國立臺灣大學
資訊管理學研究所

博士論文

時間序列資料庫之封閉性樣式探勘

吳惠雯 撰

99
1

國立臺灣大學管理學院資訊管理學研究所

博士論文

Department of Information Management
College of Management
National Taiwan University
Doctoral Dissertation

時間序列資料庫之封閉性樣式探勘

Mining Closed Patterns in Time-Series Databases

吳惠雯

Huei-Wen Wu

指導教授：李瑞庭 博士

Advisor: Anthony J. T. Lee, Ph.D.

中華民國 99 年 1 月

January, 2010

國立臺灣大學管理學院資訊管理學研究所

博士論文

Department of Information Management

College of Management

National Taiwan University

Doctoral Dissertation

時間序列資料庫之封閉性樣式探勘

Mining Closed Patterns in Time-Series Databases

吳惠雯

Huei-Wen Wu

指導教授：李瑞庭 博士

Advisor: Anthony J. T. Lee, Ph.D.

中華民國 99 年 1 月

January, 2010

時間序列資料庫之封閉性樣式探勘

Mining Closed Patterns in Time-Series Databases

By Huei-Wen Wu

A dissertation submitted to
the Graduate School of Information Management
of National Taiwan University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

January, 2010

國立臺灣大學博士學位論文
口試委員會審定書

時間序列資料庫之封閉性樣式探勘

Mining Closed Patterns in Time-Series Databases

本論文係吳惠雯君（學號 D95725007）在國立臺灣大學資訊管理學系、所完成之博士學位論文，於民國 99 年 01 月 07 日承下列考試委員審查通過及口試及格，特此證明

口試委員：李瑞庭

呂永和　　陳建錦

諶家蘭　　魏志平

所　　長：陳靜枝

謝辭

三年半的求學生涯即將在論文完成之際劃下完美的句點，看著投注所有心力的研究成果，心中備感欣慰及榮耀。回首過去苦澀艱辛的日子，一切都是值得與甘美的。

這一路走來，衷心感謝指導教授李瑞庭老師，在論文研究過程中細心、耐心地給予殷殷指導，每當我遇到瓶頸時，老師總能引領我分析、思考與解決問題，並提出許多建議，幫助我克服許多困難，讓我得以順利完成論文。老師對研究認真、熱忱、嚴謹的態度，以及對學生的親切關懷，更是我學習的最佳典範。另外，我要特別感謝系上謝清佳老師，謝老師不僅在知識層面上給予我許多啟發，也提升了我心靈層面上的體悟，而謝老師對我的鼓勵與肯定，更給予我繼續堅持的勇氣與信心。

論文口試期間，承蒙口試委員魏志平老師、諶家蘭老師、呂永和老師及陳建錦老師的悉心指正及建議，使得論文更臻完善與嚴謹，在此謹致上最誠摯的謝意。

求學期間，感謝一直陪伴在我身邊的好友們嘉韡、柏吟、及明俊，不時為我加油打氣、給予協助，令我感動至深。另外，感謝實驗室的同儕及學弟妹，帶給我許多溫暖及歡樂。

在此，我謹將論文獻給我最敬愛的父親及母親，感謝父親對我的關愛及栽培，讓我能心無旁騖地在學業上衝刺；感謝母親時時刻刻陪伴在我身旁，不斷地鼓勵、安慰及包容，讓我更有力量去面對挑戰，也感謝最疼愛我的哥哥，在我苦惱煩悶時，總是不厭其煩地傾聽與開導，讓我能保有開朗的心情，在此衷心感謝家人為我付出的一切。

最後，感謝老天的眷顧，豐富了我生命的版圖。

吳惠雯 謹識

于國立台灣大學資訊管理學研究所

中華民國九十九年一月

論文摘要

論文題目：時間序列資料庫之封閉性樣式探勘

作者：吳惠雯　　　　　　　　　　　　　　　　　　九十九年一月

指導教授：李瑞庭 博士

近年來，探勘封閉性樣式已成為知識探索領域中重要的研究議題，其主要目的在於找出隱藏在大量資料中具有代表性的樣式。本論文針對如何從時間序列資料庫中探勘封閉性樣式，提出了三個有效率的演算法，分別為 CMP (Closed Multi-sequence Patterns mining)、 CFP (Closed Flexible Patterns mining)與 CNP (multi-resolution Closed Numerical Patterns mining)。

CMP 演算法主要著重於多時間序列資料庫中樣式的分析，而 CFP 演算法可從時間序列資料庫中探勘具「彈性間隔」的封閉性樣式。CMP 與 CFP 演算法先將時間序列轉換成符號序列，然後再探勘封閉性樣式。將時間序列轉換成符號序列可減少雜訊並簡化探勘程序，然而可能導致樣式遺失或差異很大的序列卻支持相同樣式的問題。

為避免將時間序列轉換到符號序列所造成的問題，CNP 演算法可從多時間序列資料庫中直接找出具有代表性的樣式。此外，CNP 演算法加入多層解析度(multi-resolution)的概念以找出封閉性樣式，可讓使用者以不同的觀點來檢視資料。

上述三個演算法皆以深度優先探勘的方式，並配合投影資料庫以減少搜尋空間，除了加速探勘的過程外，透過有效的修剪策略及檢查封閉性的機制，可避免產生不必要的候選樣式。

本研究結果顯示 CMP 演算法比改良式 Apriori 以及 BIDE 演算法快了數十倍; CFP 演算法比改良式 Apriori 演算法更有效率; CNP 演算法執行速度亦較改良式 A-Close 演算法快。

關鍵詞: 資料探勘、封閉性樣式、時間序列資料庫。

Dissertation Abstract

Mining Closed Patterns in Time-Series Databases

By Huei-Wen Wu

JANUARY 2010

ADVISOR: Dr. Anthony J. T. Lee

Closed pattern mining is a critical research issue in the area of knowledge discovery and data mining with the aim of discovering interesting patterns hidden in a large amount of data. In this dissertation, we propose three algorithms, called CMP (Closed Multi-sequence Patterns mining), CFP (Closed Flexible Patterns mining), and CNP (multi-resolution Closed Numerical Patterns mining) to solve various issues extended from the problem of mining closed patterns.

The CMP algorithm is designed to find closed patterns in a multi-sequence time-series database. The CFP algorithm is developed to solve the problem of mining closed flexible patterns in a time-series database. Both the CMP and CFP algorithms involve a transformation of time-series sequences into symbolic sequences in the first phase. Although analyzing on symbolic sequences is ideal to reduce the effect of noises and ease the mining process, these approaches may lead to pattern lost and the sequences supporting the same pattern may look quite different.

To overcome the problem raised in symbolic sequence analysis, the CNP algorithm is proposed to mine closed patterns without any transformation from time-series sequences to symbolic sequences. The method also employs the Haar wavelet transform to discover patterns in the multiple resolutions in order to provide different perspectives on datasets.

All the proposed algorithms have employed the concept of projected databases to localize the pattern extension that leads to a significant runtime improvement. Moreover, effective closure checking schemes and pruning strategies are devised respectively in each of the proposed algorithms to avoid generating redundant candidates.

The experimental results show that the CMP algorithm significantly outperforms the modified Apriori and BIDE algorithms. The CFP algorithm achieves better performance than the modified Apriori algorithm in all cases. And, the CNP algorithm has demonstrated a significant runtime improvement in comparison to the modified A-Close algorithm.

Keywords: Data mining, closed pattern, time-series database.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

Insightful analysis of business trends and accurate forecasts of future patterns are critical strategic weapons which can be used to build competitive advantages for individuals or enterprises. These trends or patterns are usually hidden in a large amount of data and have not yet been articulated. Thus, mining such patterns has become a critical research area in recent years.

Data mining is a process of extracting or mining meaningful patterns from large amounts of data [19]. In this discipline, one major line of research aims at developing efficient algorithms for mining frequent patterns. A pattern is frequent if its support is not less than a user-specified minimum support threshold, where the support of a pattern is defined as the number of transactions containing the pattern in the database. Mining such frequent patterns helps decision-makers to perceive significant relationships hidden within data and assists in solving problems for various domains, such as finance, medical science, marketing, biology, meteorology, etc.

Several main streams of pattern mining, such as time-series mining, sequential pattern mining, have been introduced to cope with different types of datasets in mining frequent patterns. Time-series mining methods incorporate concrete notions of time in the process of finding patterns. Previous studies in this field include searching similar patterns in time-series databases [1], matching query patterns in time-series databases [17][25], and mining long periodic patterns [18]. Regardless of notions of time, sequential pattern mining methods consider only order of events in data. A wide range of problems derived from classic sequential pattern mining are investigated, such as mining long sequential patterns [45], mining patterns with quantities [27], and mining inter-sequence patterns [60].

Although frequent pattern mining has played a pivotal role in the field of data mining, it suffers from two disadvantages: (1) the number of frequent patterns discovered may be overwhelming and it is hard for users to comprehend and (2) it consumes more time and space to mine all frequent patterns. To overcome these disadvantages, closed pattern mining has been proposed to condense the frequent patterns while preserving the same information. A closed pattern is defined as a pattern that is frequent and has no super-pattern with the same support. Generally speaking, the algorithms of mining closed patterns are more efficient than those of mining frequent patterns [43][44]. Moreover, mining closed patterns is lossless because a complete set of frequent patterns and their exact supports can be derived from the closed patterns mined. Therefore, closed pattern mining is an alternative approach that preserves the same information as frequent pattern mining but provides a succinct and yet more efficient solution.

In this dissertation, we focus on the problem of mining closed patterns in time-series databases in three aspects. The first aspect is to mine closed multi-sequence patterns in multi-sequence time-series databases. A multi-sequence time-series database is a database that contains time-series records (or transactions), where each transaction contains multiple time-series sequences. The second is to mine closed flexible patterns in time-series databases. The third is to mine multi-resolution closed numerical patterns in multi-sequence time-series databases.

In this chapter, the motivations of the dissertation are discussed in Section 1.1, the contributions are presented in Section 1.2, and the dissertation layout is organized in Section 1.3.

1.1 Motivations

In the field of medical science, chronic liver disease is one of the major health problems in Asia. Most patients routinely take blood tests to determine the level of liver enzymes, such as alanine aminotransferase (ALT) and aspartate aminotransferase (AST) in order to diagnose their liver cell damage. The measurements of these two liver enzymes have been widely recognized as the indicators of liver disease [3][34][49].

According to [34], elevated ALT and AST are associated with a significantly increased standardized mortality ratio. Pockros et al. [48] evaluated the effect of oral IDN-6556, an antiapoptotic caspase inhibitor, on the patients with elevated ALT and AST, where the levels of both liver enzymes were recorded in two time-series sequences. The result indicated that IDN-6556 progressively lowered the ALT and AST levels of the patients in a treatment period of 14 days.

Many diseases are believed to be associated with different indicators. If we could identify which indicators are likely to relate to a disease, we could routinely track the measurements of these indicators to prevent the illness or to predict the possible risks.

We further speculate other real-world applications that need the closed patterns of multi-sequence time-series transactions. In the field of seismology, earthquake prediction is still the greatest challenge. Song et al. [51] analyzed the groundwater anomaly before and after the Chi-Chi Earthquake in Taiwan and found that groundwater level and groundwater chemistry can be used to forecast earthquakes. We have doubted whether there is a pattern that corresponds to an unusual event before earthquakes. Thus, we may focus on the factors like changes in groundwater levels, changes in the concentration of anions in groundwater, etc. We could visualize a database with historical earthquake records where each record contains two sequences: the values of ground water level and the values of anion concentration in ground water and these values are recorded on a daily basis for a month before an earthquake takes place. We may apply the CMP algorithm to discover meaningful patterns and expect that these patterns could help seismologists predict earthquakes.

In microelectronics, wafers are the critical components in the fabrication of semiconductor devices. To manufacture such components, many factors are taken into consideration, including temperature and humidity [42]. Manufacturers usually rely on past experience to determine the adequate temperature and humidity in order to produce high quality wafers. In order to address the issue of finding the adequate temperature and humidity, we may apply the CMP algorithm to mine time-series patterns from a database where each record contains both temperature and humidity readings (multiple sequences) during the process of producing a wafer in a semiconductor facility. Accordingly, based on the frequency of these patterns, we can determine the acceptable range of temperature and humidity to produce the best wafer.

Likewise, weather forecasts are required to predict the possibility of deteriorating weather conditions, especially approaching tornados or hurricanes. However, making such accurate predictions is difficult because many factors are involved, such as temperature, humidity, pressure, etc [2]. If we could recognize what conditions are likely to cause a tornado or a hurricane, we could take actions before the storm arrives, and save a lot of costs or lives.

Therefore, these applications have inspired us to propose an algorithm, called CMP, to efficiently mine closed patterns in time-series databases, where each transaction in the database contains multiple time-series sequences. For example, Fig. 1.1 shows an example database containing three transactions, where each transaction contains two time-series sequences.

Transaction 1	Sequence 1	-2.50	0.33	1.80	-3.90	-0.21	-0.08
	Sequence 2	-0.41	-0.25	-0.54	-1.39	0.23	-0.88
Transaction 2	Sequence 1	-7.80	6.52	0.17	-3.68	0.19	0.02
	Sequence 2	-0.06	-2.89	0.13	4.30	0.42	-0.47
Transaction 3	Sequence 1	-2.42	5.37	-1.84	-3.55	0.32	-4.64
	Sequence 2	-0.42	3.12	0.11	-0.99	0.39	-5.97

Fig. 1.1. A database containing three multi-sequence transactions.

The CMP algorithm consists of three phases. First, we transform every sequence in the time-series database into the Symbolic Aggregate approXimation (SAX) representation [37]. Second, we scan the transformed database to find all frequent

1-patterns (the patterns of length one), and build a projected database for each frequent 1-pattern, where the projected database of a pattern comprises all transactions containing the pattern. Third, for each frequent pattern found, we recursively use its projected database to generate its frequent super-patterns in a depth-first search manner. Moreover, we apply some closure checking and pruning strategies to prune frequent but non-closed patterns during the mining process. By using the projected databases, the CMP algorithm can localize support counting, candidate pruning, and closure checking in the projected databases. Therefore, it can efficiently mine the closed patterns from a time-series database.

Most previous studies [45][61][66] require exact sequence alignments in discovering patterns from sequence databases. However, they are not capable of dealing with noisy environments, such as a set of deviated sequences in a database. In order to address this issue, we incorporate the idea of allowing flexible gaps between items in a pattern. For example, *a*[0, 3]*b* is a flexible pattern where the number of gaps between *a* and *b* ranges from 0 to 3. Mining such sequential patterns with flexible gaps can provide decision-makers a broader view on data. For example, given a web log recording URL that each user visits at each minute, a web analyzer would like to determine how many users spend less than 5 minutes browsing web page *A* before switching to web page *B*. That can be denoted as a flexible pattern *A*[0, 5]*B*, which means that users who are browsing web page *A* are likely to traverse to web page *B* within 5 minutes. However, the previously proposed approaches cannot mine such a pattern since they can only mine the patterns to show that users browse web page *A* before web page *B* [35].

The subprime mortgage crisis in the United States has had a major impact on house prices and stock markets worldwide [5]. How house price index varies over time since the outbreak of the subprime crisis would become a major concern for consumers. In this case, we could analyze the house price index data to see if there is a significant variation in house price for all cities in the United States. Since the timing of house price declination or acceleration is different among cities, no patterns may be discovered by traditional pattern mining. Dealing with flexibility, we aim to give consumers a big picture on the underlying trend in house prices.

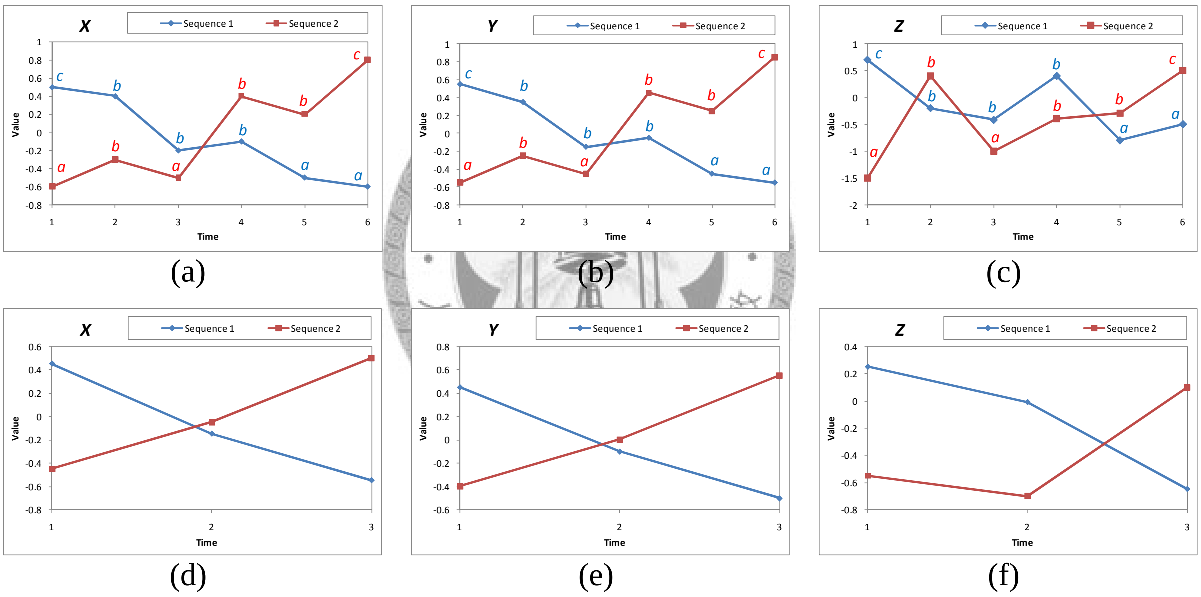
Inspecting whether the crime rate increases or drops in the same trend among different cities in a country provides polices a clue to find the major causes, such as unemployment, economic recession, drugs prevailing, etc. A flexible pattern found could be “*south cities increase in crime lagged behind the rise in the north cities’ crime rate for several months when unemployment rate drops sharply*”. However, traditional data mining techniques, such as clustering analysis, association rule mining, and sequential pattern mining, are not capable of retrieving such information [9].

Effective inventory control is vital to the success of the business in many industries. For example, predicting the bestselling books at the right time and ensuring sufficient inventory have a major impact on profitability for a book store. An interesting flexible pattern could be “*if a book has been made into a movie, it is likely that the movie will resurrect the sales of the book; the book would become the bestseller in one or two weeks after the movie is in theatres*”. Since none of the previously developed algorithms consider flexibility in the mining process, they are not capable of discovering such a pattern.

The problems outlined above have sparked our motivations to extend the problem of closed pattern mining to a second aspect; therefore, the second algorithm, called CFP, is designed to mine closed flexible patterns in a time-series database.

The CFP algorithm has three phases involved in mining closed flexible patterns in a time-series database. Given a time-series database, we first transform it into a symbolic database based on the SAX representation [37]. From the transformed database, we recursively mine closed flexible patterns in a depth-first search manner. Specifically, we first obtain all frequent patterns of length one (frequent 1-patterns) and meanwhile, we build a projected database for each 1-pattern. Then, we grow each frequent *k*-pattern *p* by joining it to each nominee in *p*'s projected database, where a *k*-pattern is a pattern of length *k*, a nominee is a frequent 1-pattern in the projected database, and the number of gaps between *p* and the nominee is bounded by a user-specified maximum gap threshold. The process is repeated until no more closed flexible patterns can be generated.

Both the CMP and CFP algorithms involve a transformation of time-series sequences into symbolic sequences in the first phase. Although analyzing on symbolic sequences is ideal to reduce the noises and ease the mining process, these approaches may lead to pattern lost and the sequences supporting the same pattern may look quite different.



(a) (b) (c)

(d) (e) (f)

Fig. 1.2. Multi-sequence time-series *X*, *Y*, and *Z*.

In the SAX representation, two closed values may fall into different symbols if a breakpoint is set in between them. For example, the values 0.42 and 0.43 are very close to each other but they are assigned to different symbols if the breakpoint is set to 0.43. Let the breakpoints be -0.43 and 0.43 in the SAX representation, that is, we assign symbol *a* to each value less than -0.43, symbol *b* to each value greater than or equal to -0.43 and less than 0.43, and symbol *c* to each value greater than or equal to 0.43. As shown in Figs. 1.2a, 1.2b, and 1.2c, three multi-sequences have the same pattern $\begin{Bmatrix} c & b & b & b & a & a \\ a & b & a & b & b & c \end{Bmatrix}$. However, multi-sequence *Z* is quite different from *X* and *Y*.

To overcome the issue raised in symbolic sequence analysis, we have intrigued to mine closed patterns directly from the raw data. That is, we mine the patterns without any transformation from numerical values to symbols. For the sequences shown in Figs. 1.2a and 1.2b, we will have the pattern $\begin{Bmatrix} 0.5 & 0.4 & -0.2 & -0.1 & -0.5 & -0.6 \\ -0.6 & -0.3 & -0.5 & 0.4 & 0.2 & 0.8 \end{Bmatrix}$ since both sequences of *X* and *Y* are close to each other.

Visualizing data in the multiple resolutions helps decision-makers to make high-quality decisions. The multi-resolution views provided by Discrete Wavelet Transformation (DWT), such as the Haar wavelet transform, improve visualization and make patterns, trends, surprises, and relationships easier to identify [50]. Figs. 1.2a, 1.2b, and 1.2c show time-series *X*, *Y*, and *Z* in the high resolution, respectively, whereas Figs. 1.2d, 1.2e, and 1.2f show the transformed time-series *X*, *Y*, and *Z* in the low resolution, respectively. As observed, different perspectives of data are captured in different resolutions; the time-series in the low resolution show overall trends, whereas the time-series in the high resolution involve some fluctuations and reveal more detailed information. This awareness has motivated us to integrate the concept of multi-resolution visualizations in the mining process.

Therefore, we propose an algorithm, called CNP, for mining multi-resolution closed numerical patterns in a time-series database, where each time-series consists of multiple sequences. Given a time-series database, we first apply the Haar wavelet transform [19] to convert each time-series in the database into a sequence in the low resolution. Second, we find all frequent patterns of length one (frequent 1-patterns) from the transformed database (low resolution). Third, we recursively extend each frequent *k*-pattern to form frequent (*k*+1)-patterns. Subsequently, we restore closed patterns back to the high (original) resolution. Finally, we obtain all closed patterns in the low and high resolutions. The advantage of applying a wavelet transform on data is that we can view data from different perspectives by discovering patterns in different resolutions. Moreover, we have shown that the mining process is speeded up and the final outcomes would be the same as those without wavelet transform.

To inspire the study of the CNP algorithm, let us consider the following examples. Initially, a hospital has applied the CNP algorithm to visualize the medical data related to disease *X* and find significant patterns in different resolutions about symptoms, such as a pattern found in the low resolution could be “*body temperature movements appear to be cyclic and meanwhile the blood pressure remains high in the first few weeks*”, whereas a pattern found in the high resolution could be “*body temperature remains above normal for four consecutive days and drops in the normal range for five consecutive days and then increases again. Meanwhile, the blood pressure has little fluctuations throughout the day but shows an increasing trend in overall*”. To diagnose whether a patient has disease *X*, a doctor may first use the pattern in the low resolution to reject the possibility if the patient’s symptoms do not conformed to the pattern. Otherwise, the doctor may take a closer look on the pattern in the high resolution to confirm the diagnosis.

Let us consider a case of embracing the use of the CNP algorithm in the business world. Corporation *W* has over hundred fast food chain stores. In order to improve the operational efficiency and assist managers in decision making, data analyzers apply the CNP algorithm to find patterns in a database where each record in the database represents a branch and each record contains two sequences: the number of customers and the delay to fulfill the meal order and these values are recorded on a hourly basis for each day. Since the CNP algorithm is able to generate patterns in the multiple resolutions, decision-makers can focus on a targeted area of their interests. For instance, the chief executive officer (CEO) of the corporation may be only interested in a broader view on data, and hence he/she focuses on the patterns found in the low resolution, whereas the branch managers may concern more detail information in data and thus they concentrate on the patterns found in the high resolution. These patterns may help the corporation to make staffing decisions and improve service delivery efficiency. As seen from the above examples, the study on mining patterns in the multiple resolutions can be used for many real-world applications.

The choice of symbolic or numerical methods depends on the visualizations required by the users or the applications in different areas. For instance, we may apply the symbolic method to discover patterns in the fields of finance or meteorology because users simply demand an overview of trends, whereas in the field of medical science or disaster forecasting (e.g. hurricane), the numerical method is adopted because doctors or experts need rigorous patterns that can help them to diagnose diseases or predict disasters.

1.2 Contributions

The work of the CMP algorithm in this dissertation has been published on Data and Knowledge Engineering Journal [32] and the work of the CFP algorithm has been published on Expert Systems with Applications Journal [64]. We summarize the contributions of this dissertation as follows.

1. A novel concept of mining closed multi-sequence patterns from a time-series database is presented.
2. An efficient algorithm, called CMP, is proposed to mine closed multi-sequence patterns and it requires only one database scan and can localize support counting, candidate pruning, and closure checking in the projected databases. Therefore, it can efficiently mine closed patterns.
3. The innovative idea of using flexible-range of consecutive gaps is stated to overcome the limitations of exact sequence alignments.
4. A novel algorithm, called CFP, is proposed to mine closed flexible patterns in a

time-series database and two pruning strategies and a closure checking scheme are designed to reduce the search space and thus speed up the algorithm.

5. To overcome the issue raised in the symbolic sequence mining, we tackle the problem of mining numerical patterns in a time-series database.
6. A novel algorithm, called CNP, is proposed to mine multi-resolution closed numerical multi-sequence patterns in a time-series database. It integrates the concept of the Haar wavelet transform to mine patterns in different resolutions. It also adopts two pruning strategies to accelerate the mining process.
7. The performance of the proposed algorithms is evaluated with both synthetic and real datasets. The experimental results show that the CMP algorithm outperforms the modified Apriori and BIDE algorithms by one or two orders of magnitude; the CFP algorithm outperforms the modified Apriori algorithm by an order of magnitude; and the CNP algorithm outperforms the modified A-Close algorithm by one or two orders of magnitude.

1.3 Dissertation layout

The remainder of this dissertation is organized as follows. Chapter 2 provides a literature review to support the study undertaken in this dissertation. Chapter 3 presents the CMP algorithm for mining closed multi-sequence patterns. Chapter 4 introduces the CFP algorithm for mining closed flexible patterns. Chapter 5 explains the CNP algorithm for mining multi-resolution closed numerical patterns. Chapter 6 illustrates the performance evaluation of the proposed algorithms. Finally, we conclude our works and suggest some future research directions in Chapter 7.

Chapter 2 Literature Review

In this chapter, a review of existing studies is presented to support the research undertaken in this dissertation.

2.1 Time-series mining

A time-series database consists of sequences of values (or events) changing over time. The values are typically measured at equal time intervals [19]. The analysis of time-series is often associated with the discovery of patterns, such as query matching patterns, periodic patterns, and similar patterns, etc.

To find patterns in a time-series database, Faloutsos et al. [17] proposed a method to find the subsequences similar to the query pattern in the database. Kontaki et al. [28] used the IDC-index to provide an efficient access method for query processing. Berndt and Clifford [6] presented a dynamic programming approach to find patterns in a time-series database. Lee et al. [33] developed a method to find fuzzy candlestick patterns for financial prediction. Teoh et al. [59] proposed a hybrid fuzzy time-series model, which combines cumulative probability distribution approach and rough set rule induction to forecast stock markets. Han et al. [18] designed an algorithm to find periodic patterns. Yang [67] showed an approach to detect intrusions. Takeuchi and Yamanishi [58] proposed a probabilistic model for detecting outliers and change points from time-series databases. Wang et al. [63] presented an approach to derive group patterns of mobile users based on their movement data.

