VŠB – Technical University of Ostrava
Faculty of Electrical Engineering and
Computer Science

Efficient Algorithms for Mining
Colossal Patterns in High
Dimensional Databases

PHD THESIS

2021

Thanh Long Nguyen

Efficient Algorithms for Mining Colossal Patterns in High Dimensional Databases

Thanh Long Nguyen

PhD Thesis

Supervisor: Prof. RNDr. Václav Snášel, CSc.

Assoc. Prof. Bay Vo

Ostrava, 2021

Abstract

With the rapid development of information technology and the application of information technology in many areas of life and socio-economy, for many years the information of humanity has been stored in database system is increasing, the accumulation of this data occurs at a burst speed. This huge amount of data is really a valuable source of "resources" because information is a key element in many areas. Data mining has helped users gain valuable insights from huge databases and data warehouses.

Data mining has been widely applied in many fields. In the field of data mining, the association rule is used to indicate the association or correlation between the "conditional → consequent" data elements between data elements. Detecting association rules is to detect those relationships within the scope of a given set of data. Association rule was introduced in 1993 by Agrawal et al. [1] and has become one of the major data mining studies, especially in recent years. Linkage detection has been successfully applied in many socio-economic fields such as trade, health, biology, finance and banking.

In association rule, frequent pattern mining is a key and an important task. Frequent pattern mining refers to the patterns that frequently occur in databases. In the last two decades, researchers have proposed many techniques and algorithms for extracting the frequent patterns, in which the downward closure property plays a fundamental role.

One of the challenges in pattern mining is the computational costs besides that is the potentially huge number of extracted patterns. In this thesis, we present an overview of the work done for frequent pattern mining, especial colossal pattern mining and develop methods for mining frequent colossal patterns in high dimensional databases that can tackle emerging data processing workloads while coping with larger and larger scales. Firstly, we develop CP (colossal pattern)-tree for efficient storing colossal patterns. Next, we propose CP-Miner algorithm to mine colossal patterns. CP-Miner is based on CP-tree, early pruning transactions and dynamic bit vectors to mine frequent colossal patterns. PCP-Miner, an improved version of CP-miner is also developed to reduce runtime and memory usage. In PCP-Miner, we develop theorems to prune non-colossal patterns in the mining process. We also develop methods for mining colossal with constraints. In our proposal, two case of constraints are developed including pattern constraint and length constraint.

Keyword: Frequent Pattern Mining, Colossal Pattern Mining, Constraints, High Dimensional 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 between VŠB-Technical University of Ostrava and Ton Duc Thang University 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 Ho Chi Minh City University of Food Industry for their supporting me on during research time.

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

BVBUC	<i>Bitwise vertical bottom up colossal</i>
CHARM	<i>Closed Association Rule Mining</i>
CMP	<i>Colossal Pattern Miner</i>
CP-Miner	<i>Colossal Pattern-Miner</i>
CP-tree	<i>Colossal Pattern tree</i>
CPCP-Miner	<i>Constraint-Based Method for Mining Colossal Patterns</i>
DBV	<i>Dynamic bit vector</i>
DNA	<i>Deoxyribo Nucleic Acid</i>
Eclat	<i>Equivalence CLAss Transformation</i>
FIN	<i>Frequent Itemset based on Nodesets</i>
FP	<i>Frequent Pattern</i>
FPM	<i>Frequent Pattern Mining</i>
FP-array	<i>Frequent Pattern array</i>
FP-growth	<i>Frequent Pattern growth</i>
FP-tree	<i>Frequent Pattern tree</i>
IT-tree	<i>Itemset Tidset-tree</i>
KDD	<i>Knowledge Discovery and Data mining</i>
LCCP	<i>Length Constraints for mining Colossal Patterns</i>
LP-tree	<i>Linear Prefix tree</i>
MCFPTree	<i>Multi-Constrained Frequent Pattern Tree mining</i>
PCP-Miner	<i>Pruning Colossal Pattern Miner</i>
k-sequence	<i>Sequence with length k</i>
minsup	<i>User-specified minimum support threshold</i>

Annotation

I	$I = \{i_1, i_2, \dots, i_m\}$	Set of items
X	$X = \{l_1, l_2, \dots, l_u\}, l_k \subseteq I, 1 \leq k \leq u$	Itemset
D	$D = \{T_1, T_2, \dots, T_n\}, T_i = \langle tid, X \rangle, 1 \leq i \leq n$	Transaction database
$sup(X)$	$sup(X) = \{T_i \in D \mid X \subseteq T_i\} $	Minimum support (<i>minsup</i>)
τ	$0 < \tau \leq 1$	The core ratio

Table of Contents

Abstract.....	i
Acknowledgments	ii
List of Abbreviations	iii
Annotation	iii
Table of Contents.....	iv
List of Figures.....	vi
List of Tables.....	ix
Chapter 1. Introduction and goal of dissertation.....	1
1.1. Knowledge discovery process	1
1.2. Data mining	2
1.3. Data mining tasks	2
1.4. Data mining applications.....	3
1.5. Colossal pattern mining.....	5
1.6. Motivation and goal of dissertation.....	5
1.7. Contributions.....	6
1.8. Organization of thesis	6
Chapter 2. Background	8
2.1. Preliminaries.....	8
2.2. Bit vectors.....	10
2.2.1. Dynamic bit vectors.....	10
2.2.2. Dynamic bit vector data structure	10
2.3. Pattern mining.....	11
2.4. Algorithms for mining Colossal patterns.....	12
2.5. Algorithms for mining colossal closed patterns	13
Chapter 3. Efficient Algorithms for Mining Colossal Patterns.....	14
3.1. Introduction	14
3.2. CP-Miner algorithm.....	15
3.2.1. CP-tree.....	16
3.2.2. Algorithm.....	18
3.2.3. Complexity analysis	19

3.2.4. Illustration	19
3.3. Pruning techniques	21
3.3.1. PCP-Miner algorithm.....	21
3.3.2. Complexity analysis.....	23
3.3.3. Illustration	23
3.4. Experiments.....	26
3.4.1. Experimental environment and databases	26
3.4.2. Comparison of the runtimes	26
3.4.3. Comparison of the number of nodes in the trees.....	32
3.4.4. Scalability of CP-Miner and PCP-Miner	41
Chapter 4. Mining Colossal Patterns with Pattern Constraint	44
4.1. Introduction	44
4.2. Method for mining pattern with pattern constraint	45
4.3. Proposed algorithm	45
4.4. Experiments	48
Chapter 5. Mining Colossal Patterns with Length Constraints.....	51
5.1. Introduction	51
5.2. Mining patterns with constraints	52
5.3. Proposed algorithm	53
5.4. An example	57
5.5. Experimental study	62
5.5.1. Environment and databases.....	62
5.5.2. Comparison of the runtimes	63
Chapter 6. Conclusions and Future works	71
6.1. Conclusions	71
6.2. Future works	72
6.3. Publications	72
References.....	73

List of Figures

Figure 1.1. The knowledge discovery process.....	1
Figure 1.2. Data mining tasks.....	3
Figure 1.3. Predictive modeling	3
Figure 3.1. Search tree using BVBUc	17
Figure 3.2. CP-Miner algorithm for mining colossal patterns	18
Figure 3.3. Search tree using CP-Miner (the supports of the patterns are the levels of the nodes in the tree)	21
Figure 3.4. PCP-Miner algorithm for mining colossal patterns	22
Figure 3.5. CP-tree after extending to level 2 of node {011111, 1}.	24
Figure 3.6. CP-tree after extending to level 3 of node {011111, 1}.	25
Figure 3.7. Final CP-tree of the PCP-Miner algorithm.....	26
Figure 3.8. Runtime in the Retails database when the number of transactions is 50 (the number of items is 275).....	27
Figure 3.9. Runtime in the Retails database when the number of transactions is 100 (the number of items is 532).....	27
Figure 3.10. Runtimes in the Retails database when the number of transactions is 150 (the number of items is 773).....	28
Figure 3.11. Runtimes in the Accidents database when the number of transactions is 50 (the number of items is 151).....	28
Figure 3.12. Runtimes in the Accidents database when the number of transactions is 100 (the number of items is 168).....	28
Figure 3.13. Runtimes in the Accidents database when the number of transactions is 150 (the number of items is 177).....	29
Figure 3.14. Runtimes in the Connect database when the number of transactions is 50 (the number of items is 65).....	29
Figure 3.15. Runtimes in the Connect database when the number of transactions is 100 (the number of items is 69).....	29
Figure 3.16. Runtimes in the Connect database when the number of transactions is 150 (the number of items is 75).....	30
Figure 3.17. Runtimes in the Pumsb* database when the number of transactions is 50 (the number of items is 344).....	30
Figure 3.18. Runtimes in the Pumsb* database when the number of transactions is 150 (the number of items is 456).....	30
Figure 3.19. Runtimes in the Pumsb* database when the number of transactions is 150 (the number of items is 574).....	31

Figure 3.20. Runtimes in the T10I4D100K database when the number of transactions is 50 (the number of items is 335).....	31
Figure 3.21. Runtimes in the T10I4D100K database when the number of transactions is 100 (the number of items is 470).....	31
Figure 3.22. Runtimes in the T10I4D100K database when the number of transactions is 150 (the number of items is 554).....	32
Figure 3.23. The log scale of Retail dataset with 50 transactions.....	34
Figure 3.24. The log scale of Retail dataset with 100 transactions.....	34
Figure 3.25. The log scale of Retail dataset with 150 transactions.....	35
Figure 3.26. The log scale of Accidents dataset with 50 transactions	35
Figure 3.27. The log scale of Accidents dataset with 100 transactions	36
Figure 3.28. The log scale of Accidents dataset with 150 transactions	36
Figure 3.29. The log scale of Connect dataset with 50 transactions	37
Figure 3.30. The log scale of Connect dataset with 100 transactions	37
Figure 3.31. The log scale of Connect dataset with 150 transactions	38
Figure 3.32. The log scale of Pumsb* dataset with 50 transactions	38
Figure 3.33. The log scale of Pumsb* dataset with 100 transactions	39
Figure 3.34. The log scale of Pumsb* dataset with 150 transactions	39
Figure 3.35. The log scale of T10I4D100K dataset with 50 transactions	40
Figure 3.36. The log scale of T10I4D100K dataset with 100 transactions	40
Figure 3.37. The log scale of T10I4D100K dataset with 150 transactions	41
Figure 3.38. Runtimes in the Retails database when the number of transactions is 500 (the number of items is 2,058).....	42
Figure 3.39. Runtimes in the Retails database when the number of transactions is 1,000 (the number of items is 3,182).....	42
Figure 3.40. Runtimes in the Retails database when the number of transactions is 2,000 (the number of items is 4,775).....	43
Figure 4.1. CPCP-Miner algorithm for mining colossal patterns with pattern constraint.....	47
Figure 4.2. Comparison of two methods in Retails database with the number of transaction is 2000.....	48
Figure 4.3. Comparison of two methods in Accidents database with the number of transaction is 150...	49
Figure 4.4. Comparison of two methods in Connect database with the number of transaction is 150	49
Figure 4.5. Comparison of two methods in Pumsb* database with the number of transaction is 150	50
Figure 5.1. LCCP algorithm for mining colossal patterns with length constraints	56
Figure 5.2. CP-tree after expending node {BCDEGHJ}	59
Figure 5.3. CP-tree after expanding node {BCDEGHJ}	61

Figure 5.4. Final CP-tree for mining colossal patterns with length constraints	62
Figure 5.5. Accidents dataset with 100 transactions, various threshold φ , fix $\xi = 20$ and $\Omega = 35$	63
Figure 5.6. Accidents database with 150 transactions, various threshold φ , fix $\xi = 20$ and $\Omega = 35$	64
Figure 5.7. Accidents database with 200 transactions, various threshold φ , fix $\xi = 20$ and $\Omega = 35$	64
Figure 5.8. Accidents database with 200 transactions, various min length constraints ξ , fix $\varphi=5$ and $\Omega=35$	65
Figure 5.9. Chess database with 300 transactions, various threshold φ , fix $\xi = 26$ and $\Omega = 35$	65
Figure 5.10. Chess database with 400 transactions, various threshold φ , fix $\xi = 26$ and $\Omega = 35$	66
Figure 5.11. Chess database with 500 transactions, various threshold φ , fix $\xi = 26$ and $\Omega = 35$	66
Figure 5.12. Chess database with 500 transactions, various min length constraints ξ , fix $\varphi=14$ and $\Omega=35$	67
Figure 5.13. Connect database with 300 transactions, various threshold φ , fix $\xi = 34$ and $\Omega = 35$	67
Figure 5.14. Connect database with 400 transactions, various threshold φ , fix $\xi = 34$ and $\Omega = 35$	68
Figure 5.15. Connect database with 500 transactions, various threshold φ , fix $\xi = 34$ and $\Omega = 35$	68
Figure 5.16. Connect database with 500 transactions, various min length constraints ξ , fix $\varphi=14$ and $\Omega=40$	68
Figure 5.17. Mushroom database with 500 transactions, various threshold φ , fix $\xi = 16$ and $\Omega = 20$	69
Figure 5.18. Mushroom database with 600 transactions, various threshold φ , fix $\xi = 16$ and $\Omega = 20$	69
Figure 5.19. Mushroom database with 700 transactions, various threshold φ , fix $\xi = 16$ and $\Omega = 20$	70
Figure 5.20. Mushroom database with 700 transactions, various min length constraints ξ , fix $\varphi=16$ and $\Omega=20$	70

List of Tables

Table 2.1. An example database D.....	8
Table 2.2. Bit vectors of transactions in Table 2.1 with fixed 6 bits	8
Table 2.3. Example of 24-byte bit vector	10
Table 2.4. Conversion of bit vector to DBV	11
Table 3.1. Results after removing items for which their supports do not satisfy minsup	19
Table 3.2. Results after removing transactions and re-arranging transaction IDs.....	20
Table 3.3. Bit vectors of transactions in Table 3.2	20
Table 3.4. Bit vectors of transactions after sorting.....	23
Table 3.5. Characteristics of the experimental databases	26
Table 3.6. Comparison of the number of nodes	32
Table 4.1. An example database	45
Table 4.2. Colossal patterns with DE-constraint	45
Table 5.1. A transaction database	53
Table 5.2. Colossal patterns with length constraints.....	54
Table 5.3. Items and their support	57
Table 5.4. A transaction database after filtering items	57
Table 5.5. A transaction database after removing transactions	57
Table 5.6. Items and their support after removing transactions	58
Table 5.7. A transaction database after removing transactions, items and rearranged Tid	58
Table 5.8. Characteristics of the experimental databases	62

Chapter 1. Introduction and goal of dissertation

The explosion of data in the recent years led to a growth in the size of databases. With the huge database was collected, it contains much hidden useful information need exploits. However, extracting useful information is challenging because traditional data analysis methods cannot due to the massive size of databases.

Data mining [2] is the element core in knowledge discovery and data mining (KDD) [3] process, a research field that is focused on the automatic extraction of hidden useful information from databases. Data mining is a necessary task because extracted information is usually used to support for intelligent applications such as marketing decisions, targeted marketing, and fraud detection.

