{7}$$

In (5), item a_k appears after a_u, so item a_k is called a forward extension and β is called a forward extension sequence. In (6) and (7), item a_k appears before a_u, so item a_k is called a backward extension and β is called a backward extension sequence. For example, sequence ⟨*AC*⟩:4 is a forward extension of sequence ⟨*A*⟩:4 because sequence ⟨*C*⟩ is extended after sequence ⟨*A*⟩. Sequence ⟨*CAC*⟩:2 is a backward extension of sequence ⟨*CC*⟩:2 because sequence ⟨*A*⟩ is extended in the middle of sequence ⟨*CC*⟩.

Definition 3.5. Consider $\alpha = \langle a_1\ a_2 ...\ a_u \rangle$, the starting position of sequence α is the position of the first appearance of itemset a_1. For example, in the sequence ⟨*AB*(*BC*)*AD*⟩, the starting position of sequence ⟨(*BC*)⟩ is 3 and sequence ⟨*ABD*⟩ is 1.

Consider a *SDB*, the relations of subsequences are typically represented as a search tree, defined recursively as follows. The root of the tree is at level zero and is labeled as the null sequence. A node labeled sequence α at level k, called the k-sequence is repeatedly extended by adding one item I to generate a child node at the next level called (k+1)-sequence. A (k+1)-sequence can be extended via sequence extension or itemset extension. In sequence extension, an item is appended to the sequential pattern to create a new itemset. In itemset extension, an item is added to the last itemset in the pattern.

In the parallel mining of FSPs, each branch of the search tree can be regarded as a single task, which can be processed independently to generate FSPs. An example is given in Figure 3.10, there are three tasks on level 1 of the task tree. Tasks 1, 2 and 3 process branches *A*, *B* and *C*, respectively.

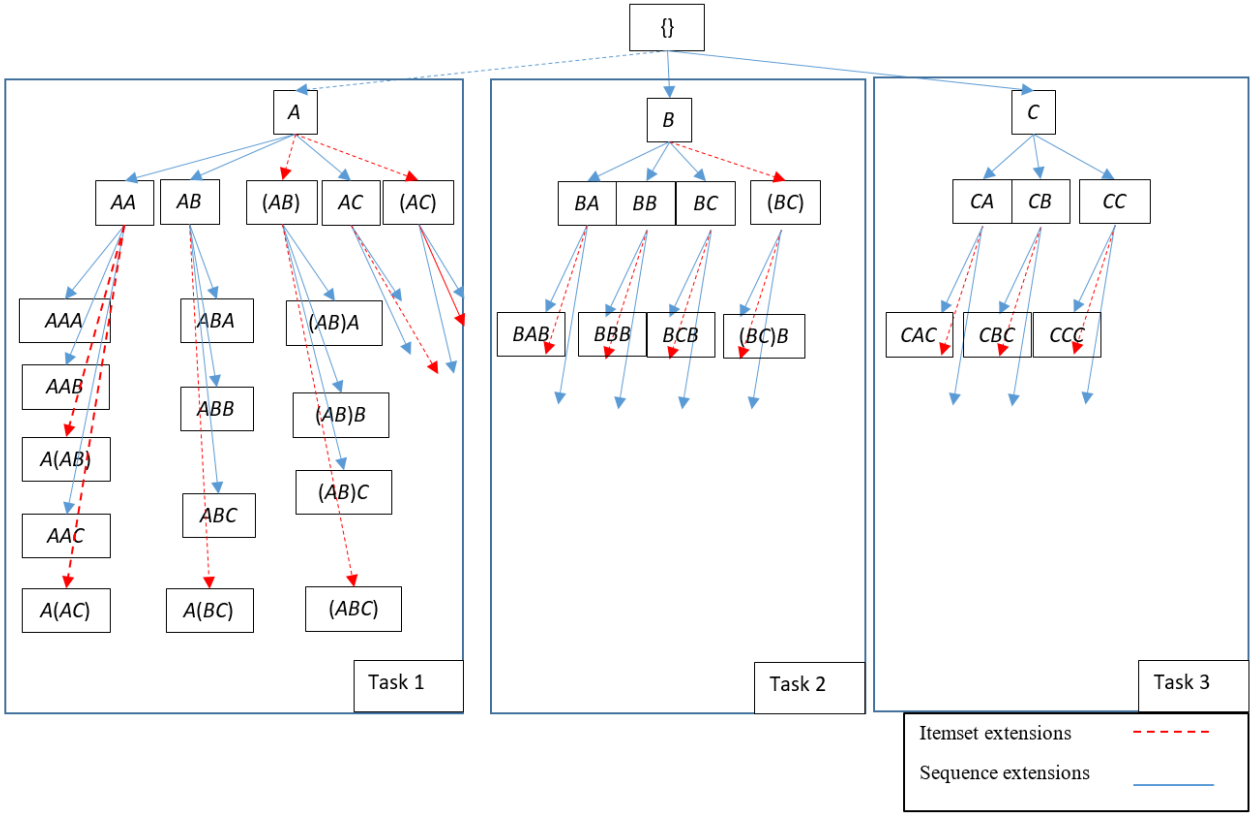


Figure 3.10. Example of task tree

The main steps of the pDBV-SPM algorithm are as follows.

1. Conversion of the sequence database to the DBV-Pattern structure.
2. Early pruning of prefix sequences.
3. Extension of sequences.

Since pDBV-SPM uses the DBV-Pattern structure, it can check the backward and forward extensions quickly. For each transaction, pDBV-SPM considers only the start position or the last position of the sequence. Therefore, if the database has N transactions, pDBV-SPM requires only N operations to check each candidate.

The mining of FSPs from the tree is divided into separate branches as follows.

1. Construct the search tree from the set of frequent 1-sequences.
2. Assign each branch of the tree to a task to parallel mine FSPs, the mining process is performed independently on each branch.
3. Synchronize results. This is done only at the end of the mining pattern on each branch.

The pseudo code of the pDBV-SPM strategy is shown in Figure 3.11. *root* is initialized to *NULL*. The **Generate-SPs** procedure is called to identify all frequent 1-sequences from *SDB* (line 3). These sequences are stored in L_1. The items in L_1 are sorted in ascending order by support (line 4) to reduce the steps in the extension phase of the itemset and to more effectively balance the workload in the system. Next, pDBV-SPM creates new tasks corresponding to each pattern in L_1 (line 7). Each CPU core p_i is assigned a set of

$M = I/p$ task, where p is the number of CPU cores and I is the number of frequent 1-sequences (equal to the number of tasks). Each task executes the procedure **DBV-SPM-Extension** (line 8) to extend itemsets and sequences. Tasks are run in parallel (i.e., independently) to generate a partial set of FSPs. The final set of FSPs is the union of the partial results. The procedure of **Generate-SPs** is shown in Figure 3.12.

```
Procedure pDBV-SPM(SDB, minsup)
Input SDB, minsup
Output: All FSPs that satisfy minsup
1       Begin
2           Let root = NULL
3           L1= Generate-SPs(SDB, minsup)
4           Sort (L1) in increasing order of support
5           Add L1 to child node of root
6           For (each node i in root) do
7               Create new task ti
8               call DBV-SPM-Extension(i, minsup)
9       End begin
```

Figure 3.11. pDBV-SPM strategy

First, the **Generate-SPs** procedure finds frequent 1-sequences (L_1) based on the DBV structure for each item that satisfies the *minsup* threshold (line 4). Consider *SDB* in Table 2.2 and *minsup* = 50%. After this procedure is executed, three frequent 1-sequences are stored and thus $L_1 = \{\langle A\rangle{:}4, \langle B\rangle{:}\ 3, \langle C\rangle{:}4\}$, as shown in Table 3.3.

```
Procedure Generate-SPs(SDB, minsup)
Input: SDB, minsup
Output: Set of frequent 1-sequences
1  Begin
2   For (each t in T ) do
3       if (σ(t) ≥ minsup) then
4           L1 = L1 ∪ t
5   End for
6  End
```

Figure 3.12. Procedure Generate-SPs.

The procedure **DBV-SPM-Extension** is shown in Figure 3.13. This procedure expands the search tree by executing procedures **extendItemset** (line 6) and **extendSequence** (line 9). For each node in the search tree, the pattern for that node is extended by calling **extendItemset** and **extendSequence** to create a new pattern. This process is repeated until no frequent sequences are generated (line 13). The results of executing sequence extension and itemset extension for *SDB* in Table 2.2 are shown in Table 3.4 and Table 3.5, respectively.

```
Procedure DBV-SPM-Extension(p, minsup)
1   Begin
2     Let list= child nodes of p
3     For each S_x in list do
4       If (S_x is not pruned) then
5         For each S_y in list do
6           If (σ(S_xy = extendItemset(S_x, S_y)) then
7             Add S_xy to child nodes of S_x
8           End if
9           If (σ(S_xy = extendSequence(S_x,S_y)) then
10            Add S_xy to child nodes of S_x
11          End if
12        End for
13        Call DBV-SPM-Extension(S_x, minsup)
14      End if
15    End for
16  End begin
```

Figure 3.13. Procedure DBV-SPM-Extension

Consider a pattern with prefix ⟨*A*⟩. Pattern with prefix ⟨*A*⟩ extends using **extendSequence** by concatenation with ⟨*A*⟩, ⟨*B*⟩ and ⟨*C*⟩ to make new patterns ⟨*AA*⟩, ⟨*AB*⟩ and ⟨*AC*⟩, respectively, as shown in Table 3.4 and extends using **extendItemset** by concatenation with ⟨*B*⟩ and ⟨*C*⟩ to make new patterns ⟨(*AB*)⟩ and ⟨(*AC*)⟩, respectively, as shown in Table 3.5. It is then checked whether the support count of these patterns satisfies the threshold based on the DBV-Pattern structure. This process is repeated for each task until all FSPs are obtained, as shown in Table 3.6.

Table 3.3. Sequences A, B and C in SDB after conversion to DBV

Sequence	⟨***A***⟩				⟨***B***⟩			⟨***C***⟩			
Start bit	1				2			1			
Value	15				7			15			
Index	4	3	2	1	4	3	2	4	3	2	1
Position	1:{1,4}	1:{1,3}	1:{1,3}	2:{2,3,4}	2:{2,3}	2:{2,4}	2:{2,3,4}	3:{3}	2:{2,5}	3:{3}	1:{1,4}

Table 3.4 Sequence extension (minsup = 50%)

Sequence	⟨***AA***⟩			
Start bit	1			
Value	15			
Index	4	3	2	1
Position *A*	1:{1,4}	1:{1,3}	1:{1,3}	2:{2,3,4}

Position *A*	1:{1,4}	1:{1,3}	1:{1,3}	2:{2,3,4}
Position *AA*	1:{4}	1:{3}	1:{3}	2:{3,4}
Sequence	⟨***AB***⟩			
Start bit	2			
Value	7			
Index	4	3	2	1
Position *A*	1:{1,4}	1:{1,3}	1:{1,3}	2:{2,3,4}
Position *B*	2:{2,3}	2:{2,4}	2:{2,3,4}	∅
Position *AB*	1:{2,3}	1:{2,4}	1:{2,3,4}	∅
Sequence	⟨***AC***⟩			
Start bit	1			
Value	15			
Index	4	3	2	1
Position *A*	1:{1,4}	1:{1,3}	1:{1,3}	2:{2,3,4}
Position *C*	3:{3}	2:{2,5}	3:{3}	1:{1,4}
Position *AC*	1:{3}	1:{2,5}	1:{3}	2:{4}

Table 3.5. Itemset extension (minsup = 50%)

Sequence	⟨(***AB***)⟩			
Start bit	2			
Value	1			
Index	4	3	2	1
Position *A*	1:{1,4}	1:{1,3}	1:{1,3}	2:{2,3,4}
Position *B*	2:{2,3}	2:{2,4}	2:{2,3,4}	∅
Position (*AB)*	∅	∅	3:{3}	∅
Sequence	⟨(***AC***)⟩			
Start bit	1			
Value	3			
Index	4	3	2	1
Position *A*	1:{1,4}	1:{1,3}	1:{1,3}	2:{2,3,4}
Position *C*	3:{3}	2:{2,5}	3:{3}	1:{1,4}
Position (*AC)*	∅	∅	3:{3}	4:{4}

Table 3.6. Set of FSP from Table 2.2 with minsup = 50%

No.	Item	FSPs – Support
1	⟨*A*⟩	⟨*A*⟩:4, ⟨*AA*⟩:4, ⟨*AB*⟩:3, ⟨*AC*⟩:4, ⟨(*AC*)⟩:2, ⟨*AAB*⟩:2, ⟨*AAC*⟩:2, ⟨*A*(*AC*)⟩:2, ⟨*ABA*⟩:3, ⟨*ABB*⟩:3, ⟨*ABC*⟩:3, ⟨*A*(*BC*)⟩:3, ⟨*ACA*⟩:2, ⟨*ACB*⟩:2, ⟨*ABAB*⟩:2, ⟨*AB*(*BC*)⟩:2, ⟨*A*(*BC*)*A*⟩:2, ⟨*A*(*BC*)*B*⟩:2
2	⟨*B*⟩	⟨*B*⟩:3, ⟨*BA*⟩:3, ⟨*BB*⟩:3, ⟨*BC*⟩:3, ⟨(*BC*)⟩:3, ⟨*BAB*⟩:2, ⟨*B*(*BC*)⟩:2, ⟨(*BC*)*A*⟩:2, ⟨(*BC*)*B*⟩:2
3	⟨*C*⟩	⟨*C*⟩:4, ⟨*CA*⟩:3, ⟨*CB*⟩:2, ⟨*CC*⟩:2, ⟨*CAC*⟩:2

3.3.2. Experimental results

Experiments were conducted to evaluate the proposed parallel algorithm. The experiments were performed on a personal computer with an Intel Core i7-4500 1.8-GHz CPU with 4 cores, 3 MB of L3 cache, 4 GB of RAM and running 64-bit Windows 10 Home. The algorithm was implemented using .Net Framework 4.5.

Two of the five databases used for comparison were generated using the IBM synthetic data generator. The third database was provided by Ferenc Bodon (http://fimi.ua.ac.be/data/). The fourth and fifth databases were provided by Fournier-Viger (http://www.philippe-fournier-viger.com/spmf/). Information of the databases is shown in Table 3.7. The definitions of parameters used to generate the databases are shown in Table 2.4.

Table 3.7. Three databases used in experiments.

Database	# sequences	#items	
C6T5N1kD1k	1,000	1,000	Synthetic databases
C6T5N1kD10k	10,000	1,000	
Kosarak25k	25,000	41,270	Click stream data of a Hungarian online news portal
BMSWebView1	59,601	497	Click stream data from an e-commerce site
BMSWebView2	77,512	3,340	Click stream data from the KDD-CUP 2000

Although many sequence mining algorithms have been proposed, none performs sequence mining on multi-core processors. The PIB-PRISM algorithm was proposed [47] based on a multi-core processors architecture for SPM and it used prime encoding theory for fast determining the support values. We also implemented DBV-SPM (a mining method that uses the DBV structure) for comparison with the proposed algorithm.

Experiments were conducted to compare PIB-PRISM, DBV-SPM and the proposed pDBV-SPM for various *minsup* values (0.5-0.8%). The runtime results are shown in Figure 3.14 - Figure 3.18 and memory usage results are shown in Figure 3.19 - Figure 3.23 for the five databases, respectively.

Parallel mining improved performance without affecting the results. Table 3.8 shown the mining results of the PIB-PRISM, DBV-SPM and pDBV-SPM. We can see that the results of those algorithms

have been always the same for all databases to prove the correctness of pDBV-SPM.

Table 3.8. The mining results of PIB-PRISM, DBV-SPM, pDBV-SPM.

#FSPs	Database	*minsup* (%)			
		0.5	0.6	0.7	0.8
PIB-PRISM **DBV-SPM** **pDBV-SPM**	C6T5S4I4N1kD1k	13631	7035	4304	2856
	C6T5S4I4N1kD10k	4761	2912	1914	1382
	Kosarak25k	1668	1148	826	593
	BMSWebView1	201	162	133	105
	BMSWebView2	408	257	187	138

3.3.3. Runtime

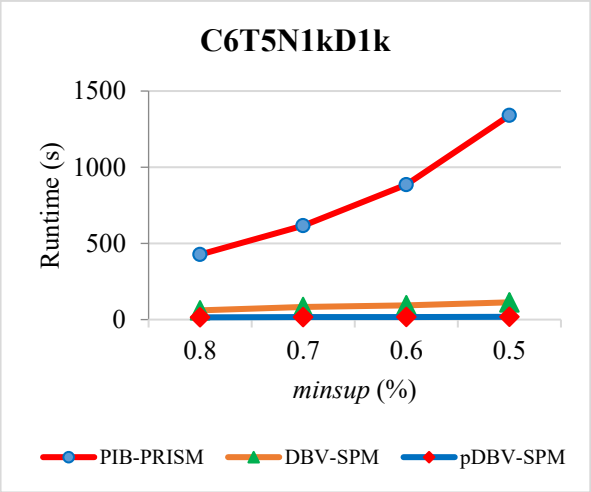


Figure 3.14. Runtime on C6T5N1kD1k

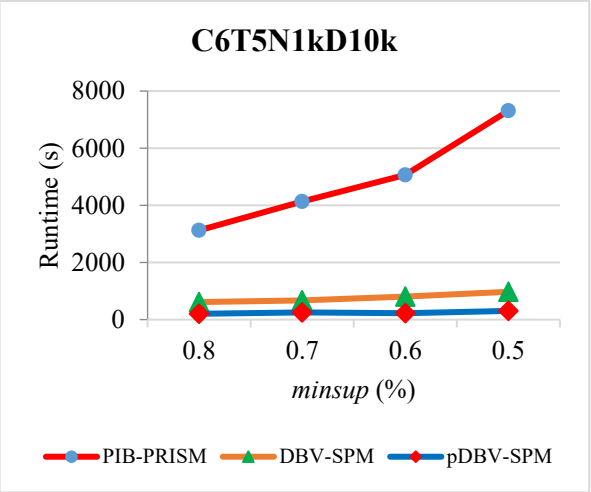


Figure 3.15. Runtime on C6T5N1kD10k

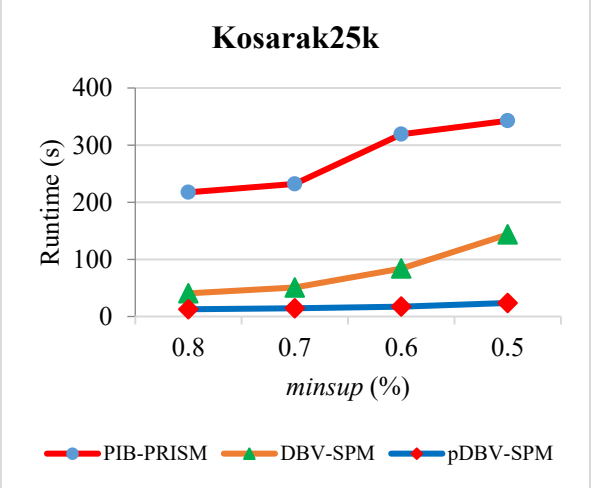


Figure 3.16. Runtime on Kosarak25k

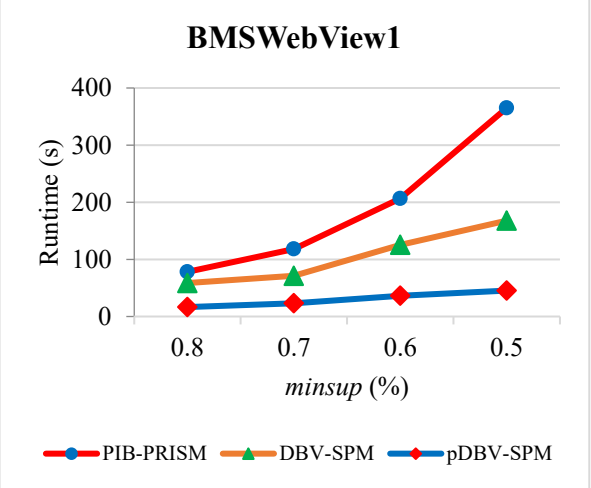


Figure 3.17. Runtime on BMSWebView1

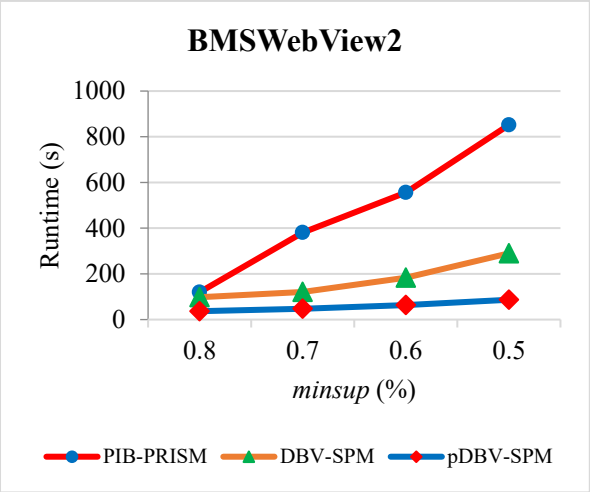


Figure 3.18. Runtime on BMSWebView2

The experimental results show that pDBV-SPM is faster than PIB-PRISM for all five databases. In Figure 3.14, we compare the runtime for the *C6T5N1kD1k* database with *minsup* changes from 0.5% to 0.8%, we easily found that the runtime of pDBV-SPM is faster than PIB-PRISM and DBV-SPM (14.69, 427 and 60.59 seconds, respectively). In a similar way, pDBV-SPM is always the best algorithm for *C6T5N1kD10k* database, as shown in Figure 3.15 (runtimes are 201.75, 3123.25 and 613.29 seconds). Especially, when we decrease *minsup*, the runtime of PIB-PRISM and DBV-SPM have significantly increase while that of pDBV-SPM has softly increase. In general, pDBV-SPM outperform PIB-PRISM and DBV-SPM for this database. Figure 3.16 compares the runtime of pDBV-SPM, PIB-PRISM and DBV-SPM for *Kosarak25k* database. With *minsup* changes from 0.5% to 0.8%, pDBV-SPM is always the best algorithm for mining frequent sequential patterns. Especially, with the smaller *minsup*, the time gaps between the runtime of pDBV-SPM and those of the PIB-PRISM and DBV-SPM is larger. Similar results were obtained for the other databases (runtimes are 16.53, 78.24 and 58.60 seconds for *BMSWebView1*, as shown in Figure 3.17, runtimes are 36.73, 119.77 and 97.88 seconds for *BMSWebView2*, as shown in Figure 3.18, respectively). When *minsup* was changed from 0.8% to 0.5%, the runtimes of pDBV-SPM have been always faster than those of PIB-PRISM.