2.2 Sequential pattern mining

Sequential pattern mining has emerged since Srikant and Agrawal [52] first defined the problem of mining sequential patterns in 1995. The GSP algorithm [54] adds time constraints to find all sequential patterns. However, it suffers from poor performance with respect to long sequences. Han et al. [20] introduced the idea of data projection and developed the FreeSpan algorithm to recursively mine sequential patterns. Pei et al. [45] proposed the PrefixSpan algorithm for mining long sequential patterns in large sequence databases. It continuously mines the patterns from projected databases, which speed up the candidate subsequence generation. Kim et al. [27] considered the importance of quantitative information associated with each item. They extended naïve algorithms, both Apriori and PrefixSpan, to mine sequential patterns with quantities. Wang and Lee [60] presented an approach, called EISP-Miner, to mine inter-sequence patterns, where they considered not only the relationships between items in a sequence but also the relationships between sequences in inter-sequence patterns. Masseglia et al. [39] pushed the time constraints into the process of mining sequential patterns. They built sequence graphs with respect to the constraint of window size and gap constraints, and used them to count the support of each candidate pattern. Chu et al. [13] proposed an algorithm, called EFI-Mine, to mine emerging frequent patterns from data streams. The EFI-Mine algorithm is to find patterns that are infrequent in the previous time window and may become frequent in the following time windows. Masseglia et al. [40] presented an algorithm for incremental mining of sequential patterns when new transactions are added to the database. Lee and Wang [31] developed a method for mining calling path patterns in GSM networks. Chen and Hsu [10] proposed an approach to discover tree-like patterns in a large dataset. Other researchers adopted different representations to improve the efficiency of mining long sequential patterns. SPADE [69] is based on a vertical id-list format and uses a lattice-theoretical approach to decompose the original search space into smaller spaces. SPAM [4] adopts a vertical bitmap representation. It has been shown that SPAM is more efficient in mining long sequential patterns than SPADE and PrefixSpan; however, it consumes more space. Lin et al. [38] presented the METISP algorithm to mine sequential patterns with various time constraints, including minimum gaps, maximum gaps, exact gaps, sliding windows, and durations. They also introduced the idea of using time-indexes to minimize the search space of potential patterns.

Recently, several variations and applications on sequential pattern mining are proposed. Perera et al. [47] analyzed online collaborative learning data and exploited group interactions in order to improve the teaching activities. They have applied clustering and sequential pattern mining in seeking interesting patterns. Huang et al. [23] considered both time and location information and developed a framework to mine sequential patterns in a large spatio-temporal database. Peng and Liao [46] believed that events may occur in multiple domains and thus they introduced a novel mining task, called the multi-domain sequential pattern mining problem. Chen and Huang [11] applied a concept hierarchy and fuzzy techniques for mining fuzzy multi-level sequential patterns. Although the performance of their method is worse than the traditional models, they could discover more interesting patterns. Huang et al. [22] proposed a progressive algorithm, called Pisa, to mine sequential patterns in both dynamic and static types of databases.

2.3 Closed pattern mining

A closed pattern is defined as a pattern that is frequent and has no super-pattern with the same support. The concept of mining closed patterns has been proposed to avoid unnecessary frequent patterns while preserving the same information. Pasquier et al. [43] proposed an algorithm, called A-Close, which adopts the same join step as the Apriori algorithm to form candidate generators and then compute closures of these generators to find all frequent closed itemsets. Although some pruning strategies have been employed, the cost of candidate generation and closure computation still lead to an efficiency bottleneck. To reduce the memory and search space for closure checking, TFP [21] adopts a two-level hash indexed tree structure to store the already mined closed itemset candidates. CHARM [70] adopts a vertical data format to mine closed itemsets and uses four pruning properties to remove non-closed itemsets. The CLOSET

[44] and CLOSET+ [62] algorithms both adopt the FP-growth framework to find frequent closed itemsets, where a FP-tree is built to speed up pattern discovery. Yan et al. [66] proposed an algorithm, called CloSpan, for mining closed sequential patterns. It first generates a set of closed candidates stored in a hash indexed tree and then performs post-pruning on those candidates. Because CloSpan needs to maintain the set of closed candidates, it consumes a large amount of memory for closure checking. Wang and Han [61] introduced an algorithm, called BIDE, for mining closed sequential patterns without maintaining candidates. They used forward and backward directional extension checking to perform closure checking and to prune the redundant patterns in the mining process. Lee et al. [30] introduced a method, called ICMiner, that efficiently mines closed inter-transaction itemsets. ICMiner involves the concept of an ID-tree and domain attributes and applies effective pruning strategies to avoid generating unnecessary candidates. In order to meet the dynamic characteristic of online data streams, Li et al. [36] proposed an algorithm, called NewMoment, to mine closed patterns. The NewMoment algorithm uses an effective bit-sequence representation to simplify the support calculation, and hence results in less memory and execution time. Ji et al. [24] introduced the idea of partitioning the original data sets to accelerate the task of mining frequent closed patterns on dense data sets. Yuan et al. [68] proposed an algorithm, called CISpan, to find closed sequential patterns in an incremental database. The key idea is to build an incremental lattice to store current closed patterns and handle the insertion and the removal of sequences using different strategies, such as merge and split. Chang et al. [8] presented an effective algorithm to mine closed sequential patterns over sliding windows in a data stream environment. They designed a synopsis structure IST to store closed sequential patterns in memory and adopted some strategies to prune search space.

2.4 SAX representation

The objective of the Symbolic Aggregate approXimation (SAX) representation [37] is to transform a time-series sequence into a symbolic sequence. The method contains two steps as follows.

In the first step, a time-series of length *n* is divided into *m* equal segments, and the average of each segment is calculated by:

$$\overline{T_i} = \frac{\sum_{j=x(i-1)+1}^{xi} T_j}{x}$$

The numerator represents the sum of all elements within a segment while the denominator *x* represents the number of elements falling within a segment (*x* = *n*/*m*). Thus, we obtain $\overline{T_1}$ as the average of the first segment and $\overline{T_2}$ as the average of the second segment, and so forth. As a result, a new representation of the time-series sequence is presented as $\overline{T} = \overline{T_1}, \overline{T_2}, ..., \overline{T_m}$, also known as Piecewise Aggregate Approximation (PAA) representation.

Table 2.1. The breakpoint table [29].

α B_i	3	4	5
B_1	-0.43	-0.67	-0.84
B_2	0.43	0	-0.25
B_3		0.67	0.25
B_4			0.84

In the second step, the PAA representation is transformed into a symbolic sequence according to the breakpoints in a breakpoint mapping table. According to empirical experiments [37], the normalized subsequences have highly Gaussian distributions. Thus, the "breakpoints" can be determined so that equal-sized areas are produced under a Gaussian curve. The breakpoints are defined as a list of ordered numbers B = B_1, B_2, …, B_{a-1} such that the area under an $N(0,1)$ Gaussian curve from B_i to B_{i+1} = 1/α. The statistical table [29] is used to determine the breakpoint for different values of α, as shown in Table 2.1.

For example, if α = 3, three breakpoint regions are [-∞, -0.43), [-0.43, 0.43), [0.43, ∞]. All PAA coefficients falling within the first region are mapped to symbol "*a*" and the others falling within the second and third regions are mapped to symbols "*b*" and "*c*", respectively. Fig. 2.1 illustrates how a time-series sequence is transformed into a symbolic sequence, where n = 30, m = 6, and α = 3. The time-series sequence is mapped to the symbolic sequence "*ccbabc*."

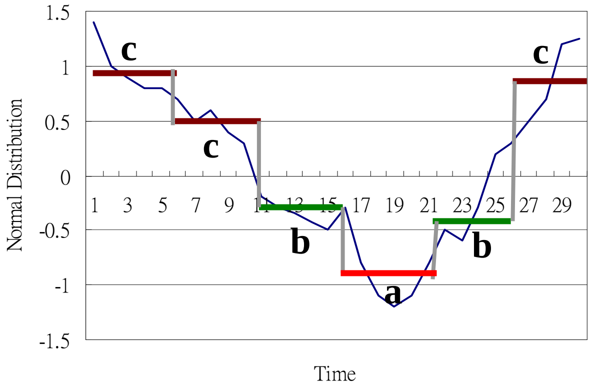


Fig. 2.1. A SAX example.

It is important to note that some time-series data may violate the Gaussian assumption. To deal with such data, several discretization methods can be used to transform time-series sequences into symbolic sequences. The Equal Frequency (EQF) method [19] divides the value range into bins, where each bin has equal frequency of values and is represented by a symbol. For the Equal Width (EQW) method [19], the range of data is divided into segments of equal length and each segment is represented by a symbol. The clustering-based method [15] discretizes time-series by grouping similar data values in a cluster and assigning a symbol to each cluster. The Persist algorithm [41] determines cut points in an adaptive way. It continuously chooses the best cut point from a set of candidates until the desired number of bins is obtained.

2.5 Discussion

In this chapter, we have surveyed the literatures of time-series mining, sequential pattern mining, and closed pattern mining, and the concept of SAX representation. In time-series mining, the main research interests are finding the most similar time-series in a database to a query time-series, finding groups of similar time-series, finding periodic patterns, and detecting anomalies contained in a given time series. In sequential pattern mining, researchers focus on the problem of discovering frequent patterns in one-sequence databases, that is, each transaction consists of single sequence in the database. Moreover, they require exact sequence alignments in discovering patterns without the consideration of noisy environments. In closed pattern mining, researchers recognize the drawback of sequential pattern mining, that is, identifying all frequent patterns is very time-consuming due to a large number of patterns generated. Thus, they incorporate the concept of closed patterns to overcome this drawback.

Most previously proposed methods in these areas did not consider multiple sequences in a transaction and the gaps between the events in a pattern, and hence they are not suitable for mining closed patterns in a multi-sequence time-series database. Therefore, in this dissertation, we focus on the problem of mining closed patterns in a database, where each transaction contains multiple time-series sequences. To the best of our knowledge, there is no method specially designed to relax the constraint of exact sequence alignments in discovering patterns. Thus, in this dissertation, we introduce the problem of mining closed flexible patterns in a time-series database, where a flexible number of gaps between items in a pattern are allowed. Although a significant amount of research efforts has been devoted to pattern mining, no mining results can provide multiple perspectives on datasets to decision-makers. In this dissertation, we explore the idea of mining multi-resolution patterns in time-series databases.

Based on the above analysis, in this dissertation, we propose three algorithms, CMP, CFP, and CNP algorithms, to mine closed multi-sequence patterns, closed flexible patterns, and multi-resolution closed numerical patterns in time-series databases. These algorithms are described in detail in chapters 3, 4, and 5.

Chapter 3 Mining Closed Patterns in Multi-Sequence Time-Series Databases

In this chapter, we propose an efficient algorithm, called CMP (Closed Multi-sequence Patterns mining), to mine closed patterns in a time-series database, where each transaction in the database contains multiple time-series sequences. The proposed algorithm consists of three phases. First, we transform each time-series sequence in a transaction into a symbolic sequence. Second, we scan the transformed database to find frequent patterns of length one. Third, for each frequent pattern found in the second phase, we recursively enumerate frequent patterns by a frequent pattern tree as described in Section 3.2 in a depth-first search manner.

3.1 Preliminaries and problem definitions

Given a database where each transaction consists of multiple time-series sequences, we first transform each sequence into a SAX symbolic sequence and then mine closed patterns in the transformed database. Note that the database may contain transactions with different lengths. As well, multiple time-series sequences contained in each transaction can be in different lengths.

Definition 3.1 Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of distinct items. A *sequence S* is an ordered list of items, denoted as $<e_1e_2\ldots e_m>$, where e_i is an item, $e_i \in I$, for $1 \leq i \leq m$. Assume there exists a lexicographical ordering $\leqq$ among the items in *I*.

For example, $a \leqq b \leqq c$ or $x \leqq y \leqq z$. To represent a sequence flexibly, we introduce a gap symbol "_", which is a wild card symbol and may be substituted by any item in *I*. The lexicographical order of the gap symbol is greater than any item in *I*. Moreover, multiple time-series sequences of different lengths contained in a transaction can be padded with gap symbols until they have the equal length. For example, a transaction is consisted of two time-series sequences $\begin{Bmatrix} a & b & a & a \\ x & y & \_ & \_ \end{Bmatrix}$, where the second sequence is padded with gap symbols to achieve the same length with the first sequence.

Another important feature to point out is that we allow time non-alignment between multiple sequences in a transaction and missing values in the sequences. We simply use gap symbols to adjust the non-alignment and replace all missing values with gaps. In other words, the non-aligned parts of the sequences may be filled up with gap symbols. For example, non-aligned sequences in a transaction $\left\{\begin{array}{ccccc} a & b & b & & \\ x & & z & x & y \end{array}\right\}$ can be replaced by $\left\{\begin{array}{ccccc} a & b & b & \_ & \_ \\ x & \_ & z & x & y \end{array}\right\}$.

Since a transaction in the database consists of multiple sequences, a pattern in the database is formed by multiple sequences.

Definition 3.2 $\left\{\begin{array}{cccc} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & & \vdots \\ a_{s1} & a_{s2} & \cdots & a_{sk} \end{array}\right\}$ is a pattern of length k, where a_{ij} is an item in I or a gap symbol. A pattern of length k is called a *k-pattern*.

For example, both $\left\{\begin{array}{c} a \\ x \end{array}\right\}$ and $\left\{\begin{array}{c} b \\ x \end{array}\right\}$ are 1-patterns; $\left\{\begin{array}{ccc} a & a & c \\ x & y & x \end{array}\right\}$ is a 3-pattern; $\begin{array}{c} S_1 \\ S_2 \end{array}\left\{\begin{array}{cccccccccc} a & b & \_ & \_ & b & c & & & & \\ & & & & x & y & \_ & \_ & y & z \end{array}\right\}$ is a 10-pattern, where *ab* occurs in sequence S_1, after two time units *bc* occurs in S_1 and *xy* simultaneously occurs in sequence S_2, and after two time units *yz* occurs in S_2. Note that the gap symbols after *bc* in S_1 and before *xy* in S_2 may be omitted.

Definition 3.3 The *concatenation* between patterns p_1 and p_2 is denoted as $p_1 \Theta p_2$.

For example, $\left\{\begin{array}{ccc} a & b & b \\ y & y & x \end{array}\right\} \Theta \left\{\begin{array}{cc} c & b \\ x & y \end{array}\right\} = \left\{\begin{array}{ccccc} a & b & b & c & b \\ y & y & x & x & y \end{array}\right\}$.

Definition 3.4 A pattern p = $\left\{\begin{array}{cccc} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{s1} & a_{s2} & \cdots & a_{sn} \end{array}\right\}$ is *contained* by another pattern *p*’

$= \left\{\begin{matrix} b_{11} & b_{12} & \cdots & b_{1m} \\ b_{21} & b_{22} & \cdots & b_{2m} \\ \vdots & \vdots & & \vdots \\ b_{s1} & b_{s2} & \cdots & b_{sm} \end{matrix}\right\}$ if there exists an integer j such that $a_{ik} = b_{i(k+j)}$ or a_{ik} = "_", $0 \le j \le m–n$, $n \le m$, $1 \le i \le s$, $1 \le k \le n$. We may also say that p' is a *super-pattern* of p, or that p is a *sub-pattern* of p'.

Definition 3.5 The *support* of a pattern p is defined as the number of transactions containing p in the database.

Definition 3.6 A pattern is *frequent* if its support is not less than a user-specified minimum support threshold, δ.

Definition 3.7 A frequent pattern p is *closed* if there does not exist any frequent super-pattern of p with the same support.

Note that the gap symbol in a pattern may be substituted by any item. If one pattern with the substituted items is a super-pattern of the other, and both patterns have the same support, we would delete the latter one. For example, $\left\{\begin{matrix} a \\ x \end{matrix}\right\}$ is a super-pattern of $\left\{\begin{matrix} a \\ \_ \end{matrix}\right\}$. If $\left\{\begin{matrix} a \\ x \end{matrix}\right\}$ and $\left\{\begin{matrix} a \\ \_ \end{matrix}\right\}$ have the same support, we should delete the pattern $\left\{\begin{matrix} a \\ \_ \end{matrix}\right\}$ because it is less representative.

Definition 3.8 The *projection* of a pattern p in a transaction t is defined as the rest of t behind p, where t contains p.

Note that since a pattern p may appear more than once in a transaction, there exists a projection for each occurrence of p. That is, a transaction may contain multiple projections of p. For example, the projections of $\left\{\begin{matrix} a \\ x \end{matrix}\right\}$ in a transaction $\left\{\begin{matrix} a & b & a & a \\ x & y & x & y \end{matrix}\right\}$ are $\left\{\begin{matrix} b & a & a \\ y & x & y \end{matrix}\right\}$ and $\left\{\begin{matrix} a \\ y \end{matrix}\right\}$.

Definition 3.9 The *projected database* of a pattern p contains all the projections of p in the database.

Since a pattern may contain many gaps, discovering all such patterns may require a lot of resources, but a user may only be interested in patterns with a certain number of gaps. Therefore, to avoid wasting resources by mining unwanted patterns, we introduce a parameter called a user-specified maximum gap threshold τ. When mining closed patterns, we only mine the patterns so that the number of gaps between any two adjacent 1-patterns in the patterns is not greater than τ.

To mine closed patterns, we build the projected database of a pattern and check if there exists any frequent 1-pattern in its projected database. Since the number of gaps between any two adjacent 1-patterns is not greater than τ, to grow a pattern *p*, we may find the frequent 1-patterns only from the first τ+1 positions in *p*'s projected database.

For example, assume that $\delta = 2$ and $\tau = 1$. Given two transactions $\begin{Bmatrix} a & b & a & c \\ x & y & x & z \end{Bmatrix}$ and $\begin{Bmatrix} a & c & b & c \\ x & z & z & x \end{Bmatrix}$, we first find all 1-frequent patterns, namely, $\begin{Bmatrix} a \\ x \end{Bmatrix}$, $\begin{Bmatrix} a \\ \_ \end{Bmatrix}$, $\begin{Bmatrix} b \\ \_ \end{Bmatrix}$, $\begin{Bmatrix} c \\ z \end{Bmatrix}$, $\begin{Bmatrix} c \\ \_ \end{Bmatrix}$, $\begin{Bmatrix} \_ \\ x \end{Bmatrix}$ and $\begin{Bmatrix} \_ \\ z \end{Bmatrix}$, where their supports are all equal to 2. From the projected database of $\begin{Bmatrix} a \\ x \end{Bmatrix}$, we know that $\begin{Bmatrix} a \\ x \end{Bmatrix}$ appears at the first and third positions of the first transaction, and at the first position of the second transaction. In the projected database of $\begin{Bmatrix} a \\ x \end{Bmatrix}$, formed by the first position of the first transaction and the first position of the second transaction, we cannot find any frequent 1-pattern from the first two positions in the projected database since $\tau = 1$. However, in $\begin{Bmatrix} a \\ x \end{Bmatrix}$'s projected database formed by the third position of the first transaction and the first position of the second transaction, we find three frequent 1-patterns from the first two positions in the projected database, namely, $\begin{Bmatrix} c \\ z \end{Bmatrix}$, $\begin{Bmatrix} c \\ \_ \end{Bmatrix}$, and $\begin{Bmatrix} \_ \\ z \end{Bmatrix}$. Therefore, we obtain three frequent 2-patterns, namely, $\begin{Bmatrix} a & c \\ x & z \end{Bmatrix}$, $\begin{Bmatrix} a & c \\ x & \_ \end{Bmatrix}$, and $\begin{Bmatrix} a & \_ \\ x & z \end{Bmatrix}$, where their supports are all equal to 2. Since $\begin{Bmatrix} a & c \\ x & \_ \end{Bmatrix}$ and $\begin{Bmatrix} a & \_ \\ x & z \end{Bmatrix}$ are sub-patterns of $\begin{Bmatrix} a & c \\ x & z \end{Bmatrix}$, they are not closed. Since we cannot find any frequent 1-patterns in the projected databases of $\begin{Bmatrix} a & c \\ x & z \end{Bmatrix}$, $\begin{Bmatrix} a & c \\ x & \_ \end{Bmatrix}$, and $\begin{Bmatrix} a & \_ \\ x & z \end{Bmatrix}$, we stop growing these patterns.

To find the closed patterns in $\begin{Bmatrix} a \\ \_ \end{Bmatrix}$'s projected database is quite similar to that in $\begin{Bmatrix} a \\ x \end{Bmatrix}$'s projected database. Next, we examine $\begin{Bmatrix} b \\ \_ \end{Bmatrix}$'s projected database to see if any frequent 1-patterns exist in the projected database, and find a frequent 1-pattern $\begin{Bmatrix} \_ \\ x \end{Bmatrix}$. Thus, we obtain a frequent 2-pattern $\begin{Bmatrix} b & \_ \\ \_ & x \end{Bmatrix}$. Since we cannot find any frequent 1-pattern in $\begin{Bmatrix} b & \_ \\ \_ & x \end{Bmatrix}$'s projected database, we stop growing the pattern. Similarly, we may obtain all frequent patterns by growing all frequent 1-patterns as described above. Finally, we may obtain two closed patterns, namely, $\begin{Bmatrix} a & c \\ x & z \end{Bmatrix}$, and $\begin{Bmatrix} b & \_ \\ \_ & x \end{Bmatrix}$.

3.2 Frequent pattern tree

The CMP algorithm uses a frequent pattern tree to enumerate frequent patterns, which may be constructed in the following way. The root of the tree is labeled by $\varnothing$. Next, we scan the database and find all frequent 1-patterns in the database and add these 1-patterns to level 1 of the frequent pattern tree. Then, we can recursively extend a frequent pattern p at level l ($\geq$ 1) to get its frequent super-pattern at the next level by appending g gaps and a frequent 1-pattern in p's projected database, where $0 \leq g \leq \tau$. Note that g = 0 means that no gap is appended.

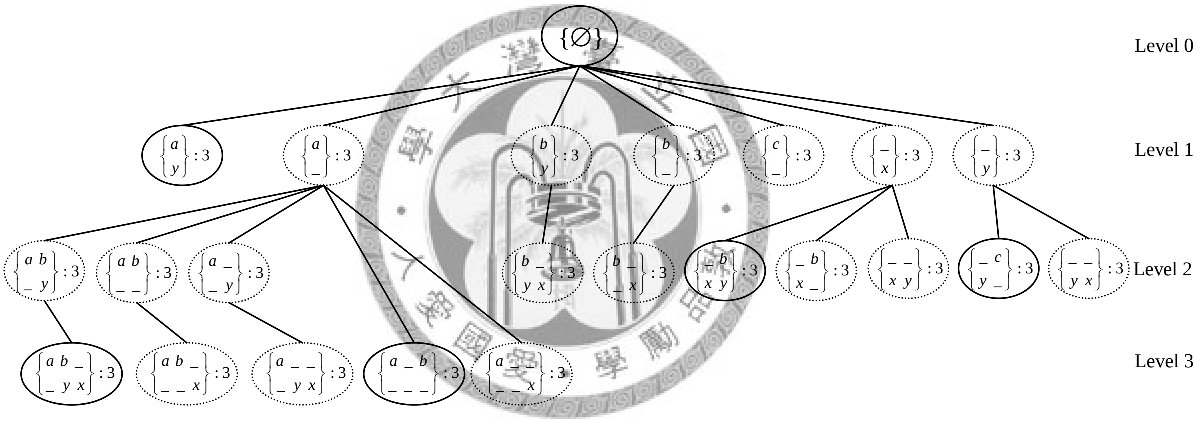
It has been shown that the algorithm based on the depth-first search (DFS) approach is more efficient in mining long patterns than that based on the breadth-first search (BFS) approach [4]. Hence, the CMP algorithm enumerates the frequent patterns by a frequent pattern tree in a DFS manner. Fig. 3.2 illustrates the frequent pattern tree for the example database in Fig. 3.1, where δ = 3, τ = 1, the number after the pattern is the support of the pattern, and the patterns in the bold circles are closed patterns, and those in the dotted circles are not. Note that the children of a node are sorted lexicographically and the gap symbol “_” may be included in a pattern.

$$\mathrm{T}_1\text{:}\ \begin{Bmatrix} a & b & c & a & b & b \\ y & y & x & x & y & x \end{Bmatrix}$$

$$\mathrm{T}_2\text{:}\ \begin{Bmatrix} a & c & b & a & b & b \\ y & x & y & z & y & x \end{Bmatrix}$$

$$\mathrm{T}_3\text{:}\ \begin{Bmatrix} a & c & a & a & b & a \\ y & z & y & x & y & x \end{Bmatrix}$$

Fig. 3.1. An example of a transformed multi-sequence database.



$\{\varnothing\}$
Level 0
$\begin{Bmatrix} a \\ y \end{Bmatrix}:3$
$\begin{Bmatrix} a \\ \_ \end{Bmatrix}:3$
$\begin{Bmatrix} b \\ y \end{Bmatrix}:3$
$\begin{Bmatrix} b \\ \_ \end{Bmatrix}:3$
$\begin{Bmatrix} c \\ \_ \end{Bmatrix}:3$
$\begin{Bmatrix} \_ \\ x \end{Bmatrix}:3$
$\begin{Bmatrix} \_ \\ y \end{Bmatrix}:3$
Level 1
$\begin{Bmatrix} a & b \\ \_ & y \end{Bmatrix}:3$
$\begin{Bmatrix} a & b \\ \_ & \_ \end{Bmatrix}:3$
$\begin{Bmatrix} a & \_ \\ \_ & y \end{Bmatrix}:3$
$\begin{Bmatrix} b & \_ \\ y & x \end{Bmatrix}:3$
$\begin{Bmatrix} b & \_ \\ \_ & x \end{Bmatrix}:3$
$\begin{Bmatrix} \_ & b \\ x & y \end{Bmatrix}:3$
$\begin{Bmatrix} \_ & b \\ x & \_ \end{Bmatrix}:3$
$\begin{Bmatrix} \_ & \_ \\ x & y \end{Bmatrix}:3$
$\begin{Bmatrix} \_ & c \\ y & \_ \end{Bmatrix}:3$
$\begin{Bmatrix} \_ & \_ \\ y & x \end{Bmatrix}:3$
Level 2
$\begin{Bmatrix} a & b & \_ \\ \_ & y & x \end{Bmatrix}:3$
$\begin{Bmatrix} a & b & \_ \\ \_ & \_ & x \end{Bmatrix}:3$
$\begin{Bmatrix} a & \_ & \_ \\ \_ & y & x \end{Bmatrix}:3$
$\begin{Bmatrix} a & \_ & b \\ \_ & \_ & \_ \end{Bmatrix}:3$
$\begin{Bmatrix} a & \_ & \_ \\ \_ & \_ & x \end{Bmatrix}:3$
Level 3

Fig. 3.2. The frequent pattern tree.

3.3 Closure checking and pruning strategies

When we generate a new frequent pattern, we need to do some closure checking to assure whether the pattern generated is closed. Following a similar concept used in BIDE [61], we use bi-directional (forward and backward) checking to determine if the pattern generated is closed.

Lemma 3.1 (Forward checking). *A pattern p is not closed if a frequent 1-pattern q exists such that* (1) *q appears after p and occurs in every transaction of p’s projected database, and* (2) *the distance between p and q is the same in every transaction of p’s* *projected database and not greater than* τ+1 *positions*.