The goal of dissertation is focus on research and propose new approaches to improve the performance of the extracting information process, especially are colossal patterns in massive databases.

1.1. Knowledge discovery process

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 . It also includes the preprocessing, sampling, and projections of the data prior to the data mining step. Data mining refers to a particular step in KDD process.

Figure 1.1. The knowledge discovery process¹

The KDD process starts with the developing an understanding of the application domain, the relevant prior knowledge and the goals of the end-user. Then, a target of the database is created by selecting from a data source 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

¹ <https://data-flair.training/blogs/data-mining-and-knowledge-discovery/>

fusing data from multiple sources, cleaning data to remove noise or duplicated, collecting the necessary information to model or account for noise, handling 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. In the end, as a post-processing phase, mining patterns have to be interpreted and validated, thus consolidating the discovered knowledge.

1.2. Data mining

Data mining is an essential process in the knowledge discovery process where intelligent methods are applied in order to extract data patterns. It is used to extracting interesting information from large data [2]. Data mining also 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 [4], bioinformatics [5], recommender systems [6], web-mining [7] [8], medicine [9], weather prediction [10], fraud detection [11], network intrusion detection [3], smart health [12] and others. It can be performed on the different type of data repositories such as 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.

Data mining has attracted a great deal of attention in the information industry 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 services, for production control and science exploration.

1.3. Data mining tasks

The tasks of data mining are very diverse and distinct, 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 tasks can be divided 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 (includes association analysis, clustering analysis, summarization, sequence discovery), while predictive mining tasks perform inference on the current data in order to make predictions (includes classification, prediction, regression and time series) [2].

The relative importance of prediction and description for particular data mining applications can vary considerably. However, in the context of KDD, description tends to be more important than prediction. This is in contrast to pattern recognition and machine learning applications where prediction is often the primary goal of the KDD process.

Figure 1.1. Data mining tasks

Among the various descriptive data mining tasks, frequent pattern mining, association rule mining and sequential pattern mining have been the most commonly employed tasks to facilitate business development. The patterns that occur frequently in the database are called frequent patterns. These patterns may include itemsets that appear together, correlated and frequent.

Figure 1.2. Predictive modeling²

1.4. Data mining applications

Data mining is an important technology that has been integrated in many industrial, governmental and academic applications such as market-basket analysis, bioinformatics, recommender systems, web-mining, medicine, weather prediction, fraud detection, network intrusion detection, smart health and others.

Healthcare: Data mining has a potential to improve health systems. It uses data and analytics to identify best practices that improve care and reduce costs. Mining can be used to

² <https://www.predictiveanalyticstoday.com/predictive-modeling/>

predict the volume of patients in every category. Data mining can also help healthcare insurers to detect fraud and abuse.

Market Basket Analysis (MBA): MBA is a technique based on a theory that if you buy a certain group of items you are more likely to buy another group of items. This technique may allow the retailer to understand the purchase behavior of a buyer. This information may help the retailer to know the buyer's needs and change the store's layout accordingly.

Educational Data Mining (EDM): EDM is a newly emerging field, concerns with developing methods that discover knowledge from data originating from educational environments. The goals of EDM are identified as predicting students' future learning behavior, studying the effects of educational support, and advancing scientific knowledge about learning. Data mining can be used by an institution to take accurate decisions and also to predict the results of the student. With the results, the institution can focus on what to teach and how to teach.

Manufacturing Engineering: Data mining tools can be very useful to discover patterns in the complex manufacturing process. Data mining can be used in system-level designing to extract the relationships between product architecture, product portfolio, and customer needs data. It can also be used to predict the product development span time, cost, and dependencies among other tasks.

CRM: Customer Relationship Management is a tool for management and analysis of customer database to improving customers' loyalty and implementing customer-focused strategies. To maintain a proper relationship with a customer, a business needs to collect data and analyze the information. With data mining technologies, the collected data can be used for analysis and predict the trends of customer.

Fraud Detection: Traditional methods of fraud detection are time-consuming and complex. Data mining providing meaningful patterns and turning data into useful information. A perfect fraud detection system should protect information of all the users by analysing a collection of sample records was stored and classified as fraudulent or non-fraudulent.

Intrusion Detection: intrusion is any action that will compromise the integrity and confidentiality of a resource. Data mining can help improve intrusion detection by adding a level of focus to anomaly detection. It helps an analyst to distinguish an activity from common everyday network activity. Data mining also helps to extract data which is more relevant to the problem.

Financial Banking: With e-banking everywhere, a huge amount of data is generated with new transactions. Data mining can contribute to solving business problems in banking and finance by finding patterns, causalities, and correlations in business information and market prices that are not immediately apparent to managers because the volume data is too large or is generated too quickly. The managers may find this information for better segmenting, targeting, acquiring, retaining and maintaining a profitable customer.

Research Analysis: Data mining is helpful in data cleaning, data pre-processing and integration of databases. The researchers can find any similar data from the database that might bring any change in the research. Identification of any co-occurring sequences and the correlation between any activities can be known. Data visualization and visual data mining provide us with a clear view of the data.

Criminal Investigation: crime analysis includes exploring and detecting crimes and their relationships with criminals. The high volume of crime datasets and also the complexity of relationships between these kinds of data have made criminology an appropriate field for applying data mining techniques.

Bioinformatics: Data Mining is an approach suited for Bioinformatics since it is data-rich. Mining biological data helps to extract useful knowledge from massive datasets gathered in biology, and in other related life sciences areas such as medicine.

1.5. Colossal pattern mining

The frequent pattern mining methods presented handle large datasets having a small number of dimensions with a reasonable minsup. However, some applications may need to mine high dimensional data (i.e., data with hundreds or thousands of dimensions) with very small minsup. But the methods studied so far cannot mine high-dimensional data because the search spaces of such typical methods grow exponentially with the number of dimensions.

Currently, two solutions are used to overcome this problem. One direction extends a pattern growth approach by further exploring the vertical data format to handle datasets with a large number of dimensions but a small number of rows. This is useful in applications like the analysis of gene expressions in bioinformatics, for example, where we often need to analyze microarray data that contain a large number of genes but only a small number of samples. The other direction develops a new mining methodology, called Pattern-Fusion, which mines colossal patterns, that is, patterns of very long length.

In the first direction, a pattern growth-based row enumeration approach. The original dataset D can easily be transformed into a transposed dataset D' . A dataset with a small number of rows but a large number of dimensions is then transformed into a transposed dataset with a large number of rows but a small number of dimensions. Efficient pattern growth methods can then be developed on such relatively low-dimensional datasets.

In Pattern-Fusion [16], a new mining methodology that mines colossal patterns. This method takes leaps in the pattern search space, leading to a good approximation of the complete set of colossal frequent patterns. Pattern-Fusion algorithm was proposed to mine K best patterns from the database with the number of items is large.

1.6. Motivation and goal of dissertation

Although many algorithms and techniques have been proposed to deal with the frequent pattern mining. However, frequent pattern mining still faces tough challenges in both effectiveness and efficiency. There could be a large number of patterns in a huge database, dense databases, especially is large databases with high dimensional.

In 2007, Zhu et al. [16] proposed a method for mining colossal patterns called the Pattern-Fusion algorithm, a method for mining colossal patterns in high dimensional databases. In 2010, Dabbiru and Shashi [33] proposed CMP (Colossal Pattern Miner) algorithm for mining such patterns. Next, in 2012, Sohrabi and Barforoush [32] also proposed the BVBU (Bitwise vertical bottom up colossal) algorithm for mining colossal patterns in high dimensional databases.

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 reducing response time is sampling. But, in some cases reducing data might result in inaccurate models, in some other cases it is not useful.

To deal with these above problems, our focus on research and development efficient methods to improve the performance of mining process, especially for colossal patterns in high dimensional databases.

Specifically, we proposed 4 new algorithms CP-Miner, PCP-Miner, CPCP-Miner and LCCP for mining colossal patterns. Experimental results show that these algorithms are more efficient than BVBUc in both the mining time and search space. This is the main motivates and goals of dissertation.

1.7. Contributions

This section describes the contributions of this work to develop a model for the mining patterns in the high dimensional database. The effective methods for mining colossal patterns are the main subject of our thesis.

The main goal is to introduce data mining techniques and to show how large-scale data mining and knowledge discovery applications can achieve scalability and efficiency. Our approach is research and proposed the efficient methods for mining colossal pattern from the high dimensional database.

The contributions of this research as follows

1. Presenting an overview of pattern mining and data structure for representing compact transaction databases.
2. Developing effective methods for mining colossal pattern in high dimensional database.
3. Developing a method for mining colossal patterns with pattern constraint.
4. Statement the problem of extracting colossal patterns with length constraints (min-length and max-length) and developing a method for mining colossal patterns with length constraints.
5. Experiments and evaluation of the proposed methods to confirm the efficient of these methods.

1.8. Organization of thesis

In this section, we summarize the content of the remaining chapters of this thesis.

Chapter 2: Background. In this chapter, we present the basic concepts and models of pattern mining, theoretically we show the kinds of patterns and some properties of them. We also present overview some traditional pattern mining framework and a compact data structure.

Chapter 3: Efficient algorithms for colossal pattern mining. This chapter introduces efficiency approaches for mining colossal pattern. We present the experimental results of our proposed methods for proven the efficiency.

Chapter 4: Mining colossal patterns with pattern constraint. We propose a method for mining colossal patterns with pattern constraint. We present the test results of the proposed methods which are proven to be more effective.

Chapter 5: Mining colossal patterns with length constraints. We introduce an approach for mining colossal patterns with constraints on the high dimensional database. This chapter also proposed effective approaches to mining colossal patterns with min-length and max-length constraints.

Chapter 6: Conclusion and future works. Summarizing the whole thesis and giving directions for future works.

Chapter 2. Background

2.1. Preliminaries

Definition 2.1 (itemset). Let $I = \{i_1, i_2, \dots, i_m\}$ be a set of m distinct attributes, also called *items*. An itemset $X = \{I_1, I_2, \dots, I_u\}$, such that $I_k \in I$ ($1 \leq k \leq u$) 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.

Definition 2.2 (transaction database). A transaction database is a set of transactions, denoted $D = \{T_1, T_2, \dots, T_n\}$, where n is the number of transactions in D , 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 (length of itemset). Given an itemset $X = \{a_1 a_2 \dots a_u\}$, 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.

Table 2.1. An example database D

<i>Tid</i>	<i>Items</i>
1	H J
2	B C D E F
3	A D E F L
4	G K
5	A D E J
6	K L
7	A B C D F G
8	A B C D F H

Table 2.2. Bit vectors of transactions in Table 2.1 with fixed 6 bits

<i>Tid</i>	Bit vector
1	000000
2	011111
3	100111
4	000000
5	100110
6	000000
7	111101
8	111101

Definition 2.4 (support of an itemset). Given a transaction dataset D , the support of an itemset X , denoted by $sup(X)$, is the number of transactions containing X .

Definition 2.5 (frequent pattern). Given a transaction dataset D , a pattern X is frequent if $sup(X) \geq minsup$, where *minsup* is the value user-specified minimum support.

Definition 2.6 (core pattern) [13]. Given a pattern Y , a pattern $X \subseteq Y$ is said to be a τ -core pattern of Y if $\text{sup}(Y)/\text{sup}(X) \geq \tau$, $0 < \tau \leq 1$ (τ is called the core ratio).

For a pattern Y , let C_Y be the set of all its core patterns, i.e., $C_Y = \{X \mid X \subseteq Y \text{ and } X \text{ is a } \tau\text{-core pattern of } Y\}$.

Definition 2.7 (colossal pattern). Frequent itemset (FI) is a set of all frequent patterns in a transaction database D . An itemset X is called a colossal pattern in FI if and only if there does not exist an itemset Y such that $X \subset Y$ and X is a τ -core pattern of Y .

The problem of mining colossal patterns is to find colossal patterns that satisfy the core ratio τ .

Definition 2.8 (The pattern contained in a set of transactions). For a set of transactions $\{t_{i1}, t_{i2}, \dots, t_{ik}\}$, the pattern contained in $\{t_{i1}, t_{i2}, \dots, t_{ik}\}$ is $t_{i1} \cap t_{i2} \cap \dots \cap t_{ik}$.

For example: Consider Table 2.1, the pattern contained in transactions $\{2, 3, 5\}$ is $\{B C D E F\} \cap \{A D E F L\} \cap \{A D E J\} = \{A D E\}$ and that contained in transactions $\{2, 3, 7\}$ is $\{B C D E F\} \cap \{A D E F L\} \cap \{A B C D F G\} = \{A D\}$.

Theorem 2.1: Given P_1 is the pattern contained in $T_1 = \{t_{i1}, t_{i2}, \dots, t_{ik-1}, t_{ik}\}$ and P_2 is the pattern contained in $T_2 = \{t_{i1}, t_{i2}, \dots, t_{ik-1}, t_{ik+1}\}$, then the pattern contained in $T = \{t_{i1}, t_{i2}, \dots, t_{ik-1}, t_{ik}, t_{ik+1}\}$ is $P = P_1 \cap P_2$.

Proof: According to Definition 2.8, we have $P_1 = t_{i1} \cap t_{i2} \cap \dots \cap t_{ik-1} \cap t_{ik}$, $P_2 = t_{i1} \cap t_{i2} \cap \dots \cap t_{ik-1} \cap t_{ik+1}$, and $P = t_{i1} \cap t_{i2} \cap \dots \cap t_{ik} \cap t_{ik+1} \Rightarrow P_1 \cap P_2 = t_{i1} \cap t_{i2} \cap \dots \cap t_{ik-1} \cap t_{ik} \cap t_{i1} \cap t_{i2} \cap \dots \cap t_{ik-1} \cap t_{ik+1} = (t_{i1} \cap t_{i1}) \cap (t_{i2} \cap t_{i2}) \cap \dots \cap (t_{ik-1} \cap t_{ik-1}) \cap t_{ik} \cap t_{ik+1} = t_{i1} \cap t_{i2} \cap \dots \cap t_{ik} \cap t_{ik+1} = P$.

Based on Theorem 2.1, the pattern of a k -transaction can be generated from the patterns of $(k-1)$ -transactions.

Theorem 2.2: If a pattern X of a k -transaction T is a colossal pattern, then all patterns $X' \neq X$ created from T are not colossal patterns.

Proof: Assume that X is a pattern of T and X' is a pattern of any $T \cup T'$ ($T' \subset D \setminus T$), this implies that $X' \subset X$. There are two cases:

- i) If $\text{sup}(X)/\text{sup}(X') < \tau$ then according to Definition 2.6, X' is not a τ -core pattern and therefore X' cannot be a colossal pattern, according to Definition 2.7.
- ii) If $\text{sup}(X)/\text{sup}(X') \geq \tau$. Because of $X' \subset X$, according to Definition 2.7, X' cannot be a colossal pattern.

Definition 2.9 (Subsuming of a pattern). Given two patterns P_1 and P_2 , if $P_1 \subseteq P_2$ then P_1 is subsumed by P_2 .