In short, through the above experiments, we can conclude that pDBV-SPM outperform PIB-PRISM and DBV-SPM for mining frequent sequential patterns.

3.3.4. Memory usage

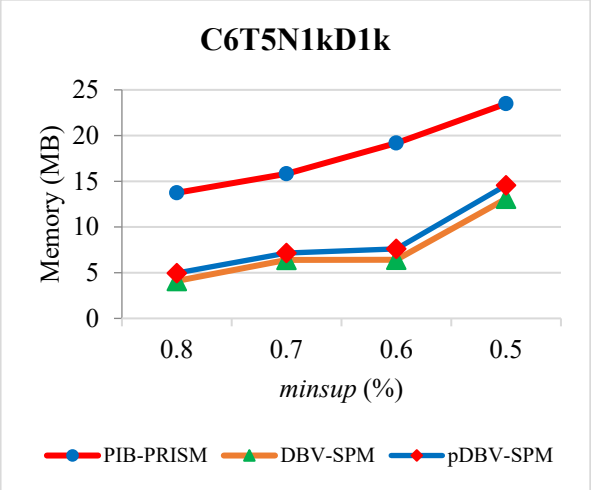


Figure 3.19.Memory usage for C6T5N1kD1k

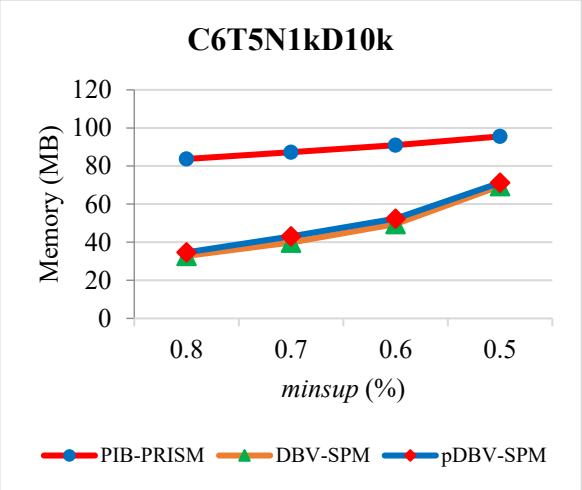


Figure 3.20.Memory usage for C6T5N1kD10k

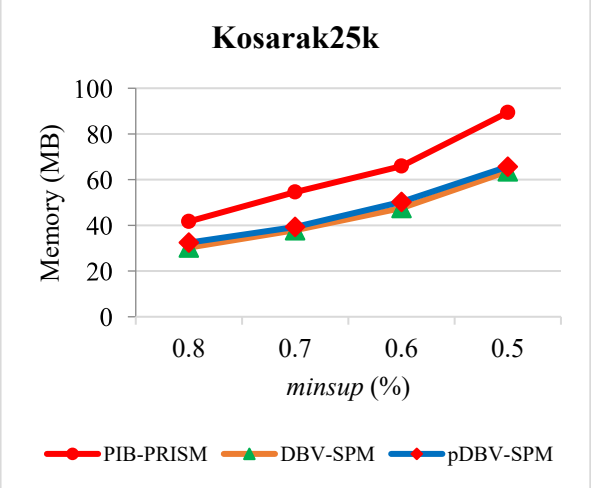


Figure 3.21.Memory usage for Kosarak25k

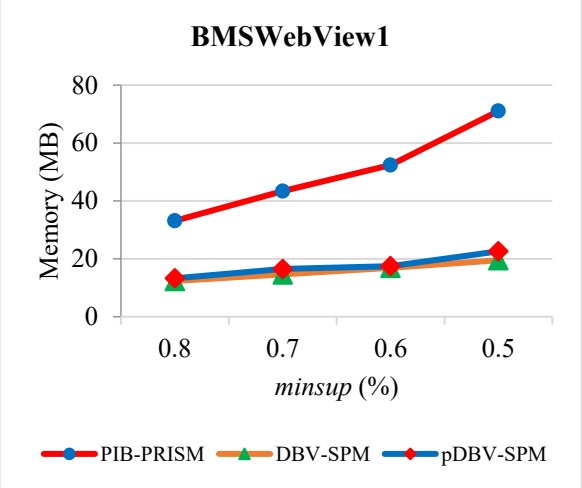


Figure 3.22. Memory usage for BMSWebView1

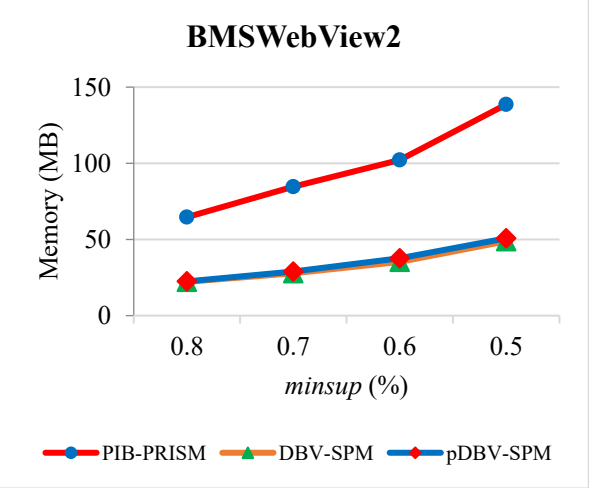


Figure 3.23. Memory usage for BMSWebView2

The memory usage of pDBV-SPM was lower than that of PIB-PRISM for all five databases. Figure 3.19 reports the memory usage of pDBV-SPM, PIB-PRISM and DBV-SPM for the *C6T5N1kD1k* database with various *minsup* settings. With *minsup* = 0.8%, the memory usage values of the PIB-PRISM is the largest (pDBV-SPM is 4.96 and PIB-PRISM is 13.74). In a similar way, the results were obtained for the other databases (memory usage values of 83.71 and 34.67 MB for *C6T5N1kD10k*, as shown in Figure 3.20, 41.70 and 32.43 MB for *Kosarak25k*, as shown in Figure 3.21, 33.14 and 13.27 MB for *BMSWebView1*, as shown in Figure 3.22, 64.65 and 22.46 MB for *BMSWebView2*, as shown in Figure 3.23, respectively). The results confirm that statement again.

However, the gap between the memory usage of DBV-SPM and that of pDBV-SPM was insignificant for all databases. The memory usage values are 4.10 and 4.96 MB for *C6T5N1kD1k*, 32.81 and 34.67 MB for *C6T5N1kD10k,* 30.28 and 32.43 MB for *Kosarak25k*, 12.37 and 13.27 MB for *BMSWebView1*, 22.19 and 22.46 MB for *BMSWebView2*, as shown in Figure 3.19 - Figure 3.23, respectively.

pDBV-SPM had lower memory usage because parallel processing divides the tasks to be processed into independent branches. The experimental results show that the proposed algorithm significantly reduces execution time and memory usage compared to those of PIB-PRISM for the five databases. When *minsup* was decreased, more FSPs were obtained and thus the runtime and memory usage increased.

The numbers of nodes in the search tree of these methods were the same. The main difference between pDBV-SPM and PIB-PRISM is the data structure. pDBV-SPM uses DBV-Pattern and thus can check the backward and forward extensions quickly. Moreover, pDBV-SPM operations are based on bit manipulation and thus it requires less memory. Therefore, pDBV-SPM is more efficient in terms of execution time and memory usage. PIB-PRISM, based on prime theory, requires more processing time than that of pDBV-SPM because the greatest common divisor operation has to switch between binary value and primes.

A disadvantage of PIB-PRISM is its unbalanced search tree (i.e., some branches have more nodes and thus need more processing time). This problem is avoided in pDBV-SPM by sorting items according to their support values in ascending order. In pDBV-SPM, the set of frequent 1-sequences are sorted before task assignment. Most nodes on the leftmost branches are infrequent and are pruned during the search. Nodes in the rightmost branches are frequent and not pruned during the search. This strategy balances the search tree for parallel processing and thus the search times for branches of the search tree are similar.

3.4. Parallel mining inter-sequence patterns with item constraints

The problem of mining inter-sequence pattern is a sub-task in data mining. Several algorithms for such mining have been proposed in the recent years. However, these algorithms only focus on the problem of mining frequent inter-sequence patterns without constraints. Most frequent inter-sequence patterns are either redundant or insignificant.

In essence, end users usually consider a small number of patterns which contain at least one item from a set of given item constraints. For example, when analyzing the sequence databases of supermarkets, there are a lot of inter-sequence patterns. However, at a time, managers only concerned the patterns contained a number of specific items. Therefore, the problem of mining inter-sequence patterns with item constraints requires thorough studies. However, this problem has not been previously considered.

This section, we propose two novel algorithms for fast mining inter-sequence patterns with item constraints. First, the problem of mining inter-sequence patterns with item constraints is introduced. Then, we develop a theorem to reduce the time of determining whether a candidate sequence satisfies the constraints. The ISP-IC (Inter-Sequence Pattern with Item Constraint mining) algorithm based on the above theorem for mining inter-sequence patterns with item constraints is then proposed. Next, a lemma and the second algorithm, named *i*ISP-IC, are proposed. Then, we present a parallel version of *i*ISP-IC, named *pi*ISP-IC algorithm. Final, experiments are conducted to show the effectiveness of the *pi*ISP-IC in terms of mining time and memory usage compared to the post-processing of the-state-of-the-art method for mining inter-sequence patterns (EISP-Miner) which is called by POST-EISP-Miner, ISP-IC and *i*ISP-IC algorithms.

3.4.1. Inter-sequence pattern mining

Consider the set of items $I = \{i_1, i_2, i_3,\ldots, i_m\}$, where i_i is an item ($1 \le i \le m$). A sequence $S = \langle I_1, I_2, I_3,\ldots, I_u\rangle$ is the set of ordered itemsets, where $I_k \subseteq I$ ($1 \le k \le u$) is an itemset. A sequence database $SDB = \{S_1, S_2, S_3,\ldots, S_n\}$, where n is the number of sequences in SDB and S_i ($1 \le i \le n$) is a tuple $\langle DAT, Sequence\rangle$, where DAT is the property of S_i used to describe the contextual information, such as time stamp or space location associated with Table 3.9 shows an example sequence database. This database is used throughout this section.

Table 3.9. Example sequence database

Dat	***Sequences***
1	$\langle C(AB)\rangle$
2	$\langle C(ABC)A\rangle$
3	$\langle AD\rangle$
4	$\langle ABD\rangle$
5	$\langle AC\rangle$
6	$\langle BCD\rangle$
7	$\langle (AB)C\rangle$

Let d_1 and d_2 be two DAT values associated with sequences S_1 and S_2, respectively. Let d_1 be the reference point, $[d_2 - d_1]$ is the span between S_2 and S_1. The sequence S_2 at domain attribute (DAT) d_2 with respect to d_1, denoted by $S_2[d_2 - d_1]$, is called an *e*-sequence (extended sequence). For example, with SBD shown in Table 3.9, let the first sequence be the reference point. Then, the *e*-sequence of the second transaction is $\langle C(ABC)A\rangle[1]$.

Let $S[d] = \langle I_1, I_2, I_3,\ldots, I_u\rangle[d]$ be a sequence where I_k is an itemset ($1 \le k \le u$) and $[d]$ is the span of S. The I_k associated with $[d]$ is defined as an extended itemset (*e*-itemset), denoted by $\langle I_k\rangle[d]$. If $I_k = (i_1, i_2, i_3,\ldots, i_m)$, where each i_j is the items ($1 \le j \le m$), then the i_j associated with $[d]$ is defined as an extended item (*e*-item), denoted by $(i_j)[d]$. For example, $\langle C(ABC)A\rangle[1]$ has 3 *e*-itemsets ($\langle C\rangle[1]$, $\langle (ABC)\rangle[1]$ and $A[1]$) and 3 *e*-items ($(C)[1]$, $(A)[1]$ and $(B)[1]$).

Given the set of k sequences $\langle d_1, S_1\rangle$, $\langle d_2, S_2\rangle$,…, $\langle d_k, S_k\rangle$ in *SDB*, $\Psi = S_1[0] \cup S_2[d_2 - d_1] \cup \ldots \cup S_k[d_k - d_1]$ is called a mega-sequence with $k \geq 1$. In a mega-sequence, the span between *DAT* of the first transaction and that of the last transaction have to be less than or equal to *maxspan* (i.e., $d_k - d_1 \leq maxspan$ in Ψ). For *SDB* shown in Table 3.9, with *maxspan* = 1 and *Dat* = 1 as the reference point, the list of mega-sequences is shown in Table 3.10.

Table 3.10. Megasequences of SDB with maxspan = 1

Dat	***Megasequences***
1	$\langle C(AB)\rangle[0]\ \langle C(ABC)A\rangle[1]$
2	$\langle C(ABC)A\rangle[0]\langle AD\rangle[1]$
3	$\langle AD\rangle[0]\ \langle ABD\rangle[1]$
4	$\langle ABD\rangle[0]\ \langle AC\rangle[1]$
5	$\langle AC\rangle[0]\langle BCD\rangle[1]$
6	$\langle BCD\rangle[0]\ \langle (AB)C\rangle[1]$
7	$\langle (AB)C\rangle[0]$

Consider the pattern $\beta = (u_1)[v_1], (u_2)[v_2],\ldots, (u_n)[v_n]$, where $(u_i)[v_i]$ $(1 \leq i \leq n)$ is an *e*-item (extended item). The number of *e*-items in β is called the length of β and a sequence with length k is denoted as a k-sequence. There are three types of relationship between two *e*-items $(u_i)[v_i]$ and $(u_{i+1})[v_{i+1}]$ in β $(1 \leq i < n)$.

1. **Itemset extension**: if $v_i = v_{i+1}$ and $(u_i u_{i+1})$ is an itemset, then $(u_{i+1})[v_{i+1}]$ is an itemset extension $(+_I)$ of $(u_i)[v_i]$.
2. **Sequence extension**: if $v_i = v_{i+1}$ and $\langle u_i u_{i+1}\rangle$ is a sequence, then $(u_{i+1})[v_{i+1}]$ is a sequence extension $(+_S)$ of $(u_i)[v_i]$.
3. **Inter extension**: if $v_i < v_{i+1}$, then $(u_{i+1})[v_{i+1}]$ is an inter extension $(+_T)$ of $(u_i)[v_i]$.

For example, in *SDB*, $\langle BCD\rangle[0]\ \langle (AB)C\rangle[1]$ is a 6-sequence. $(B)[0]$ is an itemset extension of $(CD)[0]$. $(B)[0]$ is an inter extension of $(C)[1]$ and $AB[1]$ is a sequence extension of $(C)[1]$. In other words, $(B)[0] +_I (CD)[0] = \langle (BCD)\rangle[0]$, $(AB)[0] +_S (C)[0] = \langle (AB)C\rangle[0]$ and $(B)[0] +_T (C)[1] = \langle B\rangle[0]\langle C\rangle[1]$.

Given two sequences, $S = \langle S_1, S_2,\ldots, S_n\rangle$ and $S' = \langle S'_1, S'_2,\ldots, S'_m\rangle$ with $n \leq m$, S is the subsequence of S' if and only if $\exists n, j_1, j_2,\ldots, j_n$ such that (1) $1 \leq j_1 < j_2 < \ldots < j_n \leq m$ and (2) $S_1 \subseteq S'_{j_1}$, $S_2 \subseteq S'_{j_2}$, …, $S_n \subseteq S'_{j_n}$. For example, a sequence $\langle A(BC)DF\rangle$ is the subsequence of $\langle A(ABC)(AC)D(CF)\rangle$, but it is not a subsequence of $\langle (ABC)(AC)D(CF)\rangle$. Consider two patterns $\alpha = S_1[i_1], S_2[i_2], \ldots, S_n[i_n]$ and $\beta = S'_1[j_1], S'_2[j_2],\ldots., S'_m[j_m]$, with $1 \leq n \leq m$. α is a subsequence of β if and only if there exist n e-sequences denoted by $S'_{k1}[j_{k1}]$, $S'_{k2}[j_{k2}],\ldots, S'_{kn}[j_{kn}]$ in β, such that $i_1 = j_{k1}$ and S_1 is a subsequence of S'_{k1}; $i_2 = j_{k2}$ and S_2 is a subsequence of $S'_{k2};\ldots.;$ $i_n = j_{kn}$ and S_n is a subsequence of S'_{kn}. In this case, β is a super-pattern of α. For example, both $\langle D\rangle[0]\langle BAC\rangle[3]$ and $\langle (AD)\rangle[0]\langle (AC)C\rangle[1]$ are subsequences of $\langle D(AD)\rangle[0]\ \langle B(AC)C\rangle[1]\langle B(A)CC\rangle[3]$.

Definition 3.6: Let $a = (i_m)[x]$ and $b = (i_n)[y]$ be 2 *e*-items $a = b$ if and only if $(i_m = i_n) \wedge (x = y)$ and $a < b$, if and only if $x < y$ or $((x = y) \wedge (i_m < i_n))$.

For example, $(C)[1] = (C)[1]$, $(B)[1] < (A)[2]$ and $(A)[1] < (C)[1]$.

Definition 3.7: Let p be a pattern. Function $sub_{ij}(p)$ is defined as (j-i+1) subset e-items of p from position i to position j.

For example, $sub_{1,4}(\langle(AD)\rangle[0]\langle(AC)C\rangle[1]) = \langle(AD)\rangle[0]\langle(AC)\rangle[1]$ and $sub_{5,5}(\langle(AD)\rangle[0]\ \langle(AC)C\rangle[1]) = (C)[1]$.

Definition 3.8: Let $\alpha = \langle u\rangle[0]$ and $\beta = \langle v\rangle[0]$ be two frequent 1-patterns. α is joinable to β in any instance. There are three types of join operation:

1. **itemset extension**: $\alpha \cup_I \beta = \{\langle(uv)\rangle[0]\} | \{\langle(uv)\rangle[0]\}$
2. **sequence extension**: $\alpha \cup_S \beta = \{\langle(uv)\rangle[0]\}$
3. **inter extension**: $\alpha \cup_T \beta = \{\langle u\rangle[0]\langle v\rangle[x] \mid 1 \le x \le maxspan\}$

For example, let $maxspan = 2$, $\langle A\rangle[0] \cup_I \langle B\rangle[0] = \langle(AB)\rangle[0]$; $\langle A\rangle[0] \cup_S \langle B\rangle[0] = \langle AB\rangle[0]$ and $\langle A\rangle[0] \cup_T \langle B\rangle[0] = \{\langle A\rangle[0]\langle B\rangle[1], \langle A\rangle[0]\langle B\rangle[2]\}$.

Definition 3.9: Let α and β be 2 frequent k-patterns, where $k > 1$, $sub_{k,k}(\alpha) = (u)[i]$ and $sub_{k,k}(\beta) = (v)[j]$. α is joinable to β if $sub_{1,k-1}(\alpha) = sub_{1,k-1}(\beta)$ and $i \le j$, which generates three types of join operations: (1) itemset extension: $\alpha \cup_I \beta = \{\alpha +_I (v)[j] \mid (i = j) \wedge (u < v)\}$; (2) sequence extension: $\alpha \cup_S \beta = \{\alpha +_S (v)[j] \mid (i = j)\}$; (3) inter extension: $\alpha \cup_T \beta = \{\alpha +_T (v)[j] \mid (i < j)\}$.

For example, $\langle AB\rangle[0] \cup_I \langle AC\rangle[0] = \langle A(BC)\rangle[0]$, $\langle AB\rangle[0] \cup_S \langle AC\rangle[0] = \langle ABC\rangle[0]$ and $\langle AB\rangle[0] \cup_T \langle A\rangle[0]\langle C\rangle[2] = \langle AB\rangle[0]\langle C\rangle[2]$.

3.4.2. DBV-PatternList data structure

DBV structure is based on the BitTable structure including two elements. (1) Start position: the position of the first non-zero byte in the bit vector; (2) Bit-vector: the list of bytes after removing all zero byte at the beginning. For example, give a BitTable "000011", the DBV is {Start: 5, Bit-vector: "11"}.

Using DBV structure reduces the memory usage and computational operations in intersection between two bit vectors. Using a look-up table [107], the algorithm traverses the DBV once to determine the support of its pattern and therefore the complexity is $O(n)$ where n is the number of elements in its Bit-vector.

Based on the DBV concept, the DBV-PatternList structure that combines DBV and PatternList was proposed [106]. The structure includes: (1) Sequence: storage information of the sequence; (2) Block sequence: one DBV and a list of transaction positions corresponding to each sequence. For example, in *SDB*, consider the structure of sequence $\langle A\rangle[0]$, where A appears at sequences 1, 2, 3, 4, 5 and 7. DBV-PatternList of $\langle A\rangle[0]$ is shown in Figure 3.24.

Sequence	⟨A⟩[0]					
DAT	1	2	3	4	5	7
	↓	↓	↓	↓	↓	↓
Transaction position	2	2 3	1	1	1	1

(a)

Sequence	⟨A⟩[0]					
Start	1					
DBV	15				10	
	↓	↓	↓	↓	↓	↓
Transaction position	2	2 3	1	1	1	1

(b)

Figure 3.24. Structures of (a) PatternList and (b) DBV-PatternList of ⟨A⟩[0]

In Figure 3.24 (b), the structure of a DBV-PatternList contains a number of block sequences. Each block sequence is placed in a cell containing information regarding the position of sequences that appear in the sequence database and the position of the sequence in each transaction. For detail, A appears at sequences 1, 2, 3, 4, 5 and 7, so, bit vector of A is 11111010 and therefore, DBV of A is {15, 10} (note that we use 4 bits for an integer). To find the appropriate extended sequence based on the sequence extension operation, the intersection operation has to be carried out on each block sequence. In Figure 3.24, PatternList of $\langle A \rangle[0]$ requires 26 bytes, whereas DBV-PatternList of $\langle A \rangle[0]$ requires only 20 bytes.

3.4.3. Mining inter-sequence patterns with item constraints

Various strategies have been proposed for mining frequent pattern with itemset constraints. There are three main approaches including post-processing, pre-processing and constrained patterns filtering. Post-processing approaches for mining patterns with item constraints firstly mine patterns and then check them against the constraints [63, 75]. Pre-processing approaches firstly restricts the source database for records that contain the constraints and then find patterns in the filtered database [63]. Constrained patterns filtering approaches integrate the constraints into the actual mining process to generate only patterns that satisfy the constraints. CAP [75] and MFS_DoubleCons [29] algorithms use this strategy for mining frequent patterns with item constraints.