Proof. If a frequent 1-pattern *q* exists such that (1) *q* appears after *p* and occurs in every transaction of *p*'s projected database, and (2) the distance between *p* and *q* is not greater than τ+1 positions, it means that we can always find another frequent pattern formed by *p* and *q* whose support is equal to *p*'s support. Therefore, *p* must not be closed.

Note that *p* may appear more than once in a transaction. If we cannot find any *q* after the current appearance of *p*, we need to perform the same check on the next appearance of *p* in the transaction.

Next, we illustrate the forward checking strategy as follows. Assume $\left\{\begin{array}{cccccc} a & \_ & b & c & & \\ & & x & y & \_ & z \end{array}\right\}$ is a frequent pattern. If we find a 1-pattern $\left\{\begin{array}{c} \_ \\ y \end{array}\right\}$ at the first position of every transaction in its projected database, then we can assure that $\left\{\begin{array}{cccccc} a & \_ & b & c & & \\ & & x & y & \_ & z \end{array}\right\}$ is not closed because $\left\{\begin{array}{ccccccc} a & \_ & b & c & & & \\ & & x & y & \_ & z & y \end{array}\right\}$ is its super-pattern and has the same support.

Lemma 3.2 (**Backward checking**). *A pattern p does not need to be grown if a frequent 1-pattern q exists such that* (1) *q appears before p and occurs in every transaction of p's projected database*, *and* (2) *the distance between q and p is the same in every transaction of p's projected database and not greater than* τ+1 *positions*.

Proof. If a frequent 1-pattern *q* exists such that (1) *q* appears before *p* and occurs in every transaction of *p*'s projected database, and (2) the distance between *q* and *p* is the same in every transaction of *p*'s projected database and not greater than τ+1 positions, it means that we can always find another frequent pattern formed by *q* and *p* and its support is equal to *p*'s support. Therefore, *p* is not closed and it does not need to be grown.

Note that *p* may appear more than once in a transaction. Thus, we need to check if *q* exists before every appearance of *p* in the transaction.

Next, we illustrate the backward checking strategy as follows. Assume pattern

$\begin{Bmatrix} a & b & \_ & \_ & & \\ & & & & x & y \end{Bmatrix}$ is a frequent pattern. If we find a pattern (for example, $\begin{Bmatrix} b \\ \_ \end{Bmatrix}$) right before $\begin{Bmatrix} a & b & \_ & \_ & & \\ & & & & x & y \end{Bmatrix}$ in every transaction of the projected database, we can conclude that $\begin{Bmatrix} a & b & \_ & \_ & & \\ & & & & x & y \end{Bmatrix}$ does not need to be grown since there must exist another frequent pattern $\begin{Bmatrix} b & a & b & \_ & \_ & & \\ & & & & & x & y \end{Bmatrix}$ whose support is equal to the support of $\begin{Bmatrix} a & b & \_ & \_ & & \\ & & & & x & y \end{Bmatrix}$. Thus, pattern $\begin{Bmatrix} a & b & \_ & \_ & & \\ & & & & x & y \end{Bmatrix}$ may be pruned because the pattern $\begin{Bmatrix} b & a & b & \_ & \_ & & \\ & & & & & x & y \end{Bmatrix}$ would be generated later.

Lemma 3.3 (**Same projections).** *A pattern p can be pruned during the process of mining closed patterns if* (1) *p shares the same parent with a pattern q in the frequent pattern tree,* (2) *q contains p, and* (3) *p's projections are identical to q*'*s projections.*

Proof. Since both patterns *p* and *q* share the same projections and *q* contains *p*, every pattern generated from *p* is contained by a pattern generated from *q* and both patterns generated have the same support. That is, every pattern generated from *p* is not closed. Thus, *p* may be pruned during the process of mining closed patterns.

For example, as shown in Fig. 3.2, pattern $\begin{Bmatrix} a & b \\ \_ & y \end{Bmatrix}$ contains pattern $\begin{Bmatrix} a & b \\ \_ & \_ \end{Bmatrix}$ and both patterns share the same parent in the frequent pattern tree. Moreover, both patterns have the same projections since both patterns appear at the first and fourth positions of the first transaction, at the fourth position of the second transaction, and at the fourth position of the third transaction. Therefore, pattern $\begin{Bmatrix} a & b \\ \_ & \_ \end{Bmatrix}$ may be pruned.

3.4 The CMP algorithm

The proposed CMP algorithm is shown in detail in Fig. 3.3. First, we transform every sequence in the time-series database into the SAX representation [37] as described in Section 2.4. That is, the time-series database is transformed into a database where each transaction in the database contains multiple symbolic sequences. Second, we scan the transformed database to find all frequent 1-patterns, and build a projected database for each frequent 1-pattern. Third, for each frequent pattern found, the algorithm calls the CMP-Growth procedure to recursively use a frequent *k*-pattern and its projected database to generate its frequent super-patterns at the next level in the frequent pattern tree, where $k \geq 1$.

```
Algorithm: CMP
Input: an input database DB, a minimum support threshold $\delta$, a maximum gap
       threshold $\tau$
Output: all closed patterns CP
Transform each sequence in DB into a symbolic sequence by using the SAX
    representation;
Let CP be $\emptyset$;
Scan the transformed database to find all frequent 1-patterns. Let P1 be all
    frequent 1-patterns found in DB;
for each 1-pattern p in P1 do
    Let PDB be the projected database of p;
    call CMP-Growth (PDB, p, $\delta$, $\tau$, CP);
end for
return CP;
```

Fig. 3.3. The CMP algorithm.

Fig. 3.4 shows the CMP-Growth procedure. In step 1, for each frequent *k*-pattern *p*, we build its projected database and find candidate frequent 1-patterns that occur within the first τ+1 positions in the projected database. In steps 3-8, we use the backward checking strategy to check if *p* can be pruned. If not, we use the forward checking strategy to check if *p* is closed. If so, it is added to the closed pattern pool (*CP*). In steps 10-16, for each frequent 1-pattern *c* found in step 1, we concatenate *p*, *g* gaps and *c* together to generate a frequent (*k*+*g*+1)-pattern, where $0 \leq g \leq \tau$. Then, we check if the concatenated pattern is contained by any sibling pattern and both share the same projected database. If not, the CMP-Growth procedure is called recursively to enumerate the frequent super-patterns of the newly generated frequent (k+g+1)-pattern. The process is repeated until no more patterns can be generated.

```
Algorithm: CMP-Growth (PDB, p, $\delta$, $\tau$, CP)
Input: a projected database PDB, a prefix pattern p, a minimum support threshold $\delta$, a
       maximum gap threshold $\tau$
Output: a set of closed patterns CP
Let candidate be a set of frequent 1-patterns within the first $\tau$+1 positions in PDB;
Let sup be the support of p;
if (sup is not less than $\delta$ and p passes the backward checking strategy) then
    if (p passes the forward checking strategy) then
        if (p is closed with respect to CP) then
            Add p to CP;
        end if
    end if
    if (candidate is not empty) then
        for each 1-pattern c in candidate do
            for g = 0 to $\tau$ do
                Let PDB' be the projected database of $p \ominus G \ominus c$ where G
                    contains g gaps in each sequence;
                Check if $p \ominus G \ominus c$ is contained by any sibling pattern and
                    both share the same projections;
                if not, call CMP-Growth (PDB', $p \ominus G \ominus c$, $\delta$, $\tau$, CP);
            end for
        end for
    end if
end if
return CP;
```

Fig. 3.4. The CMP-Growth procedure.

Lemma 3.4 *Each pattern found by the CMP algorithm is frequent and closed*.

Proof. In step 3 of the CMP algorithm, we generate all frequent 1-patterns by scanning the database once. In step 1 of the CMP-Growth procedure, we find a set of candidate frequent 1-patterns, which appear within the first τ+1 positions in the projected database of a frequent *k*-pattern *p*. These candidates are then concatenated with *p* and with gaps to form frequent (*k*+*g*+1)-patterns, $k \geq 1$. Since we have always checked the support of each newly pattern found against the minimum support threshold, we assure each pattern found by the CMP algorithm is frequent. Moreover, at each level of extension, we adopt the forward and backward checking strategies to eliminate non-closed patterns. Therefore, every pattern found by the CMP algorithm is frequent and closed.

Lemma 3.5 *Every closed pattern can be found by the CMP algorithm*.

Proof. Since we scan the database once to find all frequent 1-patterns, every frequent 1-pattern in the database can be found by the CMP algorithm. To extend each frequent *k*-pattern *p*, we combine it with gaps and with all possible candidate frequent 1-patterns found in the *p*'s projected database to generate all its frequent super-patterns of *p*. Thus, every frequent *k*-pattern in the database can be found by the CMP algorithm. Once a frequent *k*-pattern is found, we adopt the forward and backward checking strategies to eliminate non-closed patterns. Therefore, every closed pattern can be found by the CMP algorithm.

Theorem 3.1 *The CMP algorithm enumerates all closed patterns in the database*.

Proof. By Lemma 3.4, every pattern found by the CMP algorithm is frequent and closed. By Lemma 3.5, every closed pattern can be found by the CMP algorithm. Therefore, we can conclude that the CMP algorithm enumerates all closed patterns in the database.

Theorem 3.2 *The time and space complexities of the CMP algorithm are bounded by* $O(|N|*|D|)$ *and* $O(lp*|D| + |CP|)$*, respectively, where the size of a time-series database, the number of nodes in a frequent pattern tree, the length of the longest frequent pattern, and the number of closed patterns are* $|D|$, $|N|$, *lp*, *and* $|CP|$*, respectively.*

Proof. To transform a time-series database into a symbolic database, the SAX operation is applied to each numerical value, and hence the time complexity of transforming all time-series in the database into symbolic sequences is bounded by $O(|D|)$. By scanning the transformed database, we obtain all frequent 1-patterns and this requires $O(|D|)$ time. Next, let us consider a pattern p of various lengths. If p is a frequent 1-pattern, the number of extensions is bounded by $|C|$, where $|C|$ is the average number of local frequent 1-patterns in a projected database. Likewise, if p is a frequent 2-pattern, the number of extensions is bounded by $|C|$. Thus, we know that if p is a frequent k-pattern, the number of extensions is again bounded by $|C|$. Moreover, for each frequent k-pattern, its projected database is scanned once to find candidate frequent 1-patterns that occur within the first τ+1 positions in the projected database. Thus, the time complexity of scanning a projected database is bounded by $O(|D|)$. Since the number of total nodes in the frequent pattern tree is $|N|$, the time complexity of the CMP algorithm is bounded by $O(|D| + |D| + |N|*(|C| + |D|)) = O(|N|*|D|)$. As we have shown that the CMP algorithm mines patterns in a DFS manner, the maximum number of nodes that kept in the memory is bounded by lp, where each node also maintains a projected database that requires $O(|D|)$ space in the worse case. Moreover, the closed pattern pool requires $O(|CP|)$ space. Therefore, the space complexity of the CMP algorithm is bounded by $O(|D| + lp*|D| + |CP|) = O(lp*|D| + |CP|)$.

3.5 An example

Let us take the database shown in Fig. 3.1 as an example. Assume that δ = 3 and τ = 1. First, we scan the database to find all frequent 1-patterns $\begin{Bmatrix} a \\ y \end{Bmatrix}$, $\begin{Bmatrix} a \\ \_ \end{Bmatrix}$, $\begin{Bmatrix} b \\ y \end{Bmatrix}$, $\begin{Bmatrix} b \\ \_ \end{Bmatrix}$, $\begin{Bmatrix} c \\ \_ \end{Bmatrix}$, $\begin{Bmatrix} \_ \\ x \end{Bmatrix}$, and $\begin{Bmatrix} \_ \\ y \end{Bmatrix}$. After that, we check $\begin{Bmatrix} a \\ y \end{Bmatrix}$ against the backward checking strategy and find that it cannot be pruned. We further use the forward checking strategy to check if it is closed. Consequently, it is a closed pattern and is inserted to the closed pattern pool. Since there is no frequent 1-pattern in $\begin{Bmatrix} a \\ y \end{Bmatrix}$'s projected database, we can stop growing the pattern.

Next, we grow the frequent 1-pattern $\begin{Bmatrix} a \\ \_ \end{Bmatrix}$ to find its frequent super-patterns by using the frequent 1-patterns in its projected database. There are four frequent 1-patterns within the first two positions in $\begin{Bmatrix} a \\ \_ \end{Bmatrix}$'s projected database, namely, $\begin{Bmatrix} b \\ y \end{Bmatrix}$, $\begin{Bmatrix} b \\ \_ \end{Bmatrix}$, $\begin{Bmatrix} \_ \\ y \end{Bmatrix}$, and $\begin{Bmatrix} \_ \\ x \end{Bmatrix}$. We first grow the frequent 1-pattern $\begin{Bmatrix} a \\ \_ \end{Bmatrix}$ by appending $\begin{Bmatrix} b \\ y \end{Bmatrix}$ to it and obtain the frequent 2-pattern $\begin{Bmatrix} a & b \\ \_ & y \end{Bmatrix}$. Since $\begin{Bmatrix} a & b \\ \_ & y \end{Bmatrix}$ does not pass the forward checking strategy, it is not closed. We then continue to grow $\begin{Bmatrix} a & b \\ \_ & y \end{Bmatrix}$ by appending a frequent 1-pattern in its projected database and obtain a frequent 3-pattern $\begin{Bmatrix} a & b & \_ \\ \_ & y & x \end{Bmatrix}$, which passes the forward and backward checking strategies. Thus, it is a closed pattern. Since we could not find any frequent 1-patterns in $\begin{Bmatrix} a & b & \_ \\ \_ & y & x \end{Bmatrix}$'s projected database, we can stop growing the pattern.

Furthermore, we grow the frequent 1-pattern $\begin{Bmatrix} a \\ \_ \end{Bmatrix}$ by appending $\begin{Bmatrix} b \\ \_ \end{Bmatrix}$ to it and find that $\begin{Bmatrix} a & b \\ \_ & \_ \end{Bmatrix}$ has the same projected database as $\begin{Bmatrix} a & b \\ \_ & y \end{Bmatrix}$. Thus, pattern $\begin{Bmatrix} a & b \\ \_ & \_ \end{Bmatrix}$ may be pruned by using Lemma 3.3. Similarly, pattern $\begin{Bmatrix} a & \_ \\ \_ & y \end{Bmatrix}$ may be pruned as well. After that, we continue to grow the frequent 1-pattern $\begin{Bmatrix} a \\ \_ \end{Bmatrix}$ by appending one gap and $\begin{Bmatrix} \_ \\ x \end{Bmatrix}$. Consequently, we obtain a frequent 3-pattern $\begin{Bmatrix} a & \_ & \_ \\ \_ & \_ & x \end{Bmatrix}$. However, it is not closed since it is a sub-pattern of pattern $\begin{Bmatrix} a & b & \_ \\ \_ & y & x \end{Bmatrix}$ and both patterns have the same support. By growing other frequent patterns in a similar manner as described above, we can get all closed patterns. Finally, we obtain five closed patterns $\begin{Bmatrix} a \\ y \end{Bmatrix}$, $\begin{Bmatrix} \_ & b \\ x & y \end{Bmatrix}$, $\begin{Bmatrix} \_ & c \\ y & \_ \end{Bmatrix}$, $\begin{Bmatrix} a & b & \_ \\ \_ & y & x \end{Bmatrix}$, and $\begin{Bmatrix} a & \_ & b \\ \_ & \_ & \_ \end{Bmatrix}$ as enumerated in the frequent pattern tree shown in Fig. 3.2.

Chapter 4 Mining Closed Flexible Patterns in Time-Series Databases

In this chapter, we propose an efficient algorithm, called CFP, for mining closed flexible patterns in a time-series database, where flexible gaps are allowed in a pattern. The proposed algorithm involves three phases: transforming a time-series database into a symbolic database, generating all frequent patterns of length one from the transformed database, and mining closed flexible patterns in a depth-first search manner.

4.1 Preliminaries and problem definitions

Given a time-series database, we first transform each time-series sequence in the database into the SAX representation [37], where each transaction in the database contains only one time-series sequence. That is, each time-series sequence is transformed into a symbolic sequence. Then, we mine the closed flexible patterns from those transformed symbolic sequences. Note that the database may contain transactions with different lengths. And, each transaction may contain gap symbols "_", which can be used to replace any missing values.

Definition 4.1 Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items (or events), and $S = <e_1e_2\ldots e_m>$ be a sequence, where i_j is an item, and $e_k \in I$, $1 \leq j \leq n$, $1 \leq k \leq m$.

Definition 4.2 A flexible pattern, denoted as $e_1[x_1, y_1]e_2[x_2, y_2]\ldots[x_{m-1}, y_{m-1}]e_m$, contains a sequence of items and gap intervals, where $e_i \in I$, x_j and y_j are non-negative integers, $1 \leq i \leq m$, $1 \leq j \leq m-1$, $0 \leq x_j \leq y_j \leq \tau$, and τ is a user-specified maximum gap threshold.

Definition 4.3 A gap interval $[x_i, y_i]$ between e_i and e_{i+1} denotes that the number of gaps between both items ranges from x_i to y_i.

Note that (1) if $x_i = y_i = 0$, the gap interval $[x_i, y_i]$ can be omitted; (2) if $m = 1$, there is no gap interval in the pattern, that is, the pattern is denoted as e_1. For example, $a[0, 3]b$ illustrates that the number of gaps between a and b ranges from 0 to 3. That is,

$a[0, 3]b$ contains the following patterns: *ab*, *a_b*, *a_ _b*, and *a_ _ _b*. *ab* means that b appears right after (or one position after) a while *a_b* means that there is a gap between a and b, or b appears two positions after a.

Definition 4.4 The length of a pattern p is defined as the number of items in p. A pattern of length k is called a *k-pattern*.

For example, $a[0, 3]b$ is a 2-pattern.

Definition 4.5 A pattern $e_1[x_1, y_1]e_2[x_2, y_2]\ldots[x_{m-1}, y_{m-1}]e_m$ is *contained* by a sequence if every item in the pattern appears in order in the sequence and the number of gaps between e_i and e_{i+1} in the sequence is between x_i and y_i.

For example, $a[0, 1]b$ is contained by <*a c b d c*>.

Definition 4.6 A gap interval $[x_1, y_1]$ contains another gap interval $[x_2, y_2]$ if $x_1 \leq x_2 \leq y_2 \leq y_1$.

Definition 4.7 A pattern $p = p_1[x_{p1}, y_{p1}]p_2[x_{p2}, y_{p2}]\ldots[x_{p(m-1)}, y_{p(m-1)}]p_m$ is a *sub-pattern* of a pattern $q = q_1[x_{q1}, y_{q1}]q_2[x_{q2}, y_{q2}]\ldots[x_{q(n-1)}, y_{q(n-1)}]q_n$ if there exists an integer j such that $p_i = q_{i+j}$, and $[x_{pi}, y_{pi}]$ contains $[x_{q(i+j)}, y_{q(i+j)}]$, where $m \leq n$, $0 \leq j \leq n-m$, $i = 1, 2, \ldots, m$. We can also say that q is a *super-pattern* of p.

Definition 4.8 The *support* of a pattern is defined as the number of sequences containing the pattern in the database.

Definition 4.9 A pattern is said to be *frequent* if its support is not less than a user-specified minimum support threshold, δ.

Definition 4.10 A pattern is *closed* if it is frequent and there exists no super-pattern with the same support.

For example, if $a[0, 3]b$ and $a[0, 3]b[0, 2]c$ are frequent and both have the same support, $a[0, 3]b$ is not closed. Let us consider another example where $a[0, 3]b$ and $a[0, 1]b$ are frequent and both have the same support. Since $a[0, 3]b$ and $a[0, 1]b$ have the same support, $a[0, 3]b$ is less representative. Hence, $a[0, 3]b$ is not closed.

Definition 4.11 Given a sequence S and a pattern p, a subsequence s of S is called a *projection* of *S* with respect to *p* if *p* appears in *S* and *s* contains all items in *S* that occur after the last item of *p*.

Definition 4.12 The *projected database* of a pattern *p* contains the projection of each sequence in a database with respect to *p*.

For example, the projected database of *a*[0, 1]*c* in a database {<*a b c d e*>, <*a c d e b a*>} is {<*d e*>, <*d e b a*>}.

Definition 4.13 Given a pattern *p*, *nominee*(*p*) contains all frequent 1-patterns that occur in *p*'s projected database.

To find nominee(*p*), we only need to scan the first τ+1 items for each sequence in *p*'s projected database since the number of gaps between two successive items in a pattern should not be greater than τ. Consider the last example. If $\delta = 2$ and $\tau = 2$, nominee(*a*[0, 1]*c*) = {*d*, *e*}.

4.2 Sequence tree

In order to enumerate all frequent patterns, we adopt a tree structure, called *sequence tree*, to store all frequent patterns, where each node represents a frequent pattern and the root node represents an empty pattern. All frequent *k*-patterns are recorded at level *k* of the tree, where $k \geq 1$. That is, all frequent 1-patterns are recorded at level 1 of the tree and all frequent 2-patterns are recorded at level 2, and so on. Moreover, if a pattern at level *k* is derived from the pattern of its parent node at level *k*–1, we use an edge to connect both nodes. Specifically, we extend frequent patterns from the root node level down to the leaf node level.

The CFP algorithm also enumerates the frequent patterns by a sequence tree in a DFS manner. For instance, an example database is illustrated in Fig. 4.1 and the corresponding sequence tree is shown in Fig. 4.2, where the number after a pattern denotes the support of the pattern, the pattern in a dotted circle is non-closed, the other patterns are closed, $\delta = 3$, and $\tau = 2$.

Sequence ID	Sequence
S_1	<*a b c d b a*>
S_2	<*e b e a c a*>
S_3	<*e b e c b d*>
S_4	<*a b f c c d*>

Fig. 4.1. An example database containing four sequences.

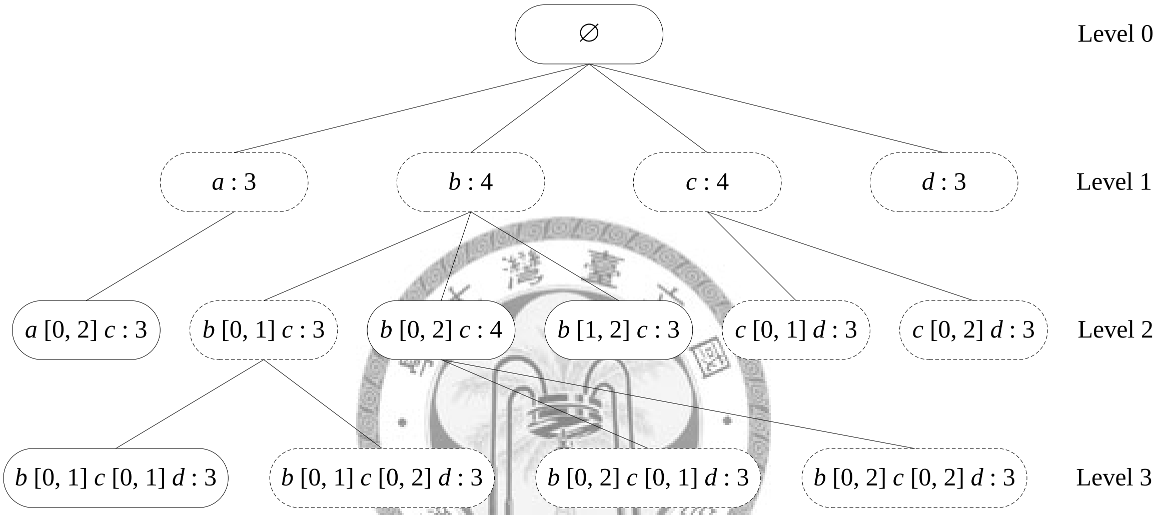


Fig. 4.2. The frequent patterns mined from the database in Fig. 4.1.

4.3 Support counting

In order to calculate the support of a pattern $p[x, y]r$, we investigate p's projected database and create an index table to facilitate the computation, where $r \in$ nominee(p). The column i of the index table records a list of sequences in p's projected database so that the number of gaps between p and r is equal to i in these sequences, $i \geq 0$. Since the maximum gap threshold is τ, we have τ+1 columns in the table. For example, assume that $\tau = 3$, $r = b$, and there are four sequences {S_1:<*d b f c c*>, S_2:<*b b b b f*>, S_3:<*e c b d c*>, S_4:<*d b f b e*>} in p's projected database. Since there is one b in S_1 and the number of gaps between p and b is 1, we add S_1 to the column 1 of the index table. There are four b's in S_2. Thus, we add S_2 to the columns 0, 1, 2, and 3. Similarly, we can add S_3

and S_4 to the index table. Finally, we obtain the index table as illustrated in Fig. 4.3.

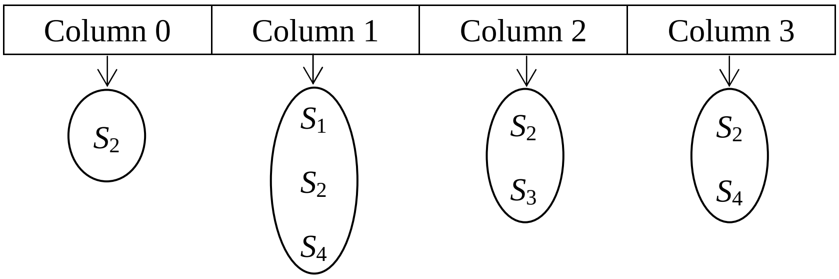


Fig. 4.3. An index table used to count the support of $p[x, y]b$.

From Fig. 4.3, we know that the supports of $p[0, 0]b$, $p[1, 1]b$, $p[2, 2]b$, and $p[3, 3]b$ are 1, 3, 2, and 2, respectively. To determine the supports of other patterns, such as $p[0, 1]b$, $p[0, 2]b$, etc, we simply merge the lists of the corresponding columns based on the gap interval. For example, to count the support of $p[1, 2]b$, we merge the lists of columns 1 and 2 and obtain a list containing S_1, S_2, S_3, and S_4. Since there are four sequences in the list, the support of $p[1, 2]b$ is 4.

4.4 Closure checking and pruning strategies

To efficiently mine closed flexible patterns, we adopt a closure checking scheme and two pruning strategies: redundant extension pruning and redundant projection pruning. Lemma 4.1 is a closure checking scheme that helps to exclude non-closed patterns during the mining process. It uses the concept of projected databases as described in BIDE [61] to localize the checking. Upon generating a new frequent pattern, we check whether it is closed by applying this lemma.

Lemma 4.1 (Closure checking). *Given a frequent pattern p, it is non-closed if it meets one of the following conditions*: (1) *if there exists a nominee that appears in the first* τ+1 *positions of each sequence in p's projected database*, *or* (2) *if there exists another frequent pattern q so that p is a sub-pattern of q and both have the same support*.

Proof. In the first condition, since a frequent 1-pattern *r* appears in the first τ+1 positions of each sequence in *p*'s projected database, there must exist another pattern formed by *p* and *r* with a certain gap interval, which has the same support as *p*. Therefore, *p* is not closed. In the second condition, since pattern *p* is the sub-pattern of *q* and both have the same support, it is not closed.

Note that *p* may appear more than once in a sequence. If we cannot find any *r* after the current appearance of *p*, we need to perform the same check on the next appearance of *p* in the sequence.