Theorem 2.3 (Check duplication based on subsuming). Given P_1 is the pattern of $T_1 = \{t_{i1}, t_{i2}, \dots, t_{ik-1}, t_{ik}\}$, P_2 is the pattern of $T_2 = \{t_{i1}, t_{i2}, \dots, t_{ik-1}, t_{ik+1}\}$, if $P_1 \subseteq P_2$ then the pattern that is created from T_1 and T_2 is a duplication.

Proof: It is easy to see that the pattern that is created from T_1 and T_2 is $P = P_1 \cap P_2 = P_1$. This means that P is a duplication of P_1 .

Based on Theorem 2.3, we can prune a node containing a subsumed pattern without losing information.

Theorem 2.4: If pattern X of a node is subsumed by any colossal pattern then all patterns created from this node cannot be colossal patterns.

In Table 2.3, the first non-zero byte appears at position 6. The conversion of the bit vector to a DBV is shown in Table 2.4.

Table 2.4. 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, 00001011, is represented by number 11 in byte. For example, value of item *A* is 11 (1011) and that of item *B* is 4 (0100). The intersection of 11 & 4 is equal to 1011 & 0100. Using DBV helps to reduce memory storage and computational cost in bitwise AND operator.

2.3. Pattern mining

The problem of mining frequent patterns was first proposed by Agrawal et al. in 1993 [1], and this is the main problem of association rule mining. In 1994, Agrawal and Srikant developed the downward closure property to prune candidates that do not satisfy *minsup* [18]. The Apriori algorithm, a level-wise approach, was also proposed. Using the downward closure property, Apriori generates candidate (k+1)-itemsets from frequent k-itemsets and also uses this property to prune candidates. In 1997, Zaki et al. developed the Eclat algorithm for mining frequent itemsets using the IT-tree (Itemset Tidset-tree) [19]. Eclat applies a depth first search scheme and vertical data format to mine frequent itemsets. The advantages of this approach are that it only scans the database once and rapidly compute the supports of patterns based on the intersections of tidsets. Because tidsets consume more memory in dense databases, in 2003 Gouda and Zaki proposed using diffsets instead to reduce memory usage and time [20]. The IT-tree was also used in CHARM [21] to mine frequent closed patterns. Early in the process CHARM uses subset checking to omit patterns that cannot be closed, and checks whether a candidate is closed or not using hash table. Diffsets are also used in another study [21]. In 2000, Han et al. proposed the FP-tree structure and used it for mining frequent patterns [22]. In this, the FP-tree compresses the database in a prefix tree and then uses projections in this to mine frequent patterns. The authors also used an FP-tree to mine frequent closed patterns [23]. Grahne and Zhu used an FP-array to reduce the number of traverses and projections in an FP-tree, and applied an FP-array to mine frequent (closed) patterns [24]. Bit vector-based algorithms were then also developed [25, 26, 17]. In 2007, Dong and Han proposed the BitTableFI algorithm for mining frequent patterns [25]. This is based on Apriori, but uses bit vectors to store tidsets of patterns and computes the supports of these by computing the intersections of bit vectors. The support of a pattern is the number of bits 1 in a bit vector. By using bit vectors, BitTableFI only scans the database once (Apriori scans the database *k* times, where *k* is the longest pattern). In 2008, Song et al. improved BitTableFI by using the subsume concept to quickly determine the support of subsumed

patterns, and thus the supports do not need to be computed for these patterns [26]. However, BitTableFI and its improved algorithm (Index-BitTableFI) use fixed bit vectors, which means that the number of bits in each bit vector does not change and is the number of transactions in the database. When the number of transactions in the database is large, bit vectors consume more memory to store and time to compute their intersections. In 2012, Vo et al. proposed the dynamic bit vector (DBV) concept, and used it to mine frequent closed patterns [17]. A dynamic bit vector is also a bit vector, but it removes zero bits at the beginning and end. The authors also proposed an algorithm for computing the intersection between two DBVs, and the way to determine whether a pattern is not frequent in the process of computing the intersections of DBVs. In 2012, Deng and Lv proposed a method for mining frequent patterns using the N-list [27]. N-list has a structure like an FP-tree (adding Pre and Post for each node, and not storing a header table as FP-tree does). Unlike FP-Growth, an N-list based algorithm (PrePost) uses a vertical data format to mine frequent patterns. An improved PrePost algorithm using the subsume concept to reduce the search space has also been proposed [28], while N-lists have been used to mine frequent closed patterns [29]. Deng and Lv then proposed the Nodesets structure. Nodesets only use Pre (or Post) to compress the database, and the FIN algorithm uses this structure to mine frequent patterns. DiffNodesets, a different Nodesets strategy that can reduce both memory usage and computing time, was proposed in 2016 [30]. Yun et al. also proposed an LP-tree structure for mining frequent itemsets [31].

2.4. Algorithms for mining Colossal patterns

There are many algorithms that have been developed to efficiently mine frequent patterns, such as Apriori, Eclat, FP-Growth, PrePost, etc. However, these are only efficient with a small number of items in the database. When a database has a large number of items (from thousands to hundreds of thousands) but the number of transactions is small, these algorithms cannot run when the minimum support threshold is also small because the search space is huge, especially for high dimensional databases.

The concept of colossal patterns was thus developed to solve the problem of high dimensional databases. The concept of core-fusion was then developed, and the authors proposed the Pattern-Fusion algorithm for mining colossal patterns. In 2012, Sohrabi and Barforoush proposed the BVBUC algorithm [32], which uses bit vectors to present the pattern in each transaction, and applies vertical bottom-up traversing in transactions to mine colossal patterns.

To solve this problem, Zhu et al. [16] proposed a method for mining colossal patterns called the Pattern-Fusion algorithm. To overcome the huge search space, Pattern-Fusion uses an approximation approach to mine K good frequent patterns. This means that Pattern-Fusion may not mine all colossal patterns that satisfy [Definition 4.1](#) (chapter 4). In 2010 Dabbiru and Shashi [33] proposed the CMP (Colossal Pattern Miner) algorithm for mining such patterns. Next, Sohrabi and Barforoush [32] proposed a method for mining colossal patterns in high dimensional databases in 2012. The authors also proposed the BVBUC algorithm, which uses a bottom-up strategy based on transactions. The BVBUC joins 1-transactions together to generate 2-transactions, and so on, until the number of transactions reaches *minsup* (which means that the patterns in a set of transactions are frequent). Moreover, the authors also

proposed a formula to prune branches that cannot expand to reach *minsup* to reduce the search space.

2.5. Algorithms for mining colossal closed patterns

Recently, Frequent Colossal Closed Itemsets (FCCI) mining is also focused, especially in the field of bioinformatics. Instead of mining all colossal patterns, mining colossal closed patterns were also attention because it precludes more cost. Some algorithm for mining colossal closed patterns was proposed such as DisClose [34] extracts colossal closed itemsets by enumerating large cardinality itemsets and then builds smaller itemsets based on data structure named Compact Row-Tree (CRTree) stores the itemsets and to traverse the search tree in a bottom-up strategy.

The existing preprocessing techniques fail to prune the complete set of irrelevant features and irrelevant rows. The DSFCCIM [35] algorithm proposed a novel efficient to mine FCCI from datasets consisting of a large number of rows and a large number of features based on the Effective Improvised Pre-processing (EIP) technique to prune the complete set of irrelevant features and irrelevant rows. The algorithm is integrated with a novel Rowset Cardinality Table, Itemset Support Table, and two efficient pruning strategies to cut down the search space.

The DREFCCIM [36] algorithm is the first distributed algorithm to mine FCCI, it is a good strategy to overcome the inefficiency of the existing sequential algorithm. The row enumerated mining search space is cut down by the pruning strategy enclosed to increasing the efficiency of the algorithm.

The RARE [37] algorithm mining colossal closed itemsets in high dimensional based on the strategy's breadth-first bottom-up row-enumeration. The algorithm uses a minimum cardinality threshold to prune the search space and efficient closure checking and pruning strategies are employed.

Chapter 3. Efficient Algorithms for Mining Colossal Patterns

There are many algorithms that have been developed to efficiently mine frequent patterns. However, these algorithms are then inefficient when the minimum support threshold is also small because the search space is huge, especially for high dimensional databases. In this section, we present novel algorithms for mining colossal patterns on high dimensional databases.

3.1. Introduction

To solve the problem colossal pattern mining, Zhu et al. [16] proposed a method for mining colossal patterns called the Pattern-Fusion algorithm. To overcome the huge search space, Pattern-Fusion uses an approximation approach to mine K good frequent patterns. This means that Pattern-Fusion may not mine all colossal patterns that satisfy [Definition 2.7](#). In 2010, Dabbiru and Shashi [33] proposed the CMP (Colossal Pattern Miner) algorithm for mining such patterns. Next, Sohrabi and Barforoush [32] proposed a method for mining colossal patterns in high dimensional databases in 2012. The authors also proposed the BVBUC algorithm, which uses a bottom-up strategy based on transactions. The BVBUC joins 1-transactions together to generate 2-transactions, and so on, until the number of transactions reaches *minsup* (which means that the patterns in a set of transactions are frequent). Moreover, the authors also proposed a formula to prune branches that cannot expand to reach *minsup* to reduce the search space. Although BVBUC is faster than CMP and Pattern-Fusion it has some limitations, as follows:

1. The patterns of a set of transactions are computed many times, and this makes BVBUC inefficient.
2. BVBUC uses the downward closure property to prune items that do not satisfy *minsup*, but it does not remove transactions (after removing infrequent items, some transactions do not contain any items).
3. BVBUC generates a lot of duplications, and thus more time is needed to check these.
4. BVBUC finds patterns based on a set of transactions (tidset) by computing the intersections among them. In fact, two tidsets only differ by one transaction if they have parent-child relationship. For example: tidsets $\{1, 2, 3\}$ and $\{1, 2, 3, 4\}$ only differ in transaction 4. In this case, if we have pattern X of tidset $\{1, 2, 3\}$, we can get pattern of tidset $\{1, 2, 3, 4\}$ by computing the intersection between X and pattern of transaction 4.
5. In high dimensional databases, using bit vectors consumes more memory and time to join them.

Based on some of the limitations of BVBUC, in this work we propose a new method for mining colossal patterns. The main contributions of this paper are as follows:

1. We develop a method for computing the pattern of a set of transactions once.

2. Based on the downward closure property, we remove 1-items for which their supports do not satisfy *minsup*. After that, we remove transactions that do not contain any item to reduce the number of transactions that need to be traversed.
3. We use patterns at the parent level to compute patterns at the child levels to reduce the number of patterns that need to compute the intersections and reduce the duplications.
4. We develop theorems to prune non-colossal patterns early in the process.
5. We use dynamic bit vectors [17] instead of bit vectors to save memory and computational time.

3.2. CP-Miner algorithm

In this section, we present a new algorithm for mining colossal patterns. First, the downward closure property is used to remove items for which their supports do not satisfy *minsup*. After that, rows that do not contain any item will be removed to reduce the number of transactions (reducing the number of transactions will reduce the search space). Next, we encode the pattern in each transaction using DBV instead of a bit vector, as in BVBUC. To reduce checking duplications, patterns are sorted in descending order according to their length (the number of items in each pattern).

Before giving the definitions, theorems and data structure, we present an example to show the disadvantages of the BVBUC (with *minsup* = 3 according to $\tau = 0.375$, in this case we consider both null items to present the core pattern as in [32]). The process for traversing the tree for mining colossal patterns according to BVBUC is shown in Figure 3.1.

The results from Figure 3.1 show that to obtain three colossal patterns (including patterns (1), (6) and (9)), we must traverse at least 31 nodes (some zero bit vectors are not shown in the tree). From the 10 patterns in level 3, the following issues are raised:

- i) There are a lot of duplications. For example, nodes (2) and (3) have the same patterns; (4), and (5) have the same patterns; node (7) has the same pattern as nodes (8) and (10).
- ii) Superset checking: BVBUC traverses the tree according to the tids order, so it cannot check colossal patterns based on previous patterns. Therefore, some patterns in the results may not be colossal. To remove redundant patterns, when a new pattern Y is inserted into the result set then BVBUC must check it against each pattern X in the result set, and if Y is a superset of X then it is removed from the results. For example: Consider Figure 3.1, pattern 000101 (node (2)) will be inserted into the results first. After that, 011101 (node 6) will be inserted. Because 011101 is a superset of 000101, 000101 is removed before 011101 is inserted.
- iii) Subset checking: When a pattern in the tree is inserted into the result set, BVBUC must check whether it is a subset of any pattern in the results, if not, it is inserted. For example: Assume that nodes (1) and (2) are inserted into the results. Consider node (3): Because (3) is a subset of (2), (3) is not inserted into the results. A similar

process occurs with nodes (4) and (5). Consider node (6): Because (6) is not a subset of any node in the result, it is inserted into the results.

All nodes that contain less than *minsup* transactions are infrequent, and node *X* that contains *minsup* transactions contains the longest pattern if we compare it with all its child nodes (i.e., nodes contain more than *minsup* transactions). Therefore, based on Theorem 2.2, the proposed algorithm would not traverse nodes that have more than *minsup* transactions.

For example: Consider Figure 3.1 with *minsup* = 2 ($\tau = 2/8 = 0.25$), all nodes that contain one transaction are infrequent patterns (Level 1). Consider nodes (1), (2), and (3) in Level 3, these nodes are created from node *X* = 000111 in Level 2 (support = 2):

- With node (1), *Y* = 000110 cannot be a colossal pattern because $Y \subset X$, according to Definition 2.7.
- With node (2), *Y* = 000101 cannot be colossal pattern because $Y \subset X$, according to Definition 2.7.
- With node (3), *Y* = 000101 cannot be colossal pattern because $Y \subset X$, according to Definition 2.7.

3.2.1. CP-tree

In this section, we present a tree structure, named the CP-tree, to store nodes to quickly mine colossal patterns.

Vertex: Each vertex in this tree contains three elements

- tids: A set of transaction IDs.
- pattern: The pattern contained in the set of transaction IDs.
- sup: The support of a pattern.

For example: Consider node (1) in Figure 3.1, this node is created from transactions {2, 3, 4}, its pattern is 000110 and its support is 3.

Arc: Connect from a node *X* at level *k* to node *Y* at level (*k*+1) if the tids of *X* is the prefix of the tids of *Y*.

For example: Node 23×000111 connects to node 235×000110 because tids 23 is the prefix of tids 235.

In fact, tids are used to easily present the arcs of nodes. However, based on Theorem 2.1, the pattern of a new node is created from those of two previous nodes, so we do not need to store them in the tree and can thus save memory.

Figure 3.1. Search tree using BVBC

3.2.2. Algorithm

Input: Transaction database D and core ratio τ .

Output: CP contains colossal patterns.