This section presents the problem of inter-sequence pattern mining with item constraints and proposes two efficient algorithms (ISP-IC and *i*ISP-IC algorithms) for its solving. In addition, we will use multi-core processors architecture for parallel mining inter-sequence patterns with item constraints to speed up the execution process, thereby reducing cost and improving the efficiency of systems. In the parallel mining of ISPs, each branch of the search tree can be regarded as a single task, which can be processed independently to generate ISPs.

3.4.4. Problem statement

Given a sequence database (*SBD)*, the minimum support (*minsup*) and a set of items constraint $\chi = \{u_1, u_2, \ldots, u_k\}$, the problem of mining inter-sequence patterns with item constraints is to find all sequences $\alpha = S_1[w_1], S_2[w_2], \ldots, S_m[w_m]$ such that $\exists S_i[w_i] \in \alpha, \exists\, b_j \in S_i[w_i]: b_j \in \chi$.

For example, let $\chi = \{C\}$. Then, the sequence $\langle C(AB) \rangle[0]\ \langle C(ABC)A \rangle[1]$ satisfies the constraints, whereas the sequence $\langle AD \rangle[0] \langle A \rangle[1]$ does not.

3.4.5. ISP-IC algorithm

Theorem 3.3. *If sequence α satisfies constraint χ, then the sequence γ, generated from α, also satisfies constraint χ.*

Proof. If α satisfies the constraint, it means that $\exists S_i[w_i] \in \alpha, \exists b_j \in S_i[w_i]: b_j \in \chi$. There are three cases to consider:

1. **Itemset extension**: There are two sub-cases. (i) If itemset extension $\notin$ itemset$_i$ of α, then $\exists$itemset$_i$ in γ that contains item $b_j \in \chi$. (ii) If itemset extension $\in$ itemset$_i$, then itemset$_i(\alpha) \subset$ itemset$_i(\gamma)$ and thus $\exists b_j \in$ itemset$_i(\gamma)$ such that $b_j \in \chi$.

2. **Itemset extension**: itemset_i is not changed. Therefore, $\exists \text{itemset}_i \in \gamma$ includes item b_j such that $b_j \in \chi$.

3. **Inter extension**: Based on the inter extension definition, itemsets in the sequence just have their indices changed and therefore $\exists \text{itemset}_i$ contains item $b_j \in \chi$

Based on Theorem 3.3, we propose the ISP-IC algorithm for mining inter-sequence patterns with item constraints. First, the ISP-IC algorithm finds all items (F_1) that satisfy the threshold. Then, the algorithm inserts $p \in F_1$ into the DBV-tree. $\forall p \in F_1$ such that $p \in \chi$, $cs(p)$ = true. Next, the algorithm calls **Join_1-PatternList** to combine 1-PatternLists. Finally, for each $p \in$ *DBV-Tree.root*, the algorithm calls the **Join_*k*-PatternList** function to combine p with other ***k*-PatternLists** that follow it. The details of the ISP-IC algorithm are presented in Figure 3.25.

Algorithm 1. ISP-IC algorithm

Input: *SDB*, *minsup*, *maxspan* and $\chi = \{u_1, u_2, \ldots, u_k\}$
Output: *DBV-results*

```
1   DBV-Tree = NULL
2   F1 = {DBV-PatternList(i) | i ∈ I and σ(i) ≥ minsup}
3   For each p in F1 do
4       Add p to childnodes(DBV-Tree)
5       If IC-Check (p, χ) = true then
6           Add p to DBV-results
7           Mark cs(p) as true
8   Join_1-PatternList(DBV-Tree, minsup, maxspan, DBV-results, χ)
9   For each p in childnodes(DBV-Tree) do
10      Join_k-PatternList (p, minsup, maxspan, DBV-results, χ)
11  Return DBV-results

12  Function Join_1-PatternList (DBV-Tree, minsup, maxspan, DBV-results, χ)
13  Let lists = childnodes(DBV-Tree)
14  For each X in lists do
15      For each Y in lists do
16          If (σ(Z = X ∪I Y) ≥ minsup then
17              Add Z to childnodes(X)
18              If cs(X) = true or cs(Y) = true or IC-Check(Z, χ) = true then
19                  Add Z to DBV-results
20                  Mark cs(Z) as true
21          If (σ(Z = X ∪S Y) ≥ minsup then
```

```
22          Add Z to childnodes(X)
23          If cs(X) = true or cs(Y) = true or IC-Check(Z, χ) = true then
24              Add Z to DBV-results
25              Mark cs(Z) is true
26      For d = 1 to maxspan do
27          If (σ(Z = X ∪T Y) ≥ minsup then
28              Add Z to childnodes(X)
29              If cs(X) = true or cs(Y) = true or IC-Check(Z, χ) = true then
30                  Add Z to DBV-results
31                  Mark cs(Z) is true

32  Function Join_k-PatternList (p, minsup, maxspan, DBV-results, χ)
33  Let lists = childnodes(p)
34  For each X in lists do
35      For each Y in lists do
36          If (σ(Z = X ∪I Y) ≥ minsup then
37              Add Z to childnodes(X)
38              If cs(X) = true or cs(Y) = true or IC-Check(Z, χ) = true then
39                  Add Z to DBV-results
40                  Mark cs(Z) as true
41          If (σ(Z = X ∪S Y) ≥ minsup then
42              Add Z to childnodes(X)
43              If cs(X) = true or cs(Y) = true or IC-Check(Z, χ) = true then
44                  Add Z to DBV-results
45                  Mark cs(Z) is true
46          If (σ(Z = X ∪T Y) ≥ minsup then
47              Add Z to childnodes(X)
48              If cs(X) = true or cs(Y) = true or IC-Check(Z, χ) = true then
49                  Add Z to DBV-results
50                  Mark cs(Z) is true
51      Join_k-PatternList(X, minsup, maxspan, DBV-results, χ)

52  Function IC-Check(sequence, χ)
53  For each S in sequence do
54      For each γ in S do
```

```
55        If γ ∈ χ then
56            Return true
57    Return false
```

Figure 3.25. ISP-IC algorithm

3.4.6. An illustrative example

Consider *SDB*, which includes 7 sequences with items $I = \{A, B, C, D\}$, *maxpan* = 1, *minsup* = 25% and $\chi = \{C\}$. DBV-PatternList of each item was shown in Table 3.11.

Table 3.11. Binary vectors of BitTable and their transformation into BitArray

(a)

Item	Bit array	Bit vector
A	1111,1010	15,10
B	1101,0110	13,6
C	1100,1110	12,14

(b)

Dat	*A*	*B*	*C*
1	2	2	1
2	2, 3	2	1, 2
3	1		
4	1	2	
5	1		2
6		1	2
7	1	1	2

Table 3.11(a) shows the bit arrays for the items in *SDB*. In this example, assume that each block is encoded by 4 bits. In *SDB*, bit array of *A* is {1111, 1010} and bit vector of *A* is {15, 10}. For each sequence, the position of each item in the sequence is stored as Table 3.11(b).

Once the sequences have been encoded into binary form and the positions in the sequence have become sequence blocks (with each sequence block containing 4 bits), the sequence database is transformed into a vertical format database and DBV-PatternList structures are created, as shown in Figure 3.26. Note that index column the transaction position and come from Table 3.11(b).

Sequence	<A>[0]					
Start	1					
DBV	15				10	
Index	1(2)	2(2,3)	3(1)	4(1)	1(1)	3(1)

Sequence	<B>[0]				
Start	1				
DBV	13			6	
Index	1(2)	2(2)	4(2)	2(1)	3(1)

Sequence	<C>[0]				
Start	1				
DBV	12		14		
Index	1(1)	2(1,2)	1(2)	2(2)	3(2)

Figure 3.26. DBV-PatternList of A[0], B[0] and C[0]

Then, the branch of ⟨*A*⟩[0] with two levels is obtained. The nodes on the second-level branch are then computed, followed by those on the third-level branch and so on, until no extended sequences are found. Figure 3.27 shows a part of a DBV-tree extended on branch ⟨*A*⟩[0]. Note that the node ⟨*C*⟩[0] (red node) is the constraint and the nodes ⟨*AC*⟩[0] and ⟨*A*⟩[0]⟨*C*⟩[1] (gray node) are inserted into the results without being checked whether they satisfy the constraint because their parent *C*[0] satisfies the constraint.

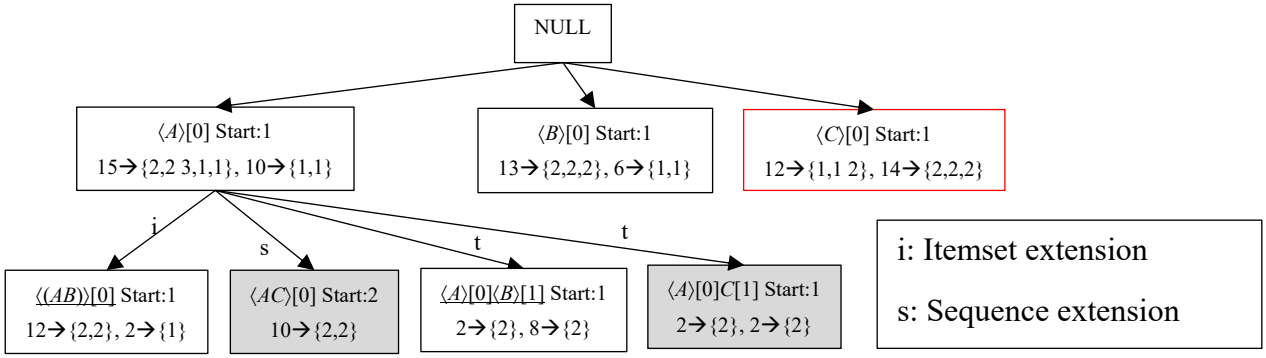


Figure 3.27. DBV-tree extended on ⟨A⟩[0]

3.4.7. iISP-IC algorithm

Lemma 3.1. *Let α satisfy constraint χ then $\forall\beta$, following sequences $\gamma_I = \alpha \cup_I \beta$, $\gamma_S = \alpha \cup_S \beta$ and $\gamma_T = \alpha \cup_T \beta$ also satisfy constraint χ.*

Based on Lemma 3.1, we propose the *i*ISP-IC algorithm. For each $p \in$ *childnodes*(*DBV-Tree*), if p satisfies χ, *i*ISP-IC calls the **Join_*k*-PatternList_NoChecking** function to combine p with other ***k*-PatternLists** that follow it. Otherwise, *i*ISP-IC calls the **Join_*k*-PatternList_Plus** function. The details of the *i*ISP-IC algorithm are presented in Figure 3.28.

Algorithm 2. *i*ISP-IC algorithm

Input: *SDB*, *minsup*, *maxspan* and $\chi = \{u_1, u_2, \ldots, u_k\}$
Output: *DBV-results*

```
1   Let DBV-Tree = NULL
2   F1 = {DBV-PatternList(i) | i ∈ I and σ(i) ≥ minsup}
3   For each p in F1 do
4       Add p to childnodes(DBV-Tree)
5       If IC-Check (p, χ) = true then
6           Add p to DBV-results
7           Mark cs(p) as true
8   Join_1-PatternList(DBV-Tree, minsup, maxspan, DBV-results, χ)
9   For each p in childnodes(DBV-Tree) do
10      If cs(p) = true then
11          Join_k-PatternList_Plus(p, minsup, maxspan, DBV-results, χ)
12      Else
13          Join_k-PatternList _NoChecking(p, minsup, maxspan, DBV-results)
14  Return DBV- results

15  Function Join_k-PatternList_Plus(p, minsup, maxspan, DBV-results, χ)
```

```
16    Let lists = childnodes(p)
17    For each X in lists do
18       For each Y in lists do
19          If (σ(Z = X ∪_I Y) ≥ minsup then
20             Add Z to childnodes(X)
21             If cs(Y) = true or IC-Check(Z, χ) = true then
22                Add Z to DBV-results
23                Mark cs(Z) as true
24          If (σ(Z = X ∪_S Y) ≥ minsup then
25             Add Z to childnodes(X)
26             If cs(Y) = true or IC-Check(Z, χ) = true then
27                Add Z to DBV-results
28                Mark cs(Z) is true
29          If (σ(Z = X ∪_t Y) ≥ minsup then
30             Add Z to childnodes(X)
31             If cs(Y) = true or IC-Check(Z, χ) = true then
32                Add Z to DBV-results
33                Mark cs(Z) is true
34       If cs(X) = true then
35          Join_k-PatternList_NoChecking(X, minsup, maxspan, DBV-results)
36       Else
37          Join_k-PatternList_Plus(X, minsup, maxspan, DBV-results, χ)

38    Function Join_k-PatternList_NoChecking(p, minsup, maxspan, DBV-results)
39    Let lists = childnodes(p)
40    For each X in lists do
41       For each Y in lists do
42          If (σ(Z = X ∪_I Y) ≥ minsup then
43             Add Z to DBV-results and childnodes(X)
44          If (σ(Z = X ∪_S Y) ≥ minsup then
45             Add Z to DBV-results and childnodes(X)
46          If (σ(Z = X ∪_T Y) ≥ minsup then
47             Add Z DBV-results and childnodes(X)
48       Join_k-PatternList_NoChecking(X, minsup, maxspan, DBV-results)
```

Figure 3.28. iISP-IC algorithm

In the example, suppose that $\chi = \{A\}$, the red node in Figure 3.29. The *i*ISP-IC algorithm calls the **Join_*k*-PatternList_NoChecking** function to expand the node ⟨*A*⟩[0], as shown in Figure 3.29. Based on Lemma 3.1, this function does not check all its child nodes.

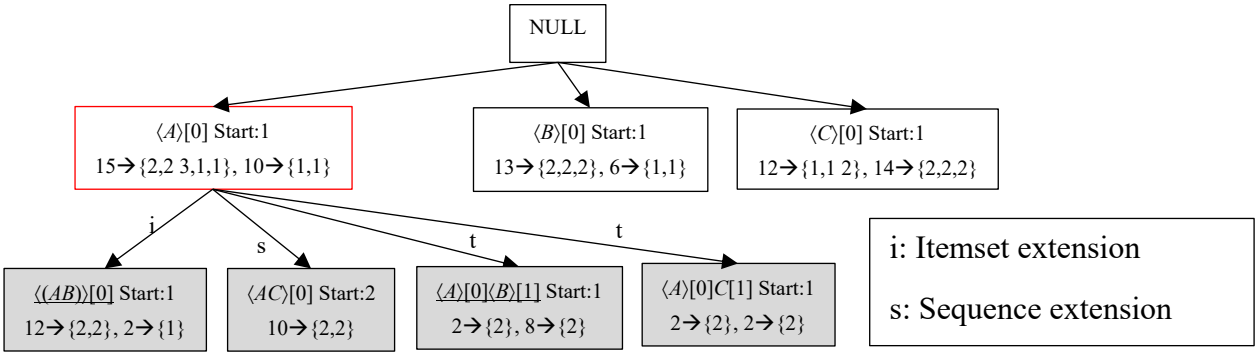


Figure 3.29. DBV-tree extended on ⟨A⟩*[0] using* i*ISP-IC algorithm*

3.4.8. piISP-IC: a parallel version of iISP-IC algorithm

In the parallel mining of ISPs, each branch of the search tree can be regarded as a single task, which can be processed independently to generate ISPs. An example is given in Figure 3.30. There are three tasks on level 1 of the task tree. Tasks 1, 2 and 3 process branches *A*, *B* and *C*, respectively.

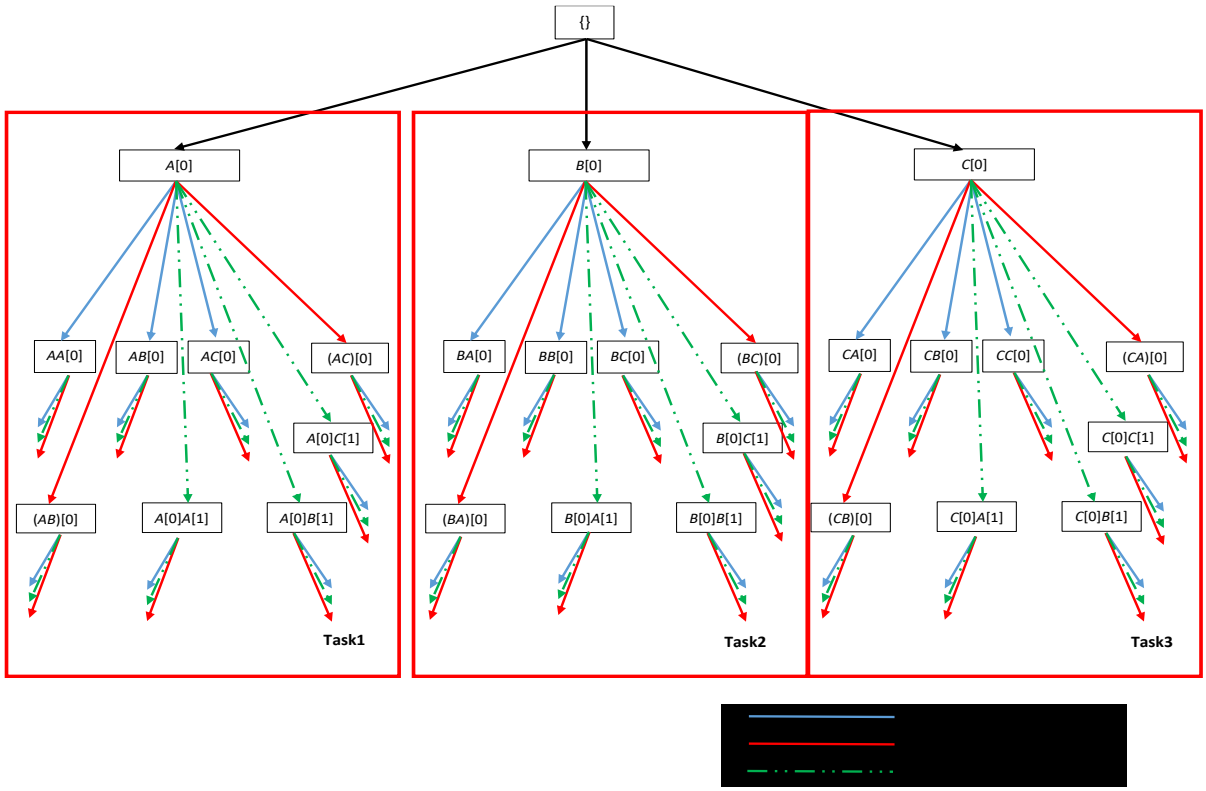


Figure 3.30. Example of task tree

*pi*ISP-IC algorithm is generally based on *i*ISP-IC algorithm and add a parallel implementation of tasks instead of threads. The advantage of *pi*ISP-IC is that each task is assigned to searching a branch of the tree

and is processed independently. The advantages over using threads are as follows. First, tasks require less memory than do threads. Second, a thread runs on only one core, whereas a task can run on multiple cores. Finally, threads require more processing time than do tasks because the operating system needs to allocate data structures for threads, such as initialization and destruction and must perform context switching between threads.

The time complexity of sequential pattern mining algorithms depends on the number of patterns in the search space and the cost of the operations of generating and processing each itemset. In multi-core processors, a key factor is task scheduling. With the widespread use of multi-core processors, designing an effective task scheduling strategy has been a hot issue. Currently, the time complexity of task scheduling for multi-core processors is considered as an NP (Non-Deterministic Polynomial) problem and no exists the optimal solutions. The existing scheduling algorithms can only get the suboptimal solutions based on solution approximated by heuristics. Heuristic is generally considered as the most suitable method to find the suboptimal solutions for NP problems.

3.4.9. Experimental results

This section compares the mining time of POST-EISP-Miner, ISP-IC, *i*ISP-IC and *pi*ISP-IC, to confirm the effectiveness of the proposed methods. All the experiments were performed on a PC with Intel Xeon Processor E5-2680 v2 (25M Cache, 2.80 GHz, 20 threads) CPUs installed with 768 GB of main memory and coded in C# in Visual Studio 2015. Synthetic databases were generated using the IBM synthetic data generator to mimic transactions in a retail environment. The definitions of parameters used to generate the databases are shown in Table 2.4.

Two synthetic databases (C6T5S4I4N1KD10K and T10I4D100K) used for comparison were generated using the IBM synthetic data generator, was provided by Bodon (http://fimi.ua.ac.be/data/) and other databases were provided by Fournier-Viger (http://www.philippe-fournier-viger.com/spmf/). Information of the databases is shown in Table 3.12.