Assume that δ= 3 and τ= 1. When a new frequent pattern *p* = *a*[0, 1]*c* with support equal to 3 is generated, by checking the first two positions of each sequence in *p*'s projected database which contains three sequences {<*d e f*>, <*a d b*>, <*d c b*>}, we find that nominee(*p*) = {*d*}. Thus, we assure that *p* is not closed since we can find a frequent pattern *a*[0, 1]*c*[0, 1]*d* that is a super-pattern of *p* and has the same support as *p*. Let us consider another example where the second condition is held. Assume there exists a frequent pattern *a*[0, 1]*c* with support equal to 3. When a new frequent pattern *b*[0, 1]*a*[0, 1]*c* with the same support is found, we know that *a*[0, 1]*c* is not closed since it is a sub-pattern of *b*[0, 1]*a*[0, 1]*c*.

The main idea for the redundant extension pruning strategy is to avoid the cost of extending non-closed patterns in advance. To be precise, for each frequent pattern *p*, we determine whether it should be further extended by checking if there exists an item that is within τ+1 positions before *p* in each sequence containing *p*. If such an item can be found, this pattern is not closed and any patterns extended from *p* would also be non-closed.

Lemma 4.2 (Redundant extension pruning). *A frequent pattern p does not need to be extended if there exists an item that appears within* τ+1 *positions before p in each sequence containing p.*

Proof. Since an item *q* appears within τ+1 positions before *p* in each sequence containing *p*, there must exist another pattern *q*' formed by *q* and *p* with a certain gap interval so that *q'* has the same support as *p*. The projected database of *q'* is the same as that of *p*. It means that each frequent pattern extended from *p* is a sub-pattern of a frequent pattern extended from *q'* and both have the same support. Therefore, *p* does not need to be extended.

Note that *p* may appear more than once in a sequence. Thus, we need to check if *q* exists before every appearance of *p* in the sequence.

For example, assume that a database contains three sequences: <*c b d e f*>, <*c a b e a*>, <*c b a g e*> and there exists a frequent pattern *b*[0, 2]*e* with support equal to 3. By checking each sequence in the database, we find that *c* appears 1-2 positions before *b*[0, 2]*e* in each sequence. It means that we have another pattern *c*[0, 1]*b*[0, 2]*e* which has the same support as *b*[0, 2]*e*. Therefore, *b*[0, 2]*e* does not need to be extended and can be pruned.

Lemma 4.3 (Redundant projection pruning). *If* $p[x_1, y_1]r$ *and* $p[x_2, y_2]r$ *are frequent and share the same projected database,* $p[x_1, y_1]r$ *can be pruned, where p is a frequent pattern,* $r \in nominee(p)$, $0 \leq x_1 \leq x_2 \leq y_2 \leq y_1$.

Proof. Since $p[x_1, y_1]r$ is a sub-pattern of $p[x_2, y_2]r$ and both share the same projected database, any pattern extended from $p[x_1, y_1]r$ can also be extended from $p[x_2, y_2]r$ and both patterns extended have the same support. It means that any pattern extended from $p[x_1, y_1]r$ is non-closed. Thus, $p[x_1, y_1]r$ can be pruned.

For example, assume that $\delta = 3$, $\tau = 2$, *c*[0, 1]*a* is a frequent flexible pattern and its projected database contains three sequences: {<*c e d f g*>, <*a a e g a*>, <*b e a b c*>}. We find that $e \in$ nominee(*c*[0, 1]*a*). Thus, patterns *c*[0, 1]*a*[0, 2]*e* and *c*[0, 1]*a*[1, 2]*e* can be generated and both share the same projected database {<*d f g*>, <*g a*>, <*a b c*>}. Since *c*[0, 1]*a*[0, 2]*e* and *c*[0, 1]*a*[1, 2]*e* share the same projected database and the former is a sub-pattern of the latter, any pattern extended from *c*[0, 1]*a*[0, 2]*e* is not closed. Therefore, *c*[0, 1]*a*[0, 2]*e* can be pruned.

4.5 The CFP algorithm

The proposed CFP algorithm consists of three phases as shown in Fig. 4.4. First, we transform each time-series sequence in the database into the SAX representation as described in Section 2.4. Second, we scan the transformed database once to find all frequent 1-patterns and build a projected database for each 1-pattern found. Third, we recursively call the CFP-Extension procedure to extend each 1-pattern found in the second phase to mine closed flexible patterns in a depth-first search manner.

```
Algorithm: CFP
Input: a time-series database TDB, a minimum support threshold $\delta$, a maximum gap
       threshold $\tau$
Output: a complete set of closed flexible patterns, CP
Transform each sequence in TDB into a symbolic sequence by the SAX
        representation;
Scan the transformed database once to find all frequent 1-patterns. Let P1 be a set
        of frequent 1-patterns found;
Let CP = $\varnothing$;
for each 1-pattern p in P1 do
        Construct the projected database of p, PDB;
        call CFP-Extension (PDB, p, $\delta$, $\tau$, CP);
end for
return CP;
```

Fig. 4.4. The CFP algorithm.

The CFP-Extension procedure is shown in Fig. 4.5. In steps 1-4, if *p* passes the redundant extension pruning strategy and closure checking scheme, it is added to the closed flexible patterns pool (*CP*). Then, we find the nominees of *p* in step 5. For each nominee found, we combine it with *p* to generate the closed flexible super-patterns of *p* in steps 6-21. In steps 7-15, we combine *p* and each nominee of *p* with every possible gap interval, and use the index table to count the support of the combined pattern, and check if the combined pattern is frequent. If this is the case, the combined pattern is added to *NP*. Next, we use the redundant projection pruning strategy to eliminate unnecessary patterns in step 16. In steps 17-20, for each combined pattern in *NP*, we construct the projected database of the pattern and recursively call the CFP-Extension procedure to enumerate the closed flexible patterns.

Let us elaborate the concept of the pattern extension. Once all nominees are found, we combine *p* and each of these nominees with all possible combinations of gap intervals. Recall that τ is the maximum gap threshold. The number of possible combinations of gap intervals is thus $1+2+\ldots+\tau+(\tau+1) = (\tau+1)(\tau+2)/2$. We then combine *p* and one of the nominee r_i by taking into account all possible combinations of gap intervals and represent it in the form of $p[x_i, y_i]r_i$, where x_i and y_i are non-negative integers, $0 \leq x_i \leq y_i \leq \tau$. These newly combined patterns are candidate 2-patterns. Next, we need to check if they are frequent before inserting them into the second level of the tree. Assume $p = a$, $\delta = 2$, $\tau = 1$, and the projected database of *a* contains two sequences, {S_1:<*b e c f e*>, S_2:<*d e a b c*>}. We find nominee(*p*) = {*b*, *e*}. Since $\tau = 1$, we have three possible combinations of gap intervals, namely, [0, 0], [0, 1], [1, 1]. Hence, we combine *p* with the first nominee *b* with these gap intervals and obtain 3 candidate 2-patterns: *a*[0, 0]*b*, *a*[0, 1]*b*, *a*[1, 1]*b*.

To count the supports for these candidates, we use the index table as described in Section 4.3. We first create an index table with two columns ($\tau+1 = 2$). Column 0 stores any sequence ID where *b* appears right after *a* in the sequence, while column 1 stores any sequence ID where *b* appears two positions after *a*. In this case, we have {S_1, S_2} in column 0 and $\varnothing$ in column 1. Therefore, the support of *a*[0, 0]*b*, *a*[1, 1]*b* are 2 and 0, respectively. To count the support of *a*[0, 1]*b*, we simply merge the lists in columns 0 and 1, and obtain a list containing S_1 and S_2. Hence, the support of *a*[0, 1]*b* is 2. Since we find that *a*[0, 0]*b* and *a*[0, 1]*b* have the same support and share the same projected database, *a*[0, 1]*b* is pruned by the redundant projection pruning strategy (Lemma 4.3).

Similarly, we combine *p* with the second nominee *e* and obtain one frequent 2-pattern: *a*[1, 1]*e*.

After we obtain all frequent 2-patterns, we apply the same procedure of the third phase to these frequent 2-patterns and extend them to frequent 3-patterns. Let us continue with the last example. After finding the frequent 2-patterns *a*[0, 0]*b*, and *a*[1, 1]*e*, we use these frequent 2-patterns to generate frequent 3-patterns. We first find the frequent 1-patterns in the projected database of *a*[0, 0]*b*, where nominee(*a*[0, 0]*b*) = {*c*}. Thus, we can generate three candidate 3-patterns: *a*[0, 0]*b*[0, 0]*c*, *a*[0, 0]*b*[0, 1]*c*, *a*[0, 0]*b*[1, 1]*c*. By using an index table to count the supports of these candidate 3-patterns, we find that *a*[0, 0]*b*[0, 1]*c* is frequent with support equal to 2. Similarly, we can apply the same procedure to *a*[1, 1]*e* and extend it to frequent 3-patterns.

While recursively performing the procedure of the third phase, we integrate the proposed pruning strategies and closure checking scheme to speed up the mining process. Upon getting a frequent pattern, we first use the redundant extension pruning strategy (Lemma 4.2) to check if the pattern can be pruned. If not, we apply the closure checking scheme (Lemma 4.1) to check whether the pattern is closed. If the pattern is closed, we add it to the pool of closed flexible patterns and extend it. As we have seen, the effort of mining frequent patterns is determined by the parameter τ. As τ is set to a large value, the number of possible combinations of gap intervals to form candidate patterns is large. However, we may use the redundant projection pruning strategy (Lemma 4.3) to prune unnecessary combinations.

In summary, we use a frequent *k*-pattern *p* to generate candidate (*k*+1)-patterns, each of which is formed by concatenating *p*, a gap interval, and one nominee of nominee(*p*). For each candidate (*k*+1)-pattern generated, we check if it is frequent and closed. If it is, we recursively perform the CFP-Extension procedure to mine its frequent super-patterns in a DFS manner. Accordingly, we mine all closed flexible patterns.

```
Procedure: CFP-Extension ($PDB, p, \delta, \tau, CP$)
Input: a projected database $PDB$ of $p$, a frequent flexible pattern $p$, a minimum support
    threshold $\delta$, a maximum gap threshold $\tau$
Output: a complete set of closed flexible patterns, $CP$
if ($p$ passes the redundant extension pruning strategy) then
    if ($p$ passes the closure checking scheme) then
        Add $p$ to $CP$;
    end if
    Find nominee($p$) from $PDB$;
    for each $r$ in nominee($p$) do
        Let $\Sigma$ be all possible combinations of gap intervals generated from $\tau$,
            where $0 \leq x_i \leq y_i \leq \tau$;
        Let $NP = \emptyset$;
        for each $[x_i, y_i]$ in $\Sigma$ do
            Combine $p$, $[x_i, y_i]$, and $r$ to form $p[x_i, y_i]r$;
            Use an index table to count the support of $p[x_i, y_i]r$;
            if (the support of $p[x_i, y_i]r$ is not less than $\delta$) then
                Add $p[x_i, y_i]r$ to $NP$;
            end if
        end for
        Apply the redundant projection pruning strategy to eliminate
            unnecessary patterns in $NP$;
        for each pattern $q$ in $NP$ do
            Construct $q$'s projected database, $PDB'$;
            call CFP-Extension ($PDB'$, $q$, $\delta$, $\tau$, $CP$);
        end for
    end for
end if
return $CP$;
```

Fig. 4.5. The CFP-Extension procedure.

Lemma 4.4 *Every pattern found by the CFP algorithm is frequent and closed*.

Proof. In the second phase, we scan the database once to find all frequent 1-patterns. In the third phase, at each level of extension, we use a frequent k-pattern p at the previous level and every nominee found in p's projected database to form frequent $(k+1)$-patterns, $k \geq 1$. Whenever a new pattern is found, we check the support of the pattern against the minimum support threshold and eliminate it if it is not frequent. Hence, every pattern found by the CFP algorithm is frequent. Moreover, at each level of extension, we adopt the closure checking scheme to eliminate non-closed patterns. Therefore, every pattern found by the CFP algorithm is frequent and closed.

Lemma 4.5 *Every closed flexible pattern can be found by the CFP algorithm*.

Proof. In step 2 of the CFP algorithm, we scan the database once to find all frequent 1-patterns. Thus, every frequent 1-pattern in the database can be found by the CFP algorithm. In steps 6-21 of the CFP-Extension procedure, for each frequent k-pattern p, we combine it with every nominee found in p's projected database and use the index table to enumerate all frequent super $(k+1)$-patterns of p, $k \geq 1$. Thus, every frequent $(k+1)$-pattern in the database can be found by the CFP algorithm. Once a frequent k-pattern is found, we adopt the closure checking scheme to eliminate non-closed patterns in step 2 of the CFP-Extension procedure. Therefore, every closed flexible pattern can be found by the CFP algorithm

Theorem 4.1 *The CFP algorithm enumerates all closed flexible patterns in the database*.

Proof. By Lemma 4.4, every pattern found by the CFP algorithm is frequent and closed. By Lemma 4.5, every closed flexible pattern can be found by the CFP algorithm. Therefore, we can conclude that the CFP algorithm enumerates all closed flexible patterns in the database.

Theorem 4.2 *The time and space complexities of the CFP algorithm are bounded by* $O(|N|*(r*g + |D|))$ *and* $O(lp*|D| + |CP|)$, *respectively*, *where the size of a time-series* *database, the number of nodes in a sequence tree, the length of the longest frequent flexible pattern, the number of nominees for a frequent flexible pattern, the number of possible combinations of gap intervals, and the number of closed patterns are* $|D|$, $|N|$, lp, r, g, *and* $|CP|$, *respectively.*

Proof. Since the size of the database is $|D|$, the time complexity of performing the SAX operation is bounded by $O(|D|)$. By scanning the transformed database, we obtain all frequent 1-patterns and this requires $O(|D|)$ time. Next, consider a pattern p of various lengths. If p is a frequent 1-pattern, the number of extensions is bounded by $r*g$. Likewise, if p is a frequent 2-pattern, the number of extensions is bounded by $r*g$. Thus, we know that if p is a frequent k-pattern, the number of extensions is again bounded by $r*g$. Moreover, for each frequent k-pattern p, the time requires to scan p's projected database to find the nominees of p is bounded by $O(|D|)$. Since the number of total nodes in the sequence tree is $|N|$, the time complexity of the CFP algorithm is bounded by $O(|D| + |D| + |N|*(r*g + |D|)) = O(|N|*(r*g + |D|))$. As we design the CFP algorithm based on the depth-first search approach, the maximum number of nodes that kept in the memory is bounded by lp, where each node also requires $O(|D|)$ space in the worse case to maintain its projected database. Moreover, the closed pattern pool requires $O(|CP|)$ space. Therefore, the space complexity of the CFP algorithm is bounded by $O(|D| + lp*|D| + |CP|) = O(lp*|D| + |CP|)$.

4.6 An example

Let us demonstrate how the CFP algorithm works. Given a time-series database containing four sequences, each sequence is transformed into a symbolic sequence as shown in Fig. 4.1. Assume that $\delta = 3$ and $\tau = 2$. After scanning the transformed database and computing the support for each 1-pattern, we find four frequent 1-patterns and add them to the level 1 of the sequence tree as shown in Fig. 4.2.

Next, we apply the redundant extension pruning strategy on a and find that it should be extended. We then construct *a*'s projected database, which is {<*b c d b a*>, <*c a*>, <*b f c c d*>}. Moreover, we scan the projected database and find nominee(*a*) = {*c*}. Since *c* appears in the first 3 positions of each sequence in *a*'s projected database, *a* does not pass the closure checking scheme, and hence it is not closed. The node of *a* is changed into a dotted circle as shown in Fig. 4.2. Subsequently, *a* is extended by concatenating it with *c* and one of the following gap intervals: [0, 0], [0, 1], [0, 2], [1, 1], [1, 2], and [2, 2]. Since δ is 3, we only obtain a frequent 2-pattern *a*[0, 2]*c* with support equal to 3. The projected database of *a*[0, 2]*c* is {<*d b a*>, <*a*>, <*c d*>}. Since we cannot find any nominee in the projected database, no more frequent pattern can be generated.

Since pattern *b* passes the redundant extension pruning strategy, it should be extended. Next, we construct *b*'s projected database, which is {<*c d b a*>, <*e a c a*>, <*e c b d*>, <*f c c d*>}. We then scan the projected database and find nominee(*b*) = {*c*}. We also perform the closure checking scheme and find that *c* appears in the first 3 positions of each sequence in *b*'s projected database, and hence *b* is not closed. The node of *b* is changed into a dotted circle as shown in Fig. 4.2. Then, we discover three frequent 2-patterns *b*[0, 1]*c*, *b*[0, 2]*c*, and *b*[1, 2]*c* with supports 3, 4, and 3, respectively. We recursively perform the pattern extension procedure on these frequent 2-patterns to get frequent super-patterns until no more frequent patterns can be generated. As shown in Fig. 4.2, the subtree under node *b* contains three closed patterns *b*[0, 2]*c*, *b*[1, 2]*c*, and *b*[0, 1]*c*[0, 1]*d*. Similarly, we can extend patterns *c* and *d* to find their frequent super-patterns. Finally, we find all closed flexible patterns in the database, including *a*[0, 2]*c*, *b*[0, 2]*c*, *b*[1, 2]*c*, *b*[0, 1]*c*[0, 1]*d*, as shown in Fig. 4.2.

Chapter 5 Mining Multi-Resolution Closed Numerical Patterns in Multi-Sequence Time-Series Databases

In this chapter, we propose an algorithm, called CNP, to mine multi-resolution closed numerical patterns in a time-series database, where each transaction consists of multiple sequences. The CNP algorithm involves three phases. First, we normalize all the time-series in the database and transform them into a low resolution by the Haar wavelet transform. Second, we generate all frequent 1-patterns from the transformed database. Third, we recursively extend each 1-pattern found in the second phase to find longer patterns. Subsequently, for each pattern found in the low resolution, we restore the pattern back to the high resolution. Finally, we determine the representative patterns as described in Section 5.1 with respect to the low and high resolutions.

5.1 Preliminaries and problem definitions

Consider a database *D* containing *n* transactions, where each transaction contains *s* sequences, and each sequence is a sequence of numerical values. Note that the database may contain transactions with different lengths. However, multiple sequences contained in each transaction must be of equal length.

Definition 5.1 A *pattern p* is defined as $\left\{\begin{matrix} v_{11} & v_{12} & \cdots & v_{1k} \\ v_{21} & v_{22} & \cdots & v_{2k} \\ \vdots & \vdots & & \vdots \\ v_{s1} & v_{s2} & \cdots & v_{sk} \end{matrix}\right\}$, where v_{ij} is a numerical value.

Definition 5.2 The length of a pattern *p* is defined as the number of numerical values in each sequence of *p*. A pattern of length *k* is called a *k-pattern*.

Note that pattern p can be regarded as a sequence of s-dimensional points, $\begin{Bmatrix} v_{11} \\ v_{21} \\ \vdots \\ v_{s1} \end{Bmatrix}$, $\begin{Bmatrix} v_{12} \\ v_{22} \\ \vdots \\ v_{s2} \end{Bmatrix}$, …, $\begin{Bmatrix} v_{1k} \\ v_{2k} \\ \vdots \\ v_{sk} \end{Bmatrix}$, where a s-dimensional point is called a *s-point*. For example, $\begin{Bmatrix} 1.0 & -0.6 & -0.4 \\ 0.6 & 0.6 & 1.4 \end{Bmatrix}$ is a 3-pattern.

Definition 5.3 Let $z_1 = \begin{Bmatrix} v_{11} \\ v_{21} \\ \vdots \\ v_{s1} \end{Bmatrix}$ and $z_2 = \begin{Bmatrix} v_{12} \\ v_{22} \\ \vdots \\ v_{s2} \end{Bmatrix}$ be two s-points. $z_1 > z_2$ if (1) $v_{11} > v_{12}$, or (2) there exists a positive integer j, such that $v_{i1} = v_{i2}$, $i = 1, 2, \ldots, j–1$, and $v_{j1} > v_{j2}$, $j \geq 1$. Moreover, $z_1 = z_2$ if $v_{i1} = v_{i2}$, $i = 1, 2, \ldots, s$.

Definition 5.4 Let p and q be two patterns. $p > q$ if (1) the first s-point of p is greater than that of q, or (2) there exists a positive integer j, such that the first j–1 s-points of p are equal to those of q, and the jth s-point of p is greater than that of q, $j \geq 2$.

Definition 5.5 A pattern $p = \begin{Bmatrix} u_{11} & u_{12} & \cdots & u_{1m} \\ u_{21} & u_{22} & \cdots & u_{2m} \\ \vdots & \vdots & & \vdots \\ u_{s1} & u_{s2} & \cdots & u_{sm} \end{Bmatrix}$ is a *sub-pattern* of $p' = \begin{Bmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ \vdots & \vdots & & \vdots \\ v_{s1} & v_{s2} & \cdots & v_{sn} \end{Bmatrix}$, if there exists an integer j such that $|u_{ik} - v_{i(k+j)}| \leq \varepsilon$, $1 \leq i \leq s$, $1 \leq k \leq m$, $0 \leq j \leq n–m$, and $m \leq n$, where ε is a user-specified minimum distance threshold. We can also say that p' is a *super-pattern* of p, or p' *contains* p. Moreover, if p is a sub-pattern of *p*' and both patterns have the same length, we can say that *p* is *similar* to *p*'.

For example, $p = \begin{Bmatrix} 0.1 & 1.1 \\ 0.6 & -0.3 \end{Bmatrix}$ is contained by $p' = \begin{Bmatrix} 0.2 & 1.2 & 0.8 \\ 0.6 & -0.3 & 0.8 \end{Bmatrix}$, where $\varepsilon = 0.1$.

Definition 5.6 The *support* of a pattern *p* is defined as the number of transactions containing *p* in the database.

Definition 5.7 A pattern is *frequent* if its support is not less than a user-specified minimum support threshold, δ.

Definition 5.8 A pattern *p* is *closed* if it is frequent and there does not exist any super-pattern of *p* with the same support.

Definition 5.9 $T[i, j]$ is a sub-transaction of $T = \begin{Bmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ \vdots & \vdots & & \vdots \\ v_{s1} & v_{s2} & \cdots & v_{sn} \end{Bmatrix}$, where $T[i, j] = \begin{Bmatrix} v_{1i} & v_{1i+1} & \cdots & v_{1j} \\ v_{2i} & v_{2i+1} & \cdots & v_{2j} \\ \vdots & \vdots & & \vdots \\ v_{si} & v_{si+1} & \cdots & v_{sj} \end{Bmatrix}$, $1 \leq i \leq j \leq n$.

Definition 5.10 If $T[i, j]$ is similar to *p*, $T[i, j]$ is called a *projection* of a pattern *p*, where *i* and *j* are called the starting and ending locations of the projection.

Definition 5.11 The *projected database* of a *k*-pattern *p* contains all projections in the database with respect to *p* and is denoted as $PDB(p) = \{<tid, j> \mid tid$ is the identifier of the transaction *T*, and $T[j-k+1, j]$ is similar to $p\}$.

For example, consider a database contains two transactions, $T_1 = \begin{Bmatrix} 0.6 & 0.3 \\ 0.5 & 0.5 \end{Bmatrix}$ and $T_2 = \begin{Bmatrix} 0.8 & 0.4 \\ 0.7 & 1.0 \end{Bmatrix}$, and $\varepsilon = 0.2$, the projected database of $p = \begin{Bmatrix} 0.6 \\ 0.5 \end{Bmatrix}$ is $\{<T_1, 1>, <T_2, 1>\}$.

Since there may exist two or more patterns that are similar to each other and have the same projected database, we may merge them together and choose one of them as the representative pattern. Here, we choose the smallest one as the representative pattern. Moreover, for a representative pattern *p*, we use a list, called *merge list*, to keep the patterns merged by *p*. For example, if $p = \left\{\begin{matrix} 0.3 \\ 0.1 \end{matrix}\right\}$ and $q = \left\{\begin{matrix} 0.4 \\ 0.2 \end{matrix}\right\}$ have the same projected database, *q* is merged by *p* and is kept in *p*'s merge list.

5.2 Haar wavelet transform

Given a transaction $T = \left\{\begin{matrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ \vdots & \vdots & & \vdots \\ v_{s1} & v_{s2} & \cdots & v_{sn} \end{matrix}\right\}$, we can transform it into a low resolution, denoted as WT(*T*), by the Haar wavelet transform, where WT(*T*) = $\left\{\begin{matrix} u_{11} & u_{12} & \cdots & u_{1(n/2)} \\ u_{21} & u_{22} & \cdots & u_{2(n/2)} \\ \vdots & \vdots & & \vdots \\ u_{s1} & u_{s2} & \cdots & u_{s(n/2)} \end{matrix}\right\}$, $u_{ij} = (v_{i(2j-1)}+v_{i(2j)})/2$, $i = 1, 2, \ldots, s$, and $j = 1, 2, \ldots, n/2$. For example, both sequences of the first transaction in Fig. 5.1 can be transformed into two sequences as shown in Fig. 5.2.

Transaction		Time stamps							
		t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
T_1	Sequence 1	-0.6	1.0	-0.4	2.8	-2.2	3.8	-2.0	4.8
	Sequence 2	4.4	-3.2	1.4	-2.0	3.6	-2.0	3.8	-4.6
T_2	Sequence 1	4.0	-0.6	1.2	-0.2	2.8	-2.0	3.8	-2.0
	Sequence 2	-4.2	4.6	-3.2	1.2	-2.2	3.4	-1.8	3.8

Fig. 5.1. A database containing two multi-sequence transactions.

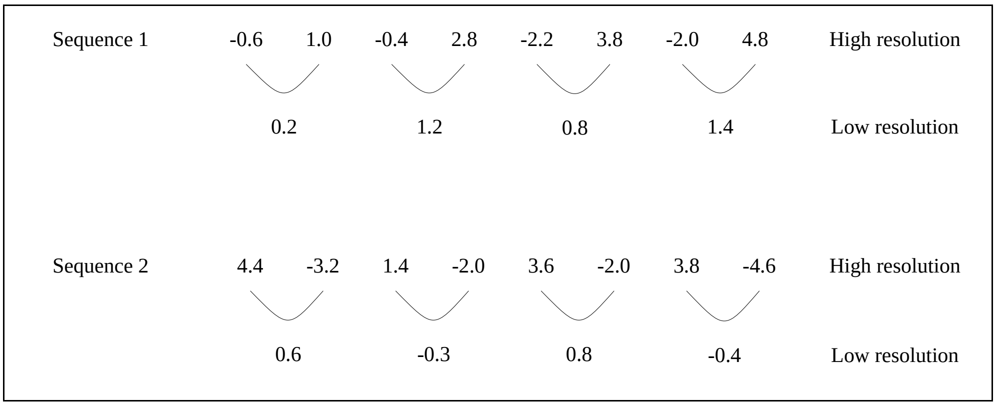


Fig. 5.2. The Haar wavelet transform.

If patterns *x* and *y* are similar to each other in the original database, WT(*x*) and WT(*y*) are similar to each other, where WT(*x*) and WT(*y*) are transformed from *x* and *y* by the Haar wavelet transform, respectively.