Method:

1. $minsup = \lceil \tau \times |D| \rceil$;
2. $D' = D$ after filtering items that do not satisfy $minsup$ and removing zero transactions;
3. $[\emptyset] = \{\{t_1, 1\}, \{t_2, 1\}, \dots, \{t_m, 1\}\}$ from D' ;
4. $CP = \emptyset$;
5. CP-Miner($[\emptyset]$, CP , $minsup$);

Procedure CP-Miner($[T]$, CP , $minsup$)

6. If $T.sup = minsup - 1$ then
7. For all n in $[T]$ do
8. If Checking-Colossal($n.pattern$) then
9. Insert $n.pattern$ and $n.sup$ to CP ;
10. Else
11. For all n_i in $[T]$ do
12. If $(n_i.sup + |[T]| - i \geq minsup)$ then
13. $[T1] = \emptyset$;
14. For all n_j in $[T]$ with $j > i$ do
15. $X.pattern = n_i.pattern \cap n_j.pattern$;
16. $X.sup = n_i.sup + 1$;
17. Add X into $[T1]$;
18. CP-Miner($[T1]$, CP , $minsup$);

Checking-Colossal($pattern$)

19. for each n in CP do
20. if $pattern \subseteq n$ then
21. return false;
22. return true;

Figure 3.2. CP-Miner algorithm for mining colossal patterns

Figure 3.2 presents the CP-Miner algorithm for mining colossal patterns that satisfy the core ratio τ in database D . First of all, CP-Miner computes the support of 1-items and removes 1-items for which their support does not satisfy *minsup*. After that, it deletes transactions that do not contain any item (after deleting items that do not satisfy *minsup*, line 2), the resulting database is D' . From D' , CP-Miner creates $[\emptyset]$ (the support of node $\emptyset$ is 0) containing 1-items along with their supports (line 3). CP is initialized by null (line 4), and line 4 computes *minsup* based on the core ratio τ . CP-Miner is then called in line 5 to mine all colossal patterns.

Consider the CP-Miner procedure: Line 6 checks if the support of the child nodes of node T is equal to *minsup*. If it is, the algorithm will add each pattern in these nodes into CP if it is colossal (lines 7-9). Lines 11 to 14 traverse each pair (n_i, n_j) , where n_i, n_j are child nodes of T to create child nodes of n_i , and the pattern of a new node X is the intersection between n_i .pattern and n_j .pattern (line 15), the support of X is the support of $n_i + 1$ (and is the level of X in the CP-tree, line 16), and then add X to CP (line 17). Finally, CP-Miner is called on recursively to create the child nodes of $[T1]$ (line 18).

Note that T is a node in the tree and $[T]$ contains all nodes of the next level with T .tids is the prefix. Each node in $[T]$ will create its child nodes as in the CP-Miner algorithm (Figure 3.2). This is the reason why CP-Miner calls recursively to create children nodes. For example: If node $T = 2 \times 011111$ (Figure 3.1), then $[T] = \{23 \times 000111, 25 \times 000110, 27 \times 011101, 28 \times 011101\}$.

3.2.3. Complexity analysis

For BVBUc, the complexity does not change and it depends on the number of transactions (n) in the database and *minsup*. If pruning is not carried out with *minsup*, the complexity of BVBUc is $\sum_{i=1}^{minsup} C_n^i$.

Let m be the number of transactions in D' , the complexity of CP-Miner is $\sum_{i=1}^{minsup} C_m^i$. m is often smaller than n , so the complexity of CP-Miner is smaller than BVBUc (In the worst-case $m = n$, they have the same complexity). In fact, both BVBUc and CP-Miner use *minsup* to prune some nodes in the search tree.

3.2.4. Illustration

First, because $\text{sup}(G) = \text{sup}(H) = \text{sup}(J) = \text{sup}(K) = \text{sup}(L) = 2$, these items are removed. Table 3.1 shows the results after removing items for which their supports do not satisfy *minsup*.

Table 3.1. Results after removing items for which their supports do not satisfy *minsup*

Tid	Items
1	
2	B C D E F
3	A D E F
4	
5	A D E
6	
7	A B C D F

8	<i>A B C D F</i>
----------	------------------

From Table 3.1, transactions that do not contain any item will be removed, and the results are shown in Table 3.2.

Table 3.2. Results after removing transactions and re-arranging transaction IDs.

Tid	Items
1	<i>B C D E F</i>
2	<i>A D E F</i>
3	<i>A D E</i>
4	<i>A B C D F</i>
5	<i>A B C D F</i>

After this, transactions are encoded as shown in Table 3.3 (in fact, we use DBV to reduce memory usage).

Table 3.3. Bit vectors of transactions in Table 3.2

Tid	Bit vector
1	011111
2	100111
3	100110
4	111101
5	111101

As we can see from Table 3.3, CP-Miner only processes five transactions instead of the eight that BVBUC does, and thus the search space is significantly reduced.

Figure 3.3 illustrates the process of mining colossal patterns using CP-Miner. The number of nodes in the CP-tree is smaller than with BVBUC (at least 31 nodes in Figure 3.1 compared to 24 nodes in Figure 3.3); however, the number of patterns that need subset- and superset-checking is the same, because these two algorithms only check them at the minsup level (level 3, both of them have 10 nodes, as shown in Figure 3.1 and Figure 3.3).

Figure 3.3. Search tree using CP-Miner

(the supports of the patterns are the levels of the nodes in the tree)

3.3. Pruning techniques

3.3.1. PCP-Miner algorithm

Based on Theorems 2.3 and 2.4, we propose an efficient algorithm, named PCP-Miner, for mining colossal patterns. Firstly, we sort transactions in D' in descending order according to their length. After that, we traverse the tree as with CP-Miner. Consider each node n_i with each node n_j following it, and because of sorting the n_i .pattern cannot be a subset of n_j .pattern, and thus there are two cases:

- i) If n_i .pattern $\supseteq n_j$.pattern then, according to Theorem 2.3, insert $\{n_j$.pattern, n_i .sup+1 $\}$ as a subnode of n_i and delete n_j .
- ii) Elseif insert $\{n_j$.pattern $\cap n_i$.pattern, n_i .sup+1 $\}$ as a subnode of n_i .

Besides, when a pattern is inserted into CP, because we sort patterns based on their length, a pattern on the left hand side cannot be subset of all patterns on the right hand side. Therefore, only condition (i) needs checking and we may delete some nodes in the search tree to save memory.

Based on Theorems 2.3 and 2.4, an efficient algorithm named PCP-Miner is proposed in Figure 3.4.

Input: Transaction database D and core ratio τ .

Output: CP contains colossal patterns.

Method:

1. $minsup = \lceil \tau \times |D| \rceil$;
2. $D' = D$ after pruning items that do not satisfy $minsup$ and removing transactions;
3. $[\emptyset] = \{\{t_1, 1\}, \{t_2, 1\}, \dots, \{t_m, 1\}\}$ from D' ;
4. $CP = \emptyset$;
5. PCP-Miner($[\emptyset]$, CP , $minsup$);

Procedure PCP-Miner($[T]$, CP , $minsup$)

6. Sort-Pattern($[T]$);
7. if $T.sup = minsup - 1$ then
8. for all n in $[T]$ do
9. if Checking-Colossal($n.pattern$) then
10. Insert $n.pattern$ and $n.sup$ to CP ;
11. Else
12. for all ni in $[T]$ do
13. if $(ni.sup + |[T]| - i \geq minsup)$ then
14. $[T1] = \emptyset$;
15. for all nj in $[T]$ with $j > i$ do
16. $X.pattern = ni.pattern \cap nj.pattern$;
17. $X.sup = ni.sup + 1$;
18. if $(|X.pattern| = |nj.pattern|)$ then
19. Remove nj from $[T]$;
20. Add X into $[T1]$;
21. PCP-Miner($[T1]$, CP , $minsup$);

Figure 3.4. PCP-Miner algorithm for mining colossal patterns

3.3.2. Complexity analysis

In worst case, i.e., all nodes in CP-tree cannot be pruned by Theorems 2.3 and 2.4, the complexity of PCP-Miner is the same that of CP-Miner. However, in case $t_1 \supseteq t_2 \supseteq \dots \supseteq t_m$, the result is only t_1 and therefore all its following nodes are removed by Theorem 2.3. In this case, the complexity of PCP-Miner is only *minsup*.

3.3.3. Illustration

Consider the example database in Table 3.3, after transactions are sorted according to their length, we have the results as shown in Table 3.4.

Table 3.4. Bit vectors of transactions after sorting

Tid	Bit vector
1	011111
2	111101
3	111101
4	100111
5	100110

The process for mining colossal patterns based on PCP-Miner is as follows.

First, level 1 of CP-tree contains five bits vectors from Table 3.4, as shown in Figure 3.5.

Consider the process of extending node $n_i = \{011101, 2\}$ (From Figure 3.5) to find colossal patterns:

- Joins with node $n_j = \{011101, 2\}$: Because $011101 \cap 011101 = 011101$, remove node n_j and add node $X = \{011101, 3\}$ into level 3.
- Joins with node $n_j = \{000111, 2\}$: Because $011101 \cap 000111 = 000101$, add node $X = \{000101, 3\}$ into level 3.
- Joins with node $n_j = \{000110, 2\}$: Because $011101 \cap 000110 = 000100$, add node $X = \{000100, 3\}$ into level 3.

Similar with node $n_i = \{000111, 2\}$: Join with node $n_j = \{000110, 2\}$, we get node $\{000110, 3\}$. Because 000110 is the same n_j .pattern, remove n_j and add node $\{000110, 3\}$ into level 3. The tree after considering these two nodes is shown in Figure 3.6.

Figure 3.5. CP-tree after extending to level 2 of node {011111, 1}.

The final CP-tree is shown in Figure 3.7. The number of nodes in Figure 3.7 is 13, which is very small compared to that seen with CP-Miner and BVBUC (24 and 31, respectively). Besides this, the number of nodes at level 3 (i.e., the number of nodes that need to be checked with patterns in CP) is only six (while BVBUC and CP-Miner is 10). Moreover, adding these patterns into CP is also more efficient than with CP-Miner and BVBUC. Consider the process of adding these patterns into CP :

Figure 3.6. CP-tree after extending to level 3 of node {011111, 1}.

- Consider pattern 011101: Add into *CP*.
- Consider pattern 000101: Because 000101 is a subset of 011101 (in *CP*), 000101 is not added into *CP*.
- Consider pattern 000100: Because 000100 is a subset of 011101 (in *CP*), 000100 is not added into *CP*.
- Consider pattern 000110: Add 000110 into *CP*.
- Consider pattern 100101: Add 100101 into *CP*.
- Consider pattern 100100: Because 100100 is a subset of 100101 (in *CP*), 100100 is not added into *CP*.

Figure 3.7. Final CP-tree of the PCP-Miner algorithm.

3.4. Experiments

3.4.1. Experimental environment and databases

All experiments presented in this section were performed on a PC with an Intel Core i5 3.2 GHz, and 4GB of Ram, running on Windows 7 operating system, with the Visual C# 2010. We made experiments on five databases, as shown in Table 3.1, which were downloaded from <http://fimi.cs.helsinki.fi/data/>.

Table 3.5. Characteristics of the experimental databases

Database	# of items	# of transactions
Accidents	468	340,183
Connect	130	67,557
Retails	16,470	88,162
Pumsb*	7,117	49,046
T10I4D100K	1,000	100,000

3.4.2. Comparison of the runtimes

We choose the first 50, 100, and 150 transactions in each database, as shown in Table 3.5, to compare the performances of the algorithms in high dimensional databases, with Figure 3.8 to Figure 3.22 showing the runtimes for BVBUc, CP-Miner and PCP-Miner.

- i) The results show that CP-Miner and PCP-Miner are always faster than BVBUC in all experiments. For dense databases such as Connect, Accidents, and Pumsb*, CP-Miner and PCP-Miner are very efficient compared to BVBUC. BVBUC runs very slowly (786 (s) with $(\#Rows, minsup) = (150, 4)$ in Connect (Figure 3.16 left) or more than 1000 (s) in Accidents with $(\#Rows, minsup) = (50, 5)$ (Figure 3.11 left), (100, 4) (Figure 3.12 left) and (150, 4) (Figure 3.13 left) in Pumsb* with (150, 4) (Figure 3.19 left). For sparse databases, such as Retails and T10I4D100K, CP-Miner and PCP-Miner are also faster than BVBUC.
- ii) PCP-Miner is more efficient than CP-Miner in all experiments. However, we only run these experiments with a number of small transactions and items, so the gap between CP-Miner and PCP-Miner is not very large (we will compare the scalability of these two algorithms in Section 3.4.4).

Figure 3.8. Runtime in the Retails database when the number of transactions is 50
(the number of items is 275).

Figure 3.9. Runtime in the Retails database when the number of transactions is 100
(the number of items is 532)

Figure 3.10. Runtimes in the Retails database when the number of transactions is 150
(the number of items is 773)

Figure 3.11. Runtimes in the Accidents database when the number of transactions is 50
(the number of items is 151).

Figure 3.12. Runtimes in the Accidents database when the number of transactions is 100
(the number of items is 168).

Figure 3.13. Runtimes in the Accidents database when the number of transactions is 150 (the number of items is 177).

Figure 3.14. Runtimes in the Connect database when the number of transactions is 50 (the number of items is 65).

Figure 3.15. Runtimes in the Connect database when the number of transactions is 100 (the number of items is 69).

Figure 3.16. Runtimes in the Connect database when the number of transactions is 150 (the number of items is 75).

Figure 3.17. Runtimes in the Pumsb* database when the number of transactions is 50 (the number of items is 344).

Figure 3.18. Runtimes in the Pumsb* database when the number of transactions is 100 (the number of items is 456).

Figure 3.19. Runtimes in the Pumsb* database when the number of transactions is 150 (the number of items is 574).

Figure 3.20. Runtimes in the T10I4D100K database when the number of transactions is 50 (the number of items is 335).

Figure 3.21. Runtimes in the T10I4D100K database when the number of transactions is 100 (the number of items is 470).

Figure 3.22. Runtimes in the T10I4D100K database when the number of transactions is 150 (the number of items is 554).

3.4.3. Comparison of the number of nodes in the trees

We also compare the number of nodes in the search space (using BVBUc, CP-Miner and PCP-Miner). Table 3.6 shows that the search space of BVBUc is very large (for example, the highest number of nodes in the search space for Retails, Connect and T10I4D100K is more than 470 million), while the search space of PCP-Miner is very small (for example, consider Connect with $(\#Rows, minsup) = (150, 4)$, the number of nodes of BVBUc is 470,169,066 while that of PCP-Miner is 119,076). Similarly, the search space of CP-Miner is smaller than that of BVBUc in all experiments.