Table 3.12. Databases used in experiments

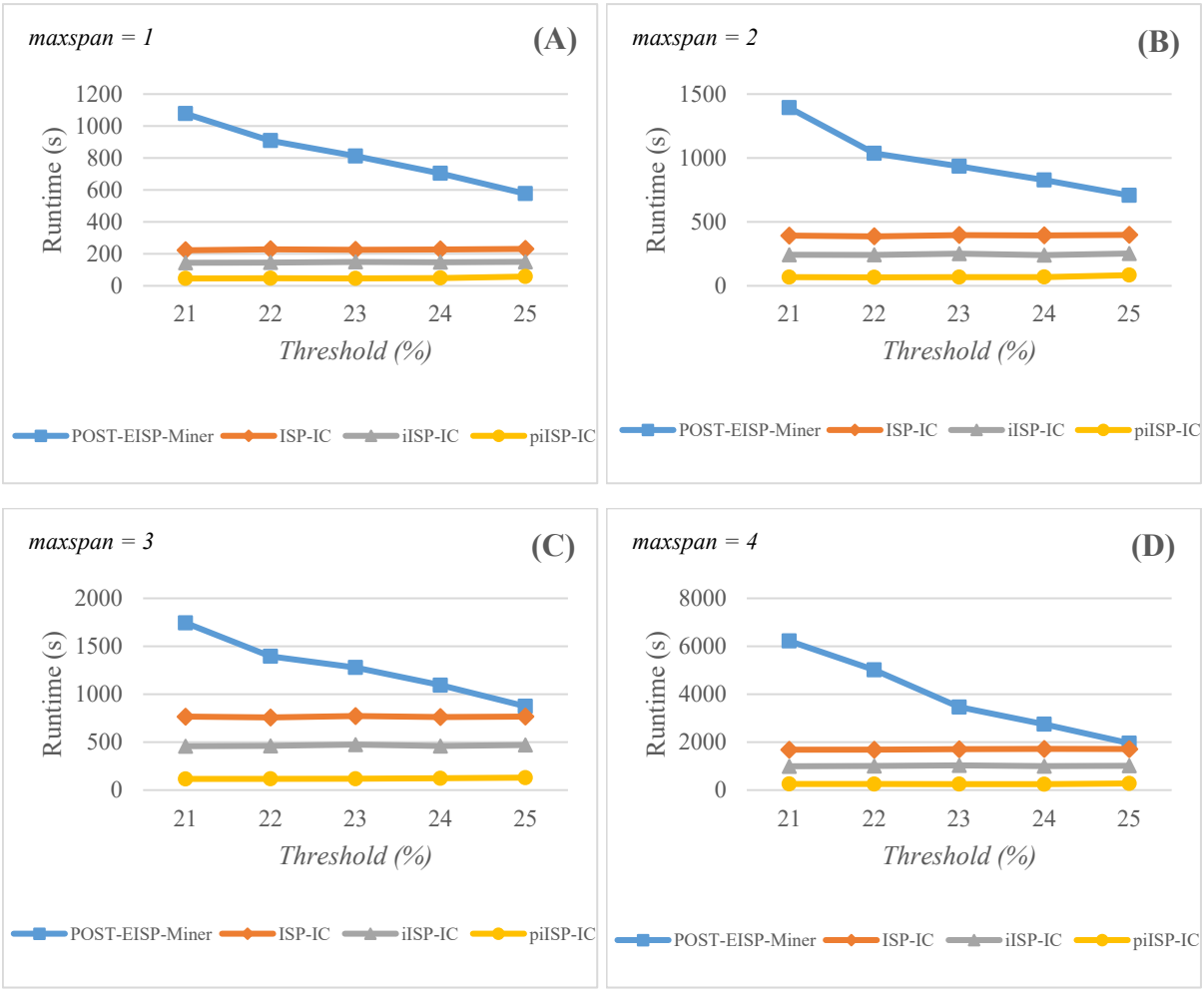
Database	#sequences	#items	
C6T5S4I4N1KD10K	10,000	1,000	Synthetic database
T10I4D100K	100,000	1,000	Synthetic database
BMSWebView2	77,512	3,340	Click stream data from the KDD-CUP 2000

3.4.9.1. Runtime

In Figure 3.31, we compare the runtime of piISP-IC, iISP-IC, ISP-IC and POST-EISP-Miner for C6T5S4I4N1KD10K database with various setting. Figure 3.31(A) fixes *maxspan* = 1 and changes *threshold* from 21 to 25 and we easily found that the runtime of iISP-IC is less than that of ISP-IC and nearly twice as that of piISP-IC, while POST-EISP-Miner was decreased so small. In a similar way, Figure 3.31(B) fixes *maxspan* = 2, Figure 3.31(C) fixes *maxspan* = 3, Figure 3.31(D) fixes *maxspan* = 4 and Figure 3.31(E) fixes *maxspan* = 5, piISP-IC is always the best algorithm for C6T5S4I4N1KD10K database. POST-EISP-Miner cannot perform with *maxspan* = 5 with any threshold, takes a long time without results. Next, in Figure 3.31(F), we fix *threshold* = 21 and change *maxspan* from 1 to 5. piISP-IC is still better than iISP-

IC, ISP-IC and POST-EISP-Miner algorithms. Especially, when we decrease *threshold*, the runtime of iISP-IC, ISP-IC and POST-EISP-Miner have significantly increase while that of piISP-IC has softly increase. In general, piISP-IC outperforms iISP-IC, ISP-IC and POST-EISP-Miner for this database.

Figure 3.32 performs the same for T10I4D100K dataset. We easy found that the runtimes of iISP-IC and ISP-IC are nearly the same, POST-EISP-Miner is highest while the runtime of piISP-IC is much better than that of iISP-IC, ISP-IC and POST-EISP-Miner. POST-EISP-Miner cannot perform with *minsup* = 4 and *maxspan* = 5.



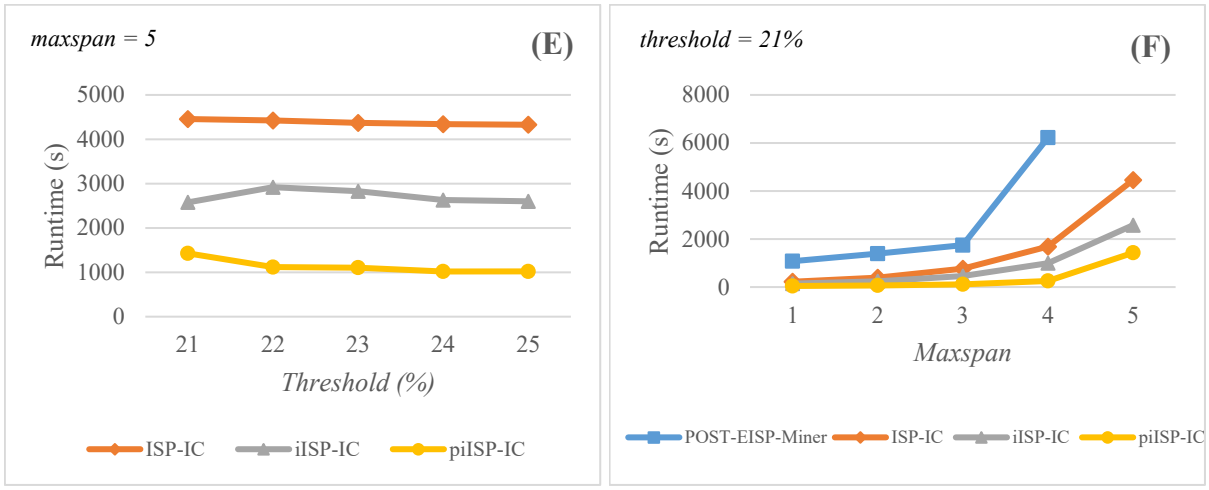
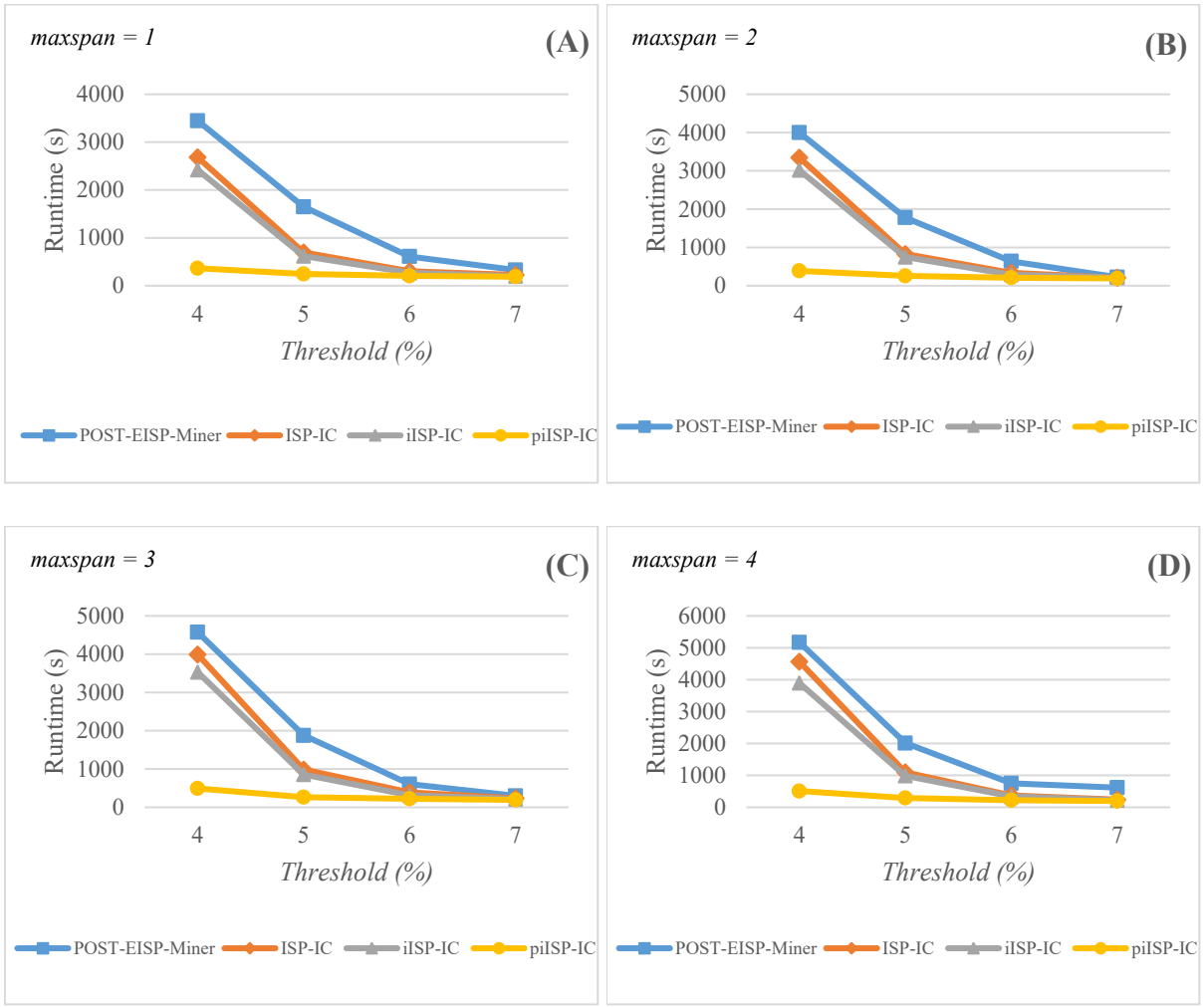


Figure 3.31. Mining time of piISP-IC, iISP-IC, ISP-IC and POST-EISP-Miner for C6T5S4I4N1KD10K database with 15 random item constraints and (A) maxspan = 1; (B) maxspan = 2; (C) maxspan = 3; (D) maxspan = 4; (E) maxspan = 5; (F) threshold = 21%.



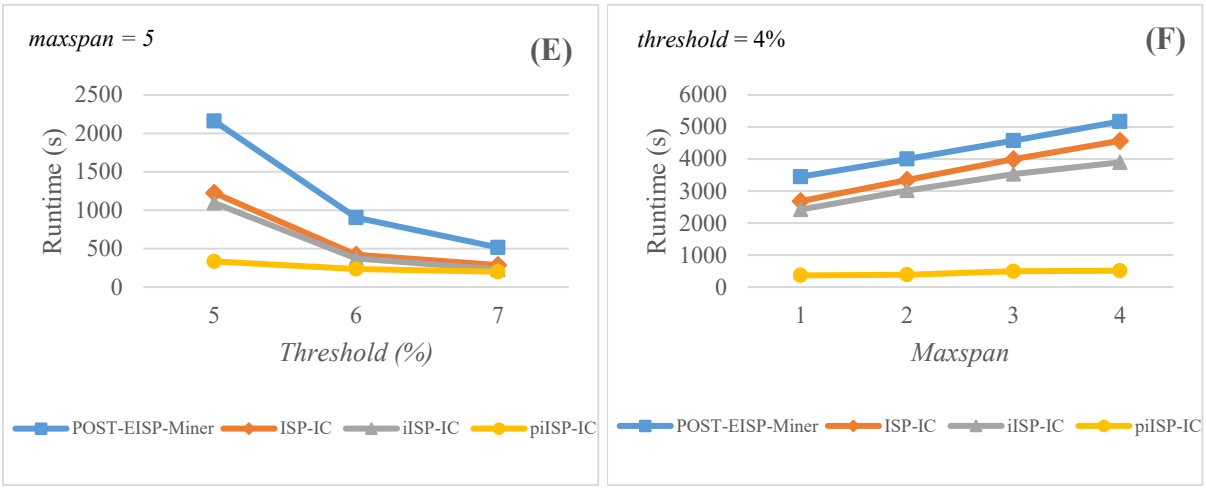
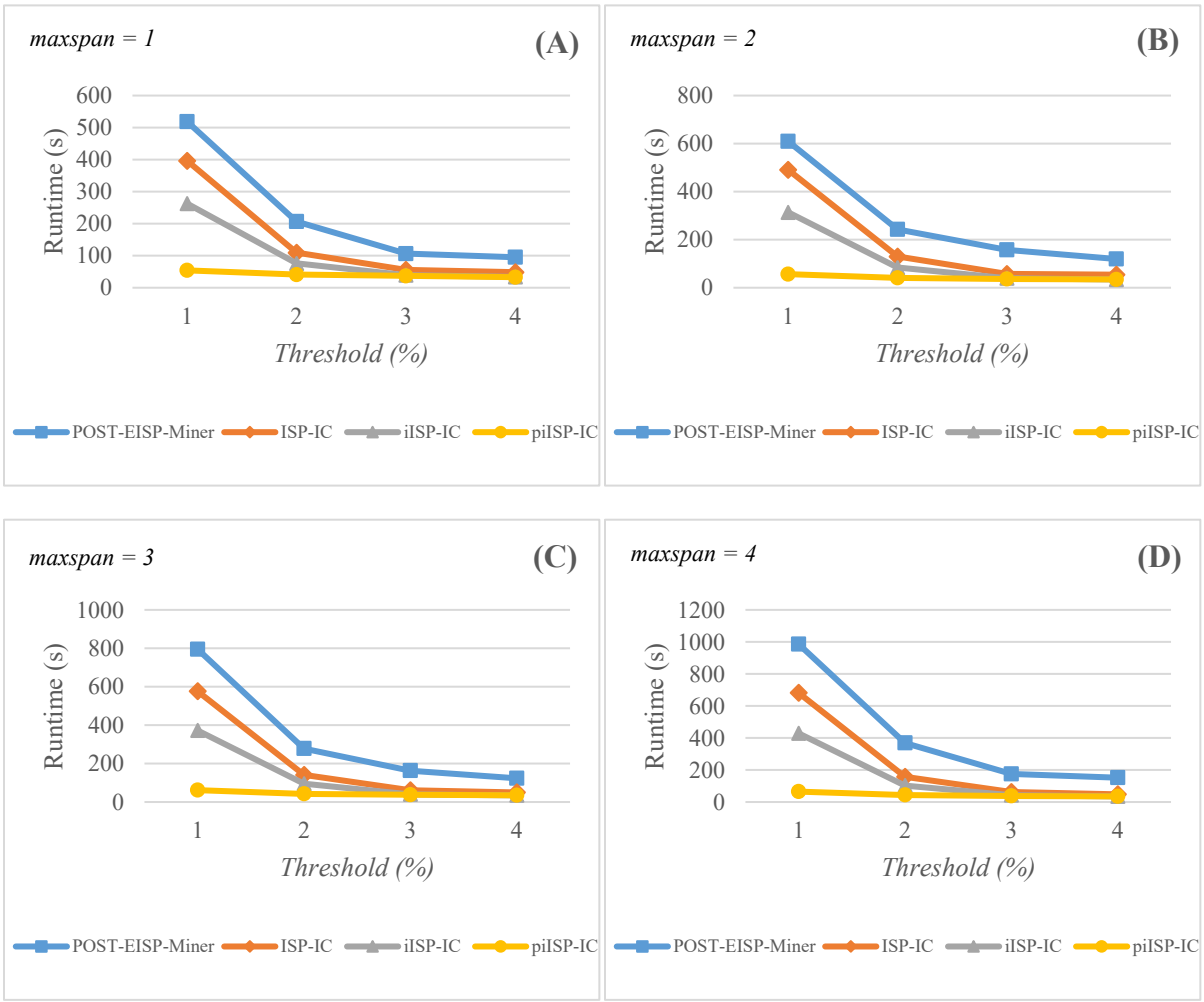


Figure 3.32. Mining time of piISP-IC, iISP-IC, ISP-IC and POST-EISP-Miner for T10I4D100K database with 15 random item constraints and (A) maxspan = 1; (B) maxspan = 2; (C) maxspan = 3; (D) maxspan = 4; (E) maxspan = 5; (F) threshold = 4%



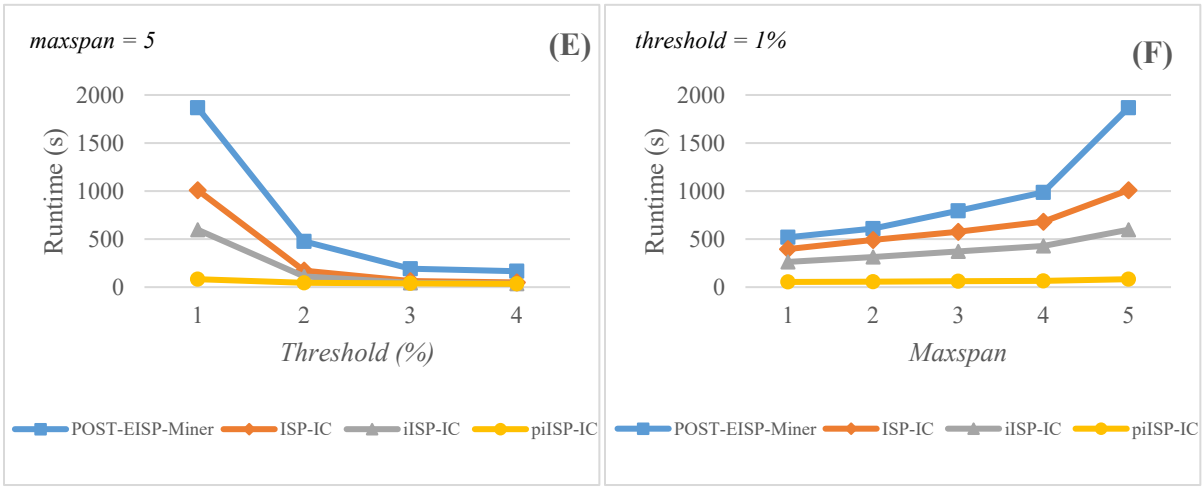
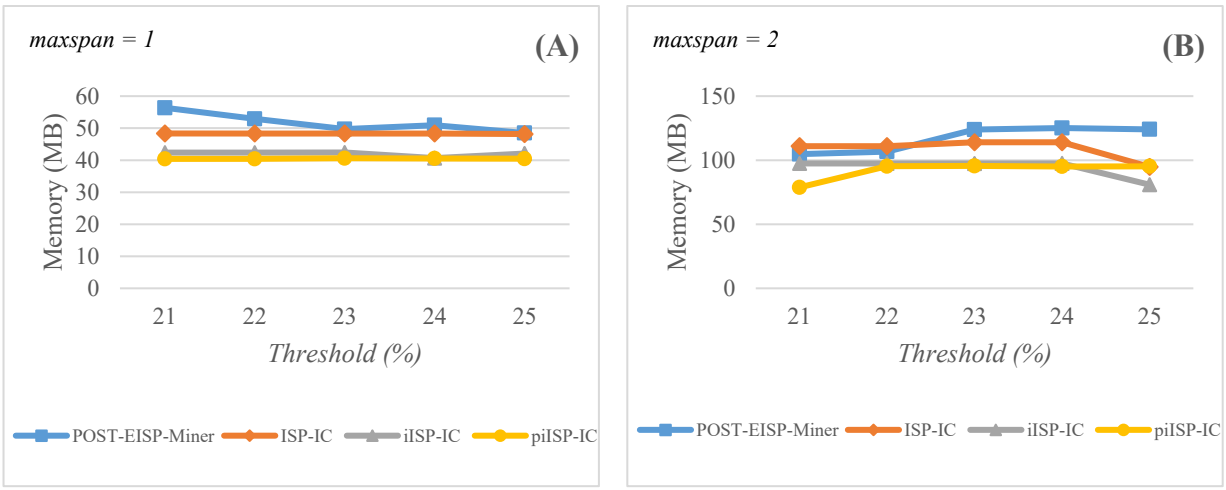


Figure 3.33. Mining time of piISP-IC, iISP-IC, ISP-IC and POST-EISP-Miner for BMSWebView2 dataset with 15 random item constraints and (A) maxspan = 1; (B) maxspan = 2; (C) maxspan = 3; (D) maxspan = 4; (E) maxspan = 5; (D) threshold = 1%.

Figure 3.33 compares the runtime of piISP-IC, iISP-IC, ISP-IC and POST-EISP-Miner for BMSWebView2 dataset with various settings. Figure 3.33(A) fixes *maxspan* = 1, Figure 3.33(B) fixes *maxspan* = 2, Figure 3.33(C) fixes *maxspan* = 3, Figure 3.33(D) fixes *maxspan* = 4 and Figure 3.33(E) fixes *maxspan* = 5, piISP-IC is always the best algorithm for mining inter-sequence patterns with item constraints for BMSWebView2 dataset. Especially, with the smaller *threshold*, the time gaps between the runtime of piISP-IC and those of iISP-IC, ISP-IC and POST-EISP-Miner are larger. In Figure 3.33(F), we fix *threshold* = 1 and change *maxspan* from 1 to 5, the runtime of piISP-IC is less than those of iISP-IC, ISP-IC and POST-EISP-Miner. Therefore, piISP-IC outperform iISP-IC, ISP-IC and POST-EISP-Miner for this dataset. In short, through the above experiments, we can conclude that piISP-IC outperform iISP-IC, ISP-IC and POST-EISP-Miner for mining inter-sequence patterns with item constraints.