Lemma 5.1 (Similar relationship). Given two *k*-patterns, $x = \begin{Bmatrix} x_{11} & x_{12} & \cdots & x_{1k} \\ x_{21} & x_{22} & \cdots & x_{2k} \\ \vdots & \vdots & & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mk} \end{Bmatrix}$ and $y = \begin{Bmatrix} y_{11} & y_{12} & \cdots & y_{1k} \\ y_{21} & y_{22} & \cdots & y_{2k} \\ \vdots & \vdots & & \vdots \\ y_{m1} & y_{m2} & \cdots & y_{mk} \end{Bmatrix}$, and two transformed patterns, WT(*x*) = $\begin{Bmatrix} u_{11} & u_{12} & \cdots & u_{1(k/2)} \\ u_{21} & u_{22} & \cdots & u_{2(k/2)} \\ \vdots & \vdots & & \vdots \\ u_{m1} & u_{m2} & \cdots & u_{m(k/2)} \end{Bmatrix}$ and WT(*y*) = $\begin{Bmatrix} v_{11} & v_{12} & \cdots & v_{1(k/2)} \\ v_{21} & v_{22} & \cdots & v_{2(k/2)} \\ \vdots & \vdots & & \vdots \\ v_{m1} & v_{m2} & \cdots & v_{m(k/2)} \end{Bmatrix}$, if $|x_{ij} - y_{ij}| \leq \varepsilon$, then $| u_{i(j/2)} - v_{i(j/2)}| \leq \varepsilon$, where $1 \leq i \leq m$ and $1 \leq j \leq k$.

Proof. $$\begin{aligned} |u_{il} - v_{il}| &= \left| \frac{x_{i(2l-1)} + x_{i(2l)}}{2} - \frac{y_{i(2l-1)} + y_{i(2l)}}{2} \right| \\ &= \left| \frac{(x_{i(2l-1)} - y_{i(2l-1)})}{2} + \frac{(x_{i(2l)} - y_{i(2l)})}{2} \right| \\ &= \frac{1}{2} \left| (x_{i(2l-1)} - y_{i(2l-1)}) + (x_{i(2l)} - y_{i(2l)}) \right| \\ &\leq \frac{1}{2} \left| x_{i(2l-1)} - y_{i(2l-1)} \right| + \left| x_{i(2l)} - y_{i(2l)} \right| \\ &\leq \frac{1}{2} (\varepsilon + \varepsilon) \\ &\leq \varepsilon \end{aligned}$$

Lemma 5.1 also indicates that if WT(x) is not similar to WT(y), it is guaranteed that x is not similar to y. However, it is not guaranteed if WT(x) is similar to WT(y), then x is similar to y. For example, assume that $\varepsilon = 0.1$. Consider two patterns, $x = \begin{Bmatrix} 0.7 & 0.5 \\ 0.9 & 0.7 \end{Bmatrix}$ and $y = \begin{Bmatrix} 0.5 & 0.5 \\ 1.0 & 0.8 \end{Bmatrix}$. By applying the Haar wavelet transform, we obtain two transformed patterns, WT(x) = $\begin{Bmatrix} 0.6 \\ 0.8 \end{Bmatrix}$ and WT(y) = $\begin{Bmatrix} 0.5 \\ 0.9 \end{Bmatrix}$. It can be observed that WT(x) and WT(y) are similar to each other, whereas x and y are dissimilar.

To mine multi-resolution closed patterns, we first mine the closed patterns from the transformed database (low resolution) and then use the closed patterns mined to restore the closed patterns in the original database (high resolution). However, the patterns restored locate at the odd positions of the original time-series sequences. To resolve such a problem, we also apply the Haar wavelet transform to the original database by starting the operation from the second position of each time-series sequence in the database as shown in light grey color in Fig. 5.3. Then, those transformed time-series sequences will be collected together to form the transformed database.

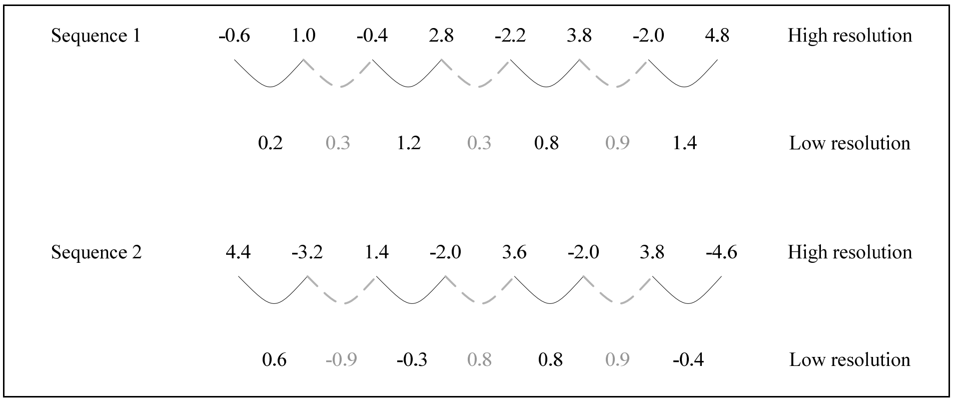


Fig. 5.3. The transformed time-series sequences.

Let S represents a time-series sequence in the transformed database, where S is transformed from a time-series sequence T in the original database. The s-points located at the odd and even positions in S are generated by the Haar wavelet transform starting from the first and second positions of T, respectively. For each s-point s_i located at the odd position in S, it is obtained by computing the average between every two adjacent values, t_i and t_{i+1} in T, where i is odd. Thus, the adjacent s-point of s_i is the value located at the next odd position in S. Likewise, if i is even, the adjacent s-point of s_i is the value located at the next even position in S. Therefore, when we grow a frequent k-pattern p in the low resolution, p is joined with its corresponding adjacent s-point to form a frequent (k+1)-pattern. For example, consider the first transaction T_1 in Fig. 5.1. When the Haar wavelet transform is applied to T_1, we obtain a transformed time-series sequence S_1 = $\begin{Bmatrix} 0.2 & 0.3 & 1.2 & 0.3 & 0.8 & 0.9 & 1.4 \\ 0.6 & -0.9 & -0.3 & 0.8 & 0.8 & 0.9 & -0.4 \end{Bmatrix}$. To extend a frequent 1-pattern p = $\begin{Bmatrix} 0.2 \\ 0.6 \end{Bmatrix}$ located at the first position in S_1, we append p's adjacent s-point $\begin{Bmatrix} 1.2 \\ -0.3 \end{Bmatrix}$ to p to form a candidate 2-pattern $\begin{Bmatrix} 0.2 & 1.2 \\ 0.6 & -0.3 \end{Bmatrix}$.

5.3 Numerical pattern tree

In order to enumerate all frequent patterns in the low resolution, we adopt a tree structure, called numerical pattern tree, where the tree consists of one root labeled as ∅

and each node represents a frequent pattern as illustrated in Fig. 5.4. The patterns in the dotted circles are non-closed, whereas the other patterns are closed. Moreover, the number following the pattern denotes the support of the pattern and the number at the top right corner of the pattern refers to its merge list. Recall that a merge list keeps all non-representative patterns merged by the representative pattern.

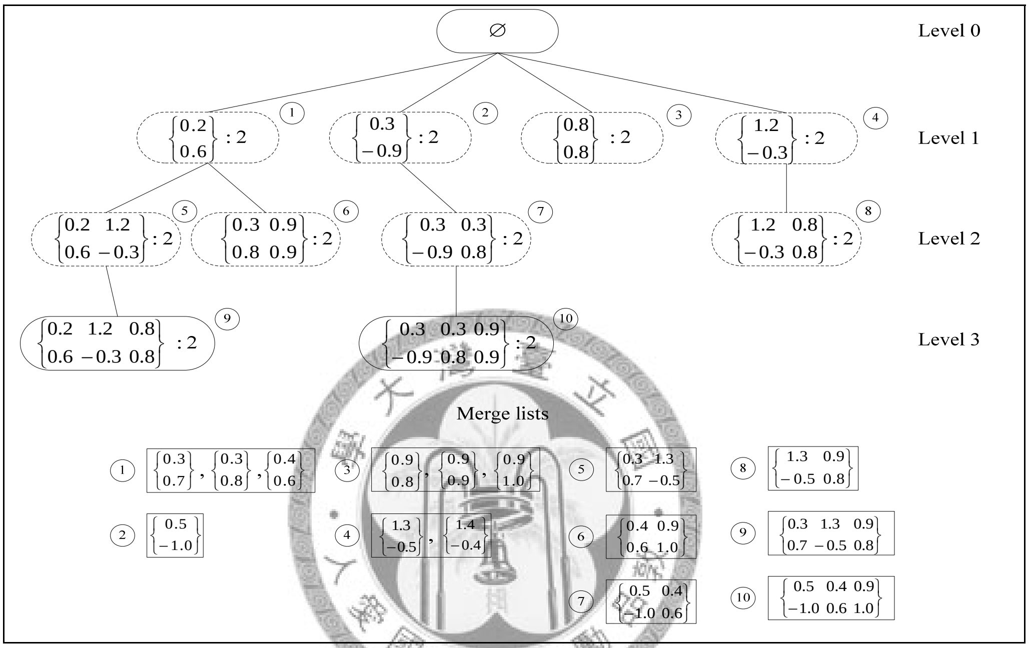


Fig. 5.4. Frequent patterns mined in the low resolution.

5.4 Comparison of projected databases

To check whether two projected databases are the same, we introduce a *max-min matching* method. When a new k-pattern p is discovered, we simultaneously build its projected database. Through the process of building a projected database, we also record the maximum and minimum (max-min for short) values of each dimension for each s-point in p. Since a s-point has s dimensions and the max-min values of each dimension must be recorded, we have a list of s max-min values, denoted as <(max_1, min_1), (max_2, min_2), …, (max_s, min_s)>. Two lists of max-min values <(max_{11}, min_{11}), (max_{12}, min_{12}), …, (max_{1s}, min_{1s})> and <(max_{21}, min_{21}), (max_{22}, min_{22}), …, (max_{2s}, min_{2s})> are *equivalent* if $max_{1i} = max_{2i}$ and $min_{1i} = min_{2i}$, i = 1, 2…, s. Since a k-pattern contains *k* *s*-points, we have *k* lists of max-min values, denotes as $\{M_1, M_2, \cdots, M_k\}$, which is called the *max-min bound* of the *k*-pattern. The max-min bounds of two patterns are *equivalent* if each corresponding list of max-min values is equivalent for both patterns.

Lemma 5.2 (Max-min matching). *The projected databases of patterns p and q are equivalent if p and q have an identical max-min bound.*

Proof. Let the projected databases of *p* and *q* be *PS* and *QS*, respectively, and the max-min bound of *p* be $\{M_1, M_2, \cdots, M_k\}$, where $1 \le i \le k$. Assume that the projected databases of patterns *p* and *q* are not equivalent. That is, there exists a projection *z* that is in *PS* but not in *QS*. Since $z \in PS$, it is certain that each *i*th *s*-point of *z* is bounded by M_i. And, since *q* has an identical max-min bound as *p*, any projection whose *s*-points are bounded by the max-min bound must be existed in *QS*. Thus, *z* must be existed in *QS*. This contradicts the assumption that *z* is in *PS* but not in *QS*. Therefore, if *p* and *q* have an identical max-min bound, the projected databases of patterns *p* and *q* are equivalent.

5.5 Pruning strategies

We adopt two pruning strategies to speed up the mining process. The *post-pruning* and *pre-pruning* strategies are used to prune the unnecessary extension of a pattern.

Lemma 5.3 (Post-pruning). *For a frequent pattern p, if there exists a frequent 1-pattern q, such that* (1) *q is the last s-point of p, and* (2) *both p and q have the same projected database, we can stop growing q.*

Proof. Since *q* is the last *s*-point of *p, p* is a super-pattern of *q*. As *p* and *q* have the same projected database, every projection found in *q*'s projected database is the last *s*-point of one and only one projection in *p*'s projected database. Hence, any pattern extended from *q* can also be extended from *p* and both patterns extended have the same support. This indicates that any pattern extended from *q* is not closed. Therefore, *q* can be pruned.

Note that we have to ensure that pruning the patterns by the post-pruning strategy in the low resolution would not cause any pattern lost in the high resolution. Thus, we double-check whether the candidate 2*k*-patterns restored from *p* and *q* also have the same projected database in the high resolution. If so, it is safe to prune pattern *q*. Otherwise, *q* is preserved.

For example, assume that $\delta = 3$, $\varepsilon = 0.1$, and a transformed database contains three transactions: $S_1 = \begin{Bmatrix} 0.5 & 0.3 & 0.8 \\ 0.6 & 0.5 & 0.9 \end{Bmatrix}$, $S_2 = \begin{Bmatrix} 0.4 & 0.4 & 0.9 \\ 0.5 & 0.4 & 1.0 \end{Bmatrix}$, and $S_3 = \begin{Bmatrix} 0.6 & 0.2 & 0.7 \\ 0.6 & 0.4 & 0.8 \end{Bmatrix}$. By scanning the database, we find three frequent 1-patterns $\begin{Bmatrix} 0.5 \\ 0.6 \end{Bmatrix}$, $\begin{Bmatrix} 0.3 \\ 0.5 \end{Bmatrix}$, $\begin{Bmatrix} 0.8 \\ 0.9 \end{Bmatrix}$, and their projected databases are {<S_1, 1>, <S_2, 1>, <S_3, 1>}, {<S_1, 2>, <S_2, 1>, <S_2, 2>, <S_3, 2>}, and {<S_1, 3>, <S_2, 3>, <S_3, 3>}, respectively. When a frequent pattern $p = \begin{Bmatrix} 0.5 & 0.8 \\ 0.6 & 0.9 \end{Bmatrix}$ is mined and *PDB*(*p*) = {<S_1, 3>, <S_2, 3>, <S_3, 3>}, we check its last *s*-point $\begin{Bmatrix} 0.8 \\ 0.9 \end{Bmatrix}$, and find that $\begin{Bmatrix} 0.8 \\ 0.9 \end{Bmatrix}$ is a frequent 1-pattern whose projected database is the same as *p*. It means that every pattern extended from $\begin{Bmatrix} 0.8 \\ 0.9 \end{Bmatrix}$ is contained by the pattern extended from *p* and is not closed. We further check the candidate 4-pattern restored from $\begin{Bmatrix} 0.5 & 0.8 \\ 0.6 & 0.9 \end{Bmatrix}$ and candidate 2-pattern restored from $\begin{Bmatrix} 0.8 \\ 0.9 \end{Bmatrix}$ and find that they have the same projected database in the high resolution. Therefore, $\begin{Bmatrix} 0.8 \\ 0.9 \end{Bmatrix}$ can be pruned and removed from the first level of the tree.

Lemma 5.4 (Pre-pruning). *For a frequent pattern p, if there exists another frequent pattern q which has already been mined such that* (1) *q is a super-pattern of p, and* (2) *both p and q have the same projected database, p can be pruned.*

Proof. Since *q* is a super-pattern of *p* and both have the same projected database, for every projection *z* in *p*'s projected database, we can find one and only one projection that contains *z* in *q*'s projected database. Hence, any pattern extended from *p* is already mined with *q*. Therefore, *p* can be pruned.

Note that, similar to the post-pruning strategy, we have to ensure that pruning the patterns by the pre-pruning strategy in the low resolution would not cause any pattern lost in the high resolution. Thus, we double-check whether the candidate 2*k*-patterns restored from *p* and *q* also have the same projected database in the high resolution. If so, it is safe to prune pattern *p*. Otherwise, *p* is preserved.

For example, assume that $\delta = 3$, $\varepsilon = 0.1$, $q = \begin{Bmatrix} 0.1 & 0.7 & 0.5 \\ 0.3 & 0.9 & 0.6 \end{Bmatrix}$ is an already mined frequent pattern, and $PDB(q)$ = {<S_1, 7>, <S_2, 3>, <S_3, 9>}. When a new frequent pattern $p = \begin{Bmatrix} 0.7 & 0.5 \\ 0.9 & 0.6 \end{Bmatrix}$ is mined and $PDB(p)$ = {<S_1, 7>, <S_2, 3>, <S_3, 9>}, *p* can be pruned since it is a sub-pattern of *q* and both patterns have the same projected database and the corresponding restored candidates generated from *p* and *q* also have the same projected database in the high resolution.

5.6 Frequent 1-pattern generation

To find frequent 1-patterns from the transformed database, we initially sort all 1-patterns in the transformed database in ascending order and store them in an array *A*, where each 1-pattern is associated with a transaction identifier containing the 1-pattern. To compute the support of a 1-pattern *p*, we find all 1-patterns similar to *p* in the array and count the number of distinct transaction identifiers of these patterns. We call *p* the *pivot* in the array. Moreover, we use two integers *i* and *j* to record the lower and upper bounds of the indices so that the difference between the value of the first dimension of *p* and that of every element between $A[i]$ and $A[j-1]$ is not greater than ε. These elements are the *candidates* that may be similar to *p*. For each candidate, we check if it is similar to *p*. Thus, we obtain all the elements similar to *p*.

Let us elaborate the concept of finding frequent 1-patterns. First, we take $A[1]$ as the pivot and both *i* and *j* are equal to 1. We keep incrementing *j* by one until the difference between the value of the first dimension of $A[j]$ and that of the pivot (*p*) is greater than *ε*. As a result, every element between *A*[*i*] and *A*[*j*–1] is the candidate that may be similar to the pivot. Then, for each candidate, we check if it is similar to the pivot. Thus, we obtain the elements similar to *p*. By counting the number of distinct transaction identifiers of these elements, we can determine the support of *p*. If *p* is frequent, the corresponding projections are stored in *p*'s projected database. Similarly, we take the second element as the new pivot and move indices *i* and *j* to the right places so that every element between *A*[*i*] and *A*[*j*–1] is the candidate that may be similar to the new pivot. We further check if these candidates are similar to the pivot. Then, we count the support of the pivot and check if it is frequent. This process is continued until the last element in the array is reached. Note that if two patterns *p* and *q* have the same projected database and *p* is less than *q*, *q* is merged by *p* and stored in *p*'s merge list.

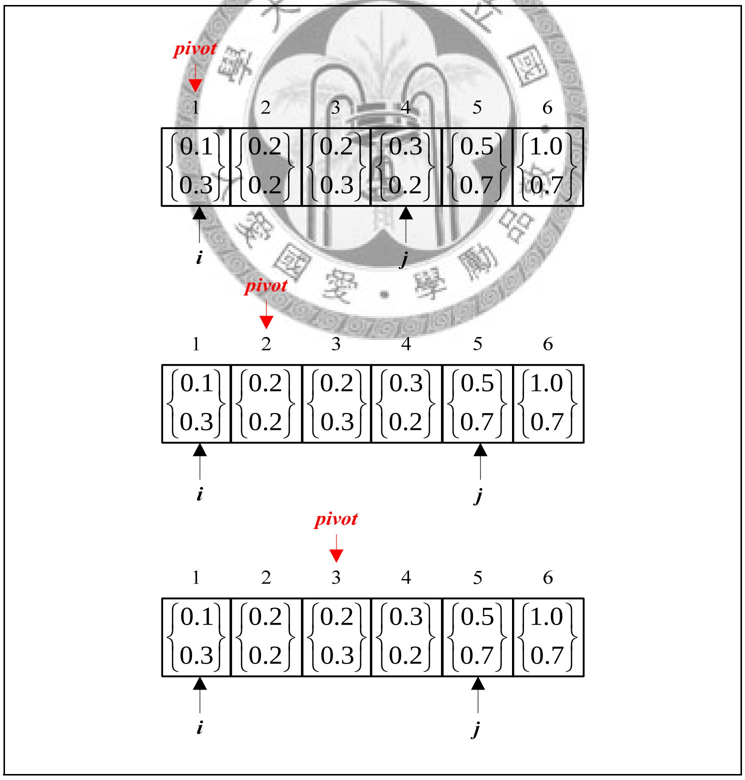


Fig. 5.5. Finding frequent 1-patterns.

Let us consider the example shown in Fig. 5.5. Assume $\varepsilon = 0.1$. We initially take $\begin{Bmatrix}0.1\\0.3\end{Bmatrix}$ to be the pivot, by checking the array, index i is not moved since the first element is similar to itself, whereas index j is moved forward to the fourth position. From i to j–1, there are three candidates which may be similar to the pivot, namely, $\begin{Bmatrix}0.1\\0.3\end{Bmatrix}$, $\begin{Bmatrix}0.2\\0.2\end{Bmatrix}$, and $\begin{Bmatrix}0.2\\0.3\end{Bmatrix}$. We further check if these candidates are similar to the pivot and find that all of them are similar to the pivot. Thus, the corresponding projections are stored in the projected database of $\begin{Bmatrix}0.1\\0.3\end{Bmatrix}$. Next, we change the pivot to $\begin{Bmatrix}0.2\\0.2\end{Bmatrix}$ and check if it is a frequent 1-pattern. We find that i is unchanged while j is moved to the fifth position. Therefore, the corresponding projections are stored in the projected database of $\begin{Bmatrix}0.2\\0.2\end{Bmatrix}$. We further check the next pivot $\begin{Bmatrix}0.2\\0.3\end{Bmatrix}$ and find that it is similar to $\begin{Bmatrix}0.2\\0.2\end{Bmatrix}$ and has the same projected database as $\begin{Bmatrix}0.2\\0.2\end{Bmatrix}$. Hence, $\begin{Bmatrix}0.2\\0.3\end{Bmatrix}$ is merged by $\begin{Bmatrix}0.2\\0.2\end{Bmatrix}$. This process is continued until the last element $\begin{Bmatrix}1.0\\0.7\end{Bmatrix}$ is checked.

5.7 The CNP algorithm

The proposed algorithm, called CNP, is illustrated in Fig. 5.6. It contains a procedure, CNP-Growth, which is shown in Fig. 5.7.

The CNP algorithm consists of three phases. The first phase involves steps to normalize each time-series sequence in the database and then apply the Haar wavelet transform to convert each time-series sequence into a low resolution (step 3). The second phase involves steps to scan the transformed database once to find all frequent 1-patterns and construct a projected database for each 1-pattern found as described in Section 5.6. If any two frequent 1-patterns have the same projected database, the larger one is merged by the smaller one (step 4). The third phase involves steps for the pattern extension and pattern restoring (steps 5-13). That is, for each frequent 1-pattern, we call the CNP-Growth procedure to enumerate the closed patterns. Then, we restore the patterns in the high resolution from the patterns mined in the low resolution. Furthermore, we find all frequent 1-patterns in the high resolution from the original database. Finally, for each closed pattern found, we check if it can be merged by the other patterns.

```
Algorithm: CNP
Input: a time-series database DB, a minimum support threshold $\delta$
Output: all closed patterns $CP_L$ and $CP_H$ in the low and high resolutions, respectively
Initialize $CP_L$ and $CP_H$ to $\emptyset$;
Normalize all time-series in DB;
Apply the Haar wavelet transform to each time-series in DB and store the
        transformed time-series in RDB;
Let P1 be all frequent 1-patterns found in RDB and merge similar 1-patterns into
        one if they have the same projected database;
for each 1-pattern $p \in P1$ do
        Let PDB be the projected database of p;
        call CNP-Growth (p, PDB, $\delta$, P1, $CP_L$, $CP_H$);
        Restore p and each of the patterns merged by p respectively to find the
                frequent 2-patterns and 3-patterns in the high resolution and add them
                to $CP_H$ if they are closed;
end for
Find frequent closed 1-patterns in the high resolution and add them to $CP_H$;
for each closed pattern p in the low and high resolutions do
        Use its max-min bound to decide if it is a representative pattern;
end for
return $CP_L$ and $CP_H$;
```

Fig. 5.6. The CNP algorithm.

```
Procedure: CNP-Growth ($p$, $PDB$, $\delta$, $P1$, $CP_L$, $CP_H$)
Input: a $k$-pattern $p$, a projected database $PDB$, a minimum support threshold $\delta$, and a
    set of frequent 1-patterns $P1$
Output: a set of closed patterns $CP_L$ in the low resolution and a set of closed patterns
    $CP_H$ in the high resolution
if ($p$ is closed) then
    Add $p$ to $CP_L$;
end if
if ($p$ passes the pre-pruning strategy and there exists an adjacent frequent
        1-pattern $q$ that appears right after $p$) then
    Append $q$ to $p$ to form a candidate ($k$+1)-pattern $p$';
    if ($p$ do not pass the post-pruning strategy) then
        Remove $q$ from $P1$;
    end if
    Build a new projected database $PDB$' for $p$' and derive the merge list of $p$';
    if ($p$' is frequent) then
        call CNP-Growth ($p$', $PDB$', $\delta$, $P1$, $CP_L$, $CP_H$);
        Restore $p$' and each of the patterns merged by $p$' respectively to find the
            frequent 2$k$-patterns and (2$k$+1)-patterns in the high resolution and
            add them to $CP_H$ if they are closed;
    end if
    for each frequent pattern $m$' that is extended from $p$'s merge list but not
            merged by $p$' do
        Build a projected database $PDB_{m'}$ for $m$' and derive the merge list of $m$';
        call CNP-Growth ($m$', $PDB_{m'}$, $\delta$, $P1$, $CP_L$, $CP_H$);
        Restore $m$' and each of the patterns merged by $m$' respectively to find
            the frequent 2$k$-patterns and (2$k$+1)-patterns in the high resolution
            and add them to $CP_H$ if they are closed;
    end for
end if
return $CP_L$ and $CP_H$;
```

Fig. 5.7. The CNP-Growth procedure.

In the CNP-Growth procedure, upon getting a frequent *k*-pattern *p*, we first check it with the already mined patterns in CP_L to determine if it is closed in steps 1-3. If it is closed, we then add it to CP_L and extend it.

Next, *p* is checked against the pre-pruning strategy and the condition of whether there exists an adjacent frequent 1-pattern *q* appears right after *p*. If so, we append *q* to *p* to form a candidate pattern *p*' in steps 4-5. Otherwise, we stop growing *p*. Subsequently, we use the post-pruning strategy to prune unnecessary frequent 1-patterns in steps 6-8.

The projected database of *p*' can be derived from the projected database of *p* in step 9. That is, we extend each projection in *p*'s projected database and check if the newly extended projection is similar to *p*'. Accordingly, we obtain a new projected database for *p*'. If the support of *p*' is not less than δ, it is a frequent (*k*+1)-pattern. To further extend *p*' to find longer patterns, we recursively call the CNP-Growth procedure in step 11.

Besides, we use the merge list of *p* to find out which patterns can be merged by *p*'. For each *k*-pattern *m* in *p*'s merge list, we extend *m* to find a candidate (*k*+1)-pattern, *m*'. If *m*' is similar to *p*' and both of them have the same projected database, *m*' is merged by *p*' and stored in the merge list of *p*'. In contrast, if *m*' has a different projected database from *p*' and its support is not less than δ, it is a frequent (*k*+1)-pattern. Thus, we build the projected database for *m*' and derive its merge list in steps 14-15. And we extend *m*' to find frequent super-patterns by calling the CNP-Growth procedure in step 16. The CNP-Growth procedure is recursively called until no more patterns can be generated.

To restore each *k*-pattern *p* found in the low resolution to the high resolution, we first recompose the *s*-points of *p*. Recall that we compute the average between two adjacent *s*-points appeared in $T[i, i+1]$ to obtain a transformed *s*-point located in $S[i, i]$ as shown in Fig. 5.3, where $1 \leq i \leq l-1$ and *l* is the length of *T*. By referring back to the original database, each *s*-point in the low resolution can be recomposed into two *s*-points in the high resolution.

Since p has k s-points, it can be restored to a candidate pattern that has $2k$ s-points. That is, assume p appears in $S[j–2k+2, j]$, we can obtain a restored pattern q_1 of length $2k$ which appears in $T[j–2k+2, j+1]$. Likewise, for a pattern p' of length $k+1$ in the low resolution, it can be restored to a candidate $(2k+2)$-pattern in the high resolution. However, no candidate of length $2k+1$ is restored. Thus, we use p to derive a pattern q_2 of length $2k +1$ which appears in $T[j–2k+2, j+2]$. In other words, we can use p to find two candidate patterns in the high resolution.