Table 3.6. Comparison of the number of nodes

Database	#Rows	minsup	BVBUc	CP-Miner	PCP-Miner
Retails	50	3	112,994	6,735	185
		4	4,988,066	48,884	232
		5	210,641,640	276,018	309
	100	2	9,901	2,492	1,226
		3	950,994	44,673	926
		4	89,461,066	590,114	609
	150	2	22,351	5,806	3,025
		3	3,263,994	162,501	2,924
		4	470,169,066	3,390,976	1,996
Accidents	50	3	112,994	20,875	14,704
		4	4,988,066	251,174	82,061
		5	#NA	2,369,884	323,455
	100	2	9,901	5,050	5,050

		3	950,994	166,750	97,209
		4	#NA	4,087,974	717,911
	150	2	22,351	11,325	11,325
		3	3,263,994	552,625	282,457
		4	#NA	20,822,899	2,293,813
Connect	50	2	2,451	1,275	1,275
		3	112,994	20,875	4,745
		4	4,988,066	251,174	10,987
	100	2	9,901	5,050	5,050
		3	950,994	166,750	18,745
		4	89,461,066	4,087,974	43,405
	150	2	22,351	11,325	11,325
		3	3,263,994	562,625	46,645
		4	470,169,066	20,822,899	119,076
Pumsb*	50	2	2,451	1,275	1,275
		3	112,994	20,703	10,330
		4	4,988,066	236,988	37,168
		5	210,641,640	2,013,532	92,279
	100	2	9,901	5,050	4,974
		3	950,994	164,477	55,045
		4	89,461,066	3,746,281	252,885
	150	2	22,351	11,325	11,216
		3	3,263,994	554,700	156,112
		4	#NA	19,081,259	795,839
T10I4D100K	50	2	2,451	301	298
		3	112,994	376	317
		4	4,988,066	312	144
		5	210,641,640	223	71
	100	2	9,901	1,020	1,995
		3	950,994	2,017	1,431
		4	89,461,066	2,747	1,350

	150	2	22,351	2,164	2,081
		3	3,263,994	5,681	3,373
		4	470,169,066	10,527	3,889

The comparison of the number of nodes by log scale chart of BVBUC, CP-Miner, and PCP-Mine algorithms are shown in Figure 3.23 - Figure 3.37.

Figure 3.23. The log scale of Retail dataset with 50 transactions

Figure 3.24. The log scale of Retail dataset with 100 transactions

Figure 3.25. The log scale of Retail dataset with 150 transactions

Figure 3.26. The log scale of Accidents dataset with 50 transactions

Figure 3.27. The log scale of Accidents dataset with 100 transactions

Figure 3.28. The log scale of Accidents dataset with 150 transactions

Figure 3.29. The log scale of Connect dataset with 50 transactions

Figure 3.30. The log scale of Connect dataset with 100 transactions

Figure 3.31. The log scale of Connect dataset with 150 transactions

Figure 3.32. The log scale of Pumsb* dataset with 50 transactions

Figure 3.33. The log scale of Pumsb* dataset with 100 transactions

Figure 3.34. The log scale of Pumsb* dataset with 150 transactions

Figure 3.35. The log scale of T10I4D100K dataset with 50 transactions

Figure 3.36. The log scale of T10I4D100K dataset with 100 transactions

Figure 3.37. The log scale of T10I4D100K dataset with 150 transactions

3.4.4. Scalability of CP-Miner and PCP-Miner

We increase the number of transactions (m) to examine the scalability of two of the proposed algorithms (BVBUC cannot run with these cases), and the results are shown in Figure 3.38 - Figure 3.40.

First, we run two algorithms with $m = 500$ (the first 500 transactions contain 2,058 items). Figure 3.38 shows that CP-Miner only runs to $minsup = 4$, when we increase $minsup$ it runs very slowly (more than 3000 (s)) while the run time of the PCP-Miner algorithm does not change much (from 0.023 (s) to 0.109 (s)). Next, we increase m to 1,000 (and the number of items is 3,182), CP-Miner also cannot run with $minsup$ greater than 4, while PCP-Miner can run to $minsup = 30$. Besides, when we increase $minsup \{3, 4, 5, 10, 20, 30\}$, the runtime decreases $\{1.068(s), 0.913(s), 0.776(s), 0.445(s), 0.113(s), 0.112(s)\}$. This is because PCP-Miner prunes a lot of nodes in the CP-tree. Finally, we increase m to 2,000 (the number of items is 4,775), and CP-Miner also cannot run with $minsup$ greater than 2, while the runtime of PCP-Miner increases to $\{10.49(s), 10.766(s), 10.791(s)\}$ at $minsup = \{2, 3, 4\}$. However, when we increase $minsup$ to $\{5, 10, 20, 30\}$, the runtime decreases significantly, as shown in Figure 3.40. The reason is that when $minsup$ is large, the algorithm can prune more patterns than when the $minsup$ is small (by Line 8, Figure 3.4). Besides, when the level is high, PCP-Miner prunes more nodes (by Theorem 4) than when the level is low. Therefore, at the maximal level ($minsup$), the number of nodes is small, leading to the number of nodes that need to check colossal (by Line 4, Figure 3.4) is small as well.

As we see in Figure 3.40, CP-Miner cannot run with $minsup > 2$ when we increase the number of transactions to 2,000. Therefore, we do not increase the number of transactions more than 2,000.

Figure 3.38. Runtimes in the Retails database when the number of transactions is 500
(the number of items is 2,058).

Figure 3.39. Runtimes in the Retails database when the number of transactions is 1,000
(the number of items is 3,182).

Figure 3.40. Runtimes in the Retails database when the number of transactions is 2,000

(the number of items is 4,775).

Chapter 4. Mining Colossal Patterns with Pattern Constraint

Constraint-based methods for mining patterns have been developed in recent years. They are based on top-down manner to prune candidate patterns. However, for colossal pattern mining, bottom-up manners are efficient methods, so the previous approaches for pruning candidate patterns based on top-down manner cannot apply to colossal pattern mining with constraint when using bottom-up manner. In this section, we develop a theorem for efficient pruning candidate patterns with bottom-up manner and propose an efficient algorithm for mining colossal patterns with pattern constraints based on this theorem.

4.1. Introduction

Frequent pattern mining (FPM) has been a vital subject in the field of data mining first introduced by Agarwal [1] and has been a focused theme in research for over three previous decades. Frequent patterns (FPs) are referred to itemsets, subsequences, or substructures whose appearance frequency in the target dataset is not less than a user-specified threshold value. FPM plays a fundamental role in association rule mining [38, 39], correlation mining [40] [41], sequential pattern mining [42, 43], episode mining [44], partial periodicity [45], classification [46, 47], clustering [48], etc. FPM has multiple applications such as market basket analysis [49], weblog analysis [10, 50], DNA sequence analysis [7, 51, 52, 53] and prediction [54].

Although numerous efficient algorithms have been proposed to solve frequent pattern mining or some of its variants, and interest in this problem remains [55], especially for FPM on huge database.

One of the main reasons for the high level of interest in FPM algorithms is due to the computational challenge of the task. The search space of FPM is enormous, which is exponential to the length of the transactions in the dataset. This is challenges for itemset generation when the support levels are low.

Although there are a lot of algorithms developed to solve the problem of mining colossal patterns, no algorithm is developed for mining colossal patterns with constraints. In fact, the users can put their constraints to mine the patterns that they concern. For example, they may want to mine patterns that contain an item in a set of items or patterns that contain a pattern.

There are many methods that can be used to mine frequent patterns with pattern constraints [14, 15, 16, 17], mine class association rules with itemset constraints [18, 19] and class constraints [20, 21], and mine erasable itemsets with subset and superset constraints [22]. However, all of them are based on top-down manner to prune candidate patterns. Therefore, proposed strategies in these works cannot apply to colossal pattern mining with pattern constraint using bottom-up manner.

In this chapter, we propose a method for mining colossal patterns with pattern constraint. The main contributions of this paper are as follows:

1. We state the problem of mining colossal patterns with pattern constraint.
2. A theorem is developed to prune nodes in CP-tree.

- Based on the theorem, we propose an algorithm for fast mining colossal patterns with pattern constraint.

4.2. Method for mining pattern with pattern constraint

There are three approaches for mining FPs with constraints: The first approach is called post-processing method. They are based on Apriori [18], Eclat [21] or FP-Growth [22] to mine all FPs and then filter out the FPs satisfy the constraint. This approach generates many candidates, meaning that it is time-consuming. In the real world, the users are interesting in a small set of FPs which satisfy conditions. The second approach is called pre-processing. In this approach, firstly, they discarded all the records do not satisfy the constraint. Then, based on the rest of database they mine all FPs. For example, dataset filtering [65] and MCFPTree [66]. And, the last approach is called constraint FPs. The authors integrated constraint into the mining process so that it was generate all the FPs which satisfy the constraint. Example CAP [67] and MINE_FS_CONS [68]. This approach can mine enough the constraint FPs for end users.

Table 4.1. An example database

Tid	Items
1	B E F H G K
2	B C D E F
3	A D E F L
4	G K
5	A D E J
6	H K L
7	A B C D E G K
8	A B C D E F H

Table 4.2. Colossal patterns with DE-constraint ($\tau = 0.25$)

#	Colossal pattern
1	CDE
2	DEF
3	CDEF
4	ADEF

4.3. Proposed algorithm

In this section, we develop a theorem to fast prune candidate patterns that cannot satisfy pattern constraint. After that, we propose an efficient algorithm for mining colossal patterns with pattern constraint.

Definition 4.1 (Colossal pattern with pattern constraint). Given a transaction dataset D , a core ratio τ , and a pattern X , mining colossal patterns with pattern constraint is to mine all colossal patterns in D that their patterns contain X .

Theorem 4.1 [23]. If pattern X of a node is subsumed by any colossal pattern, then all patterns created from this node cannot be colossal patterns.

Theorem 4.2. Given a node n in the CP-tree, if $n.pattern$ does not satisfy pattern constraint then all its child nodes cannot satisfy pattern constraint.

Proof: According to Definition 4.1 and the hypothesis, $n.pattern$ does not contain X , i.e. $n.pattern \cap X \subset X$. Besides, Y is a child node of n then $Y \subset n.pattern$. It leads to $Y \cap X \subset X$ or Y cannot satisfy pattern constraint.

For example: Consider $X = CDE$ from the database in Table 4.1. When we join two nodes $1 \times BEFHGK$ and $2 \times BCDEF$ into node $12 \times BEF$, because BEF does not satisfy X (i.e., BEF does not contain X) $\Rightarrow$ all child nodes of $12 \times BEF$ do not satisfy X -constraint.

Input: Transaction database D , core ratio τ , and pattern constraint X .

Output: Colossal patterns CP that satisfy X .

Method:

1. $minsup = \lceil \tau \times |D| \rceil$;
2. $D' = D$ after filtering items that do not satisfy $minsup$, removing zero transactions and removing items that cannot satisfy X ; // by Theorem 4.2
3. $[\emptyset] = \{\{t_1, 1\}, \{t_2, 1\}, \dots, \{t_m, 1\}\}$ from D' ;
4. $CP = \emptyset$;
5. CPCP-Miner($[\emptyset]$, CP , $minsup$);

Procedure CPCP-Miner($[T]$, CP , $minsup$)

6. Sort-Pattern($[T]$);
7. If $T.sup = minsup - 1$ then
8. For all n in $[T]$ do
9. If Checking-Colossal($n.pattern$) then
10. Insert $n.pattern$ and $n.sup$ to CP ;
11. Else
12. For all n_i in $[T]$ do
13. If $(n_i.sup + |[T]| - i \geq minsup)$ then
14. $[T1] = \emptyset$;
15. For all n_j in $[T]$ with $j > i$ do
16. $Y.pattern = n_i.pattern \cap n_j.pattern$;
17. $Y.sup = n_i.sup + 1$;
18. If $(|Y.pattern| = |n_j.pattern|)$ then
19. Remove n_j from $[T]$; // by Theorem 4.1
20. If $(X \subseteq Y.pattern)$ then // by Theorem 4.2
21. Add Y into $[T1]$;
22. CPCP-Miner($[T1]$, CP , $minsup$);

Figure 4.1. CPCP-Miner algorithm for mining colossal patterns with pattern constraint

Figure 4.1 presents the CPCP-Miner algorithm for mining colossal patterns that satisfy the core ratio τ and pattern X . Basically, it is based on PCP-Miner [56]. Main different of these two algorithms is in Lines 2 and 15. In these lines, CPCP-Miner is based on Theorem 4.2 to prune nodes early.

4.4. Experiments

4.4.1. Experimental environment and databases

All experiments presented in this section were performed on a PC with an Intel Core i5 3.2 GHz, and 4GB of Ram, running on Windows 7 operating system, with the Visual C# 2010. We made experiments on five databases, as shown in Table 3.5, which were downloaded from <http://fimi.cs.helsinki.fi/data/>.

4.4.2. Comparison of the runtimes

Figure 4.2 to Figure 4.5 show the runtimes for CPCP-Miner (PCP-Miner with post processing) and CPCP-Miner.

Figure 4.2. Comparison of two methods in Retails database with the number of transactions is 2000

Figure 4.3. Comparison of two methods in Accidents database with the number of transactions is 150

Figure 4.4. Comparison of two methods in Connect database with the number of transactions is 150

Figure 4.5. Comparison of two methods in Pumsb* database with the number of transactions is 150

Figure 4.2 to Figure 4.5 present the results from experimental databases with some $minsup$ values (We random items in X with $|X| = 3$ for all experiments). Results show that CPCP-Miner is always more efficient than PCP-Miner with post-processing. For example, consider Accidents database with $minsup = 4$, the number of transactions is 150 and the number of items is 773, the runtime of CPCP-Miner is 141.9 s while that of PCP-Miner with post-processing is 223.8 s.

Chapter 5. Mining Colossal Patterns with Length Constraints

There are many efficient approaches for extracting colossal patterns have proposed. However, the approaches cannot apply to colossal pattern mining with constraints. In the previous chapter, we develop a method for mining colossal patterns with itemset constraints. In this chapter, we continually develop theorems for efficient pruning and fast checking candidates. Based on these theorems, we propose LCCP (Length Constraints for Colossal Patterns) algorithm for mining colossal patterns with length constraints.

5.1. Introduction

Many algorithms for mining colossal patterns were proposed such as the Pattern-Fusion [16], the CMP (Colossal Pattern Miner) [33], BVBUC [32], CP-Miner and PCP-Miner [56]. The Pattern-Fusion algorithm conquers the enormous search area by only mining K good frequent patterns. The BVBUC [32] proposed for mining colossal patterns in high dimensional databases which uses a bottom-up scheme based on transactions. Although BVBUC is better than CMP and Pattern-Fusion, it still exists some drawbacks. So, two methods named CP-Miner and PCP-Miner [56] were developed to timesaving for mining colossal patterns. CP-Miner is based on CP-tree to efficient mine colossal patterns while PCP-Miner is based on CP-Miner and pruning strategies to efficient prune nodes in CP-tree.

In many problems, the users need to enter the constraints based on different problem required to mine the patterns that they concern. For example, the user wants to mine patterns that comprise an item, patterns that contain an itemset or patterns with their length is smaller than a threshold, etc.

There are many methods that can be used to mine frequent patterns with pattern constraints [14, 15, 16, 17], mine class association rules with itemset constraints [24, 18, 19] and class constraints [20, 21], mine erasable itemsets with subset and superset constraints [22]. However, the top-down strategies to prune non-colossal patterns cannot be applied to colossal pattern mining with pattern constraints.

In 2017, Nguyen et al. proposed a method for mining patterns with the itemset constraint. The authors proposed CPCP-tree for quick mining colossal patterns based on itemset constraint [72].