3.4.9.2. Memory usage



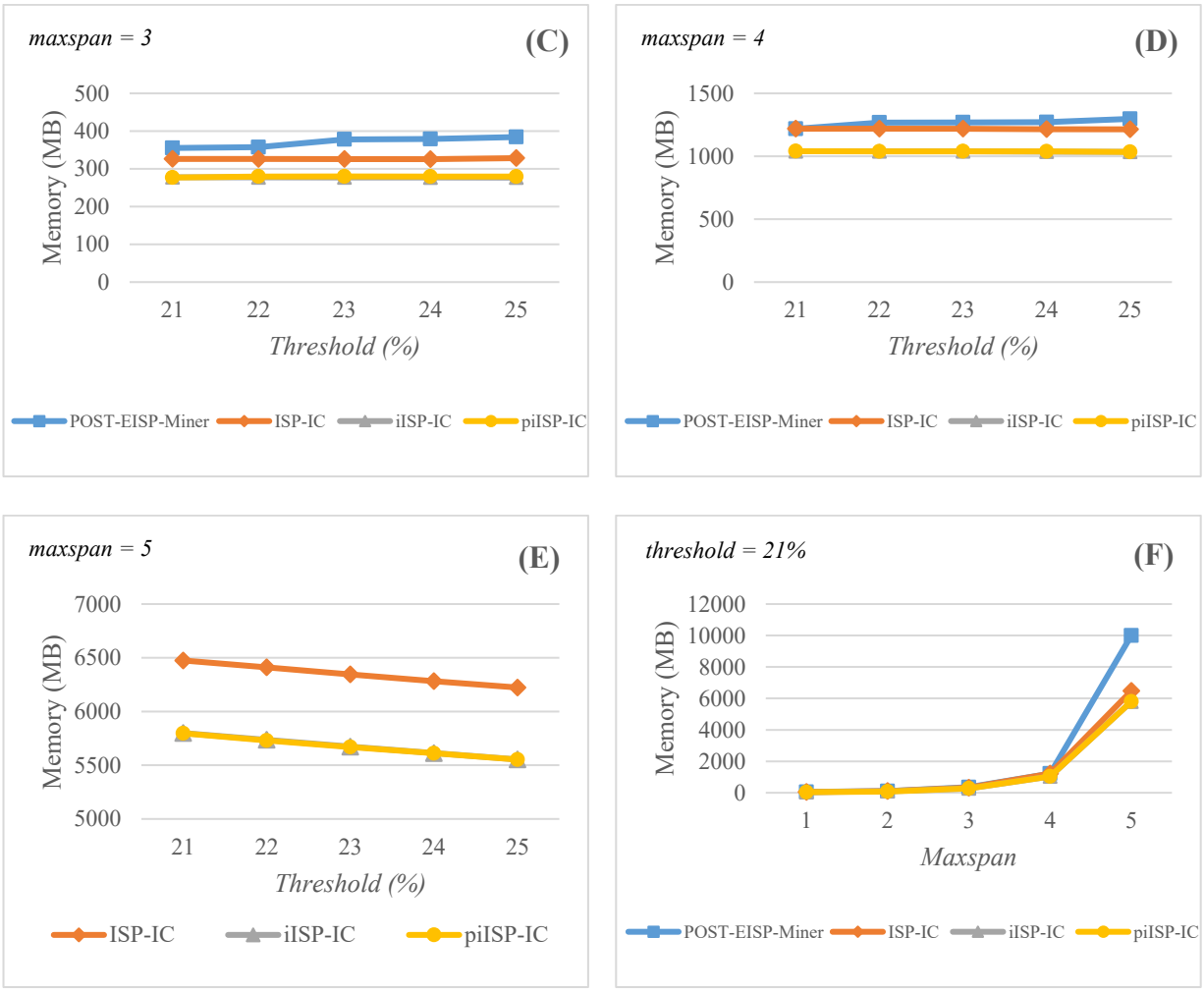


Figure 3.34. Memory usage of piISP-IC, iISP-IC, ISP-IC and POST-EISP-Miner for C6T5S4I4N1KD10K database with 15 random item constraints and (A) maxspan = 1; (B) maxspan = 2; (C) maxspan = 3; (D) maxspan = 4; (E) maxspan = 5; (F) threshold = 21%.

Figure 3.34 reports the memory usage of *pi*ISP-IC, *i*ISP-IC, ISP-IC and POST-EISP-Miner for C6T5S4I4N1KD10K database with various setting. Figure 3.34(A) fixes *maxspan* = 1, Figure 3.34(B) fixes *maxspan* = 2, Figure 3.34(C) fixes *maxspan* = 3, Figure 3.34(D) fixes *maxspan* = 4 and Figure 3.34(E) fixes *maxspan* = 5, we found that the memory usage of POST-EISP-Miner is the largest. However, the gap between the memory usage of *pi*ISP-IC and those of *i*ISP-IC and ISP-IC are insignificant for C6T5S4I4N1KD10K database. Then, Figure 3.34(F), which shows the memory usage of *pi*ISP-IC, *i*ISP-IC, ISP-IC and POST-EISP-Miner when we fix *threshold* = *21* and change *maxspan* from 1 to 5, confirm that statement again. POST-EISP-Miner cannot perform with *maxspan* = 5 and *threshold* = 21, the memory value with *maxspan* = 5 in Figure 3.34(F) is an example.

Figure 3.35 - Figure 3.36 reports the memory usage of *pi*ISP-IC, *i*ISP-IC, ISP-IC and POST-EISP-Miner for T10I4D100K and BMSWebView2 databases. The results show that the memory usages of these algorithms are nearly the same. Summary, although the memory usage of piISP-IC is largest (compared to those of iISP-IC, ISP-IC and POST-EISP-Miner), the gap between the memory usage of piISP-IC and those of iISP-IC, ISP-IC and POST-EISP-Miner are insignificant.

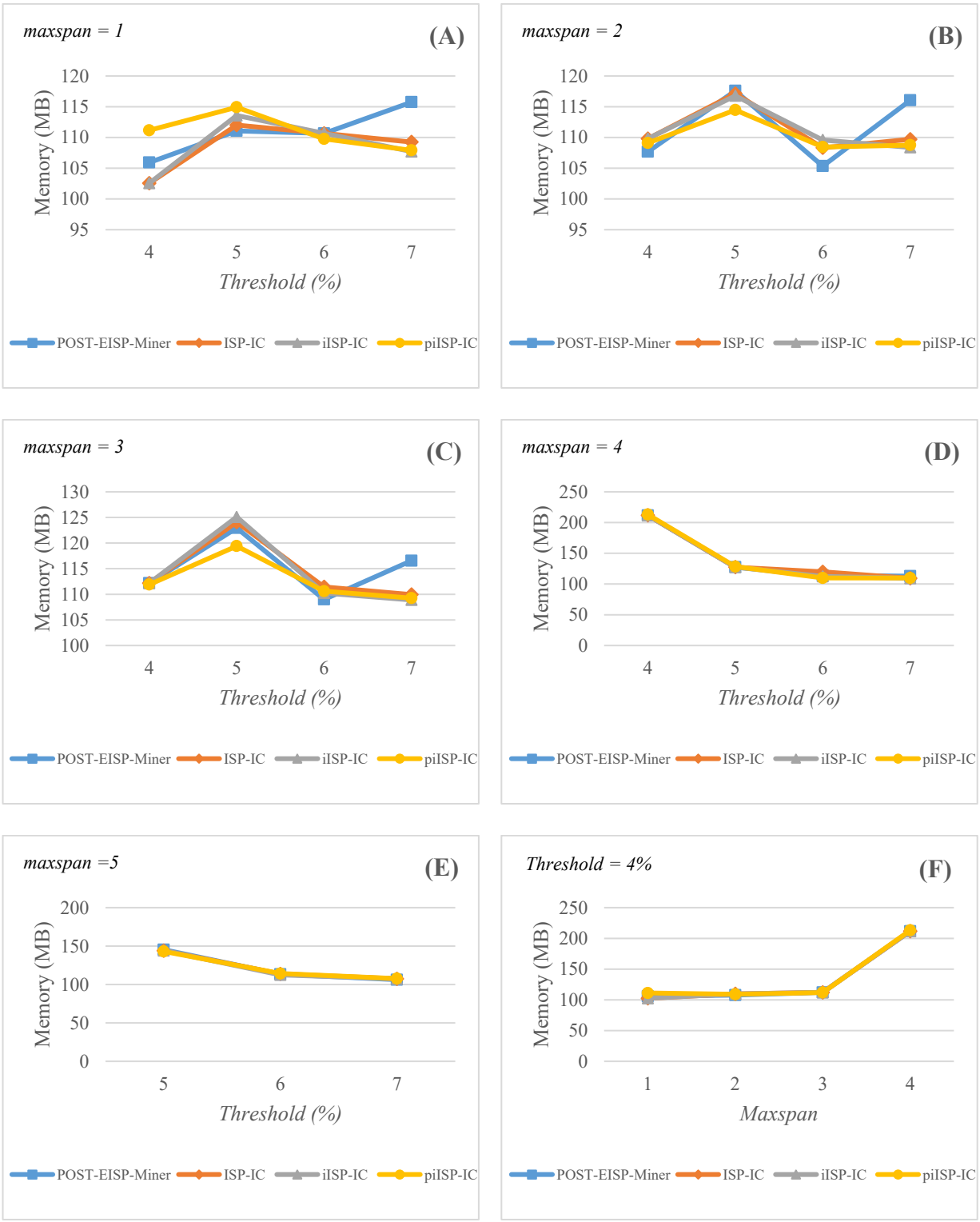


Figure 3.35. Memory usage of piISP-IC, iISP-IC, ISP-IC and POST-EISP-Miner for T10I4D100K database with 15 random item constraints and (A) maxspan = 1; (B) maxspan = 2; (C) maxspan = 3; (D) maxspan = 4; (E) maxspan = 5; (F) threshold = 4%.

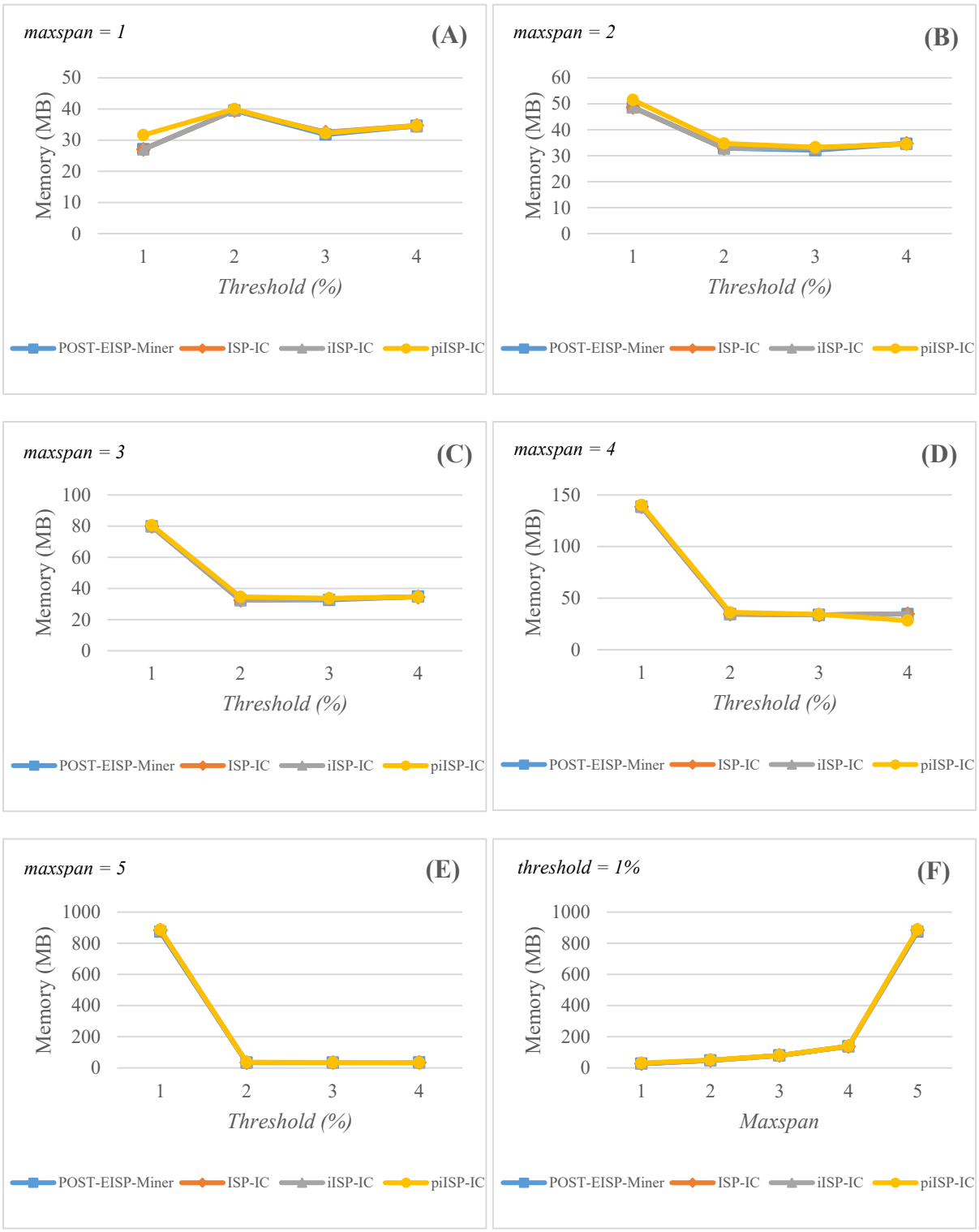


Figure 3.36. Memory usage of piISP-IC, iISP-IC, ISP-IC and POST-EISP-Miner for BMSWebView2 dataset with 15 random item constraints and (A) maxspan = 1; (B) maxspan = 2; (C) maxspan = 3; (D) maxspan = 4; (E) maxspan = 5; (F) threshold = 1%.

3.5. Summary

In this chapter, we proposed parallel methods for mining sequential pattern based on multi-core processors architecture, includes:

- Parallel Independent Branch PRISM: This section proposed a parallel method for mining frequent sequential pattern used prime theory for a fast determining support count of sequence pattern. Based on this theory, an efficient algorithm for mining sequential patterns (PIB-PRISM algorithm) was proposed.

- Parallel Dynamic Bit Vector Sequential Pattern Mining: This section proposed parallel efficient methods for mining used dynamic bit vector data structure (pDBV-SPM) for reducing computational cost and being speedup mining time. These methods also solved the load balancing workload among processors.

- Parallel mining inter-sequence patterns with item constraints: This section proposed three novel methods for mining inter-sequence patterns with item constraints. Firstly, a theorem was developed for fast determining whether an inter-sequence pattern satisfies item constraints. This theorem reduces mining time. Based on this theorem, an efficient algorithm for mining inter-sequence patterns with constraints (ISP-IC algorithm) was proposed. Secondly, a lemma was presented for improving the strategy of ISP-IC. Based on this lemma, the *i*ISP-IC algorithm was proposed. Thirdly, we presented a parallel version of *i*ISP-IC named *pi*ISP-IC to improve the performance.

Chapter 4

Parallel mining frequent closed sequential patterns

In this chapter, we propose a parallel approach called Parallel Dynamic Bit Vector Frequent Closed Sequential Patterns (pDBV-FCSP) for mining frequent closed sequential pattern (FCSP) from large databases. The pDBV-FCSP performs closure checking of prefix sequences early to reduce execution time for mining frequent closed sequential patterns. In addition, this approach also solves the load balance issues of the workload between processors by a dynamic mechanism that re-distributes the work when some processes are out of work to minimize the idle CPU time.

4.1. Introduction

Mining sequential pattern generates an exponential number of patterns when the database contains long sequences which need high computational cost in both time and space. Therefore, instead of mining the complete set of sequential patterns, it is better to mine closed sequential patterns only. A closed sequential pattern is a sequential pattern having no super sequence with the same support. Unlike the general mining of frequent sequential patterns, the mining of frequent closed sequential patterns has not been extensively studied. Although some algorithms have been proposed, such as CloSpan [117], BIDE [109, 111], ClaSP [36] and CloFS-DBV [101], their performance is not good for processing on large databases. CloSpan works under candidate maintenance-and-test paradigm, hence it is not scalable. BIDE follows a strict depth-first search in order to generate the closed sequential patterns, it does not need to keep track of any single historical frequent closed sequential (or candidate) patterns for a new pattern's closure checking. ClaSP uses a vertical database format strategy and a heuristic to prune non-closed sequences. CloFS-DBV uses a vertical data format and dynamic bit vector (DBV) structure combined with location information in the structure of the transaction CloFS-DBVPattern to mine frequent closed sequences. However, the mining time of these algorithms is still high because they have to explore a huge search space, moreover, they have very expensive computationally. The computational cost increases with increasing complexity of the patterns to be mined, especially for long sequence or dense databases.

Parallel processing has been widely applied to improve processing speed for various problems. Although there have been many parallel methods based on distributed system or shared memory system for mining frequent itemsets [122], frequent sequential patterns [49], frequent closed sequential patterns [25, 128], sub-graph mining [100]. But there are no methods for mining frequent closed sequential pattern on multi-core processors architecture.

Multi-core systems have placed more pressure on system programmers as well as application developers to make efficient use of the multiple computing cores. These challenges include determining how to divide applications into separate tasks and minimizes CPU idle time. These tasks must be balanced such that each task is doing an equal amount of work. Just as tasks must be separated, data must also be divided so that it can be accessed by the tasks running on separate cores.

In this chapter, we propose an algorithm, called pDBV-FCSP (Parallel Dynamic Bit Vector Frequent Closed Sequential Patterns), for mining frequent closed sequential pattern. The main contributions are as follows:

1. First parallel mining method for closed sequential patterns using multi-core processors architecture.
2. Partitioning the work into multiple independent tasks so that the overhead of inter-processor communication is minimized.
3. Load balance of the workload between processors using a dynamic mechanism that re-distributes the work when some processes are out of work.
4. Perform closure checking of prefix sequences early to prune infrequent and non-closed sequences.

4.2. Parallel mining frequent closed sequential patterns

This section describes the proposed pDBV-FCSP algorithm for mining frequent closed sequential patterns based on multi-core processors architecture computer that utilizes task parallel along with dynamic load balancing and uses a dynamic bit vector (DBV) data structure was presented in Chapter 2 to reduce cost.

In the parallel mining, each branch of the search tree can be regarded as a single task, the computation at each node becomes an independent task and the overall computation can be parallelized by distributing these tasks among the available processors which can be processed independently to generate FCSPs. The task parallel formulation distributes the tasks among the processors in the following way. First, the tree is expanded using the data-parallel algorithm at level $k + 1$, with $k > 0$. Then, the different nodes at level k are distributed among the processors. Once this initial distribution is done, each processor proceeds to generate the subtrees underneath the nodes to which they have been assigned.

Each node X at level k in the tree can be extended via sequence extension or itemset extension by adding one item to get a child node X at level k+1. A new node XY is created by joining nodes X and Y, which must have the same length and the same $|X|$ - 1 prefix item. Figure 4.1 shows frequent closed pattern candidates (with fill color) for *SDB* in Table 2.2 obtained using the pDBV-FCSP algorithm.

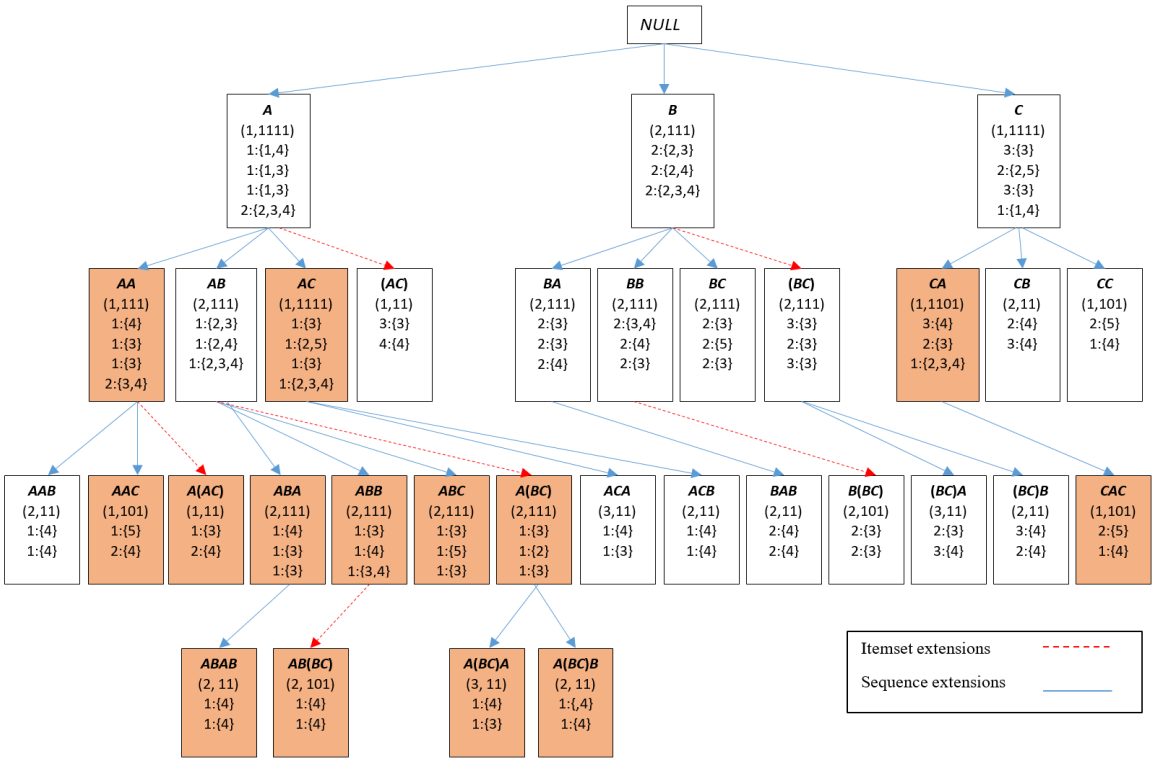


Figure 4.1. Frequent closed pattern candidates in Table 2.2 with minsup = 50%.

One of the primary challenges in parallel mining is that the candidate generation tree is usually not balanced. This skewness (one subtree is very deep compared to the others) of the tree can cause the performance of the parallel algorithm is not good. Therefore, dynamically redistributing the works when some processes are out of work is necessary.

The proposed pDBV-FCSP algorithm uses the DBV data structure and it uses a parallel form of the depth first search that relies on dynamic load balancing, where each process can perform a DFS on the subtree, since they are computed independently. In addition, a parallel implementation of tasks instead of threads because each task is assigned to searching a branch of the tree and is processed independently. Using tasks has advantages over using threads. Firstly, tasks require less memory than do threads. Secondly, a thread runs on only one core, whereas a task can run on multiple cores. Finally, threads require more processing time than do tasks because the operating system needs to allocate resources for threads, which require initialization and destruction and must perform context switching among threads.

When a process finishes finding all frequent patterns in its corresponding part of the tree, it actively requests an unexplored part of the tree from other processes. In this scheme, as a processor becomes idle, it randomly selects a donor processor and sends it a work request. The donor sends a response indicating whether or not it has additional work. If a response indicates that a donor does not have any more work, the processor selects another donor and sends work request to that donor. Otherwise, the processor receives nodes to expand, along with the portion of the database associated with those nodes. Upon receiving new works the processor starts extending newly received nodes. This process continues until every processor completely extended the nodes assigned to it. Dijkstra's token termination

algorithm [28] can be used to detect whether all processors have become idle. The work in our case is the DFS code prefixes of the patterns stored at each level of the candidate generation tree. We maintain a DFS prefix queue for each level of the tree. The queue is populated in the order of the DFS code extensions. The DFS code of a candidate pattern can be constructed by concatenating the current expanded prefixes from level 1 to the current level.

The main steps of the pDBV-FCSP algorithm are as follows.

1. Convert the sequence database to the DBV-Pattern structure.
2. Identify the frequent 1-sequences.
3. Project the database along each frequent 1-sequences and check the closure of frequent sequences to early eliminate infrequent sequences.
4. Prune the prefix sequences early.
5. Extend sequences.

Since pDBV-FCSP uses the DBV-Pattern structure, it can check the backward and forward extensions quickly. For each transaction, pDBV-FCSP considers only the start position or the last position of the sequence. Therefore, if the database has N transactions, pDBV-FCSP requires only N operations to check each candidate.

The mining of FCSPs from a tree is divided into separate branches as follows.

1. Construct the search tree from the set of frequent 1-sequences.
2. Mine FCSPs parallel, assign each branch of the tree to a task. The mining process is performed independently on each branch, no inter-processor communication is needed during the local mining.
3. Synchronize results. This is done only at the end of the mining pattern on each branch.