To determine the supports of q_1 and q_2, we first restore each projection in p's projected database and check if it is similar to q_1. If yes, it is stored in q_1's projected database. As the projected database of q_1 is built, we further use it to derive the projected database of q_2 by extending the projections in q_1's projected database. As a result, both their supports are calculated. If their supports are not less than δ, they are frequent $2k$-pattern and frequent $(2k+1)$-pattern. We then use the patterns in the closed pattern pool to check whether q_1 and q_2 are closed. For example, assume that $\delta = 2$, $\varepsilon = 0.1$, and an original database contains two transactions: $T_1 = \begin{Bmatrix} 0.5 & 0.7 & 0.1 & 0.3 \\ 0.6 & 1.0 & 0.2 & 0.6 \end{Bmatrix}$, $T_2 = \begin{Bmatrix} 0.4 & 0.8 & 0.2 & 0.4 \\ 0.5 & 0.9 & 0.3 & 0.1 \end{Bmatrix}$ which is transformed to a new database that contains: $S_1 = \begin{Bmatrix} 0.6 & 0.4 & 0.2 \\ 0.8 & 0.6 & 0.4 \end{Bmatrix}$, $S_2 = \begin{Bmatrix} 0.6 & 0.5 & 0.3 \\ 0.7 & 0.6 & 0.2 \end{Bmatrix}$. From the transformed database, we find that $p = \begin{Bmatrix} 0.6 \\ 0.8 \end{Bmatrix}$ is a frequent 1-pattern in the low resolution. When we restore it to the high resolution, we may obtain two candidate patterns, including $q_1 = \begin{Bmatrix} 0.5 & 0.7 \\ 0.6 & 1.0 \end{Bmatrix}$ and $q_2 = \begin{Bmatrix} 0.5 & 0.7 & 0.1 \\ 0.6 & 1.0 & 0.2 \end{Bmatrix}$, where their projected databases are $\{<T_1, 2>, <T_2, 2>\}$ and $\{<T_1, 3>, <T_2, 3>\}$, respectively. Since q_2 contains q_1 and their supports are equal to 2, q_1 is not closed and only q_2 is added to CP_H.

Note that, each k-pattern m merged by p in the low resolution, is also restored to generate a candidate $2k$-pattern and a candidate $(2k+1)$-pattern in the high resolution in order to verify whether they can be merged by the restored patterns generated from *p*. If not, we compare them with the already mined closed patterns and output them to the closed pattern pool if they are closed.

It is shown that frequent 1-patterns in the low resolution are restored to find frequent 2-patterns and frequent 3-patterns in the high resolution; however, no frequent 1-patterns in the high resolution are retrieved. Thus, we mine frequent closed 1-patterns from the original database and add them to the closed pattern pool in step 10 of the CNP algorithm.

Once we have obtained all closed patterns in both low and high resolutions, we then proceed to select representative patterns among all closed patterns in each resolution in steps 11-13 of the CNP algorithm. Recall that while we build the projected database of a pattern, we also compute its max-min bound. We then use this information to select the representative pattern. For a pair of closed *k*-patterns, if they have identical max-min bounds, we choose the smaller pattern to be the representative pattern. As a result, we can obtain all representative closed patterns in both low and high resolutions.

Lemma 5.5 *Each pattern found by the CNP algorithm is frequent and closed*.

Proof. In the CNP algorithm, the second phase involves steps to find all frequent 1-patterns by scanning the transformed database once. The third phase involves the CNP-Growth procedure to recursively find longer patterns. That is, we extend a frequent *k*-pattern *p* by appending an adjacent frequent 1-pattern *q* which appears right after *p* to form a candidate (*k*+1)-pattern. Then we check the support of this candidate. If its support is greater than δ, it is a frequent (*k*+1)-pattern. As a result, we have assured that every pattern found in the low resolution is frequent. On the other hand, every frequent *k*-pattern *p* found in the low resolution is restored to two candidate patterns in the high resolution. We further check if these two candidate patterns are frequent. In addition, we mine frequent 1-patterns from the original database. Therefore, every pattern found in the high resolution is frequent. Moreover, upon getting a frequent pattern in the low resolution, we always check it with the already mined patterns in CP_L to determine if it is closed. If it is closed, it is added to CP_L. Similarly, for every restored frequent pattern or frequent 1-pattern found in the high resolution, we check it with the already mined patterns in CP_H and add it to CP_H if it is closed. Therefore, every pattern found by the CNP algorithm is frequent and closed.

Lemma 5.6 *Every closed pattern can be found by the CNP algorithm*.

Proof. Since we scan the transformed database once to find all frequent 1-patterns, every frequent 1-pattern in the low resolution can be found by the CNP algorithm. For each frequent *k*-pattern *p* in the low resolution, we combine it with an adjacent frequent 1-pattern *q* which appears right after *p* to generate all its frequent super-patterns of *p*. Thus, every frequent *k*-pattern in the low resolution can be found by the CNP algorithm. For each frequent *k*-pattern found in the low resolution, we restore it to find two candidate patterns in the high resolution and eliminate the infrequent ones. Moreover, we scan the original database once to find all frequent 1-patterns in the high resolution. Therefore, every frequent *k*-pattern in the high resolution can be found by the CNP algorithm. Once a frequent *k*-pattern is found either in the low resolution or in the high resolution, we use the already mined patterns in the pool to eliminate non-closed patterns. Therefore, every closed pattern can be found by the CNP algorithm.

Theorem 5.1 *The CNP algorithm enumerates all closed patterns in both low and high resolutions in the database*.

Proof. By Lemma 5.5, every pattern found by the CNP algorithm is frequent and closed. By Lemma 5.6, every closed pattern can be found by the CNP algorithm. Therefore, we can conclude that the CNP algorithm enumerates all closed patterns in the database in both low and high resolutions.

Theorem 5.2 *The time and space complexities of the CNP algorithm are* $O(|D|*\log|D| + |N_L|*|D| + |CP_L|^2 + |CP_H|^2)$ *and* $O(lp*|D| + lv*(|CP_L| + |CP_H|))$, *respectively, where the size of a time-series database is* $|D|$, *the number of nodes in the numerical pattern tree* *in the low resolution is* $|N_L|$*, the numbers of closed patterns in the low and high resolutions are* $|CP_L|$ and $|CP_H|$*, respectively, the average length of the frequent pattern is lv, and the length of the longest frequent pattern is lp.*

Proof. According to [19], the time complexity of transforming a time-series in the high resolution into a time-series in the low resolution is bounded by $O(l)$, where l is the length of a time-series. Since the size of a time-series database is $|D|$, the time complexity of the Haar wavelet transform is bounded by $O(|D|)$. To obtain all frequent 1-patterns in the low resolution, we sort all 1-patterns in the transformed database and check whether each 1-pattern is frequent. It requires $O(|D|*\log|D|)$ time to perform this task. Next, let us consider the mining process in the low resolution. To extend a frequent k-pattern p, we append an adjacent frequent 1-pattern that appears right after p to form its super-pattern and meanwhile derive a new projected database for the super-pattern. This process takes $O(|D|)$ time since the maximum size of projected databases is $|D|$ in the worse case. Since there are $|N_L|$ nodes in the numerical pattern tree in the low resolution, the time-complexity of the CNP algorithm in the low resolution is bounded by $O(|D| + |D|*\log|D| + |N_L|*|D|) = O(|D|*\log|D| + |N_L|*|D|)$. Next, let us consider the restoring operation in the CNP algorithm. Since p can be restored into two frequent patterns at most in the high resolution, the maximum number of frequent patterns in the high resolution is bounded by $2*|N_L|$. During the restoring process, p's projected database is scanned once to check if the restored patterns are frequent, and hence the time complexity of the restoring operation is bounded by $O(|N_L|*|D|)$. The time complexity of checking if a pattern is closed is bounded by $O(|CP_L|)$ and $O(|CP_H|)$ with respect to the low and high resolutions. Moreover, the time complexity of the merge operation in the low resolution is bounded by $O(|CP_L|^2)$ because each closed pattern is checked against the other $|CP_L| - 1$ patterns to determine if it is a representative pattern. Similarly, the time complexity of the merge operation in the high resolution is bounded by $O(|CP_H|^2)$. Besides, the time complexity of mining frequent 1-patterns in the high resolution is bounded by $O(|D|*\log|D|)$. Therefore, the time complexity of the CNP algorithm is bounded by $O(|D| + |D|*\log|D| + |N_L|*|D| + 2*|N_L|*|D| + |CP_L| + |CP_H| + |CP_L|^2 + |CP_H|^2 + |D|*\log|D|) = O(|D|*\log|D| + |N_L|*|D| + |CP_L|^2 + |CP_H|^2)$. In order to select the representative patterns, we record the max-min bounds for the closed patterns in both low and high resolutions. The memory space used to preserve this information is thus bounded by $O(lv*(|CP_L| + |CP_H|))$. Since the CNP algorithm is processed in a DFS manner, the maximum number of nodes that kept in the memory is bounded by lp. Moreover, each node requires $O(|D|)$ space to maintain its projected database and $O(lv)$ space to store its max-min bounds. To store all closed patterns in both low and high resolutions, we need $O(|CP_L| + |CP_H|)$ space. Therefore, the space complexity of the CNP algorithm is bounded by $O(|D| + |D| + lv*(|CP_L| + |CP_H|) + lp*(|D| + lv) + |CP_L| + |CP_H|) = O(lp*|D| + lv*(|CP_L| + |CP_H|))$.

5.8 Mining multi-resolution patterns

The CNP algorithm can be modified to find multi-resolution patterns. Given a number of resolutions γ and a time-series database, we convert each time-series in the database into the lowest resolution by applying the Haar wavelet transform $(\gamma–1)$ times. Once we have obtained a transformed database in the lowest resolution, we perform the tasks in the second and third phases of the CNP algorithm to mine all closed patterns in the lowest resolution. However, to restore a frequent k-pattern p in the lowest resolution to the highest resolution, we have to perform the restoring operations multiple times until the highest resolution is reached. Specifically, p is first restored to find frequent $2k$-pattern e_1 and frequent $(2k+1)$-pattern e_2 in the resolution which is one level higher than the lowest resolution. Subsequently, e_1 and e_2 are restored to find frequent $4k$-pattern g_1, frequent $(4k+1)$-pattern g_2, frequent $(4k+2)$-pattern g_3, and frequent $(4k+3)$-pattern g_4 in the resolution which is two levels higher, respectively. Furthermore, we restore g_1, g_2, g_3, and g_4 to find frequent patterns in the resolution which is three levels higher, and so on. For each frequent pattern found in the resolution *i*, we insert it to the corresponding closed pattern pool CP_i if it is closed. As a result, we obtain all closed patterns in different resolutions. Finally, for each closed pattern found, we check if it can be merged by the other patterns and find all representative patterns in different resolutions.

5.9 An example

In this section, let us demonstrate how the CNP algorithm works. Consider a time-series database containing two multi-sequences as shown in Fig. 5.1, where each sequence is transformed into a new sequence in the low resolution by the Haar wavelet transform. We obtain a transformed database that contains two transactions: S_1 = $\left\{\begin{matrix} 0.2 & 0.3 & 1.2 & 0.3 & 0.8 & 0.9 & 1.4 \\ 0.6 & -0.9 & -0.3 & 0.8 & 0.8 & 0.9 & -0.4 \end{matrix}\right\}$ and S_2 = $\left\{\begin{matrix} 1.7 & 0.3 & 0.5 & 1.3 & 0.4 & 0.9 & 0.9 \\ 0.2 & 0.7 & -1.0 & -0.5 & 0.6 & 0.8 & 1.0 \end{matrix}\right\}$. Assume that δ = 2 and ε = 0.2. We first generate frequent 1-patterns from the transformed database and store them at the first level of the tree as illustrated in Fig. 5.4. Meanwhile, a projected database is built for each frequent 1-pattern found and similar frequent 1-patterns are merged together.

Next, we extend each frequent 1-pattern by a DFS manner. Consider the first frequent 1-pattern *p* = $\left\{\begin{matrix} 0.2 \\ 0.6 \end{matrix}\right\}$ with its merge list containing $\left\{\begin{matrix} 0.3 \\ 0.7 \end{matrix}\right\}$, $\left\{\begin{matrix} 0.3 \\ 0.8 \end{matrix}\right\}$, and $\left\{\begin{matrix} 0.4 \\ 0.6 \end{matrix}\right\}$. Since *p* passes the pre-pruning strategy and there exists an adjacent frequent 1-pattern *q* = $\left\{\begin{matrix} 1.2 \\ -0.3 \end{matrix}\right\}$ occurred right after *p*, we extend *p* by appending *q* to its tail and thus form a candidate 2-pattern *p*'= $\left\{\begin{matrix} 0.2 & 1.2 \\ 0.6 & -0.3 \end{matrix}\right\}$. Moreover, no pattern at the first level of the tree can be pruned by *p* because *p* passes the post-pruning strategy. The projected database of *p*' can be derived from its parent *p* by extending all projections in the *p*'s projected database and check whether the newly extended projections are similar to *p*'. As a result, *p*' is a frequent 2-pattern. Besides, we extend the patterns which are merged by *p* to find candidate 2-patterns that are potentially to be merged by *p*'. In this case, we find three candidate patterns, m_1 = $\begin{Bmatrix} 0.3 & 1.3 \\ 0.7 & -0.5 \end{Bmatrix}$, m_2 = $\begin{Bmatrix} 0.3 & 0.9 \\ 0.8 & 0.9 \end{Bmatrix}$, and m_3 = $\begin{Bmatrix} 0.4 & 0.9 \\ 0.6 & 1.0 \end{Bmatrix}$, and all of them are frequent. As m_1 and *p*' have the same projected database, m_1 is merged by *p*'. On the other hand, both patterns m_2 and m_3 no longer share the same projected database with *p*'. Hence, they independently form new nodes at the second level of the tree. As m_2 and m_3 share the same projected database, m_3 is merged by m_2.

To further extend pattern *p*' to frequent 3-patterns, we perform the CNP-Growth procedure. Consequently, we obtain a frequent 3-pattern $\begin{Bmatrix} 0.2 & 1.2 & 0.8 \\ 0.6 & -0.3 & 0.8 \end{Bmatrix}$. Likewise, we can extend other frequent 1-patterns to find its super-patterns. For each frequent *k*-pattern found, we use the already mined patterns in CP_L to check whether the pattern is closed, where CP_L is a closed pattern pool in the low resolution. Finally, we find two closed patterns in the low resolution, including $\begin{Bmatrix} 0.2 & 1.2 & 0.8 \\ 0.6 & -0.3 & 0.8 \end{Bmatrix}$ and $\begin{Bmatrix} 0.3 & 0.3 & 0.9 \\ -0.9 & 0.8 & 0.9 \end{Bmatrix}$, and we store them in CP_L.

Upon getting a frequent *k*-pattern *p* in the low resolution, we restore it to find frequent patterns in the high resolution. Recall that a pattern in the low resolution can be restored to find candidate 2*k*-pattern and (2*k*+1)-pattern in the high resolution. Consider the frequent 3-pattern *p* = $\begin{Bmatrix} 0.2 & 1.2 & 0.8 \\ 0.6 & -0.3 & 0.8 \end{Bmatrix}$, when we refer back to the original database, we find that the first *s*-point $\begin{Bmatrix} 0.2 \\ 0.6 \end{Bmatrix}$ of *p* is recomposed into $\begin{Bmatrix} -0.6 & 1.0 \\ 4.4 & -3.2 \end{Bmatrix}$, the second *s*-point $\begin{Bmatrix} 1.2 \\ -0.3 \end{Bmatrix}$ of *p* into $\begin{Bmatrix} -0.4 & 2.8 \\ 1.4 & -2.0 \end{Bmatrix}$, and the third *s*-point $\begin{Bmatrix} 0.8 \\ 0.8 \end{Bmatrix}$ of *p* into $\begin{Bmatrix} -2.2 & 3.8 \\ 3.6 & -2.0 \end{Bmatrix}$. Therefore, *p* is restored to a candidate 6-pattern, c_1 = $\begin{Bmatrix} -0.6 & 1.0 & -0.4 & 2.8 & -2.2 & 3.8 \\ 4.4 & -3.2 & 1.4 & -2.0 & 3.6 & -2.0 \end{Bmatrix}$ and a candidate 7-pattern, c_2 = $\begin{Bmatrix} -0.6 & 1.0 & -0.4 & 2.8 & -2.2 & 3.8 & -2.0 \\ 4.4 & -3.2 & 1.4 & -2.0 & 3.6 & -2.0 & 3.8 \end{Bmatrix}$ in the high resolution. To determine the supports of c_1 and c_2, we restore all projections contained in p's projected database and find that the supports of c_1 and c_2 are all equal to 2. Thus, both c_1 and c_2 are frequent. However, since c_1 is contained by c_2, c_1 is not closed. Likewise, we restore each frequent pattern found in the low resolution to find frequent patterns in the high resolution. Furthermore, we check whether a restored frequent pattern is closed by comparing it with the already mined patterns in CP_H, where CP_H is a closed pattern pool in the high resolution. A complete set of closed patterns mined in the high resolution is shown in Fig. 5.8.

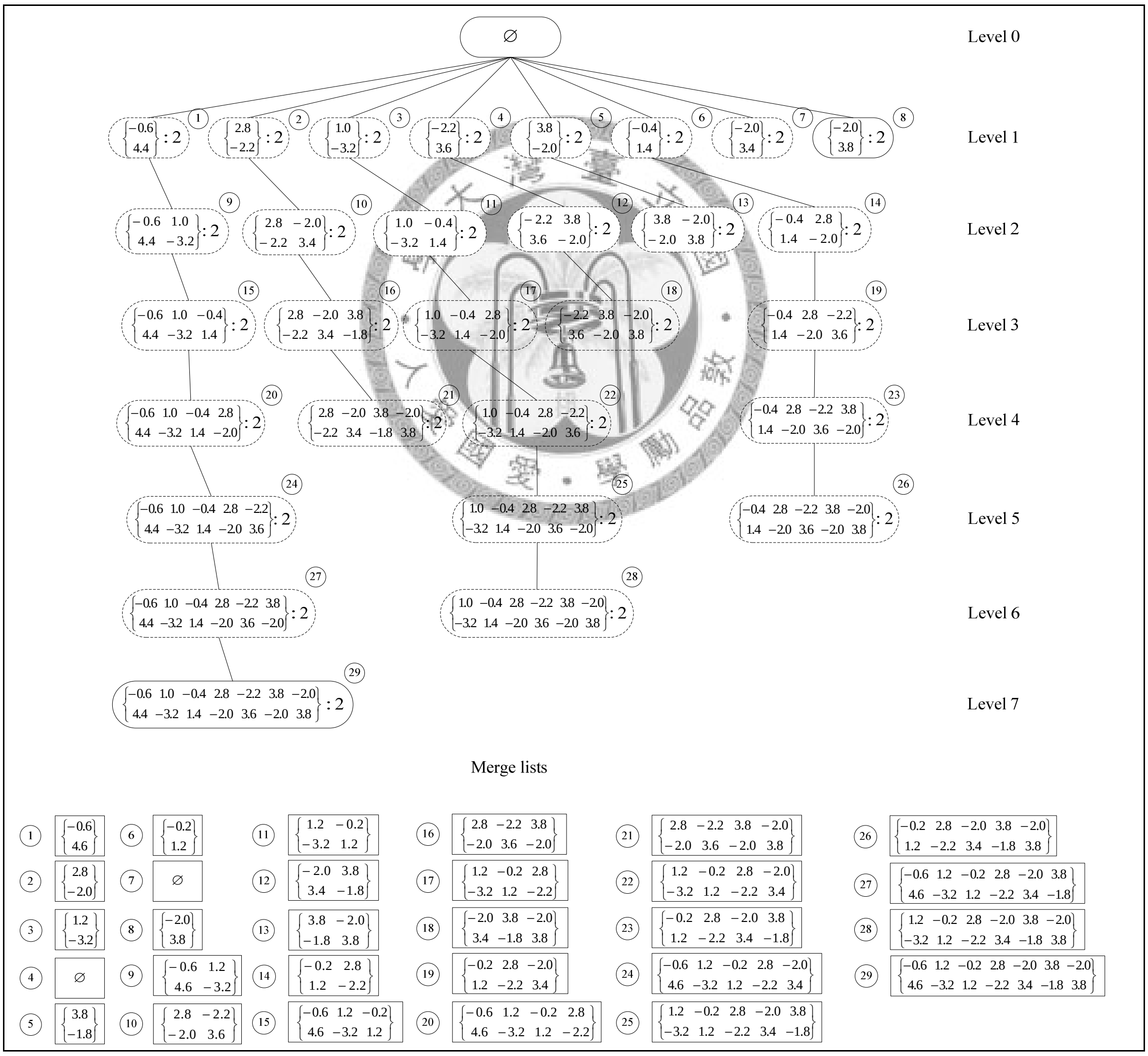


Fig. 5.8. Frequent patterns mined in the high resolution.

Chapter 6 Performance Evaluation

In this chapter, we conduct experiments on both synthetic and real data to assess the efficiency of the proposed CMP, CFP, and CNP algorithms. The experiments are performed in the different environments with respect to each algorithm as listed in Table 6.1.

Table 6.1. Environment settings for the proposed algorithms.

Algorithm	Platform	Programming Language
CMP	IBM compatible PC with Intel Core 2 Duo CPU 1.86GHz, 2GB main memory, running on MS Windows XP Professional.	Microsoft Visual C++ 6.0
CFP & CNP	IBM compatible PC with Intel Core 2 Quad CPU 2.4GHz, 2GB main memory, running on MS Windows XP Professional.	Microsoft Visual C++ 2005

6.1 Synthetic data

We use a similar approach suggested by Srikant and Agrawal [53] to generate the synthetic datasets. First, we generate potential frequent patterns and use them to construct transactions in the database. Besides the potential frequent patterns, we insert the items (either symbols for the CMP and CFP algorithms or numerical values for the CNP algorithm) into the sequence following an exponential distribution with mean equal to 1.

We conduct several experiments under various parameter settings. For each experiment, we vary one parameter and set other parameters to their default values. The default values for these parameters used in the proposed CMP, CFP, and CNP algorithms are listed in Tables 6.2-6.4, respectively. Note that the support of a pattern is defined as the fraction of transactions containing the pattern in the database in the experimental section.

Table 6.2. Parameters used in the CMP algorithm.

Parameter	Description	Default value
$\|T\|$	Number of transactions in the database	10,000
L	Length of transactions	10
P	Number of potential frequent patterns in the database	800
A	Average length of potential frequent patterns in the database	6
S	Number of sequences in a transaction	2
Sym	Number of symbols in a sequence	5
δ	Minimum support threshold	2%
τ	Maximum gap between any two symbols	2

Table 6.3. Parameters used in the CFP algorithm.

Parameter	Description	Default value
$\|T\|$	Number of transactions in the database	10,000
L	Length of transactions	8
P	Number of potential frequent patterns in the database	800
A	Average length of potential frequent patterns in the database	6
Sym	Number of symbols in a sequence	5
δ	Minimum support threshold	6%
τ	Maximum gap between any two symbols	3

Table 6.4. Parameters used in the CNP algorithm.

Parameter	Description	Default value
$\|T\|$	Number of transactions in the database	50,000
L	Length of transactions	16
P	Number of potential frequent patterns in the database	1000
A	Average length of potential frequent patterns in the database	10
S	Number of sequences in a transaction	2
δ	Minimum support threshold	0.05%
ε	Distance threshold	0.01
γ	Number of resolutions	2

6.2 Performance evaluation of the CMP algorithm

We compare the CMP algorithm with the modified Apriori and BIDE algorithms. The modified Apriori algorithm generates the frequent patterns level by level. At each level, it combines two frequent *k*-patterns to generate a candidate (*k*+1)-pattern. For each candidate (*k*+1)-pattern, the database is scanned to count its support and check if it is frequent. The process is continued until no more frequent patterns can be generated. The modified Apriori algorithm uses only the anti-monotone property to prune the unnecessary candidates.

Since the BIDE algorithm is originally designed to analyze datasets that contain only a single sequence in each transaction, a multi-sequence time-series database is transformed into a single sequence time-series database by composing multiple sequences in a transaction into one sequence. That is, for each transaction that contains multiple sequences, we append these sequences one by one to form a new transaction. For example, a transaction that contains two sequences $\begin{Bmatrix} a & b & c & c & d \\ y & y & x & y & z \end{Bmatrix}$ is transformed into a single sequence $\{a \quad b \quad c \quad c \quad d \quad y \quad y \quad x \quad y \quad z\}$. After the modified BIDE algorithm mines all the closed patterns in a single-sequence format, the mined patterns may be transformed into a multi-sequence format. For example, the patterns $\{a \quad b \quad \_ \quad \_ \quad \_ \quad y \quad y\}$ and $\{a \quad \_ \quad c \quad \_ \quad \_ \quad y \quad \_ \quad x\}$ mined by the modified BIDE algorithm, may be transformed into $\begin{Bmatrix} a & b \\ y & y \end{Bmatrix}$, and $\begin{Bmatrix} a & \_ & c \\ y & \_ & x \end{Bmatrix}$, respectively. To compare the modified Apriori, the modified BIDE and the CMP algorithms, we first mine all closed patterns and then generate a complete set of frequent patterns from the closed patterns mined.

6.2.1 Evaluations on synthetic data

Fig. 6.1 shows the runtime versus the minimum support threshold, where the minimum support threshold varies from 0.01% to 5%. The CMP algorithm runs about

33-350 times faster than the modified Apriori algorithm and runs about 14-43 times faster than the modified BIDE algorithm. The CMP algorithm outperforms the modified Apriori algorithm because the latter generates a huge number of candidates, especially when the minimum support threshold is low. By using the projected database, the CMP algorithm requires only one database scan and can localize support counting, candidate pruning, and closure checking in the projected database. It also adopts the closure checking and pruning strategies to accelerate the mining process and to avoid generating many unnecessary candidates. Even though the modified BIDE algorithm avoids candidate generation and uses similar pruning strategies as the CMP algorithm, the length of a transaction in the modified BIDE algorithm is longer than that in the CMP algorithm. Thus, the modified BIDE algorithm spends more execution time than the CMP algorithm. Therefore, the CMP algorithm outperforms the modified Apriori and BIDE algorithms.

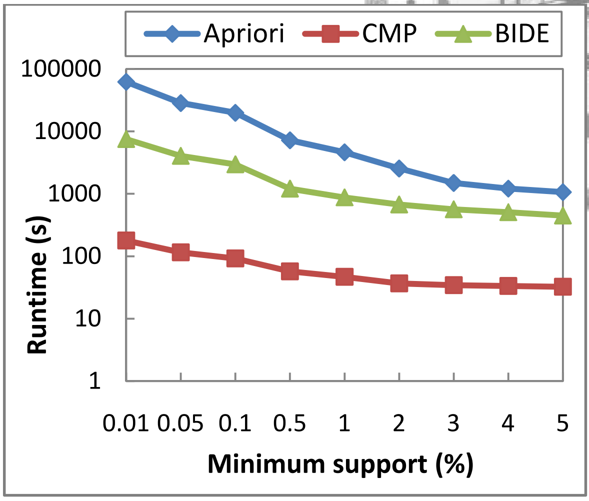


Fig. 6.1. Runtime versus minimum support (CMP).