In this chapter, we propose an efficient method for mining colossal patterns with length constraints (min-length and max-length). The main contributions of this paper are as follows:

1. Firstly, we describe the problem of mining colossal patterns with length constraints.
2. Next, a theorem based on max length is developed to early prune candidates.
3. Based on the theorem, a length constraint-based method for quick mining colossal patterns is developed.
4. Final, evaluation of the proposed method compared to previous methods in terms of mining time and memory usage on various real-life databases.

5.2. Mining patterns with constraints

Many methods for mining FPs with constraints was developed: The first strategy based on Apriori [18], Eclat [21] or FP-Growth [22] to mine all FPs satisfy the constraint is called post-processing method. This strategy has high time-consuming because of many unnecessary candidates was generated. In the real world, the users are interesting in a small set of FPs which satisfy conditions. The second strategy includes two phases is called pre-processing. They discarded all the records that do not satisfy the constraint firstly and then it mines all FPs from the rest database. For example, database filtering [73] and MCFPTree [74]. The last approach is known as constraint FPs. This strategy integrated constraint into the mining process so that it generated all the FPs which satisfy the constraint.

Methods for mining patterns with constraints have been suggested for many different types of constraints, with regard to event occurrence, timing, data, and combinations of these, as used in real-world monitoring systems [25] which automatically mine constraint candidates from event logs to monitor runtime.

The MPP [26] algorithm for mining constraint sequence patterns uses a multi-valued decision diagram (MDD) within a prefix projection algorithm, that compactly encodes the sequence of items and their attributes by leveraging symmetry. The algorithm is more efficient than a prefix-projection algorithm with constraint checks. The power of MPP becomes greater as we increase the size of the database, the number of constraints, or the number of attributes.

Recently, a query-constraint-based ARM (QARM) [77] approach for exploratory analysis of multiple, diverse clinical datasets. QARM enables rule mining on a subset of data items satisfying a query constraint. The algorithm uses the Top-k Non-Redundant (TNR) algorithm to generate association rules and then remove general and subsumed rules so that unique and non-redundant rules have resulted in a particular query constraint QARM. This is a useful method to reduce the number of uninteresting association rules generated from imbalanced datasets.

A constraint programming model [78] for mining association rules is proposed, this model can easily be extended to extract different types of rules. It can apply to any kind of user's constraint, such as mandatory or forbidden items, cardinality, etc.

Next, two algorithms named MWAPC and EMWAPC [50] to mining web access patterns with super-pattern constraint by applying the dynamic bit vector structure based on the PreWAP tree for quick pruning and reducing checking, that prunes the search space at the beginning of mining process and avoids checking the constraints one by one, each frequent pattern has to be checked if it contains at least one pattern from a user-defined set of patterns.

In addition, the CPCP-Miner [72] algorithm based on CPCP-tree structure was developed to mining colossal patterns with itemset constraint. This algorithm developed a theorem for fast eliminate non-colossal pattern candidates with constraint.

Although, there are existing many methods for mining patterns with constraint. However, in many problems, the users need to enter the constraints based on different problems required

to mine the patterns that they concern. In this section, we statement the problem of mining colossal pattern with length constraints, we proposed some theorems and a new method for effective mining colossal pattern with min and max length constraint.

5.3. Proposed algorithm

In this section, we develop a theorem (Theorem 5.1) to quickly eliminate candidate patterns that cannot gratify min-length constraint and a theorem (Theorem 5.2) to fast check candidates with max-length constraint. After that, we propose an efficient algorithm for mining colossal patterns with length constraints.

Definition 5. 1 (Colossal pattern) [27]. A pattern X is called a colossal pattern in a set of frequent patterns if and only if there does not appear an itemset Y such that $X \subset Y$ and X is a τ -core pattern of Y .

Given a transaction database D , a core ratio τ , a min-length threshold ξ , and a max-length threshold Ω . The problem of mining colossal patterns with length constraints is stated as Definitions 5.2 - 5.4.

Definition 5.2 (Colossal pattern with min-length constraint). Mining colossal patterns with min-length constraint in database D is extracting all colossal pattern X in D that $|X| \geq \xi$

Definition 5.3 (Colossal pattern with max-length constraint). Mining colossal patterns with max-length constraint in database D is extracting all colossal pattern Y in D that $|Y| \leq \Omega$.

Based on the Definitions 5.1, 5.2 and 5.3, we state the problem of mining colossal patterns with length constraints in Definition 5.4.

Definition 5.4 (Colossal pattern with length constraints). Mining colossal patterns with length constraints is to find all patterns in D that satisfy max-length constraint and min-length constraint.

Example 5.1: Consider an example database shown in Table 5.1 with $\tau = 0.25$, $\xi = 3$ and $\Omega = 5$, we have colossal patterns with length constraints as shown in Table 5.2.

Table 5.1. A transaction database

Tid	Items
1	A J K L
2	B C D E G H J
3	B C E F G H J
4	B C D E F G H J
5	B E F G H J
6	B H K
7	B D F G H
8	E F G H J
9	F G H
10	B C D E G H
11	C F K

12	A G K L
----	---------

Table 5.2. Colossal patterns with length constraints

#	Colossal patterns satisfy length constraints
1	BEGHJ
2	BGH
3	EGHJ
4	BCEGH
5	BDGH
6	BFGH
7	EFGHJ
8	FGH

Definition 5.5 (Colossal Pattern (CP)-tree). According to [39], a CP-tree is defined as follow:

23. Vertex: Each vertex in this tree contains three elements

- tids: A set of transaction IDs.
- pattern: The pattern contained in the set of transaction IDs.
- sup: The support of a pattern.

Arc: Connect from a node X at level k to nodes Y at level $(k+1)$ if the tids of X is the prefix of the tids of Y .

Theorem 5.1 (min-length constraint-based for pruning candidates). if a node n in the CP-tree does not satisfy min-length constraint then all its child nodes cannot satisfy min-length constraint.

Proof: According to Definition 5.2, n does not satisfy the min-length constraint, i.e. $|n.pattern| < \xi$. Besides, n_1 is a child node of n then $|n_1.pattern| \leq |n.pattern|$. Therefore, $|n_1.pattern| < \xi$ or n_1 cannot satisfy min-length constraint.

Based on Theorem 5.1, we need not expand a node if it does not satisfy min-length constraint. Therefore, we can prune the exploration area in the mining process.

Theorem 5.2 (max-length constraint-based for checking candidates). if a node n in the CP-tree satisfies the max-length constraint then all its child nodes satisfy the max-length constraint.

Proof: According to Definition 5.3, n satisfies the max-length constraint, i.e. $|n.pattern| \leq \Omega$. Besides, n_1 is a child node of n then $|n_1.pattern| \leq |n.pattern|$. Therefore, $|n_1.pattern| \leq \Omega$ or n_1 satisfies max-length constraint.

Based on Theorem 5.2, we need not check all child nodes of a node n if it satisfies max-length constraint. In this case, we can save time to check many nodes in CP-tree.

We suggest an efficient method, named LCCP, for quickly mining colossal patterns with length constraints (min-length and max-length). LCCP is based on PCP-Miner [28] to efficiently generate and prune colossal pattern candidates. It is also based on Theorem 5.1 to prune candidates that cannot meet the min-length constraint and based on Theorem 5.2 to eliminate candidates that do not need to have the max-length constraint checked.

Input: Transaction database D , core ratio τ , min-length ξ and max-length Ω .

Output: Colossal patterns CP that satisfy length constraints.

Method:

1. $\phi = \lceil \tau \times |D| \rceil$;
2. $D' = D$ after filtering items that do not satisfy ϕ , removing transactions that do not satisfy ξ ;
3. Initial the $[\emptyset] = \{\{t_1, 1\}, \{t_2, 1\}, \dots, \{t_m, 1\}\}$ from D' ;
4. $CP = \emptyset$;
5. LCCP($[\emptyset]$, CP , ϕ , ξ , Ω);

Procedure LCCP ($[T]$, CP , ϕ , ξ , Ω)

6. Sort-Pattern($[T]$);
7. If $T.\text{sup} = \phi - 1$ then
8. For all n in $[T]$ do
9. If ($|n.\text{pattern}| \leq \Omega$ and isColossal($n.\text{pattern}$)) then
10. Insert $n.\text{pattern}$ and $n.\text{sup}$ to CP ;
11. Else
12. For all n_i in $[T]$ do
13. If ($n_i.\text{sup} + |[T]| - i \geq \phi$) then
14. $[T1] = \emptyset$;
15. For all n_j in $[T]$ with $j > i$ do
16. $n.\text{pattern} = n_i.\text{pattern} \cap n_j.\text{pattern}$;
17. $n.\text{sup} = n_i.\text{sup} + 1$;
18. If ($|n.\text{pattern}| = |n_j.\text{pattern}|$) then
19. Remove n_j from $[T]$;
20. If ($|T.\text{pattern}| \geq \xi$) then

```

21.          Add Y into [T1];
22.      If ( $|n_i.pattern| \leq \Omega$ ) then
23.          LeCCP([T1], CP,  $\varphi$ ,  $\xi$ );
24.      Else
25.          LCCP([T1], CP,  $\varphi$ ,  $\xi$ ,  $\Omega$ );

Procedure LeCCP ([T], CP,  $\varphi$ ,  $\xi$ )
26.  Sort-Pattern([T]);
27.  If T.sup =  $\varphi-1$  then
28.      For all n in [T] do
29.          If (isColossal(n.pattern)) then
30.              Insert n.pattern and n.sup to CP;
31.  Else
32.      For all  $n_i$  in [T] do
33.          If ( $n_i.sup + |[T] - i \geq \varphi$ ) then
34.              [T1] =  $\emptyset$ ;
35.              For all  $n_j$  in [T] with  $j > i$  do
36.                   $n.pattern = n_i.pattern \cap n_j.pattern$ ;
37.                   $n.sup = n_i.sup + 1$ ;
38.                  If ( $|n.pattern| = |n_j.pattern|$ ) then
39.                      Remove  $n_j$  from [T];
40.                  If ( $|T.pattern| \geq \xi$ ) then
41.                      Add Y into [T1];
42.                  LeCCP([T1], CP,  $\varphi$ ,  $\xi$ );

```

Figure 5.1. LCCP algorithm for mining colossal patterns with length constraints

Figure 5.1 presents the LCCP algorithm for mining colossal patterns with length constraints. It is expended from PCP-Miner [56] for efficient generating candidates. Theorem 5.1 is used to prune candidates (Line 2, Lines 6-7 and 28-29). In line 2, LCCP uses Theorem 5.1 to remove all transactions that their length is less than ξ . Lines 6, 28 check the condition of Theorem 5.1, if it is true then pruning all child nodes of T (Lines 7 and 29). Line 24 checks the condition of Theorem 5.2, if it is true, LCCP is called (Line 25) to mine colossal patterns with min-length

constraint (It means we need not to check the max-length constraint). The rest of LCCP and LeCCP algorithms is the same with PCP-Miner [56].

5.4. An example

Example 5.2: We illustrate LCCP algorithm based on the information given in Example 5.1. Firstly, the support of each item in Table 5.1 is calculated and results are shown in Table 5.3.

Table 5.3. Items and their support

Item	A	B	C	D	E	F	G	H	J	K	L
support	2	7	5	4	6	7	9	9	6	4	2

With $\phi = 3$ ($= \lceil 0.25 \times 12 \rceil$), we can see that A and L do not satisfy ϕ , so they are filtered. The database in Table 5.1 after filters items that do not satisfy ϕ is shown in Table 5.4.

Table 5.4. A transaction database after filtering items

Tid	Items
1	J K
2	B C D E G H J
3	B C E F G H J
4	B C D E F G H J
5	B E F G H J
6	B H K
7	B D F G H
8	E F G H J
9	F G H
10	B C D E G H
11	C F K
12	G K

The number of items in Transactions 1 and 12 is 2, it is smaller than ξ (According to Example 5.1, $\xi = 3$), these two transactions are removed. The results are shown in Table 5.5.

Table 5.5. A transaction database after removing transactions

Tid	Items
2	B C D E G H J
3	B C E F G H J
4	B C D E F G H J
5	B E F G H J
6	B H K
7	B D F G H
8	E F G H J
9	F G H
10	B C D E G H
11	C F K

Table 5.6 presents the support of items after removing transactions.

Table 5.6. Items and their support after removing transactions

Item	B	C	D	E	F	G	H	J	K
support	7	5	4	6	7	8	9	5	2

K is deleted because its support is 2. After K is deleted, transactions 6 and 11 are removed and we have results shown in Table 5.7.

Table 5.7. A transaction database after removing transactions, items and rearranged Tid

Tid	Items
1	B C D E G H J
2	B C E F G H J
3	B C D E F G H J
4	B E F G H J
5	B D F G H
6	E F G H J
7	F G H
8	B C D E G H

LCCP uses the database in Table 5.7 to build CP-tree as shown in Figure 5.2, Figure 5.3 and Figure 5.4.

Figure 5.2. CP-tree after expending node {BCDEGHJ}

Firstly, level 1 of CP-tree will be created from 8 transactions from Table 5.7 as shown in Figure 5.2. Consider node {BCDEGHJ}, it will join with node following it to create new nodes in the tree:

- With node {BCEFGHJ}: create node {BCEGHJ} at level 2 (support = 2) $\Rightarrow [T1] = \{\{BCEGHJ\}\}$.
- With node {BCDEFGHJ}: create node {BCDEGHJ} at level 2 (support = 2) $\Rightarrow [T1] = \{\{BCEGHJ\}, \{BCDEGHJ\}\}$.
- With node {BEFGHJ}: create node {BEGHJ} at level 2 (support = 2) $\Rightarrow [T1] = \{\{BCEGHJ\}, \{BCDEGHJ\}, \{BEGHJ\}\}$.
- With node {BDFGH}: create node {BDGH} at level 2 (support = 2) $\Rightarrow [T1] = \{\{BCEGHJ\}, \{BCDEGHJ\}, \{BEGHJ\}, \{BDGH\}\}$.
- With node {EFGHJ}: create node {EGHJ} at level 2 (support = 2) $\Rightarrow [T1] = \{\{BCEGHJ\}, \{BCDEGHJ\}, \{BEGHJ\}, \{BDGH\}, \{EGHJ\}\}$.

- With node {FGH}: create node {GH} at level 2 (support = 2). Because $|GH| = 2 < \xi = 3 \Rightarrow GH$ does not add to $[T1]$ (by Theorem 5.1).
- With node {BCDEGH}: create node {BCDEGH} at level 2 (support = 2) $\Rightarrow [T1] = \{\{BCEGHJ\}, \{BCDEGHJ\}, \{BEGHJ\}, \{BDGH\}, \{EGHJ\}, \{BCDEGH\}\}$. Because pattern of the new node is the same with {BCDEGH} $\Rightarrow$ Remove {BCDEGH} from the Level 1 (by Theorem 2.4).