The pseudo code of the pDBV-FCSP strategy is shown in Figure 4.2. The value of *root* is initialized to *NULL*. The **Generate1SPs** procedure is called to identify all frequent 1-sequences from *SDB* (line 3). These sequences are stored in L_1. The items in L_1 are sorted in ascending order by support (line 4) to reduce the steps in the extension phase of the itemset and gain more effectively balance of the work load in the system. Next, pDBV-FCSP creates new tasks corresponding to each pattern in L_1 (line 7) where each core p_i receives $M = I/p$ tasks, where p is the number of CPU cores and I is the number of frequent 1-sequences (equal to the number of tasks). Each task executes the procedure **FCSP-Extension** (line 8) to extend itemsets and sequences. Tasks are configured to run in parallel (i.e., independently) to generate a partial set of FCSPs. The final set of FCSPs is the union of the partial results. The procedure of **Generate1SPs** is shown in Figure 4.3.

Procedure **pDBV-FCSP(***SDB*, *minsup***)**
Input *SDB*, *minsup*
Output: All FCSPs that satisfy *minsup*

```
1   Begin
2       Let root = NULL
3       L1= Generate1SPs(SDB, minsup)
4       Sort (L1) in increasing order of support
5       Add L1 to child node of root
6       For (each node i in root) do
7           Create new task ti
8           Call FCSP-Extension(i, minsup)
9   End begin
```

Figure 4.2. pDBV-FCSP strategy

Initially, the **Generate1SPs** procedure finds frequent 1-sequences (L_1) based on the DBV structure for each item that satisfies the *minsup* threshold (line 4). Consider *SDB* in Table 2.2 and *minsup* = 50%. After this procedure is executed, three frequent 1-sequences are stored and thus L_1 = {⟨*A*⟩:4, ⟨*B*⟩: 3, ⟨*C*⟩:4}, as shown in Table 4.1.

Procedure **Generate1SPs**(*SDB, minsup*)
Input: *SDB*, *minsup*
Output: Set of frequent 1-sequences

```
1  Begin
2   For (each t in T ) do
3       if (σ(t) ≥ minsup) then
4           L1 = L1 ∪ t
5   End for
6  End
```

Figure 4.3. Generate1SPs procedure.

The procedure **FCSP-Extension** is shown in Figure 4.4. This procedure expands the search tree by executing procedures **extendItemset** (line 6) and **extendSequence** (line 9). For each node in the search tree, the pattern for that node is extended by calling **extendItemset** and **extendSequence** to create a new pattern. Before sequence extension, the algorithm tests and eliminates prefixes that cannot extend frequent closed sequences using *Proposition 2.1* (line 4). This process is repeated (line 13) until no frequent closed sequences are generated. Lines 16 - 22 uses *Proposition 2.2* to check the prefix S_x. If S_x is not a frequent closed sequence, it will be set to NULL. After each level expansion the processors communicate between each other to determine whether the work needs to be re-balanced (line 23).

The description of dynamic load balancing is stated as the following. If a work request comes in the early stages of a counting phase, an active processor discards work done so far and shares its workload with the recipient. If a work request comes in the middle stages of the counting phase, an active processor lets the receiver know that while it has work, the work is not available immediately. Along with this message, the active processor also sends an estimated time before completion as well as an estimated

relative amount of workload at the next level. Towards the end of the counting phase, an active processor ignores the work requests. The requests received during that time are processed upon completion of the counting phase at that level.

If an idle processor does not receive work from a potential donor, it chooses another processor to inquire about work. If after polling all available processors, an idle processor did not get any work, this processor picks an active processor with work and lets it know that it will wait for work until it gets it. To ensure that idle processors will pick different active processors, an idle processor will pick top n active processors and will randomly choose one to attach itself to.

The results of executing sequence extension and itemset extension for *SDB* in Table 2.2 are shown in Table 4.2 and Table 4.3.

Consider a pattern with prefix $\langle A \rangle$, prefix $\langle A \rangle$ is not a closed sequence after the backward-extension process and prefix $\langle B \rangle$ can be pruned after the pruning prefix process. The algorithm performs sequence extension to create new frequent closed 2-sequences, pattern with prefix $\langle A \rangle$ extends using **extendSequence** by concatenation with $\langle A \rangle$, $\langle B \rangle$ and $\langle C \rangle$ to make new patterns $\langle AA \rangle$, $\langle AB \rangle$ and $\langle AC \rangle$, respectively, as shown in Table 4.2 and extends using **extendItemset** by concatenation with $\langle B \rangle$ and $\langle C \rangle$ to make new patterns $\langle (AB) \rangle$ and $\langle (AC) \rangle$, respectively, as shown in Table 4.3. It is then checked whether the support count of these patterns satisfies the threshold based on the DBV-Pattern structure or not. This process is repeated for each task until all FCSPs are obtained, as shown in Table 4.4.

```
    Procedure FCSP-Extension(p, minsup)
1   Begin
2     Let list= child nodes of p
3     For each S_x in list do
4       If (S_x is not pruned) then
5           For each S_y in list do
6               If (σ(S_xy = extendItemset(S_x, S_y)) ≥ minsup) then
7                   Add S_xy to child nodes of S_x
8               End if
9               If (σ(S_xy = extendSequence(S_x,S_y)) ≥ minsup) then
10                  Add S_xy to child node of S_x
11              End if
12          End for
13          Call FCSP-Extension(S_x, minsup)
14      End if
15      If (checkBackwardExtension(S_x,S_y) = true) then
16          S_x.isClosed = false
17      Else (checkForwardExtension(S_x,S_y) = true) then
18          S_x.isClosed = false
19      End if
20      If (S_x.isClosed = false) then
```

```
21              Sx = NULL
22            End if
23            Load_balancing()
24          End for
25        End begin
```

Figure 4.4. FCSP-Extension procedure.

Table 4.1. Sequences A, B and C in SDB after conversion to DBV

Sequence	⟨***A***⟩				⟨***B***⟩			⟨***C***⟩			
Start bit	1				2			1			
Value	15				7			15			
Index	4	3	2	1	4	3	2	4	3	2	1
Position	1:{1,4}	1:{1,3}	1:{1,3}	2:{2,3,4}	2:{2,3}	2:{2,4}	2:{2,3,4}	3:{3}	2:{2,5}	3:{3}	1:{1,4}

Table 4.2. Sequence extension (minsup = 50%)

Sequence	⟨***AA***⟩			
Start bit	1			
Value	15			
Index	4	3	2	1
Position *A*	1:{1,4}	1:{1,3}	1:{1,3}	2:{2,3,4}
Position *A*	1:{1,4}	1:{1,3}	1:{1,3}	2:{2,3,4}
Position *AA*	1:{4}	1:{3}	1:{3}	2:{3,4}
Sequence	⟨***AB***⟩			
Start bit	2			
Value	7			
Index	4	3	2	1
Position *A*	1:{1,4}	1:{1,3}	1:{1,3}	2:{2,3,4}
Position *B*	2:{2,3}	2:{2,4}	2:{2,3,4}	∅
Position *AB*	1:{2,3}	1:{2,4}	1:{2,3,4}	∅
Sequence	⟨***AC***⟩			
Start bit	1			
Value	15			
Index	4	3	2	1
Position *A*	1:{1,4}	1:{1,3}	1:{1,3}	2:{2,3,4}
Position *C*	3:{3}	2:{2,5}	3:{3}	1:{1,4}
Position *AC*	1:{3}	1:{2,5}	1:{3}	2:{4}

Table 4.3. Itemset extension (minsup = 50%)

Sequence	⟨(***AB***)⟩
Start bit	2
Value	1

Index	4	3	2	1
Position *A*	1:{1,4}	1:{1,3}	1:{1,3}	2:{2,3,4}
Position *B*	2:{2,3}	2:{2,4}	2:{2,3,4}	∅
Position (*AB)*	∅	∅	3:{3}	∅
Sequence	⟨(***AC***)⟩			
Start bit	1			
Value	3			
Index	4	3	2	1
Position *A*	1:{1,4}	1:{1,3}	1:{1,3}	2:{2,3,4}
Position *C*	3:{3}	2:{2,5}	3:{3}	1:{1,4}
Position (*AC)*	∅	∅	3:{3}	4:{4}

Table 4.4. Set of frequent closed sequential patterns from Table 2.2 with minsup = 50%

No.	Item	FCSPs – Support
1	⟨*A*⟩	⟨*AA*⟩:4, ⟨*AC*⟩:4, ⟨*AAC*⟩:2, ⟨*A*(*AC*)⟩:2, ⟨*ABA*⟩:3, ⟨*ABB*⟩:3, ⟨*ABC*⟩:3, ⟨*A*(*BC*)⟩:3, ⟨*ABAB*⟩:2, ⟨*AB*(*BC*)⟩: 2, ⟨*A*(*BC*)*A*⟩:2, ⟨*A*(*BC*)*B*⟩:2
2	⟨*B*⟩	∅
3	⟨*C*⟩	⟨*CA*⟩:3, ⟨*CAC*⟩:2

4.3. Experimental results

Experiments were conducted to evaluate the proposed parallel algorithm. The experiments were performed on a personal computer with an Intel Core i5-6200U 2.3-GHz CPU with 4 cores, 3 MB of L3 cache, 4 GB of RAM and running 64-bit Windows 10 Pro. The algorithm was implemented using .Net Framework 4.5.

The first and second databases (C6T5N1kD10k and T10I4D100K) used for comparison were generated using the IBM synthetic data generator. The third database (Kosarak25k) was provided by Bodon (http://fimi.ua.ac.be/data/). Others databases were provided by Fournier-Viger (http://www.philippe-fournier-viger.com/spmf/). Information of the databases is shown in Table 4.5. The definitions of parameters used to generate the databases are shown in Table 2.4.

Table 4.5. Databases used in experiments.

Database	#sequences	#items	
C6T5N1kD10k	10,000	1,000	Synthetic databases
T10I4D100K	100,000	870	
Kosarak25k	25,000	41,270	Click stream data of a Hungarian online news portal
BMSWebView1	59,601	497	Click stream data from an e-commerce site
BMSWebView2	77,512	3,340	Click stream data from the KDD-CUP 2000
MSNBC (Short)	31,790	17	Click stream data from the UCI repository
MSNBC (Full)	989,818	17	

Parallel mining improved performance without affected the results. Table 4.6 shown the mining results of CloFS-DBV and pDBV-FCSP. The set of results of algorithms was always the same for all databases to prove the correctness of pDBV-FCSP.

We also add a set of results of frequent sequential pattern (FSP) mining for comparison. With these thresholds, the result is similar for all databases among frequent closed sequential pattern (FCSP) and frequent sequential patterns. We just saw that the results are different between FCSP and FSP on the Kosarak25k database and on two databases BMSWebView1 and BMSWebView2 with the lowest threshold

Table 4.6. The mining results of pDBV-FCSP and CloFS-DBV

C6T5S4I4N1kD10k							
minsup (%)	6	5	4	3	2	1	
#FSP	83	117	188	281	418	863	
#FCSP	83	117	188	281	418	863	
T10I4D100K							
minsup (%)	5.0	4.5	4.0	3.5	3.0	2.5	2.0
#FSP	10	17	26	40	60	107	155
#FCSP	10	17	26	40	60	107	155

BMSWebView1						**BMSWebView2**					
minsup (%)	0.5	0.4	0.3	0.2	0.1	*minsup* (%)	0.5	0.4	0.3	0.2	0.1
#FSP	201	286	435	798	3991	**#FSP**	408	676	1340	3683	23294
#FCSP	201	286	435	798	3974	**#FCSP**	408	676	1340	3683	22245
Kosarak25k						**MSNBC (Short)**					
minsup (%)	0.6	0.5	0.4	0.3	0.2	*minsup* (%)	7	6	5	4	3
#FSP	1148	1668	2694	5167	15679	**#FSP**	721	987	1478	2352	4118
#FCSP	1026	1458	2248	3972	9005	**#FCSP**	721	987	1478	2352	4118

Some existing methods for parallel mining closed sequential patterns are designed on a distributed memory system and are performed from 4, 8, 16, 32 and 64 nodes as Par-CSP [25] and Par-ClosP [128], each node has a 1GHz Pentium III processor, 1GB main memory. Our proposed method is the first method applied to multi-core processors for mining closed sequential pattern. The newest parallel algorithm applied to multi-core processors to publish in Applied Intelligence is pDBV-FCSP [49] only implemented for sequential pattern. So it is very difficult to compare our results with that method because they are in different fields. Experiments were conducted to compare CloFS-DBV and the proposed pDBV-FCSP for various *minsup* values and the proposed method running on two figures are two and four cores. The runtime results are shown in Figure 4.5-Figure 4.10 and memory usage results are shown in Figure 4.11-Figure 4.16 for the six regarding databases.

4.3.1. Runtime

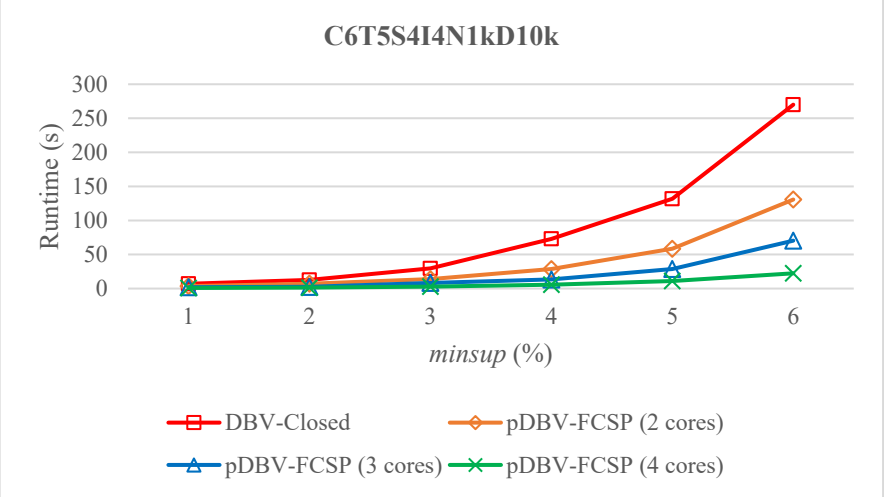


Figure 4.5. Runtimes on C6T5N1kD10k

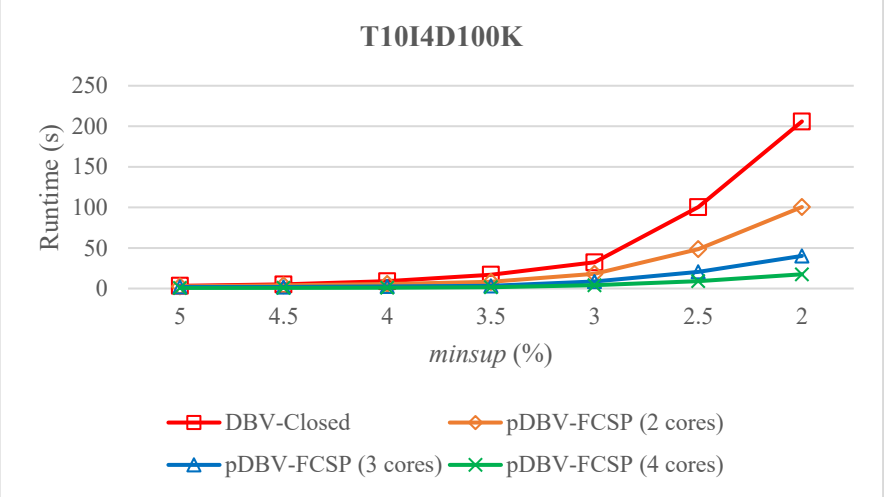


Figure 4.6. Runtimes on T10I4D100K

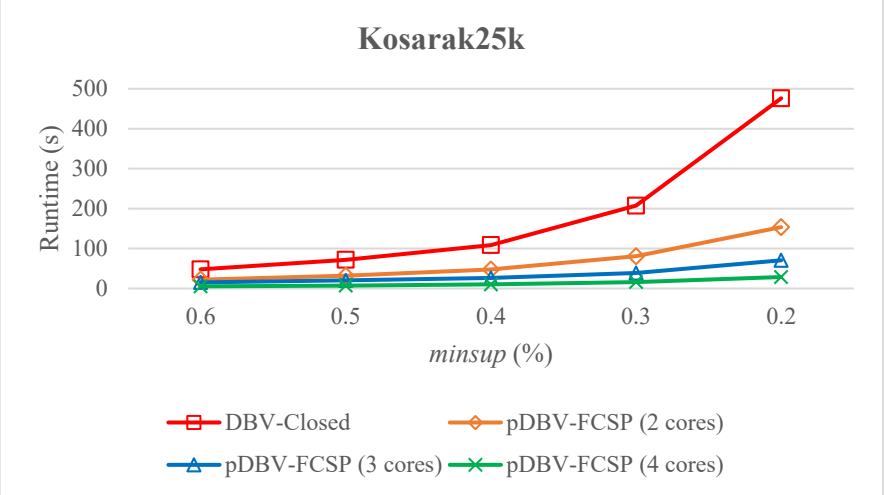


Figure 4.7. Runtimes on Kosarak25k

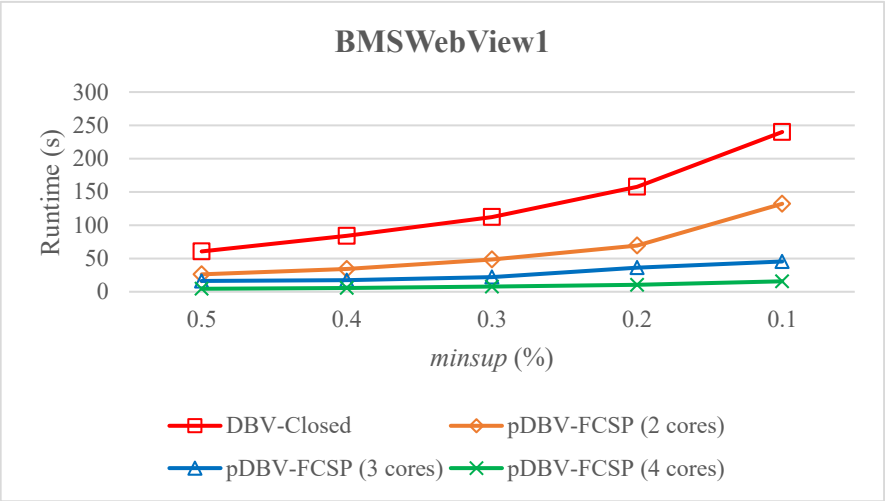


Figure 4.8. Runtimes on BMSWebView1

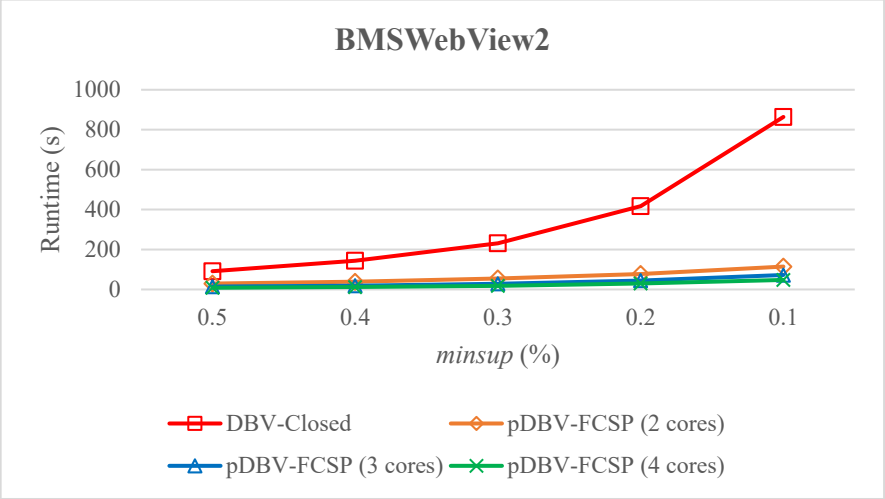


Figure 4.9. Runtimes on BMSWebView2

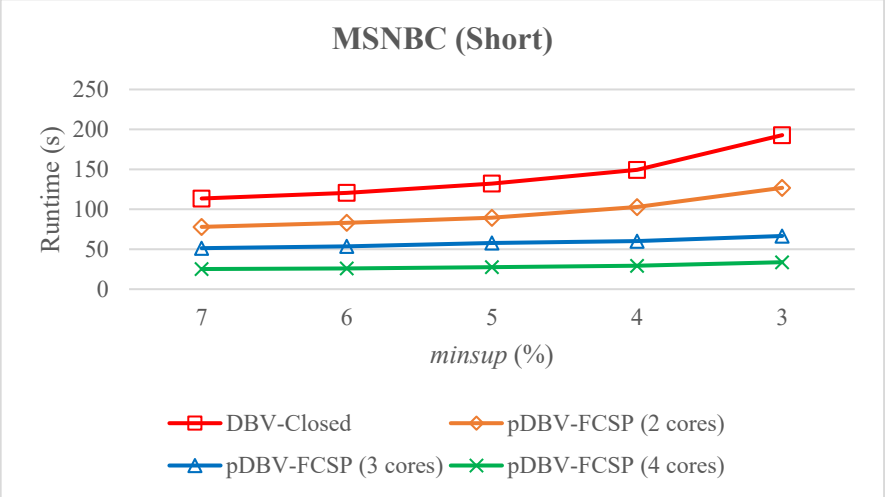


Figure 4.10. Runtimes on MSNBC_Short

The experimental results show that pDBV-FCSP is faster than CloFS-DBV in most cases, especially, executed on a computer with more cores. With a large *minsup*, pDBV-FCSP is not faster than CloFS-DBV, however, with a small *minsup*, pDBV-FCSP is much faster than CloFS-DBV. In Figure 4.5, for the C6T5S4I4N1kD10k database, the runtimes of CloFS-DBV and pDBV-FCSP were 7.06 s, 3.59 s (two cores) and 0.91 s (four cores) seconds, respectively.

The runtimes of pDBV-FCSP have been always better than the others when executed on four cores while CloFS-DBV increased rapidly when *minsup* was decreased from 6% to 1%. Similar results, the runtimes of pDBV-FCSP were slower than the CloFS-DBV as shown in Figure 4.6 for T10I4D100K database. In Figure 4.7-Figure 4.10, the runtimes of pDBV-FCSP for Kosarak25k, BMSWebView1, BMSWebView2 and MSNBC (short) databases were better than CloFS-DBV in most cases, especially with small *minsup* and running on a computer that has many cores. When *minsup* was decreased, the runtimes of pDBV-FCSP have been always faster.