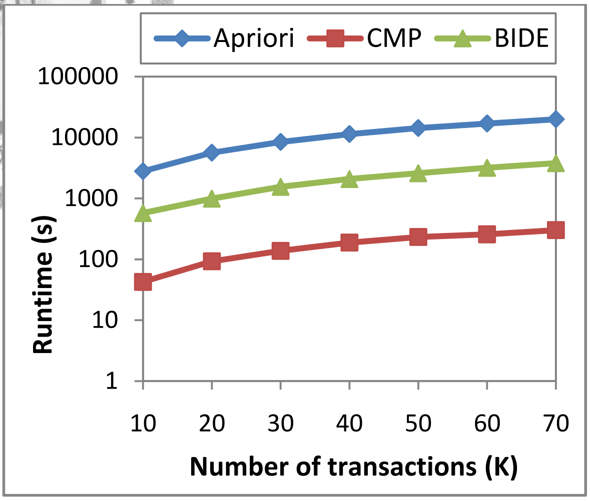


Fig. 6.2. Runtime versus number of transactions (CMP).

Fig. 6.2 shows the runtime versus the number of transactions in the database, where the number of transactions varies from 10,000 to 70,000, and the minimum support threshold is equal to 2%. The runtime of all three algorithms increases almost linearly as the number of transactions increases; however, the CMP algorithm runs about 63 times faster than the modified Apriori algorithm and about 12 times faster than the modified BIDE algorithm.

Fig. 6.3 depicts the runtime versus the length of a transaction where the length of a transaction varies from 8 to 20. As the length of a transaction increases, the number of closed patterns increases. Therefore, the runtime of all three algorithms increases. However, the CMP algorithm is about 54-85 times faster than the modified Apriori algorithm and about 6-270 times faster than the modified BIDE algorithm. Moreover, we notice that the runtime of the modified BIDE algorithm rises sharply as the transaction length increases. This is because the length of each transaction is doubled as we append multiple sequences to form a single sequence, and mining such long single sequences costs more time. The modified BIDE algorithm runs slower than the modified Apriori algorithm when the length of transactions is greater than or equal to 16.

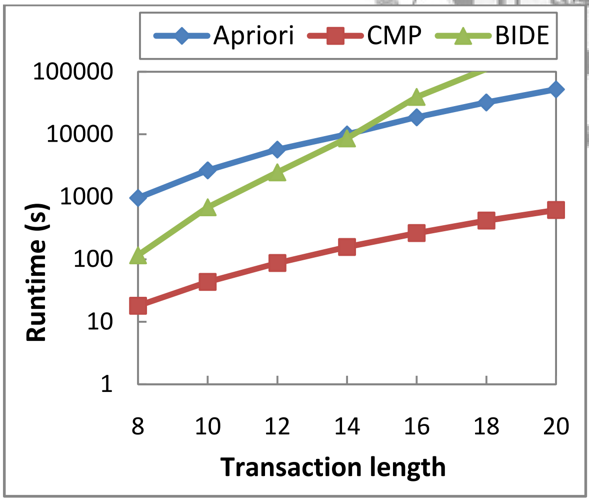


Fig. 6.3. Runtime versus transaction length (CMP).

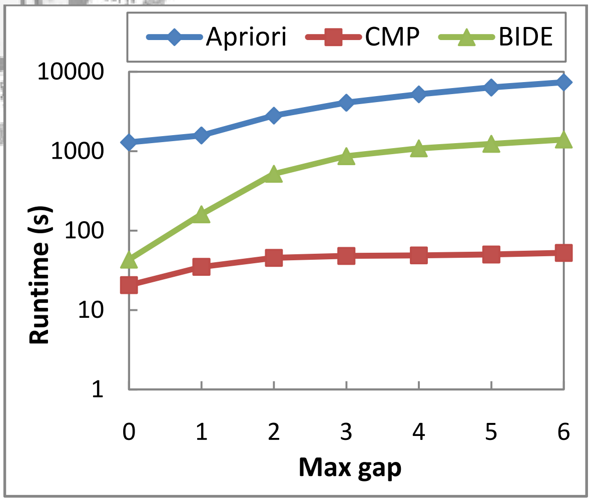


Fig. 6.4. Runtime versus maximum gap (CMP).

Next, we show the impact of the maximum number of gaps on the performance of all three algorithms in Fig. 6.4. As the maximum number of gaps increases, more candidates or patterns are generated. Thus, the runtime of all three algorithms increases. We observe that the runtime of the CMP algorithm increases slowly as the maximum number of gaps increases. The CMP algorithm is about 45-140 times faster than the modified Apriori algorithm and about 2-27 times faster than the modified BIDE algorithm.

Fig. 6.5 shows the runtime versus the number of sequences in a transaction. As the number of sequences in a transaction increases, the runtime of all three algorithms increases. When the number of sequences in a transaction increases, it takes much more time to check whether the candidate patterns are frequent for the modified Apriori algorithm. When the number of sequences in a transaction is more than five, the modified Apriori algorithm cannot find the complete set of frequent patterns in a reasonable time. As well, the modified BIDE algorithm cannot mine patterns in a reasonable time. Since multiple sequences are appended together to form a single sequence, the length of the single sequence formed is five times as long as the length of the transaction with multiple sequences. The modified BIDE algorithm cannot efficiently mine closed patterns in the database with such long sequences. However, the CMP algorithm can efficiently perform the mining task in such databases.

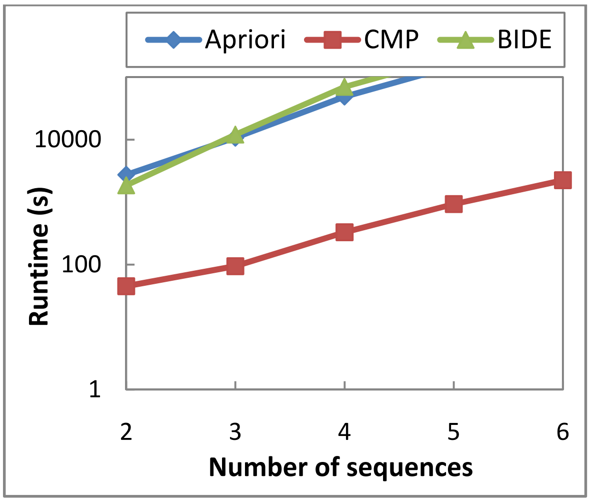


Fig. 6.5. Runtime versus number of sequences (CMP).

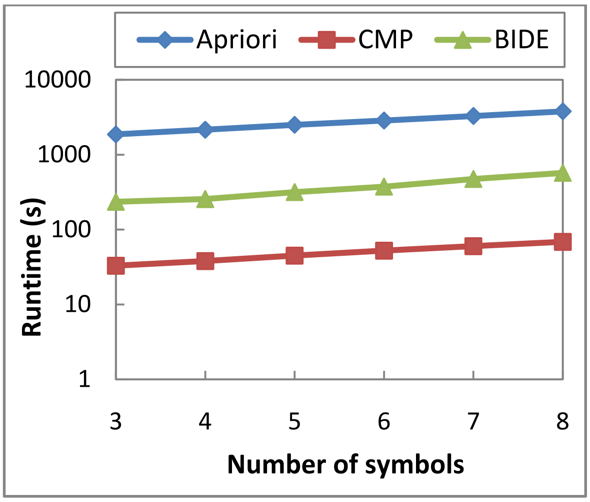


Fig. 6.6. Runtime versus number of symbols (CMP).

Fig. 6.6 shows the runtime versus the number of symbols in a sequence where the number of symbols varies from 3 to 8. Note that when the number of symbols in a sequence increases, more frequent 1-patterns may be mined from the database. Recall that the frequent super-pattern is generated by concatenating the *k*-pattern, gaps, and one of the frequent 1-patterns in the projected database. As the number of frequent 1-patterns increases, the CMP and the modified BIDE algorithms need to check more concatenated patterns. The runtime of the modified Apriori algorithm also increases as the number of symbols increases. The reason is that more candidates are generated as the number of symbols increases, and hence the algorithm spends more effort on pattern concatenation and support counting. Therefore, all three methods appear to spend more time in the mining process. However, the CMP algorithm still outperforms the other algorithms in all cases.

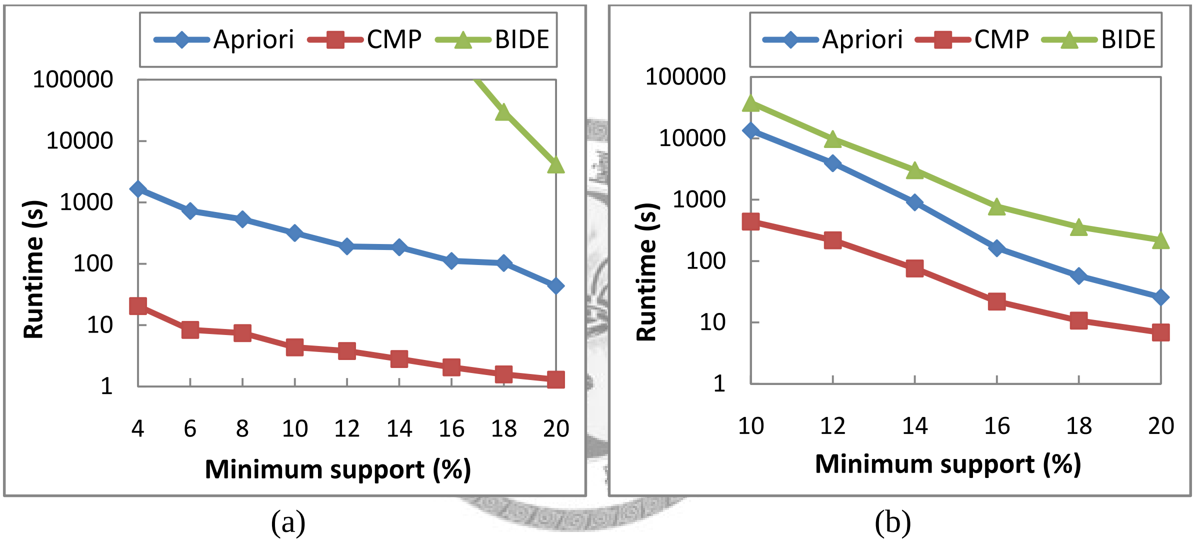
In summary, since the CMP algorithm employs the projected database, closure checking, and pruning strategies to mine closed patterns, it is more efficient and scalable than the modified Apriori and BIDE algorithms in all cases, especially when the minimum support threshold is low or the number of sequences in a transaction is large. The advantage of using the projected database is that the CMP algorithm can localize support counting, candidate pruning, and closure checking to speed up the mining process. Moreover, the CMP algorithm adopts the closure checking and pruning strategies to avoid generating many unnecessary candidates. Therefore, the experimental results show that the CMP algorithm outperforms the modified Apriori algorithm by one or two orders of magnitude. Compared with the modified BIDE algorithm, the CMP algorithm achieves better performance in all cases. Although the modified BIDE algorithm and the CMP algorithm both employ projected databases and similar pruning strategies to mine closed patterns, the transactions used in the modified BIDE algorithm is 2-6 times longer than those in the CMP algorithm. Thus, the CMP algorithm is more efficient and scalable than the modified BIDE algorithm.

6.2.2 Evaluations on real data

In this sub-section, we compare the CMP, and the modified Apriori and BIDE algorithms using two real datasets: weather and stock. The weather dataset is retrieved from the Data Bank for Atmospheric Research (DBAR) and the period of the dataset is from January 2003 to December 2007 with a total of 1825 transactions [16]. Each transaction in the database includes two sequences: temperature and relative humidity in a day. These measurements are taken at the weather station in Taipei. Both sequences are recorded hourly. In the experiment, we aim to examine the influence of temperature and relative humidity on cloud formation. Hence, we select dates with average cloudiness greater than six on a decimal scale where a value of six refers to mostly cloudy [7]. As a result, 1348 days are selected. Therefore, the database contains 1348 transactions, and the length of each transaction is 24.

Next, we transform each sequence in a transaction into a symbolic sequence in the weather dataset. The first sequence in a transaction represents temperature and it includes five different symbols L_1, ML_1, M_1, MH_1, and H_1 from low to high, where the symbols L_1, ML_1, M_1, MH_1, and H_1 represent low, medium low, medium, medium high, and high temperature, respectively. The second sequence in a transaction represents relative humidity, and also uses five different symbols L_2, ML_2, M_2, MH_2, and H_2 from low to high humidity.

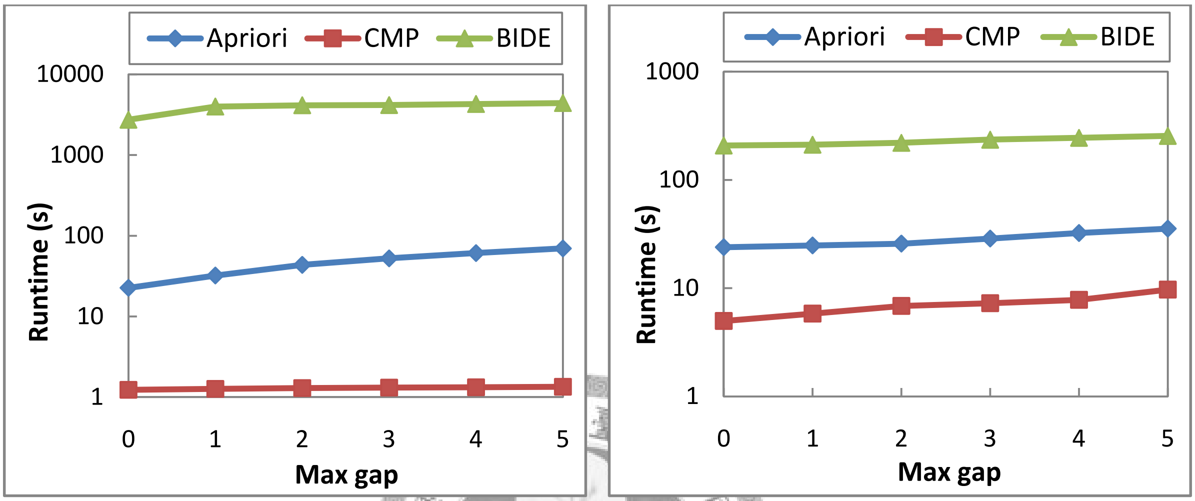
The stock dataset is collected from the Taiwan Stock Exchange Corporation (TSEC) [57] from January 2000 to December 2007. Seven stock indexes are chosen, including Nasdaq Composite Index (IXIC), Dow Jones Industrial average (DJI), Tokyo Nikkei 225 Price Index (NK-225), Korea Composite Stock Price Index (KOSPI), Hong Kong Hang Seng Index (HSI), Singapore Straits Times Index (STI), and Taiwan Stock Index (TAIEX). Each of these stock market movements is represented by one of the sequences in a transaction. Each sequence in a transaction represents the movements of the stock market in a month. In this experiment, we aim to examine if Asia stock market is likely to be affected by U.S. stock market. That is, we doubt whether there exists a pattern that contains all seven stock indexes (multiple sequences) moving in the same direction. Therefore, the database contains 96 transactions (96 months), where each transaction includes seven sequences and the length of each transaction is about 20. Then, each sequence is transformed into a symbolic sequence. The movements of a stock market are classified into three levels: rising, falling, and constant. That is, each sequence is formed by three symbols: R_i (rising), F_i (falling), and C_i (constant), where i is the sequence ID in a transaction, $1 \leq i \leq 7$.



(a) (b)

Fig. 6.7. Runtime versus minimum support: (a) weather and (b) stock.

The performance of the three algorithms using the real datasets is quite similar to that found using the synthetic data. Fig. 6.7 illustrates the runtime versus the minimum support threshold for the weather and stock datasets. In the modified BIDE algorithm, a new transaction is formed by appending the second sequence to the first sequence. The length of a transaction is 48 in the weather dataset and about 140 in the stock dataset. Since the modified BIDE algorithm cannot efficiently mine frequent closed patterns in long transactions, it runs slower than the CMP and the modified Apriori algorithms. The CMP algorithm outperforms the modified Apriori algorithm because the latter generates a large number of candidates, especially when the minimum support threshold is low. In addition, the CMP algorithm adopts closure checking and pruning strategies to accelerate the mining process and avoid generating many unnecessary candidates. Therefore, it outperforms the modified Apriori and BIDE algorithms.

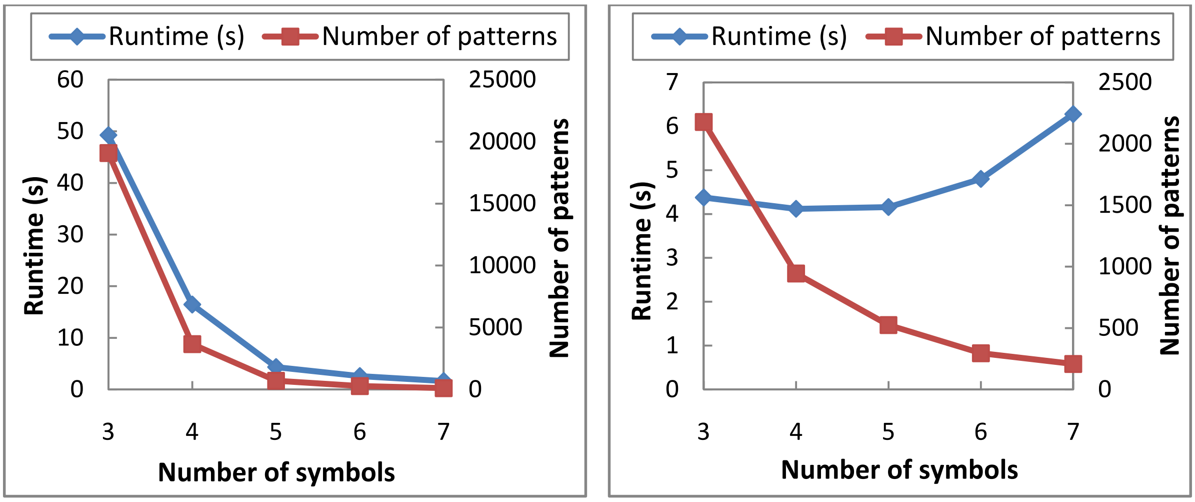


(a) (b)

Fig. 6.8. Runtime versus maximum gap: (a) weather and (b) stock.

Fig. 6.8 presents the runtime versus the maximum number of gaps for the weather and stock datasets. As the maximum number of gaps increases, more candidates or patterns are generated and therefore the runtime of all three algorithms increases. Since the runtime of the CMP algorithm increases slowly as the maximum number of gaps increases, the CMP algorithm outperforms the modified Apriori and BIDE algorithms.

In Fig. 6.9, we demonstrate how the SAX representation may affect the performance on the real datasets. We apply the CMP algorithm to the weather and stock datasets with a different number of symbols, where the minimum support threshold is 10% in the weather dataset and 30% in the stock dataset. In both real datasets, the number of closed patterns decreases as the number of symbols increases. When the number of symbols increases, the runtime decreases for the weather dataset; however, it increases for the stock dataset.



(a) (b)

Fig. 6.9. Runtime and number of closed patterns versus number of symbols: (a) weather and (b) stock.

We perceive that the trend of runtime obtained for the weather dataset is different from what we have observed from the synthetic data in which the runtime of the CMP algorithm increases as the number of symbols increases. In the weather dataset, both temperature and relative humidity are recorded hourly and the hourly fluctuations of these variables are small for most days. Thus, many values in a sequence are transformed into the same symbol in the SAX representation. As more equivalent symbols are contained in the dataset, more closed patterns but fewer frequent 1-patterns may be mined. In other words, discretizing by three symbols or more symbols does not make a big difference on the number of frequent 1-patterns generated. Thus, the runtime of the CMP algorithm depends on the number of closed patterns mined from the dataset. With smaller number of symbols, more closed patterns can be mined, and hence the CMP algorithm requires more time in the mining process.

In comparison with the weather data, the movements of stock market indexes are collected on a daily basis and have larger fluctuations. Hence, fewer closed patterns but more frequent 1-patterns are generated in the stock dataset. As more frequent 1-patterns are generated, the number of combinations of forming candidate patterns from a frequent pattern increases. This leads to more computational effort to check whether these candidate patterns are frequent and closed. Notice that the runtime in the case of three symbols is slightly greater than that in the cases of four and five symbols. In the case of three symbols, the effort on generating closed patterns dominates the total runtime. As the number of symbols increases, fewer closed patterns may be mined and the effort that dominates total runtime would shift to forming candidate patterns and checking if the candidates formed are frequent and closed. However, in the cases of four and five symbols, the effort on forming candidate patterns and checking is not yet apparent since the number of symbols only increases a little and the effort on generating closed patterns decreases because fewer closed patterns can be found. Hence, the runtime in the cases of four and five symbols may be less than in the case of three symbols. As the effort on forming candidate patterns becomes apparent in the case of six and seven symbols, the performance would be worse in comparison to the case of three symbols. Therefore, as the number of symbols increases, the runtime decreases for the weather dataset; however, it increases for the stock dataset.

Mining the weather dataset, some interesting patterns are found, such as $\left\{\begin{matrix} MH_1 & MH_1 & MH_1 & MH_1 & MH_1 \\ H_2 & H_2 & H_2 & H_2 & H_2 \end{matrix}\right\}$ and $\left\{\begin{matrix} MH_1 & MH_1 & MH_1 & MH_1 & MH_1 & \_ & H_1 & H_1 \\ H_2 & H_2 & H_2 & H_2 & H_2 & H_2 & MH_2 & MH_2 \end{matrix}\right\}$. The patterns show that when the temperature and the relative humidity are medium high or greater, it is likely to be day with high cloudiness.

In Figs. 6.10 - 6.12, we project the patterns back to the original data. Fig. 6.10 shows a continuous period of high (or medium high) temperature and high (or medium high) relative humidity from the 1st to the 10th hours, and demonstrates that warm and moist air produces clouds during this period. However, when the relative humidity starts to drop in the 11th hour and remains at a lower level afterward, the cloudiness also drops dramatically despite the temperature remaining at the high level.

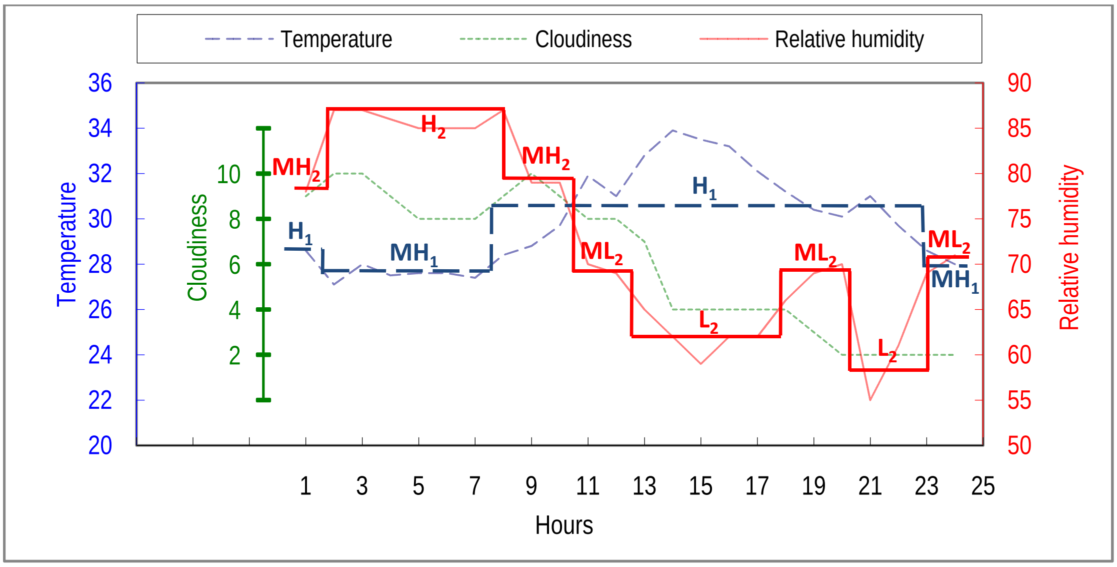


Fig. 6.10. Example I: projecting the pattern back to the raw time-series sequence.

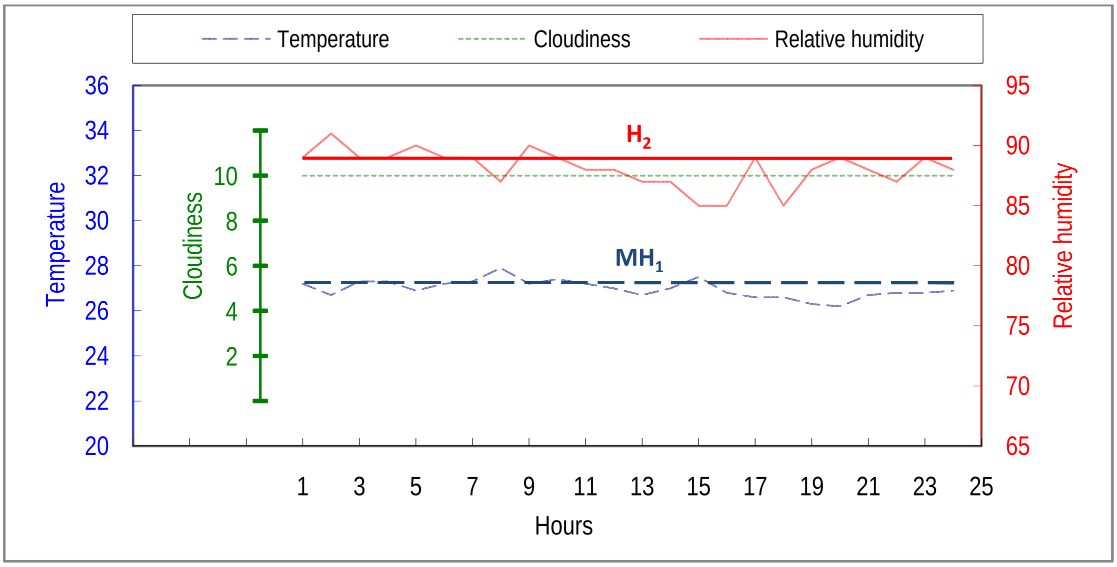


Fig. 6.11. Example II: projecting the pattern back to the raw time-series sequence.

In Fig. 6.11, the temperature remains at the medium high level and the relative humidity remains at the high level, and hence high cloudiness can be observed through the day. In Fig. 6.12, it is mostly cloudy in the early morning when the temperature and relative humidity are high (or medium high) through the first eight hours. However, the relative humidity decreases to a lower level from late morning to the afternoon while the temperature remains at a high level. Hence, a trend in decreasing cloudiness may be observed during this period. Nevertheless, from the 20th to the 24th hour, the cloudiness starts to increase as the relative humidity starts to increase and the temperature remains at a medium high level.