After creating $[T1] = \{\{BCEGHJ\}, \{BCDEGHJ\}, \{BEGHJ\}, \{BDGH\}, \{EGHJ\}, \{BCDEGH\}\}$ for node {BCDEGHJ}, the algorithm will call recursive to expend child nodes of the nodes in $[T1]$. Consider node {BCEGHJ} with all nodes following it in $[T1]$ (Figure 5.3):

- With node {BCDEGHJ}: create node {BCEGHJ} at level 3 (support = 3) $\Rightarrow [T1] = \{\{BCEGHJ\}\}$.
- With node {BEGHJ}: create node {BEGHJ} at level 3 (support = 3) $\Rightarrow [T1] = \{\{BCEGHJ\}, \{BEGHJ\}\}$. Because the pattern of the new node is the same with {BEGHJ} $\Rightarrow$ Remove {BEGHJ} from the Level 2 (by Theorem 2.4).
- With node {BDGH}: create node {BGH} at level 3 (support = 3) $\Rightarrow [T1] = \{\{BCEGHJ\}, \{BEGHJ\}, \{BGH\}\}$.
- With node {EGHJ}: create node {EGHJ} at level 3 (support = 3) $\Rightarrow [T1] = \{\{BCEGHJ\}, \{BEGHJ\}, \{BGH\}, \{EGHJ\}\}$. Because the pattern of the new node is the same with {EGHJ} $\Rightarrow$ Remove {EGHJ} from the Level 2 (by Theorem 2.4).
- With node {BCDEGH}: create node {BCEGH} at level 3 (support = 3) $\Rightarrow [T1] = \{\{BCEGHJ\}, \{BEGHJ\}, \{BGH\}, \{EGHJ\}, \{BCEGH\}\}$.

Figure 5.3. CP-tree after expanding node $\{BCDEGHJ\}$

Figure 5.4 shows the final CP-tree after expanding all nodes and using Theorems 2.4, 5.1 to prune nodes. After creating level 3, from patterns in level 3, we get the result as $\{\{BEGHJ\}, \{BGH\}, \{EGHJ\}, \{BCEGH\}, \{BDGH\}, \{BFGH\}, \{EFGHJ\}, \{FGH\}\}$.

Figure 5.4. Final CP-tree for mining colossal patterns with length constraints

5.5. Experimental study

5.5.1. Environment and databases

This section compares the runtimes of PCP-Miner-Post1, PCP-Miner-Post2, and LCCP, to confirm the effectiveness of the proposed methods. All experiments presented in this section were performed on a PC with an Intel Core i5 3.2 GHz, and 4GB of RAM, running on Windows 10, with the Visual C# 2017.

We made experiments on four databases, as shown in Table 5.8, which were downloaded from <http://fimi.cs.helsinki.fi/data/>.

Table 5.8. Characteristics of the experimental databases

Database	# of items	# of transactions
Accidents	468	340,183
Connect	130	67,557
Chess	75	3,196
Mushroom	120	8,124

5.5.2. Comparison of the runtimes

We compare the runtime of PCP-Miner-Post1, PCP-Miner-Post2 and LCCP on four databases in Table 5.8 with various settings. In the PCP-Miner-Post1 algorithm, the set of colossal patterns was mined first, after that, we selected the patterns satisfying length constraints. The PCP-Miner-Post2 algorithm was based on the PCP-Miner method, it checked the length constraints when putting a pattern to the result set. And the LCCP algorithm is a newly proposed method, this method applied Theorem 5.1 and Theorem 5.2 to eliminate the patterns that do not satisfy the constraints during the mining process. We have changed the number of transactions in incremental when executing the algorithms for each database to evaluate the effectiveness of the proposed method.

Figure 5.5 to Figure 5.7 show the experiments for Accidents database. We fix min and max length constraints ($\xi = 20, \Omega = 35$), change ϕ from 3 to 6 and we easily found that the runtime of LCCP is less than that of PCP-Miner-Post1 and PCP-Miner-Post2. When increasing ϕ , the LCCP increased very small, while both PCP-Miner-Post1 and PCP-Miner-Post2 was very fast linear increased.

Figure 5.5. Accidents dataset with 100 transactions, various threshold ϕ , fix $\xi = 20$ and $\Omega = 35$

Figure 5.6. Accidents database with 150 transactions, various threshold φ , fix $\xi = 20$ and $\Omega = 35$

Figure 5.7. Accidents database with 200 transactions, various threshold φ , fix $\xi = 20$ and $\Omega = 35$

Figure 5.8. Accidents database with 200 transactions, various min length constraints ξ , fix $\phi=5$ and $\Omega=35$

In Figure 5.8, we fix $\phi = 5$, $\Omega = 35$, change min length constraints ξ from 20 to 26. The runtime of LCCP is faster than PCP-Miner-Post2 and PCP-Miner-Post1 are six times and fourteenth times. In addition, the runtime of LCCP decreased while PCP-Miner-Post1 and PCP-Miner-Post2 are almost not decreased. In a similar way for Chess database, we fix min and max length constraints in Figure 5.9 to Figure 5.11 (with $\xi = 26$ and $\Omega = 35$), change ϕ from 8 to 14, the LCCP is always the best algorithm.

Next, in Figure 5.12, we fix $\phi = 14$, $\Omega = 35$, change min length constraints ξ from 26 to 32, the LCCP is still better than the PCP-Miner-Post1 and PCP-Miner-Post2 algorithms. Especially, when we increase the min length constraint, the runtime of LCCP has significantly decreased while the PCP-Miner-Post1 and PCP-Miner-Post2 have slight decreased. Results from Figure 5.12 show the efficient of Theorems 5.1 and 5.2 compared to without using them (LCCP compared to PCP-Miner-Post2).

Figure 5.9. Chess database with 300 transactions, various threshold ϕ , fix $\xi = 26$ and $\Omega = 35$

Figure 5.10. Chess database with 400 transactions, various threshold φ , fix $\xi = 26$ and $\Omega = 35$

Figure 5.11. Chess database with 500 transactions, various threshold φ , fix $\xi = 26$ and $\Omega = 35$

Figure 5.12. Chess database with 500 transactions, various min length constraints ξ , fix $\phi=14$ and $\Omega=35$

Figure 5.13 to Figure 5.15 perform the same for the Connect database. We easily found that the runtimes of PCP-Miner-Post1 and PCP-Miner-Post2 are nearly the same, PCP-Miner-Post1 is highest while the runtime of LCCP is much better than that of PCP-Miner-Post1 and PCP-Miner-Post2. When we fix $\phi = 14$, $\Omega = 35$, change ξ from 30 to 36, the runtime of LCCP is significantly decreased while the PCP-Miner-Post1 and PCP-Miner-Post2 have softly decreased as shown in Figure 5.16. The experiments confirmed that the LCCP is the best algorithm in three methods for Connect database.

Figure 5.13. Connect database with 300 transactions, various threshold ϕ , fix $\xi = 34$ and $\Omega = 35$

Figure 5.14. Connect database with 400 transactions, various threshold ϕ , fix $\xi = 34$ and $\Omega = 35$

Figure 5.15. Connect database with 500 transactions, various threshold ϕ , fix $\xi = 34$ and $\Omega = 35$

Figure 5.16. Connect database with 500 transactions, various min length constraints ξ , fix $\phi=14$ and $\Omega=40$

Figure 5.17 to Figure 5.19 compare the runtime of LCCP, PCP-Miner-Post1 and PCP-Miner-Post2 for the Mushroom database with various settings. We fix min and max length constraints as $\xi = 16$, $\Omega = 20$, LCCP is always the best algorithm for mining colossal patterns with length constraints for the Mushroom database. Especially, when we change ϕ , the time gaps between the runtime of LCCP and those of PCP-Miner-Post1 and PCP-Miner-Post2 are very large.

In Figure 5.20, we fix $\phi = 16$ and $\Omega = 20$, and change min length constraint from 12 to 18, the runtime of LCCP is very less than those PCP-Miner-Post1 and PCP-Miner-Post2. Therefore, LCCP outperforms PCP-Miner-Post1 and PCP-Miner-Post2 for this dataset. In short, through the above experiments, we can conclude that LCCP outperforms PCP-Miner-Post1 and PCP-Miner-Post2 for mining colossal patterns with length constraints.

Figure 5.17. Mushroom database with 500 transactions, various threshold ϕ , fix $\xi = 16$ and $\Omega = 20$

Figure 5.18. Mushroom database with 600 transactions, various threshold ϕ , fix $\xi = 16$ and $\Omega = 20$

Figure 5.19. Mushroom database with 700 transactions, various threshold ϕ , fix $\xi = 16$ and $\Omega = 20$

Figure 5.20. Mushroom database with 700 transactions, various min length constraints ξ , fix $\phi=16$ and $\Omega=20$

Chapter 6. Conclusions and Future works

Frequent pattern mining has been attracting attention in recent researches into data mining. Very large search space and data volume have made many problems for algorithms to mine frequent patterns. The thesis proposed efficient algorithms in order to mining frequent pattern effectively from high dimensional databases.

6.1. Conclusions

This thesis has proposed new efficient algorithms for mining colossal patterns, mining colossal patterns with pattern constraints and mining colossal patterns with length constraints. The results are satisfying all requires of the thesis. The contributes of thesis can be summarized as follows:

First of all, the CP-tree structure is developed. Then, two theorems for quickly mining colossal patterns and pruning nodes are designed. Based on these, we propose the CP-Miner algorithm for mining colossal patterns. CP-Miner mines colossal patterns using the CP-tree and pre-processing techniques to reduce the search space [P5].

The next, based on CP-Miner, a theorem for quickly checking duplicated patterns and one for rapidly checking non-colossal patterns are developed. These two theorems provide support for PCP-Miner, an improved version of the CP-Miner algorithm, which is able to efficiently mine colossal patterns [P3].

Then, we have continued to improve and make the CPCP-Miner algorithm more efficient than the previous two algorithms [P4].

Finally, we proposed two new theorems and a method named LCCP for mining colossal patterns with constraints. Based on these theorems, an efficient algorithm for mining colossal patterns with constraints was proposed, the theorems have early eliminated the patterns that are not satisfying the length constraints to improve the mining times [P1].

BVBUC uses bit vectors to represent itemsets, when the number of items in the database is large, it is time consuming to join itemsets. the paper uses dynamic bit vectors to improve the performance and memory usage.

One of the weaknesses of bit vectors is that when the database is very sparse and the first position of bit 1 at the beginning and the last position of bit 1 at the end of each bit vector is very far, the number of bits in each bit vector is very large, although it contains a small number of bits 1. Although we have used DBV to solve this problem it only removes bits 0 at the beginning and end. In future work, we will study how to compress this case of databases in the tree.

The last, we statement the problem of extracting colossal patterns with length constraints (min-length and max-length) and developing a method for mining colossal patterns with length constraints based on theorems min-length and max-length to early eliminate candidates.

6.2. Future works

In future work, we will study how to compress this case of databases in the tree. We will also apply the CP-tree to mining frequent closed patterns and frequent maximal patterns in high dimensional databases.

We will also apply the CP-tree to mining frequent closed patterns and frequent maximal patterns in high dimensional databases.

In addition, we will study mining colossal patterns with items and sequence constraints. The combination of many constraints will also be studied. The colossal pattern mining with constraints will be researching and implement in the distributed and/or parallel environment.

6.3. Publications

The experimental results prove that our proposed methods are better for tackling the problems mining frequent patterns on high dimensional databases. The proposed methods and their experimental results have been presented and published in high-quality international conferences / journals. The publications are listed as follows.

- [P1]. Tuong Le, Thanh-Long Nguyen, Bao Huynh, Hung Nguyen, Tzung-Pei Hong, Vaclav Snasel (2021). Mining Colossal Patterns with Length Constraints. *Applied Intelligence* (Second round revision – 01/2021).
- [P2]. Loan T. T. Nguyen, Bay Vo, Thang Mai, Thanh-Long Nguyen (2018). A Weighted Approach for Class Association Rules. *ACIIDS (Extended Posters)*, pp. 213-222
- [P3]. Thanh-Long Nguyen, Bay Vo, Václav Snásel (2017). Efficient algorithms for mining colossal patterns in high dimensional databases. *Knowledge Based Systems* 122, pp. 75-89.
- [P4]. 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 (Best paper award).
- [P5]. Thanh-Long Nguyen, Bay Vo, Loan T. T. Nguyen (2016). A new method for mining colossal patterns. *SMC*, pp. 3119-3124.
- [P6]. Bay Vo, Dang Nguyen, Thanh-Long Nguyen (2015). A Parallel Algorithm for Frequent Subgraph Mining. *ICCSAMA*, pp. 163-173.

References

- [1] Agrawal R., Imielinski T., Swami A.N., "Mining Association Rules between Sets of Items in Large Databases", *SIGMOD Conference*, pp. 207-216, 1993.
- [2] Han J., Kamber M., Pei J., Data Mining: Concepts and Techniques 3rd Edition, USA: Morgan Kaufmann. ISBN 978-0-12-381479-1, 2011.
- [3] Bhuyan M.H., Bhattacharyya D.K., Kalita J.K., "Towards Generating Real-life Datasets for Network Intrusion Detection", *I. J. Network Security*, vol. 17, no. 6, pp. 683-701, 2015.
- [4] Agrawal R., Srikant R., "Mining Sequential Patterns", *ICDE'95*, pp. 3-14, 1995.
- [5] Valle M., Ruz G. and Morrás R., "Market basket analysis: Complementing association rules with minimum spanning trees", *Expert Syst. Appl.*, vol. 97, pp. 146-162, 2018.
- [6] Jeyabharathi J., Shanthi D., "Enhanced sequence identification technique for protein sequence database mining with hybrid frequent pattern mining algorithm", *IJDMB'16*, vol. 16, no. 3, pp. 205-229, 2016.
- [7] Deng N., Chen X., Li D. and Xiong C., "Frequent Patterns Mining in DNA Sequence", *IEEE Access*, vol. 7, pp. 108400-108410, 2019.
- [8] Temple M., "An auditory display tool for DNA sequence analysis", *BMC Bioinformatics*, vol. 18, no. 1, pp. 221:1-221:11, 2017.
- [9] Isinkaye F.O., Folajimi Y.O., Ojokoh B.A., "Recommendation systems: Principles, methods and evaluation", *Egyptian Informatics Journal*, vol. 16, no. 3, pp. 261-273, 2015.
- [10] Hagen M. and Stein B., "Weblog Analysis", in *Encyclopedia of Social Network Analysis and Mining. 2nd Ed.*, 2018.
- [11] Ren Y., Tomko M., Salim F.D., Ong K., Sanderson M., "Analyzing Web behavior in indoor retail spaces", *JASIST*, vol. 68, no. 1, pp. 62-76, 2017.
- [12] 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*, vol. 17, no. S-19, pp. 87-90, 2016.
- [13] 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*, vol. 9, no. 38, DOI: 10.17485/ijst/2016/v9i38/101962, 2016.