4.3.2. Memory usage

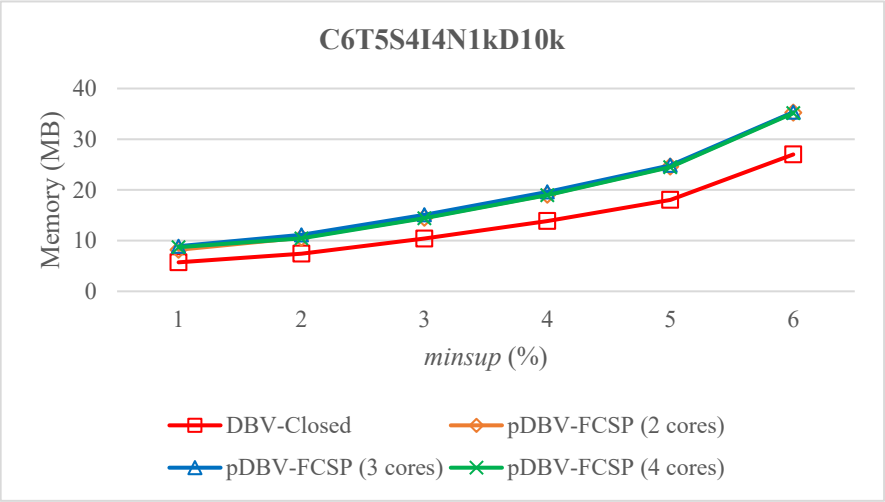


Figure 4.11. Memory usage for C6T5N1kD10k

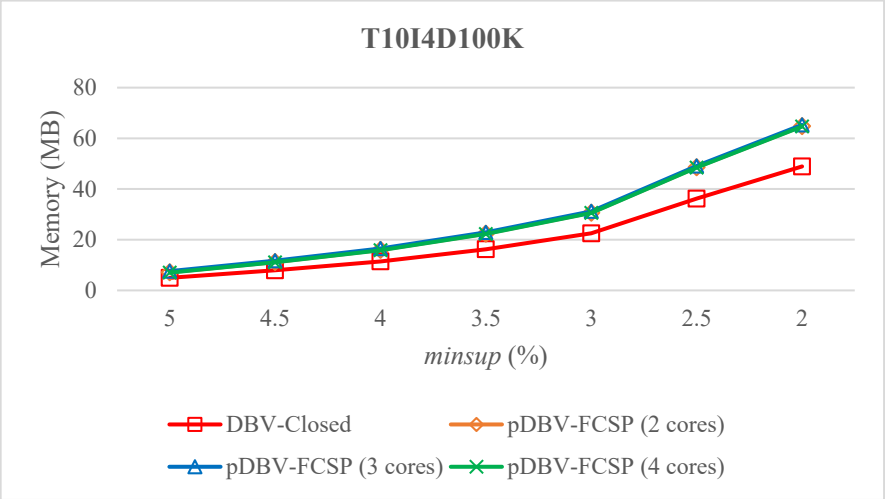


Figure 4.12. Memory usage for T10I4D100K

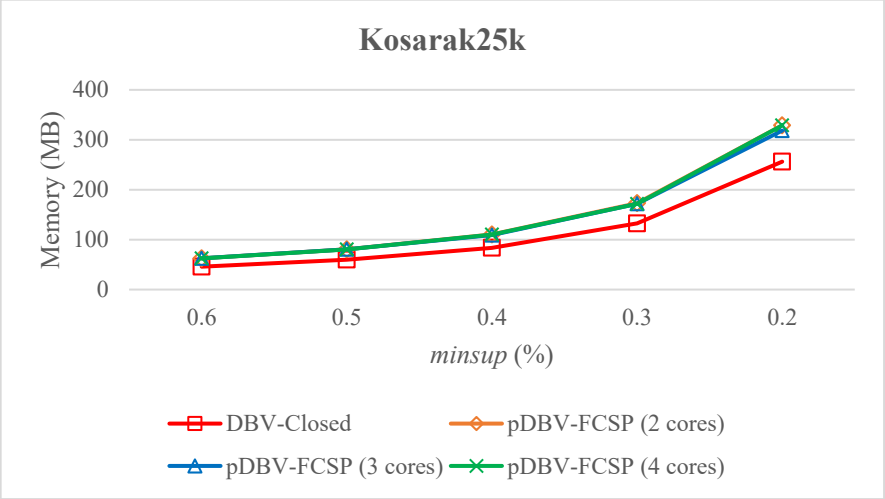


Figure 4.13. Memory usage for Kosarak25k

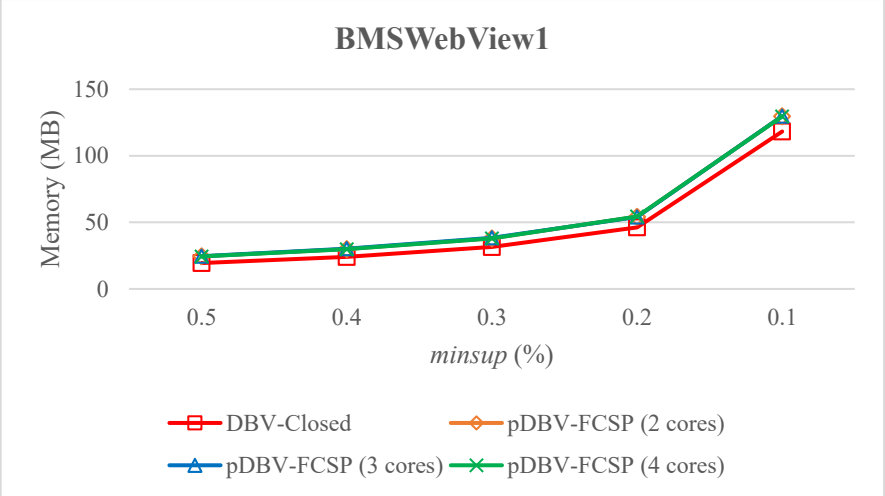


Figure 4.14. Memory usage for BMSWebView1

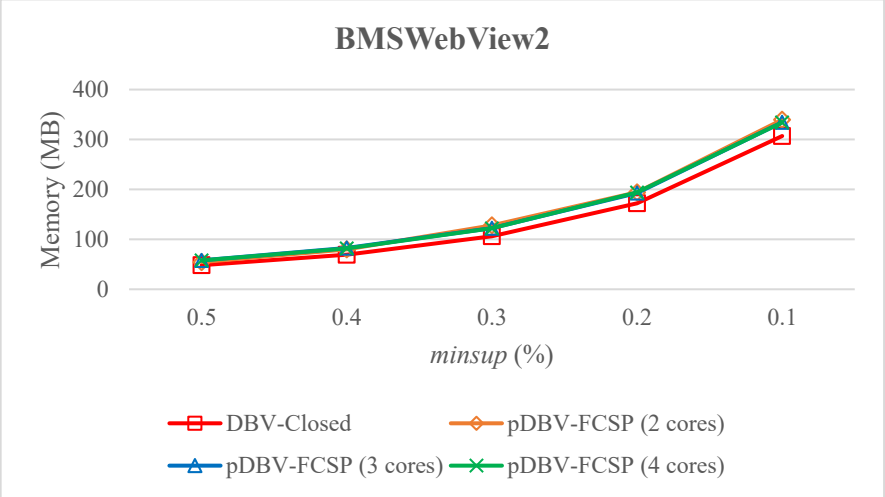


Figure 4.15. Memory usage for BMSWebView2

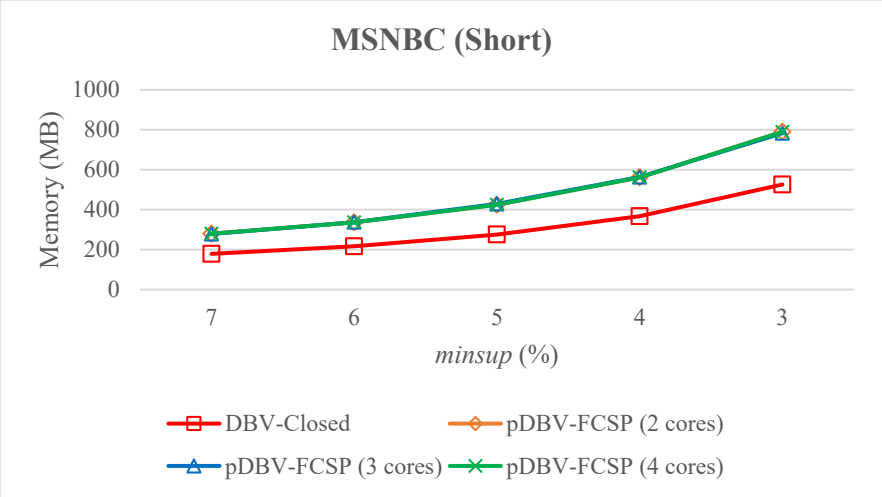


Figure 4.16. Memory usage for MSNBC (short)

On all databases, CloFS-DBV is always better than pDBV-FCSP in terms of memory usage, as shown in Figure 4.11 – Figure 4.16. This can be explained as follows. Although, both algorithms used the DBV data structure for storing sequence information. pDBV-FCSP using more memory usage because parallel processing divided the tasks to be processed into independent branches, needing more memory to store the results. When *minsup* was decreased, more FCSPs were obtained and thus the runtime and memory usage increased. The memory usage of pDBV-FCSP between two and four cores was equivalents because the numbers of nodes in the search tree of was the same.

An advantage of pDBV-FCSP is that it helps to balance the search tree using a dynamic mechanism that re-distributes the works when some processes are out of work to minimize the CPU idle time and sorting patterns according to their support values in ascending order. In pDBV-FCSP, the set of frequent 1-sequences are sorted before task assignment. Most nodes in the leftmost branches are infrequent and are pruned during the search. Nodes on the rightmost branches are frequent and not pruned during the search. This strategy balances the search tree for parallel processing and thus the search times for branches of the search tree are similar.

4.3.3. Scalability

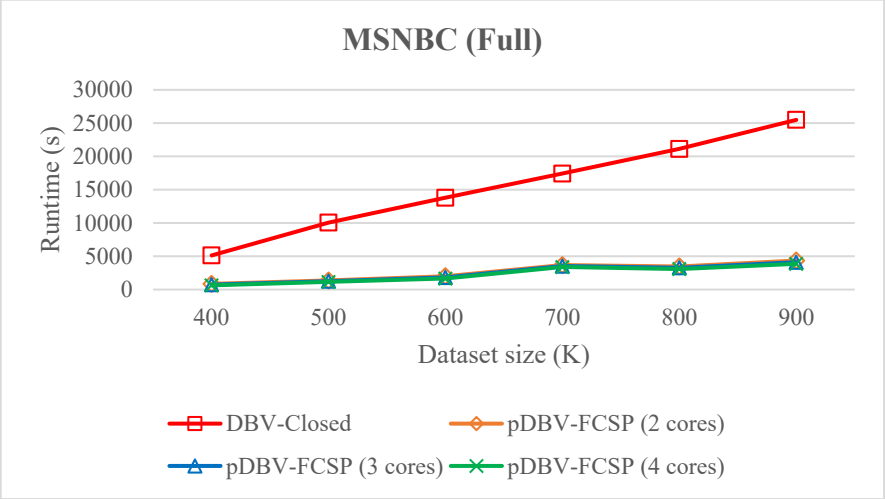


Figure 4.17. Scalability of pDBV-FCSP and CloFS-DBV for MSNBC database with minsup=7% and various database sizes

In this section, we performed scalability experiments on various numbers of sequences for MSNBC full database which is the largest one in the experimental databases. The goal of this experiment is to observe the influence of the number of sequences at execution time. The results in Figure 4.17 show that pDBV-FCSP had the best scalability. We tried this database with CM-ClaSP approach proposed by Philippe et al. [34], even the CMClaSP could not run for MSNBC database from 500K sequences.

4.4. Summary

This chapter proposed a strategy for mining FCSPs in parallel using a multi-core processors architecture. The proposed algorithm distributed SP search tasks on a multi-core processors with a dynamic load balancing and used an efficient data structure (DBV) for quickly mining FCSPs. The experimental results showed that the proposed algorithm outperformed the CloFS-DBV algorithm and the runtime reduced when the number of cores increased.

Chapter 5

Conclusions and Future works

In this chapter, we conclude the thesis by summarizing our contributions. Then, we highlight the ongoing works we are conducting to extend this thesis.

5.1. Conclusion

This thesis deals with sequential pattern mining from huge sequence databases, it 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 mine effectively, efficient parallel algorithms are necessary.

In the main part of this thesis, we are based on multi-core processors architecture and SPMD (Single Program Multiple Data) strategy to parallel sequential pattern mining. The database is partitioned into multiple continuous blocks and distributed tasks on a multi-core computer. According to this mechanism, a set of processors execute in parallel with the same algorithm on different partitions of a database.

Our approach has several main advantages over task parallel: i) tasks require less memory than threads do; ii) a thread runs on only one core, whereas a task can run on multiple cores; iii) threads require more processing time than tasks do because the operating system need to allocate data structures for threads, as initialization and destruction and also performs the context switch between threads. This simplifies programming and leads to a significant development time 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 parallel. 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 parallel.

Besides, the main goal of the algorithm is to reduce cost and increase effective processed. The load balanced workload among the processors and good scalability is interesting. We use strategy dynamic load balanced and sort the first candidates list (set of 1-sequences) to improve the main parallel algorithms. In other words, the improved algorithm is good behavior and scalability when length of patterns grows up while main parallel algorithm face to very high computation time.

The contributes of thesis can be summarized as follows:

Firstly, we propose a method named PIB-PRISM (Parallel Independent Branch PRISM) [R1]. This method applied prime theory to encode sequences, including ID encoding of sequence data and position encoding of items in each sequence. A sequential pattern appears in only some sequences of a database and in each sequence only occurs in some positions of a sequence. Therefore, a bit vector usually includes many bits of zero. To reduce storage space, PIB-PRISM retains only non-empty blocks after prime encoding. Hence, it keeps an index with each sequence block to indicate which non-empty position blocks correspond to a given sequence block, so it is more compressed and easily determined

the support value of each sequence. The primary advantages of the PIB-PRISM strategy is that each task is assigned to searching branches of the search tree and processed independently from other tasks without any synchronization.

Secondly, we propose a method called pDBV-SPM (Parallel Dynamic Bit Vector Sequential Pattern Mining) [R3] used dynamic bit vector (DBV) data structure. pDBV-SPM overcomes the drawbacks of PIB-PRISM and further reduces the computational cost of mining. The advantages of this method are: i) fast determined support and reduced storage space; ii) quick mining; iii) load balancing of the workload between processors.

The main disadvantage of the bit vector structure is its fixed size, which depends on the number of sequences in a sequence database. During sequence extension, '0' bits thus appear often, which increases the required memory and processing time. The DBV is a compact data structure, using DBV helps to reduce memory storage and computational cost in bitwise AND operator.

Thirdly, we propose a method called pDBV-FCSP (Parallel Dynamic Bit Vector Frequent Closed Sequential Patterns) [R4] also based on DBV data structure, for mining frequent closed sequential patterns with dynamic load balancing. pDBV-FCSP is the first parallel mining method for closed sequential patterns using multi-core processors architecture. This method performs closure checking of prefix sequences early to prune infrequent and non-closed sequences and it solves the load balance issues of the workload between processors with a dynamic mechanism that re-distributes the work when some processes are out of work to minimize the idle CPU time.

Next, we propose a parallel approach named MCM-SPADE [R6] uses the vertical data format and a data structure named CMAP (Co-occurrence MAP) for storing co-occurrence information. This algorithm performs early pruning of the candidates to reduce the search space and it partitions the related tasks to each processor core by using the divide-and-conquer property. The proposed algorithm also uses dynamic scheduling to avoid task idling and achieve load balancing between processor cores.

Finally, we propose three novel algorithms for fast mining inter-sequence patterns with item constraints [R5, R7]. First, the problem of mining inter-sequence patterns with item constraints is introduced. Then, we develop a theorem to reduce the time of determining whether a candidate sequence satisfies the constraints. The ISP-IC (Inter-Sequence Pattern with Item Constraint mining) algorithm based on the above theorem for mining inter-sequence patterns with item constraints is then proposed. Next, a lemma and the second algorithm, named iISP-IC, are proposed. Then, we present a parallel version of iISP-IC, named piISP-IC algorithm. Experiments are conducted to show the effectiveness of the piISP-IC in terms of mining time and memory usage compared to ISP-IC and iISP-IC algorithms.

We have implemented our parallel algorithm using .NET framework. In that, a new concept is introduced – Task. The task is part of TPL, which is a new collection of classes that focus on parallel programming as well as offer light weight object compared to traditional threading system. Using tasks has many benefits, not only are tasks more efficient, but they also abstract away from the underlying hardware and the OS specific thread scheduler. One other great feature about TPL is that it aims to use each core of CPU so no core will be idle. Our experimental results indicate that proposed algorithms have good performance, greatly reduce the storage and search space and a good work loading as well.

5.2. Future works

We are currently working on two major categories. In the first category, we are working on improving frequent subgraph mining on parallel environment. The second category is to research and apply hybrid parallel models, a combination of distributed environment and multi-core processors architecture to improve the scalability sequential pattern mining for biological sequences.

In the first category, we propose two efficient parallel algorithms for mining frequent subgraphs based on gSpan algorithm basic framework and multi-core processors architecture with good parallel and scale up [R2]. The first strategy called PMFS-IB, we distribute each branch of the DFS tree to a single task which mines assigned branch independently from other tasks. It explores all frequent subgraphs in the same way as gSpan [118] does, except that PMFS-IB mines each branch of the DFS tree in parallel. The second parallel strategy called PMFS-SB, assigns each branch of the DFS tree to multiple tasks.

The experimental results show that the proposed methods are superior to the sequential algorithm gSpan in terms of mining time on two databases, namely Chemical and Compound. However, when minimum support values are very low, the cost of context switching between tasks is still very high. The memory consumption is also high, which may cause the memory leakage.

In the future, we will continue researching this trend and scalable sequential pattern mining for biological sequences with hybrid parallel models, a combination of multi-core processors architecture and distributed environment. We also focus on distributed data mining because distributed computing plays an important role in the data mining process for several reasons. Firstly, data mining often requires huge amounts of resources in storage space and computation time. To make systems scalable, it is important to develop mechanisms that distribute the workload among several sites in a flexible way. Secondly, data are often inherently distributed into several databases, making a centralized processing of this data very inefficient and prone to security risks. Distributed data mining explores techniques of how to apply data mining in a non-centralized way.

5.3. Publications

The experimental results prove that our proposed methods are better for tackling the problems mining frequent sequential patterns and frequent closed sequential patterns, especially is on large and dense database. The proposed methods and their experimental results have been presented and published in high-quality international journals. The publications are listed as follows.

[R1]. Bao Huynh, Bay Vo (2015). *Using multi-core processors for mining frequent sequential patterns. ICIC* Express Letters, 9 (11), pp. 3071-3079 (Scopus)

[R2]. Bao Huynh, Dang Nguyen, Bay Vo (2016). *Parallel Frequent Subgraph Mining On Multi-Core Processor Systems. ICIC* Express Letters, 10 (9), pp. 2105-2113 (Scopus)

[R3]. Bao Huynh, Bay Vo, Vaclav Snasel (2017). *An Efficient Method for Mining Frequent Sequential Patterns Using Multi-Core Processors*. Applied Intelligence, 46(3), pp. 703-716 (SCI, IF = 1.904)

[R4]. Bao Huynh, Bay Vo, Vaclav Snasel (2017). *An Efficient Parallel Method for Mining Frequent Closed Sequential Patterns*. IEEE Access, 5(1), pp. 17392-17402 (SCIE, IF = 3.244)

[R5]. Tuong Le, Anh Nguyen, Bao Huynh, Bay Vo and Witold Pedrycz. *Mining Constrained Inter-Sequence Patterns: A Novel Approach to Cope with Item Constraints*. Applied Intelligence (accepted, SCI, IF = 1.904)

[R6]. Bao Huynh, Cuong Trinh, Huy Huynh, Trang Van, Bay Vo, Vaclav Snasel. *An Efficient Approach for Mining Sequential Patterns Using Multiple Threads on Very Large Databases*. Engineering Applications of Artificial Intelligence (submitted, SCIE, IF = 2.894)

[R7]. Thanh-Long Nguyen, Bay Vo, Bao Huynh, Vaclav Snasel, Loan T.T. Nguyen (2017). *Constraint-Based Method for Mining Colossal Patterns in High Dimensional Databases*. ISAT, 655(1), pp. 195-204 (AISC)

References

[1]. Abbass H., Sarker R., Newton C.S., *Data Mining: A Heuristic Approach*. Idea Group Publishing, pp. 261-289 (2002)

[2]. Agrawal R., Srikant R., *Mining Sequential Patterns*. ICDE'95, pp. 3-14 (1995)

[3]. Almasi S., Gottlieb A., *Highly parallel computing*. Benjamin-Cummings Publishing Co., Inc. Redwood City, CA, USA (2nd ed.). ISBN:0-8053-0443-6 (1994)

[4]. Akhter S., Roberts J., *Multi-Core Programming*. Intel Press (2006)

[5]. Altaf W., Shahbaz M., Guergachi A., *Applications of association rule mining in health informatics: a survey*. Artif. Intell. Rev., 47(3), pp. 313-340 (2017)

[6]. András V., *Multi-core and Many-core Processor Architectures*. Springer Science Business Media, LLC 2011, pp. 9-43 (2011). http://www.springer.com/gp/book/9781441997388

[7]. Ankerst M., Breunig M.M., Kriegel H.P., Sander J., *OPTICS: Ordering Points to Identify the Clustering Structure*. SIGMOD'99, 28(2), pp. 49-60 (1999)

[8]. Arora P., Deepali Dr., Varshney S., *Analysis of K-Means and K-Medoids Algorithm For Big Data*. Procedia Computer Science, 78, pp. 507-512 (2016)

[9]. Ayres J., Flannick J., Gehrke J., Yiu T., *Sequential pattern mining using a bitmap representation*. KDD'02, pp. 429-435 (2002)

[10]. Bartz-Beielstein T., Zaefferer M., *Model-based methods for continuous and discrete global optimization*. Applied Soft Computing, 55, pp. 154–167 (2017)

[11]. Beg A.H., Islam Z., *A novel genetic algorithm-based clustering technique and its suitability for knowledge discovery from a brain data set*. CEC, pp. 948-956 (2016)

[12]. Berkhin P., Dhillon I., *Knowledge Discovery: Clustering*. Encyclopedia of Complexity and Systems Science, pp. 5051-5064 (2009)

[13]. Bhuyan M.H., Bhattacharyya D.K., Kalita J.K., *Towards Generating Real-life Datasets for Network Intrusion Detection*. I. J. Network Security, 17(6), pp. 683-701 (2015)

[14]. Bischof C., Bucker M., Gibbon P., Joubert G., Lippert T., *Parallel Computing: Architectures, Algorithms and Applications*. Volume 15 Advances in Parallel Computing, IOS Press, ISBN-13: 978-1586037963 (2008)

[15]. Blumofe R.D., Joerg C.F., Kuszmaul B.C., Leiserson C.E., Randall K.H., Zhou Y., *Cilk: An Efficient Multithreaded Runtime System*. 37(1), pp. 55-69 (1996)

[16]. Boyd-Wickizer S., *Optimizing communication bottlenecks in multiprocessor operating system kernels*. Massachusetts Institute of Technology, Cambridge, MA, USA (2014)

[17]. Buckzak A.L., Baugher B., Guven E., Thomas L.R., Elbert Y., Babin S.M., Lewis S., *Fuzzy association rule mining and classification for the prediction of malaria in South Korea*. BMC Med. Inf. & Decision Making, 15:47 (2015)

[18]. Bureva V., Sotirova E., Chountas P., *Generalized Net of the Process of Sequential Pattern Mining by Generalized Sequential Pattern Algorithm* (*GSP*). IEEE Conf. on Intelligent Systems, 2, pp. 831-838 (2015)

[19]. Campos L.M.L., Oliveira R.C.L., Roisenberg M., *Network Intrusion Detection System Using Data Mining*. EANN, pp. 104-113 (2012)

[20]. Campbell C., Johnson R., Miller A., Toub S., *Parallel Programming with Microsoft®.NET: Design Patterns for Decomposition and Coordination on Multicore Architectures*. Microsoft Press (2010)

[21]. Casali A., Ernst C., *Extracting Correlated Patterns on Multi-core Architectures*. CD-ARES'13, pp. 118-133 (2013)

[22]. Chai L., Gao Q., Panda D.K., *Understanding the Impact of Multi-Core Architecture in Cluster Computing: A Case Study with Intel Dual-Core System*. CCGRID 2007. pp. 471-478 (2007)

[23]. Che D., Safran M.S., Peng Z., *From Big Data to Big Data Mining: Challenges, Issues and Opportunities*. DASFAA Workshops, pp. 1-15 (2013)

[24]. Choi K., Yoo D., Kim G., Suh Y., *A hybrid online-product recommendation system: Combining implicit rating-based collaborative filtering and sequential pattern analysis*. Electronic Commerce Research and Applications, 11(4), pp. 309-317 (2012)

[25]. Cong S., Han J., Padua D.A., *Parallel Mining of Closed Sequential Patterns*. KDD'05, pp. 562-567 (2005)

[26]. Congiusta A., Talia D., Trunfio P., *Parallel and Grid-Based Data Mining - Algorithms, Models and Systems for High-Performance KDD*. Data Mining and Knowledge Discovery Handbook, pp. 1009-1028 (2010)

[27]. Copp S.M., Bogdanov P., Debord M., Singh A., Gwinn E., *Base Motif Recognition and Design of DNA Templates for Fluorescent Silver Clusters by Machine Learning*. Advanced Materials, 28(33), pp.5839-5845 (2014)

[28]. Dijkstra E.W., Feijen W.H.J., Gasteren A.J.M., *Derivation of a termination detection algorithm for a distributed computation*. Inf. Process. Lett, 16(5), pp. 217-219 (1983)

[29]. Duong H.V., Truong T.C., Vo B., *An efficient method for mining frequent itemsets with double constraints*. Eng. Appl. of AI, 27, pp. 148-154 (2014)

[30]. Elmasri R., Navath S.B., *Fundamentals of Database Systems 7th Edition*. John Wiley & Sons. ISBN-13: 978-0133970777 (2016)

[31]. Fayyad U.M., Piatetsky-Shapiro G., Smyth P., *From Data Mining to Knowledge Discovery in Databases*. AI Magazine, 17(3), pp. 37-54 (1996)

[32]. Feng W., Cai X., *The Application and Improvement of ID3 Algorithm in WEB Log Data Mining*. LNEE'16, 393, pp. 577-583 (2016)

[33]. Flouri T., Iliopoulos C.S., Park K., Pissis S.P., *GapMis-OMP: Pairwise Short-Read Alignment on Multi-core Architectures*. AIAI, 2, pp. 593-601 (2012)

[34]. Fournier-Viger P., Gomariz A., Campos M., Thomas R., *Fast Vertical Mining of Sequential Patterns Using Co-occurrence Information*. PAKDD'14, 1, pp. 40-52 (2014)

[35]. Golmohammadi K., Zaïane O.R., *Time series contextual anomaly detection for detecting market manipulation in stock market*. DSAA, pp. 1-10 (2015)

[36]. Gomariz A., Campos M., Marin R., Goethals B., *ClaSP: An Efficient Algorithm for Mining Frequent Closed Sequences*. PAKDD'13, 1, pp. 50-61 (2013)

[37]. Gouda K., Hassaan M., Zaki M.J., *Prism: An Effective Approach for Frequent Sequence Mining via Prime-Block Encoding*. J. Comput. Syst. Sci., 76(1), pp. 88-102 (2010)

[38]. Goyal N., Balasubramaniam S., Goyal P., Islam S., Sati M., *A High Performance Computing Framework for Data Mining*. HiPC Workshops, pp.11-18 (2016)

[39]. Grama A., Karypis G., Kumar V., Gupta A., *Introduction to Parallel Computing 2nd Edition*. Addison Wesley, ISBN-13: 978-0201648652 (2003)

[40]. Han J., Pei J., Mortazavi-Asl B., Chen Q., Dayal U., Hsu M., *Freespan: Frequent Pattern-Projected Sequential Pattern Mining*. KDD'00, pp. 355-359 (2000)

[41]. Han J., Pei J., Yin Y., *Mining Frequent Patterns Without Candidate Generation*. ACM SIGMOD'00, pp. 1-12 (2000)

[42]. Han J., Kamber M., Pei J., *Data Mining: Concepts and Techniques 3rd Edition*. Morgan Kaufmann, USA. ISBN 978-0-12-381479-1 (2011)

[43]. Han M., Wang Z., Yuan J., *Efficient Method for Mining Patterns from Highly Similar and Dense Database based on Prefix-Frequent-Items*. JSW, 9(8), pp. 2080-2086 (2014)

[44]. Hernández J.K.F., Palancar J.H., León R.H., Uribe C.F., *SPaMi-FTS: An Efficient Algorithm for Mining Frequent Sequential Patterns*. CIARR, 8827, pp. 470-477 (2014)

[45]. Hillar G.C., *Professional Parallel Programming with C#: Master Parallel Extensions With .NET 4*. Wiley Publishing, ISBN: 978-0-470-49599-5 (2011)

[46]. Höppner F., *Association Rules*. Data Mining and Knowledge Discovery Handbook, pp. 299-319 (2010)

[47]. Huynh B., Vo B., *Using Multi-Core Processors for Mining Frequent Sequential Patterns*. ICIC Express Letters, 9(11), pp. 3071-3079 (2015)

[48]. Huynh B., Nguyen D., Vo B., *Parallel frequent subgraph mining on multi-core processor systems*. ICIC Express Letters, 10(9), pp. 2105-2113 (2016)

[49]. Huynh B., Vo B., Snasel V., *An efficient method for mining frequent sequential patterns using multi-Core processors*. Appl. Intell., 46(3), pp. 703-716 (2017)

[50]. Idrissi A., Rehioui H., Laghrissi A., Retal S., *An improvement of DENCLUE algorithm for the data clustering*. ICTA, pp. 1-6 (2015)

[51]. Inmon W.H., *Building the Data Warehouse 4th Edition*. John Wiley & Sons. ISBN: 978-0-7645-9944-6 (2005)

[52]. Isinkaye F.O., Folajimi Y.O., Ojokoh B.A., *Recommendation systems: Principles, methods and evaluation*. Egyptian Informatics Journal, 16(3), pp. 261-273 (2015)

[53]. Jeyabharathi J., Shanthi D., *Enhanced sequence identification technique for protein sequence database mining with hybrid frequent pattern mining algorithm*. IJDMB'16, 16(3), pp. 205-229 (2016)

[54]. Jung H., Chung K., *Sequential pattern profiling based bio-detection for smart health service*. Cluster Computing, 18(1), pp. 209-219 (2015)

[55]. Katti A., Fatta G.D., *Dynamic group communication for large-scale parallel data mining*. Concurrent Engineering: R&A, 21(3), pp. 227-234 (2013)

[56]. Khan S.S., Ahmad A., *Cluster center initialization algorithm for K-modes clustering*. Expert Syst. Appl. , 40(18), pp. 7444-7456 (2013)

[57]. Kumar R., Zyuban V., Tulsen D.M., *Interconnections in multi-core architecture: understanding mechanisms, overheads and scaling*. International Symposium on Computer Architecture, 1-11 (2005)

[58]. Kriegel H.P., Kröger P., Sander J., Zimek A., *Density-based clustering*. Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery, 1(3), pp. 231-240 (2011)

[59]. Laurent A., Négrevergne B., Sicard N., Termier A., *Efficient Parallel Mining of Gradual Patterns on Multi-core Processors*. Advances in Knowledge Discovery and Management, 398, pp. 137-151 (2012)

[60]. Leijen D., Schulte W., Burckhardt S., *The design of a task parallel library*. ACM SIGPLAN Not., 44(10), pp. 227–242 (2009)

[61]. Li M., Holmes G., Pfahringer B., *Clustering Large Datasets Using Cobweb and K-Means in Tandem*. Australian Conference on Artificial Intelligence, pp. 368-379 (2004)

[62]. Li R., Hu H., Li H., Wu Y., Yang J., *MapReduce Parallel Programming Model: A State-of-the-Art Survey*. Int J Parallel Prog , 44(4), pp. 832-866 (2016)

[63]. Lin W.Y., Huang K.W., Wu C.A., *MCFPTree: An FP-tree-based algorithm for multi constraint patterns discovery*. IJBIDM, 5(3), pp. 231-246 (2010)

[64]. Lin C.J., Wu C., Chaovalitwongse W.A., *Integrating Human Behavior Modeling and Data Mining Techniques to Predict Human Errors in Numerical Typing*. IEEE Trans. Human-Machine Systems, 45(1), pp. 39-50 (2015)

[65]. Liu L., Li E., Zhang Y., Tang Z., *Optimization of Frequent Itemset Mining on Multiple-Core Processor*. VLDB '07, pp. 1275-1285 (2007)

[66]. Loh W.Y., *Classification and regression trees*. Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery, 1(1), pp. 14-23 (2011)

[67]. MacQueen J., *Some Methods for Classification and Analysis of Multivariate Observations*. Proceeding 5th Berkeley Symposium Mathematics, Statistics and Probability, 1, pp. 281-297 (1967)

[68]. Magoules F., Zhao H.X., *Data Mining and Machine Learning in Building Energy Analysis: Towards High Performance Computing*. Wiley-ISTE, ISBN: 978-1-84821-422-4 (2016)

[69]. Maragatham G., Lakshmi M., *UTARM: an efficient algorithm for mining of utility-oriented temporal association rules*. IJKEDM, 3(2), pp. 208-237 (2015)

[70]. Masseglia F., Cathala F., Poncelet P., *The PSP Approach for Mining Sequential Patterns*. PKDD'98, pp. 176-184 (1998)

[71]. Mazid M.M., Shawkat Ali A.B.M., Tickle K.S., *Improved C4.5 algorithm for rule based classification*. AIKED'10, pp. 296-301 (2010)

[72]. Garcia-Molina H., Ullman J. D., Widom J., *Database Systems: The Complete Book 2nd Edition*. Pearson, ISBN-13: 978-0131873254 (2009)

[73]. Napierala K., Stefanowski J., *Types of minority class examples and their influence on learning classifiers from imbalanced data*. J. Intell. Inf. Syst., 46(3), pp. 563-597 (2016)

[74]. Negrevergne B., Termier A., Rousset M.C., Méhaut J.F., *Para Miner: A Generic Pattern Mining Algorithm for Multi-Core Architectures*. Data Min. Knowl. Discov., 28(3), pp. 593-633 (2014)

[75]. Ng R.T., Lakshmanan L.V.S., Han J., Pang A., *Exploratory mining and pruning optimizations of constrained association rules*. SIGMOD'98, pp. 13-24 (1998)

[76]. Nguyen D., Vo B., Le B., *Efficient Strategies for Parallel Mining Class Association Rules*. Expert Systems with Applications, 41(10), pp. 4716-4729 (2014)

[77]. Nobile M.S., Tangherloni A., Besozzi D., Cazzaniga P., *GPU-powered and settings-free parameter estimation of biochemical systems*. CEC, pp. 32-39 (2016)

[78]. Olaiya F., Adeyemo A.B., *Application of Data Mining Techniques in Weather Prediction and Climate Change Studies*. I.J. Information Engineering and Electronic Business, 1, pp. 51-59 (2012)

[79]. Pei J., Han J., Mortazavi-Asl B., Wang J., Pinto H., Chen Q., Dayal U., Hsu M., *Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach*. IEEE Trans. Knowl. Data Eng., 16(11), pp. 1424-1440 (2004)

[80]. Pham T.T., Luo J., Hong T.P., Vo B., *An efficient method for mining non-redundant sequential rules using attributed prefix-trees*. Eng. Appl. of AI, 32, pp.88-99 (2014)

[81]. Qardaji W.H., Yang W., Li N., *Understanding Hierarchical Methods for Differentially Private Histograms*. PVLDB, 6(14), pp. 1954-1965 (2013)

[82]. Rajaraman A., Leskovec J., Ullman J., *Mining of Massive Datasets*. Stanford University (2014)

[83]. Rehioui H., Idrissi A., Abourezq M., Zegrari F., *DENCLUE-IM: A New Approach for Big Data Clustering*. NT/SEIT, pp. 560-567 (2016)

[84]. Ren Y., Tomko M., Salim F.D., Ong K., Sanderson M., *Analyzing Web behavior in indoor retail spaces*. JASIST, 68(1), pp. 62-76 (2017)

[85]. Riid A., Preden J.S., *Design of Fuzzy Rule-based Classifiers through Granulation and Consolidation*. JAISCR, 7(2), pp. 137-147 (2017)

[86]. Sánchez F., Cabarcas F., Ramirez A., Valero M., *Long DNA Sequence Comparison on Multi-core Architectures*. Euro-Par, 2, pp. 247-259 (2010)

[87]. Schlegel B., Karnagel T., Kiefer T., Lehner W., *Scalable Frequent Itemset Mining on Many-Core Processors*. DaMoN'13, page 3. DOI:10.1145/2485278.2485281 (2013)

[88]. Schönbach C., Verma C., Bond P.J., Ranganathan S., *Bioinformatics and systems biology research update from the 15th International Conference on Bioinformatics (InCoB2016)*. BMC Bioinformatics, 17(S-19), pp. 87-90 (2016)

[89]. Scott M.L., *Transactional Memory Today*. SIGACT News, 46(2), pp. 96-104 (2015)

[90]. Serban I.V., Sordoni A., Bengio Y., Courville A.C., Pineau J., *Building End-To-End Dialogue Systems Using Generative Hierarchical Neural Network Models*. AAAI, pp. 3776-3784 (2016)

[91]. Sheikh F., Karthick S., Malathi D., Sudarsan J.S., Arun C., *Analysis of Data Mining Techniques for Weather Prediction*. Indian Journal of Science and Technology, 9(38). DOI: 10.17485/ijst/2016/v9i38/101962 (2016)

[92]. Slimani T., Lazzez A., *Sequential Mining: Patterns and Algorithms Analysis*. Corr abs/International Journal of Computer and Electronics Research, 2(5), pp. 639-647 (2013)

[93]. Song S., Lu Y., *Decision tree methods: applications for classification and prediction*. Shanghai Archives of Psychiatry, 27(2), pp. 130-135 (2015)

[94]. Solihin Y., *Fundamentals of Parallel Computer Architecture*. Chapman and Hall/CRC, ISBN 9781482211184 - CAT# K21675 (2015)

[95]. Sordoni A., Galley M., Auli M., Brockett C., Ji Y., Mitchell M., Nie J.Y., Gao J., Dolan B., *A Neural Network Approach to Context-Sensitive Generation of Conversational Responses*. HLT-NAACL, pp. 196-205 (2015)

[96]. Srikant R., Agrawal R., *Mining Sequential Patterns: Generalizations and Performance Improvements*. EDBT'96, pp. 3-17 (1996)

[97]. Srikantaiah K.C., Kumar N.K., Venugopal K.R., Patnaik L.M., *Web caching and prefetching with cyclic model analysis of web object sequences*. I. J. Knowledge and Web Intelligence, 5(1), pp. 76-103 (2014)

[98]. Su Y., Shi F., Talpur S., Wang Y., Hu S., Wei J., *Achieving self-aware parallelism in stream programs*. Cluster Computing, 18(2), pp. 949-962 (2015)

[99]. Taheri S., Mammadov M.A., *Structure learning of Bayesian Networks using global optimization with applications in data classification*. Optimization Letters, 9(5), pp. 931-948 (2015)

[100]. Talukder N., Zaki M.J., *Parallel graph mining with dynamic load balancing*. BigData, pp. 3352-3359 (2016)

[101]. Tran M.T., Le B., Vo B., *Combination of Dynamic Bit Vectors and Transaction Information for Mining Frequent Closed Sequences Efficiently*. Eng. Appl. of AI, 38, pp. 183-189 (2015)

[102]. Tung A.K.H., Lu H., Han J., Feng L., *Efficient mining of Inter-transaction association rules*. IEEE Trans. Knowl. Data Eng., 15(1), pp. 43-56 (2003)

[103]. Tzenakis G., Papatriantafyllou A., Kesapides J., Pratikakis P., Vandierendonck H., Nikolopoulos D.S., *BDDT: block-level dynamic dependence analysis for deterministic task-based parallelism*. PPOPP, pp. 301-302 (2012)

[104]. Vajda A., *Multi-core and Many-core Processor Architectures*. Programming Many-Core Chips, pp. 9-43 (2011)

[105]. Van T.T., Vo B., Le B., *IMSR_PreTree: An Improved Algorithm for Mining Sequential Rules based on The Prefix-Tree*. Vietnam J. Computer Science, 1(2), pp. 97-105 (2014)

[106]. Vo B., Tran M.T., Hong T.P., Nguyen H., Le B., *A dynamic bit-vector approach for efficiently mining inter-sequence patterns*. IBICA'12, pp. 51-56 (2012a)

[107]. Vo B., Hong T.P., Le B., *DBV-Miner: A dynamic bit-vector approach for fast mining frequent closed itemsets*. Expert Syst. Appl., 39(8), pp. 7196-7206 (2012b)

[108]. Wang W., Yang J., Muntz R., *STING: A Statistical Information Grid Approach to Spatial Data Mining*. VLDB'97, pp. 186-195 (1997)

[109]. Wang J., Han J., *BIDE: Efficient Mining of Frequent Closed Sequences*. ICDE '04, pp. 79-90 (2004)

[110]. Wang W., Yang J., *Mining Sequential Patterns from Large Data Sets*. Advances in Database Systems, 28, pp. 1-161 (2005)

[111]. Wang J., Han J., Li C., *Frequent closed sequence mining without candidate maintenance*. IEEE Trans. Knowl. Data Eng., 19(8), pp. 1042-1056 (2007)

[112]. Wang P., Liu S., Liu M., Wang Q., Wang J., Zhang C., *The Improved DBSCAN Algorithm Study on Maize Purity Identification*. CCTA, 2, pp. 648-656 (2011)

[113]. Wang S., Jiang L., Li C., *Adapting naive Bayes tree for text classification*. Knowl. Inf. Syst., 44(1), pp. 77-89 (2015)

[114]. Weitschek E., Fiscon G., Felici G., *Supervised DNA Barcodes species classification: analysis, comparisons and results*. BioData Mining, 7:4 (2014)

[115]. Wilkinson B., *Parallel Programming Techniques And Applications*. Pearson Education, ISBN-13: 978-0131405639 (2006)

[116]. Wright A.P., Wright A., McCoy A.B., Sittig D.F., *The use of sequential pattern mining to predict next prescribed medications*. Journal of Biomedical Informatics, 53, pp. 73-80 (2015)

[117]. Yan X., Han J., Afshar R., *CloSpan: Mining Closed Sequential Patterns in Large Datasets*. SDM'03, pp. 166-177 (2003)

[118]. Yan X., Han J., *gSpan: Graph-Based Substructure Pattern Mining*. ICDM'02, pp. 721-724 (2002)

[119]. Yu K.M., Wu S.H., *An Efficient Load Balancing Multi-Core Frequent Patterns Mining Algorithm*. TrustCom'11, pp. 1408-1412 (2011)

[120]. Yu D., Wu W., Zheng S., Zhu Z., *BIDE-Based Parallel Mining of Frequent Closed Sequences with MapReduce*. ICA3PP'12, 7440, pp. 177- 186 (2012)

[121]. Zaki M.J., *SPADE: An Efficient Algorithm for Mining Frequent Sequences*. Machine Learning, 42(1/2), pp. 31-60 (2001a)

[122]. Zaki M.J., *Parallel Sequence Mining on Shared-Memory Machines*. J. Parallel Distrib. Comput., 61(3), pp. 401-426 (2001b)

[123]. Zaki M.J., Ho C.T., *Large-Scale Parallel Data Mining (Lecture Notes in Computer Science)*. Lecture Notes in Artificial Intelligence, 1759 (2010).

[124]. Zhang T., Ramakrishnan R., Livny M., *BIRCH: An Efficient Data Clustering Method for Very Large Databases*. SIGMOD'96, 25(2), pp. 103-114 (1996)

[125]. Zhang S.J., Du Z., Wang J.T.L., *New techniques for mining frequent patterns in unordered trees*. IEEE Trans. Cybernetics, 45(6), pp. 1113-1125 (2015)

[126]. Zhao W., Chen J.J., Perskin R., Wang Y., Liu Z., Hong H., Tong W., Zou W., *A novel procedure on next generation sequencing data analysis using text mining algorithm*. BMC Bioinformatics, 17, pp. 17:213 (2016)

[127]. Zhao Y., *Network intrusion detection system model based on data mining*. SNPD'16, pp. 155-160 (2016)

[128]. Zhu T., Bai S., *A Parallel Mining Algorithm for Closed Sequential Patterns*. AINA Workshops, 1, pp. 392-395 (2007)

[129]. Zou Q., Zeng J.C., Cao L., Jida R., *A novel features ranking metric with application to scalable visual and bioinformatics data classification*. Neurocomputing, 173, pp. 346-354 (2016)