-
- [14] Golmohammadi K., Zaïane O.R., "Time series contextual anomaly detection for detecting market manipulation in stock market", *DSAA*, pp. 1-10, 2015.
- [15] Jung H., Chung K., "Sequential pattern profiling based bio-detection for smart health service", *Cluster Computing*, vol. 18, no. 1, pp. 209-219, 2015.
- [16] Zhu F., Yan X., Han J., Yu P., Cheng H., "Mining colossal frequent patterns by core pattern fusion", *ICDE*, pp. 706-715, 2007.
- [17] Vo B., Hong T.P., Le B., "DBV-Miner: A Dynamic Bit-Vector approach for fast mining frequent closed itemsets", *Expert Syst. Appl.*, vol. 39, no. 8, pp. 7196-7206, 2012.
- [18] Agrawal R., R. Srikant R., "Fast Algorithms for Mining Association Rules in Large Databases", *VLDB*, pp. 487-499, 1994.
- [19] Zaki M.J. , Parthasarathy S., Ogihara M., Li W. , "New algorithms for fast discovery of association rules", *KDD*, pp. 283-286, 1997.
- [20] Zaki M.J., Gouda K., "Fast vertical mining using diffsets", *KDD*, pp. 326–335, 2003.
- [21] Zaki M.J., Hsiao C.J., "Efficient algorithms for mining closed itemsets and their lattice structure", *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 4, pp. 462–478, 2005.
- [22] Han J., Pei J., Yin Y., "Mining frequent patterns without candidate generation", *SIGMOD*, pp. 1-12, 2000.
- [23] Wang J., Han J., Pei J., "CLOSET+: searching for the best strategies for mining frequent closed itemsets", *KDD*, pp. 236-245, 2003.
- [24] Grahne G., Zhu J., "Fast algorithms for frequent itemset mining using FP-trees", *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 10, pp. 1347–1362, 2005.
- [25] Dong J., Han M., "BitTableFI: an efficient mining frequent itemsets algorithm", *Knowl.-Based Syst.*, vol. 20, no. 4, pp. 329–335, 2007.
- [26] Song W., Yang B., Xu Z., "Index-BitTableFI: an improved algorithm for mining frequent itemsets", *Knowl.-Based Syst.*, vol. 21, no. 6, pp. 507–513, 2008.
- [27] Deng Z.H., WangZ., Jiang J.J., "A new algorithm for fast mining frequent itemsets using N-lists", *Information Sciences*, vol. 55, no. 9, pp. 2008-2030, 2012.
- [28] Vo B., Le T., Coenen F., Hong T.P., "Mining frequent itemsets using the N-list and subsume

- concepts", *Int. J. Machine Learning & Cybernetics*, vol. 7, no. 2, pp. 253-265, 2016.
- [29] Le T., Vo B., "An N-list-based algorithm for mining frequent closed patterns", *Expert Syst. Appl.*, vol. 42, no. 19, pp. 6648-6657, 2015.
- [30] Deng Z.H., "DiffNodesets: An efficient structure for fast mining frequent itemsets", *Appl. Soft Comput.*, vol. 41, pp. 214-223, 2016.
- [31] Pyun G., Yun U., Ryu K.H., "Efficient frequent pattern mining based on Linear Prefix tree", *Knowl.-Based Syst.*, vol. 55, pp. 125-139, 2014.
- [32] Sohrabi M.K., Barforoush A.A., "Efficient colossal pattern mining in high dimensional datasets", *Knowl.-Based Syst.*, vol. 33, pp. 41-52, 2012.
- [33] Dabbiru M. and Shashi M., "An efficient approach to colossal pattern mining", *Int. J. Comput. Sci. Network Security*, vol. 6, pp. 304-312, 2010.
- [34] Zulkurnain N., "DisClose : discovering colossal closed itemsets from high dimensional datasets via a compact row-tree", *University of Manchester*, 2012.
- [35] Vanahalli M. and Patil N., "An efficient parallel row enumerated algorithm for mining frequent colossal closed itemsets from high dimensional datasets", *Inf. Sci.*, vol. 496, pp. 343-362, 2019.
- [36] Vanahalli M. and Patil N., "Distributed Mining of Significant Frequent Colossal Closed Itemsets from Long Biological Dataset", *ISDA*, vol. 1, pp. 891-902, 2018.
- [37] Zaki F.M. and Zulkurnain N., "RARE: Mining colossal closed itemset in high dimensional data", *Knowl.-Based Syst.*, vol. 161, pp. 1-11, 2018.
- [38] Zhang S., Wu X., "Fundamentals of association rules in data mining and knowledge discovery", *Wiley Interdisc. Rev.: Data Mining and Knowledge Discovery*, vol. 1, no. 2, pp. 97-116, 2011.
- [39] Djenouri Y., Fournier-Viger P., J. Lin, Djenouri D. and Belhadi A., "GPU-based swarm intelligence for Association Rule Mining in big databases", *Intell. Data Anal.*, vol. 23, no. 1, pp. 57-76, 2019.
- [40] Badia A., Kantardzic M., "Generalizing Association Rules: Theoretical Framework and Implementation", *Intelligent Systems Design and Applications. Advances in Soft Computing*, vol. 23, pp. 283-292, 2003.

-
- [41] Yan Y. and Shyu M., "Correlation-Assisted Imbalance Multimedia Concept Mining and Retrieval", *Int. J. Semantic Computing*, vol. 11, no. 2, pp. 209-228, 2017.
- [42] Mooney C., Roddick J.F., "Sequential pattern mining - approaches and algorithms", *ACM Comput. Surv.*, vol. 45, no. 2, pp. 19:1-19:39, 2013.
- [43] Slimani T., Lazzez A., "Sequential Mining: Patterns and Algorithms Analysis", *International Journal of Computer and Electronics Research*, vol. 2, no. 5, pp. 639-647, 2013.
- [44] Ao X., Luo P., Li C., Zhuang F., He Q., "Online Frequent Episode Mining", *ICDE'15*, pp. 891-902, 2015.
- [45] Luo A., Jia X., Shang L., Gao Y., Yang Y., "Granular-Based Partial Periodic Pattern Discovery over Time Series Data", *Rough Sets and Knowledge Technology. RSKT. LNCS*, vol. 6954, pp. 706-711, 2011.
- [46] Liu H., Wu X., Zhang S., "A New Supervised Feature Selection Method for Pattern Classification", *Computational Intelligence*, vol. 30, no. 2, pp. 342-361, 2014.
- [47] Wang Y., Tang Y., Li L. and Zheng X., "Block sparse representation for pattern classification: Theory, extensions and applications", *Pattern Recognition* 88, pp. 198-209, 2019.
- [48] Li S., Wang M., Liu S. and Fu Y., "A novel pattern recognition technique based on group clustering computing and convex optimization for dimensionality reduction", *Cluster Computing*, vol. 21, no. 1, pp. 805-811, 2018.
- [49] Alibasa M., Calvo R. and Yacef K., "Sequential Pattern Mining Suggests Wellbeing Supportive Behaviors", *IEEE Access*, vol. 7, pp. 130133-130143, 2019.
- [50] Van T., Yoshitaka A. and Le B., "Mining web access patterns with super-pattern constraint", *Appl. Intell.*, vol. 48, no. 11, pp. 3902-3914, 2018.
- [51] Bachman J., Gyori B. and Sorger P., "FamPlex: a resource for entity recognition and relationship resolution of human protein families and complexes in biomedical text mining", *BMC Bioinformatics*, vol. 19, no. 1, pp. 248:1-248:14, 2018.
- [52] Sakthivel N. K., Gopalan N. P. and Subasree S., "A Comparative Study and Analysis of DNA Sequence Classifiers for Predicting Human Diseases", *ICIA*, pp. 107:1-107:5, 2016.

- [53] Dessouky M., Taha E., Dessouky M., Eltholth A., Hassan E. and El-Samie F., "Non-parametric spectral estimation techniques for DNA sequence analysis and exon region prediction", *Computers & Electrical Engineering*, vol. 73, pp. 334-348, 2019.
- [54] Iqbal W., Erradi A. and Mahmood A., "Dynamic workload patterns prediction for proactive auto-scaling of web applications", *J. Network and Computer Applications*, vol. 124, pp. 94-107, 2018.
- [55] Hyeok K., Cholyong J., Unhyok R., "Implementation of Association Rule Mining for Network Intrusion Detection", *CoRR abs/1601.05335*, 2016.
- [56] Nguyen T.L., Vo B., Snásel V., "Efficient algorithms for mining colossal patterns in high dimensional databases", *Knowl.-Based Syst.*, vol. 122, pp. 75-89, 2017.
- [57] Van T., Vo B. and Le B., "Mining sequential patterns with itemset constraints", *Knowl. Inf. Syst.*, vol. 57, no. 2, pp. 311-330, 2018.
- [58] Le T., Nguyen A., Huynh B., Vo B. and Pedrycz W., "Mining constrained inter-sequence patterns: a novel approach to cope with item constraints", *Appl. Intell.*, vol. 48, no. 5, pp. 1327-1343, 2018.
- [59] Kocak G., Akgun O., Miguel I. and Nightingale P., "Closed Frequent Itemset Mining with Arbitrary Side Constraints", *ICDM Workshops*, pp. 1224-1232, 2018.
- [60] Nguyen D., Nguyen L., Vo B. and Pedrycz W., "Efficient mining of class association rules with the itemset constraint," *Knowl.-Based Syst.*, vol. 103, pp. 73-88, 2016.
- [61] Bessiere C., Lazaar N., Lebbah Y. and M. M., "Users Constraints in Itemset Mining", *CoRR abs/1801.00345*, 2018.
- [62] Nguyen D., Nguyen L., Vo B. and Hong T., "A novel method for constrained class association rule mining", *Inf. Sci.*, vol. 320, pp. 107-125, 2015.
- [63] Cooper M. and Zivny S., "Hybrid Tractable Classes of Constraint Problems", *The Constraint Satisfaction Problem: Complexity and Approximability*, pp. 113-135, 2017.
- [64] Vo B., Le T., Pedrycz W., Nguyen G. and Baik S., "Mining erasable itemsets with subset and superset itemset constraints", *Expert Syst. Appl.*, vol. 69, pp. 50-61, 2017.
- [65] Wojciechowski M., Zakrzewicz M., "Dataset filtering techniques in constraint-based frequent pattern mining", *Pattern detection and discovery*, vol. 2447, pp. 77-91, 2002.

- [66] Lin W.Y., Huang K.W., Wu C.A., "MCFPTree: An FP-tree-based algorithm for multi-constraint patterns discovery", *International Journal of Business Intelligence and Data Mining*, vol. 5, pp. 231-246, 2010.
- [67] Ng R.T., Lakshmanan L.V.S., Han J., Pang A., "Exploratory mining and pruning optimizations of constrained associations rules", *ACM SIGMOD International Conference on Management of Data*, pp. 13-24, 1998.
- [68] Tran A.N., Duong H.V., Truong T.C., Le B.H., "Efficient Algorithms for Mining Frequent Itemsets with Constraint", *Knowledge and Systems Engineering (KSE)*, pp. 19 – 25, 2011.
- [69] Vo B., Le T., Pedrycz W., Nguyen G., Baik S.W., "Mining erasable itemsets with subset and superset itemset constraints", *Expert Syst. Appl.*, vol. 69, pp. 50-61, 2017.
- [70] Nguyen D., Nguyen L.T.T., Vo B., Hong T.P., "A novel method for constrained class association rule mining", *Inf. Sci.*, vol. 320, pp. 107-125, 2015.
- [71] Nguyen D., Vo B. and Le B., "CCAR: An efficient method for mining class association rules with itemset constraints", *Eng. Appl. of AI*, vol. 37, pp. 115-124, 2015.
- [72] Nguyen T. -L., Vo B., Huynh B., Snasel V. and Nguyen L., "Constraint-Based Method for Mining Colossal Patterns in High Dimensional Databases", in *Information Systems Architecture and Technology - ISAT*, Advances in Intelligent Systems and Computing, 2017, pp. 195-204.
- [73] Kurdija A., Silic M., Vladimir K. and Delac G., "Efficient global correlation measures for a collaborative filtering dataset", *Knowl.-Based Syst.*, vol. 147, pp. 36-42, 2018.
- [74] Lin W., Huang K. and Wu C., "MCFPTree: An FP-tree-based algorithm for multi-constraint patterns discovery", *International Journal of Business Intelligence and Data Mining*, vol. 5, pp. 231-246, 2010.
- [75] Krismayer T., Rabiser R. and Grünbacher P., "A Constraint Mining Approach to Support Monitoring Cyber-Physical Systems (CPS)", *CAiSE*, pp. 659-674, 2019.
- [76] Hosseininasab A., Hoeve W.-J. and Ciré A., "Constraint-Based Sequential Pattern Mining with Decision Diagrams", *AAAI*, pp. 1495-1502, 2019.
- [77] Abeysinghe R. and Cui L., "Query constraint based mining of association rules for exploratory analysis of clinical datasets in the National Sleep Research Resource", *BMC Med. Inf. & Decision Making*, vol. 18, no. S-2, pp. 89-100, 2018.

-
- [78] Belaid M., Bessiere C. and Lazaar N., "Constraint Programming for Association Rules", *SDM*, pp. 127-135, 2019.
- [79] Zhu F., Yan X., Han J., Yu P. and Cheng H., "Mining colossal frequent patterns by core pattern fusion", *ICDE'07*, pp. 706-715, 2007.
- [80] Nguyen L., Vo B., Nguyen L., Fournier-Viger P. and Selamat A., "ETARM: an efficient top-k association rule mining algorithm", *Appl. Intell.*, vol. 48, no. 5, pp. 1148-1160, 2018.
- [81] Huynh B., Trinh C., Huynh H., Van T., Vo B. and V. Snásel, "An efficient approach for mining sequential patterns using multiple threads on very large databases", *Eng. Appl. of AI*, vol. 74, pp. 242-251, 2018.
- [82] Lin Y., Huang C. and Tseng V., "A novel methodology for stock investment using high utility episode mining and genetic algorithm", *Appl. Soft Comput.*, vol. 59, pp. 303-315, 2017.
- [83] Littmann M., Goldberg T., Seitz S., Bodén M. and Rost B., "Detailed prediction of protein sub-nuclear localization", *BMC Bioinformatics*, vol. 20, no. 1, pp. 205:1-205:15, 2019.
- [84] Philippe Fournier-Viger P., Li Z., Lin J., Kiran R. and Fujita H., "Efficient algorithms to identify periodic patterns in multiple sequences", *Inf. Sci.*, vol. 489, pp. 205-226, 2019.
- [85] Gan W., Lin J., Chao H., Hong T. and Yu P., "CoUPM: Correlated Utility-based Pattern Mining," *BigData*, pp. 2607-2616, 2018.
- [86] Lin J., Yang L., Fournier-Viger P. and Hong T., "Mining of skyline patterns by considering both frequent and utility constraints", *Eng. Appl. of AI*, vol. 77, pp. 229-238, 2019.
- [87] Prasanna K. and Seetha M., "A Doubleton Pattern Mining Approach for Discovering Colossal Patterns from Biological Dataset", *International Journal of Computer Applications*, vol. 119, no. 21, 2015.
- [88] Prasanna K. and Seetha M., "Efficient and Accurate Discovery of Colossal Pattern Sequences from Biological Datasets: A Doubleton Pattern Mining Strategy (DPMine)", *IMCIP-2015*, vol. 54, pp. 412 – 421, 2015.