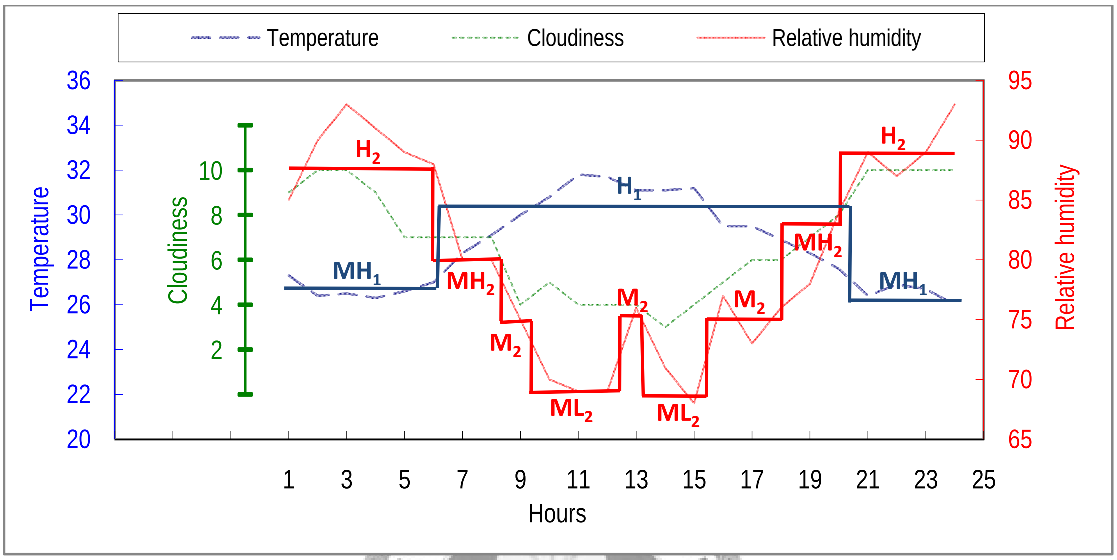


Fig. 6.12. Example III: projecting the pattern back to the raw time-series sequence.

In summary, the CMP algorithm is more efficient and scalable than the modified Apriori and BIDE algorithms. It requires much less execution time, especially when the minimum support threshold is low. This is because the CMP algorithm employs the projected database, closure checking and pruning strategies to mine closed patterns to avoid generating many unnecessary candidates. Therefore, the experimental results show that the CMP algorithm outperforms the modified Apriori and BIDE algorithms.

6.3 Performance evaluation of the CFP algorithm

To evaluate the performance of the CFP algorithm, we compare it with the modified Apriori algorithm. For the comparison reason, we adapt the Apriori algorithm [53] to mine closed flexible patterns in a time-series database. That is, we slightly modify the join step of the Apriori algorithm. Instead of combining two frequent *k*-patterns as described in [53], we join these patterns with all possible combinations of gap intervals to generate candidate (*k*+1)-patterns. Then, we scan the database once to count the support of each candidate (*k*+1)-pattern and check if it is frequent. The steps described above are repeated until no more frequent patterns can be found. To compare with the modified Apriori algorithm, we derive a complete set of frequent patterns from the closed flexible patterns found in the CFP algorithm.

6.3.1 Evaluations on synthetic data

In Fig. 6.13, we investigate the effect of varying minimum support thresholds from 2% to 12% on the runtime of the CFP and the modified Apriori algorithms. The result shows that as the minimum support threshold decreases, the runtime of both algorithms increases sharply. This is because more patterns can be mined with a smaller minimum support threshold. However, compared with the modified Apriori algorithm, the CFP algorithm has a significant performance improvement, especially when the minimum support threshold is low. This is due to the benefits of using projected databases so that the CFP algorithm can localize the pattern extension in a small number of projected databases. Moreover, using two pruning strategies and the closure checking scheme, we can eliminate many unnecessary patterns which result in a major reduction in execution time. In contrast, the modified Apriori algorithm bears more computational costs because it generates a large number of candidates and needs multiple scans of the database. Therefore, the CFP algorithm runs about 5-19 times faster than the modified Apriori algorithm.

Generally speaking, increasing the number of transactions may lead to a larger number of frequent patterns being generated. As shown in Fig. 6.14, the runtime of both algorithms increases linearly as the number of transactions increases.

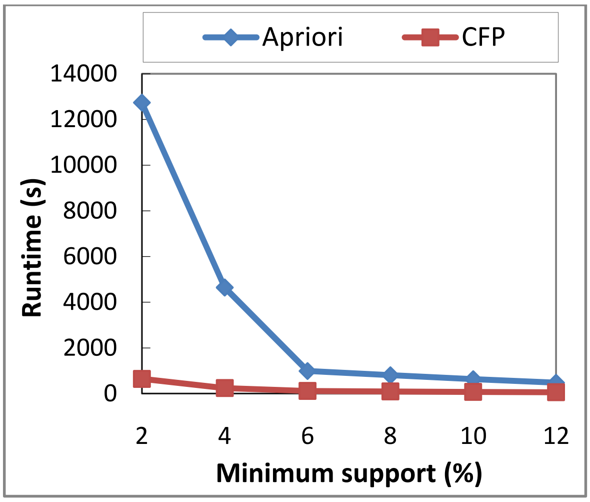


Fig. 6.13. Runtime versus minimum support (CFP).

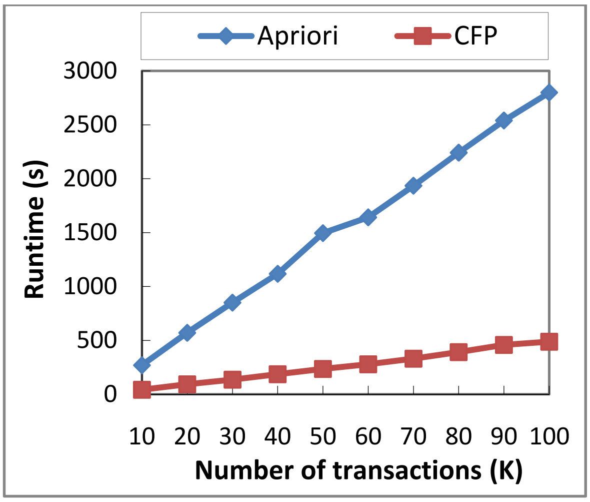


Fig. 6.14. Runtime versus number of transactions (CFP).

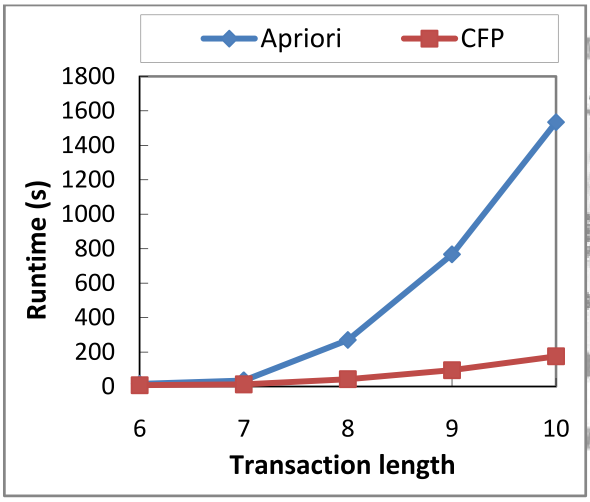


Fig. 6.15. Runtime versus transaction length (CFP).

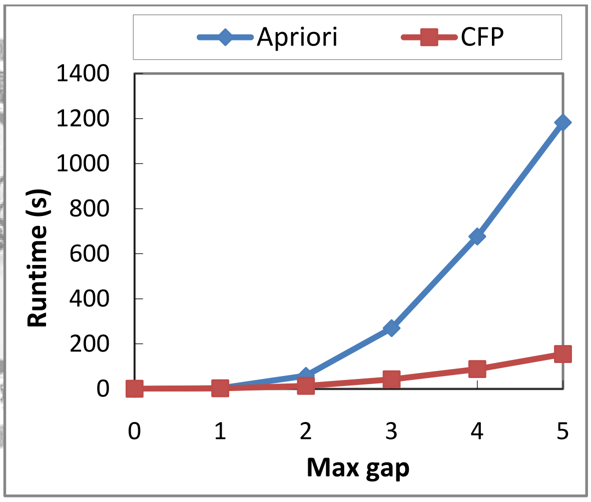


Fig. 6.16. Runtime versus maximum gap (CFP).

Fig. 6.15 illustrates the runtime versus the transaction length for both algorithms, where the transaction length varies from 6 to 10. As the transaction length increases, the runtime of both algorithms increases. The reason is that longer flexible patterns may be discovered from longer transactions and thus require more effort on the join steps and counting supports. Nevertheless, the CFP algorithm outperforms the modified Apriori algorithm in all cases.

In Fig. 6.16, we evaluate the performance on the CFP and the modified Apriori algorithms with different maximum gap thresholds. We observe that when the maximum gap threshold increases, the runtime of both algorithms increases. As the maximum gap threshold increases, the number of possible combinations of gap intervals between two successive items in a pattern also increases. This results in more frequent patterns at each level, and hence more execution time is required. However, the modified Apriori algorithm spends a lot of time on generating a large number of candidates and scanning the database to count their supports. In contrast, the CFP algorithm employs the concept of projected database and saves a lot of time on support counting. Therefore, the CFP algorithm is more efficient and scalable than the modified Apriori algorithm.

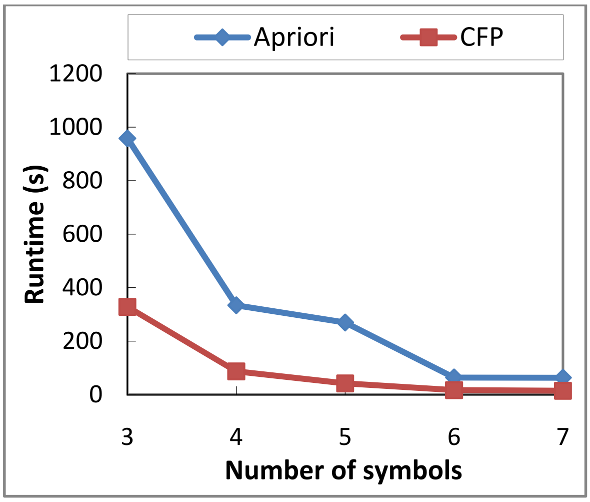


Fig. 6.17. Runtime versus number of symbols (CFP).

We further examine the effect of the number of distinct symbols on the performance of both algorithms. As shown in Fig. 6.17, both curves slope downward as the number of distinct symbols increases from 3 to 7. This is because larger number of distinct symbols may reduce the chances of forming frequent patterns. Nevertheless, the CFP algorithm results in better performance than the modified Apriori algorithm.

In summary, since the CFP algorithm employs the projected database, two pruning strategies, and the closure checking scheme to mine closed flexible patterns in a DFS manner, it is more efficient and scalable than the modified Apriori algorithm in all cases, especially when the minimum support threshold is low or the average length of transactions is large. The experimental results show that the CFP algorithm outperforms the modified Apriori algorithm by an order of magnitude.

6.3.2 Evaluations on real data

To validate the CFP algorithm on the real dataset, we perform experiments on the house price index (HPI) data of the United States. The real dataset is retrieved from Standard & Poor's official website [55] and it covers the time period from July 2005 to December 2007. This period could be characterized as a turning point in U.S. economy, generally attributed to the strong housing demand in 2005 and the subprime mortgage crisis in 2007.

The real data consists of home price indices of 20 metropolitan regions in the United States, including Boston, New York, Washington, Charlotte, Atlanta, Tampa, Miami, Detroit, Cleveland, Chicago, Minneapolis, Dallas, Denver, Phoenix, Las Vegas, Seattle, Portland, San Francisco, Los Angeles, and San Diego. Moreover, it is recorded on a monthly basis. Hence, we have 20 transactions in the database and each of which contains 30 elements. That is, there are 30 months over the period from July 2005 to December 2007. Each element in the transaction is transformed into one of the distinct symbols *HP*, *LP*, *FT*, *LN*, and *HN* in the transformation phase where these symbols denote high positive, low positive, flat, low negative, and high negative rate of change in home-price appreciation, respectively.

Fig. 6.18 shows the runtime versus the minimum support threshold which varies from 20% to 70%. The runtime of the modified Apriori algorithm grows sharply as the minimum support threshold decreases since it requires a major effort to generate a huge number of candidates and scan databases multiple times. In contrast, the runtime of the CFP algorithm increases moderately even when the minimum support threshold is low.

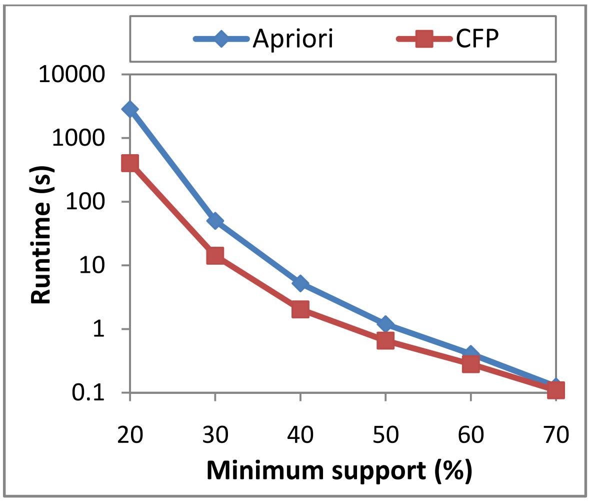


Fig. 6.18. Runtime versus minimum support for HPI data.

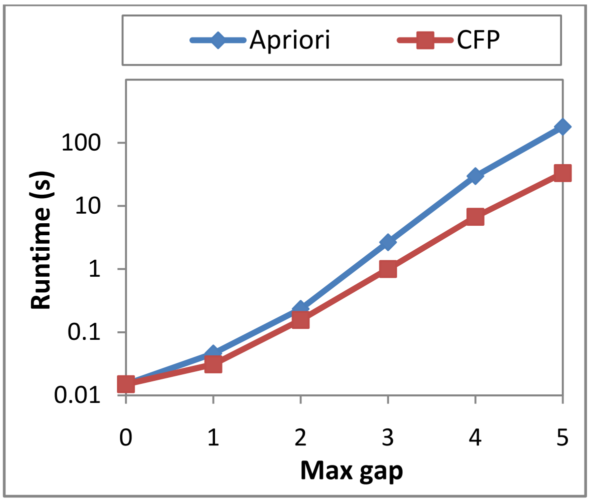


Fig. 6.19. Runtime versus maximum gap for HPI data.

Fig. 6.19 depicts the result of mining real data with respect to different values of maximum gaps, where the minimum support threshold is set to 90%. The runtime of both algorithms rises slowly with low maximum gap thresholds because there are only a limited number of patterns. However, as the value of maximum gaps increases, the performance differences between the modified Apriori algorithm and the CFP algorithm become noticeable.

Mining the real dataset, we could find some interesting patterns and project these patterns back to the original data in order to show the trend in a certain time period. For instance, one of the patterns is $\{LN\ LN\ HN\ HN\ HN\}$ as shown in Fig. 6.20. The pattern implies a continuing downward trend in the rate of house price in the wake of the subprime mortgage crisis in 2007. We obtain this pattern on West Coast cities like Las Vegas, Phoenix, Los Angeles, and San Diego and this denotes that these cities all have the same movement in house prices as the subprime mortgage crisis breaks out. The movement demonstrates that the home price starts to falter in spring 2007 and continue to have a higher negative rating in summer and winter 2007. Moreover, Las Vegas has the pattern begins in February 2007 but it is at a lag of three months for Phoenix. Los Angeles and San Diego have the pattern appeared at the same time in June

2007 that is a month later than Phoenix. In other words, we observe an earlier fall in house prices affected by the subprime mortgage crisis in Las Vegas and Phoenix than other West Coast cities.

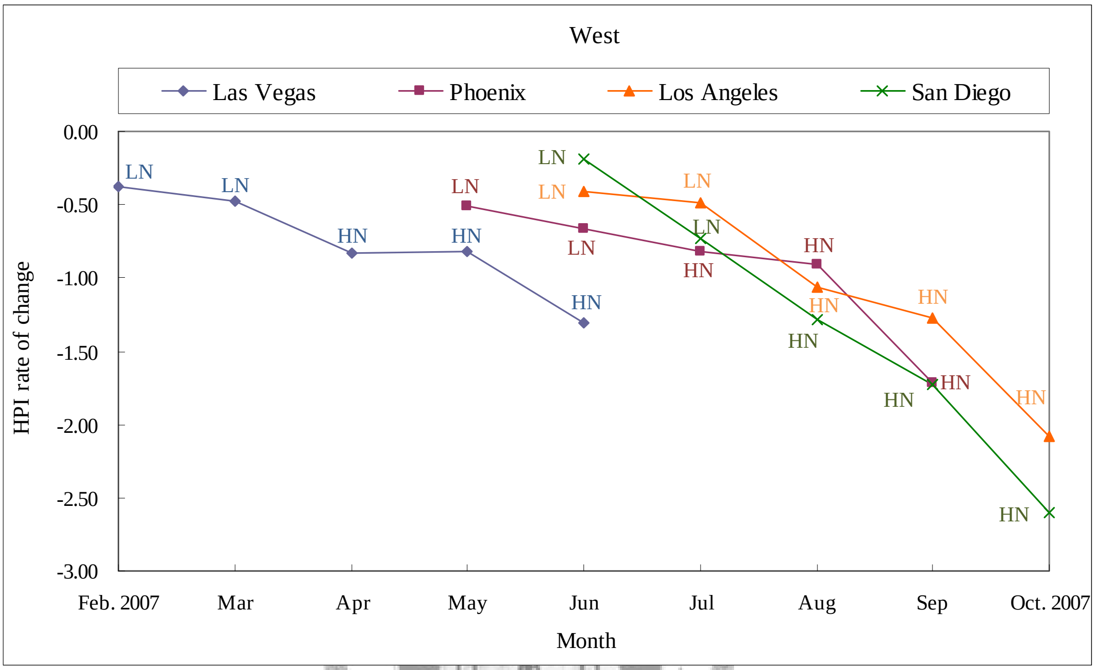


Fig. 6.20. HPI rate of change for West Coast cities.

Another pattern $\{FT\ LP\ LP\ LP\ LP\ FT\ LN\ LN\ LN\ LN\}$ is found on South cities Atlanta and Dallas as shown in Fig. 6.21. The same trend can be seen in the rate of house price movement between both cities. It indicates the small rise in house price appreciation during the period from March 2006 to August 2006 before the subprime mortgage crisis hits. However, it depicts several consecutive monthly falls in the rate of house price starting in September 2006.

The other pattern $\{FT\ FT\ LN\ [0,1]\ HN\ [0,1]\ HN\ HN\}$ is discovered on Midwest cities Minneapolis, Chicago, and Cleveland as shown in Fig. 6.22. We plot the pattern separately for these three cities in order to provide a clear view on the part of the flexible intervals. We observe that all three cities have the same movement $\{FT\ FT\ LN\}$ from June to August and then within 0-1 gap interval, the HPI rate of change would drop to high negative. Furthermore, the second gap interval [0, 1] between *HN* and *HN* indicates that within 0-1 gap interval, there is less fluctuation in house price movement and the change remains in a high negative rate.

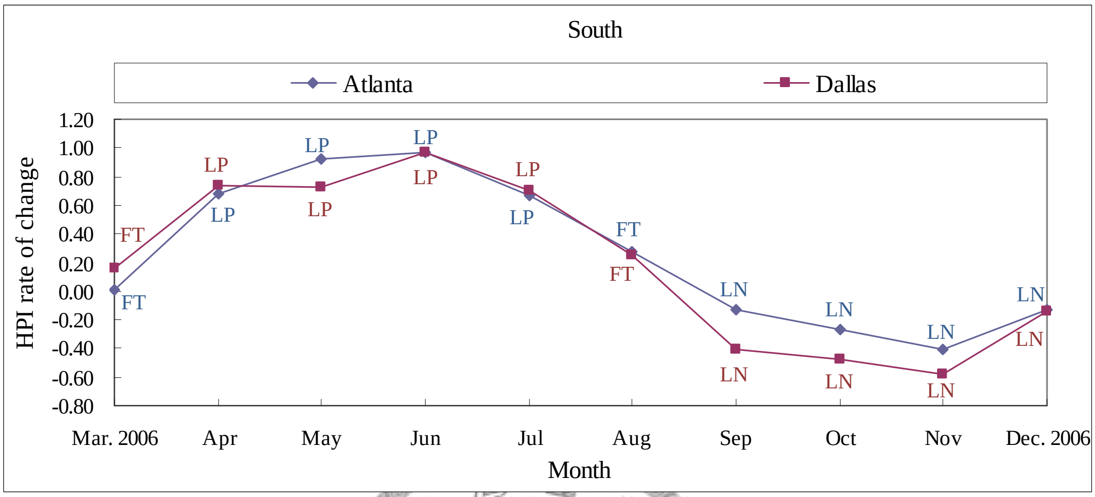


Fig. 6.21. HPI rate of change for South cities.

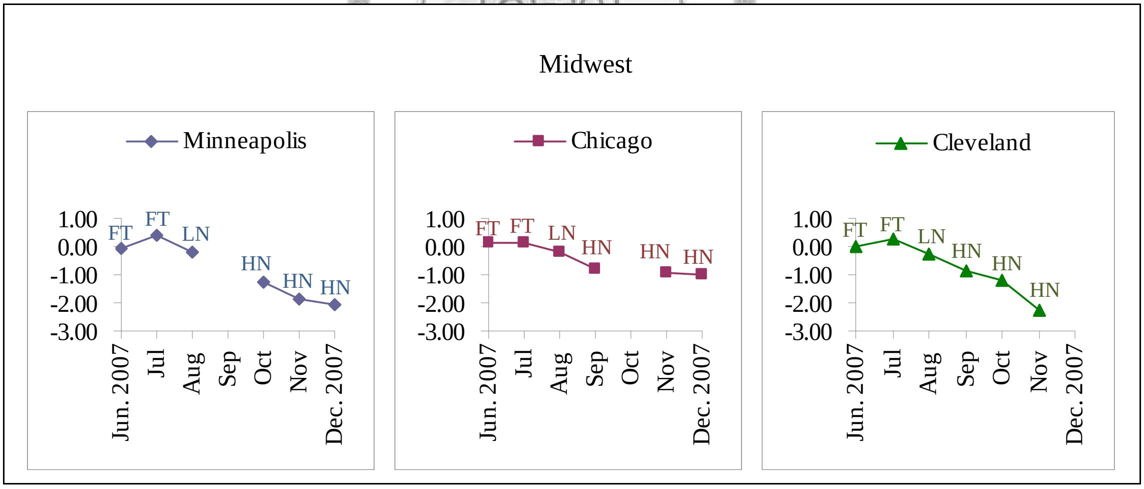


Fig. 6.22. HPI rate of change for Midwest cities.

In summary, the performance of the CFP and the modified Apriori algorithms on the real dataset is quite similar to that on the synthetic datasets. Since the CFP algorithm takes the advantage of two pruning strategies and the closure checking scheme to reduce the search space and thus speed up the algorithm, it outperforms the modified Apriori algorithm.

6.4 Performance evaluation of the CNP algorithm

The performance of the CNP algorithm is evaluated against the A-Close algorithm [43]. For the comparison, the A-Close algorithm is modified appropriately to mine frequent closed patterns in time-series databases. Specifically, we scan database once to mine all frequent 1-patterns. Next, we join two frequent *k*-patterns as described in [43] to form a candidate (*k*+1)-pattern. However, we need to check if this candidate pattern is indeed a valid pattern existed in the database. This is because the candidate pattern must be formed of a set of consecutive time points, whereas the original A-Close algorithm does not take time into account when joining the candidate patterns. If the candidate pattern is valid and its support is not less than δ, it is a frequent (*k*+1)-pattern. According to the pruning strategy in [43], we remove any frequent (*k*+1)-pattern if its support is the same as that of its sub-patterns. This process is repeated until no more patterns can be generated.

To obtain all closed patterns, we compute the closures of all frequent patterns found earlier in the database. That is, we grow each frequent pattern *p* until its super-pattern *q* has a different support count from *p*. If such super-pattern *q* can be found, we store *q* in the closed pattern pool; otherwise, we store *p* in the closed pattern pool. Note that, before inserting a new pattern in the closed pattern pool, we use the already mined patterns in the pool to double-check whether the new pattern is indeed closed. Finally, we merge similar patterns together and select the representative patterns.

6.4.1 Evaluations on synthetic data

In Fig. 6.23, we report the runtime performance of the CNP and the modified A-Close algorithms, where the minimum support threshold varies from 0.001% to 0.5%. As observed, the CNP algorithm achieves better efficiency than the modified A-Close algorithm, especially when the minimum support threshold drops to 0.001%; their performances differ by a factor of about 100. This is because the modified A-Close algorithm suffers from generating a huge number of candidates and computing the closures of the frequent patterns. On the other hand, the CNP algorithm takes the advantage of projected databases to localize the pattern extension in small projected databases and applies both pre-pruning and post-pruning strategies to stop growing unnecessary patterns and thus results in much better performance.

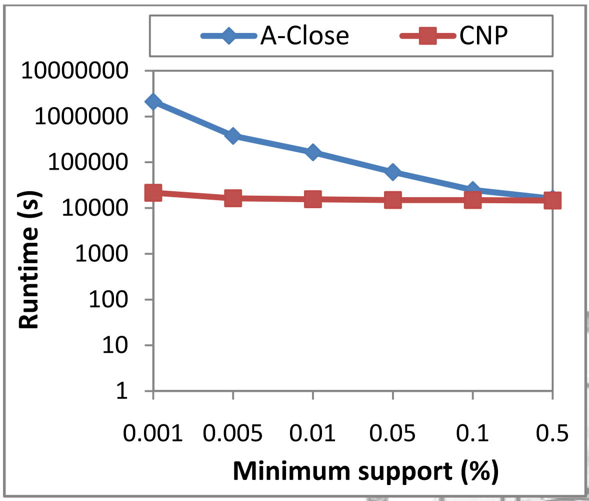


Fig. 6.23. Runtime versus minimum support (CNP).

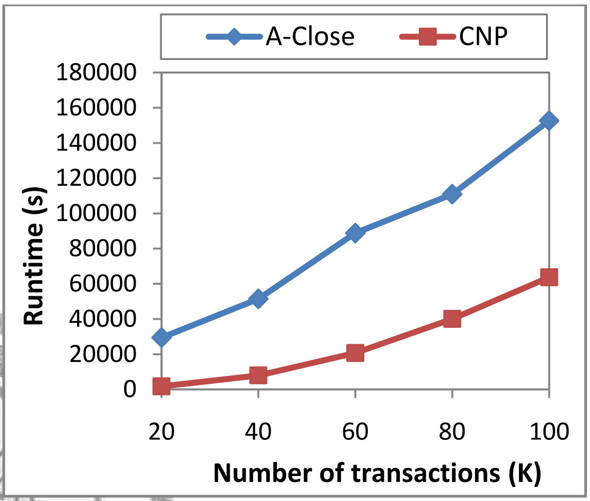


Fig. 6.24. Runtime versus number of transactions (CNP).

Fig. 6.24 illustrates the scalability of the CNP and the modified A-Close algorithms, where the number of transactions varies from 20,000 to 100,000. The runtime of both algorithms increases linearly as the number of transactions increases. This is because more closed patterns can be mined with the increasing number of transactions. The modified A-Close algorithm thus generates more candidates and requires more effort on the join steps, support counting, and closure computation, whereas the CNP algorithm reduces much search space to mine patterns by employing projected databases. However, the CNP algorithm performs consistently better than the modified A-Close algorithm in all cases.

Fig. 6.25 depicts the runtime versus the transaction length for both algorithms, where the transaction length varies from 4 to 64. Generally, as the transaction length increases, the number of closed patterns increases, and hence the runtime increases. We see that the runtime of the modified A-Close algorithm increases faster than that of the CNP algorithm, especially when the transaction length increases to 64. It is clear that the modified A-Close algorithm is not effective in dealing with long transactions because more combinations of candidate super-patterns are considered in the join step and it needs to scan the database multiple times to count the supports. It also spends a lot of time on the closure computation to find closed patterns. Nevertheless, the CNP algorithm outperforms the modified A-Close algorithm significantly since it uses the projected databases to determine the supports of the patterns and adopts two pruning strategies to accelerate the mining process for longer transactions.

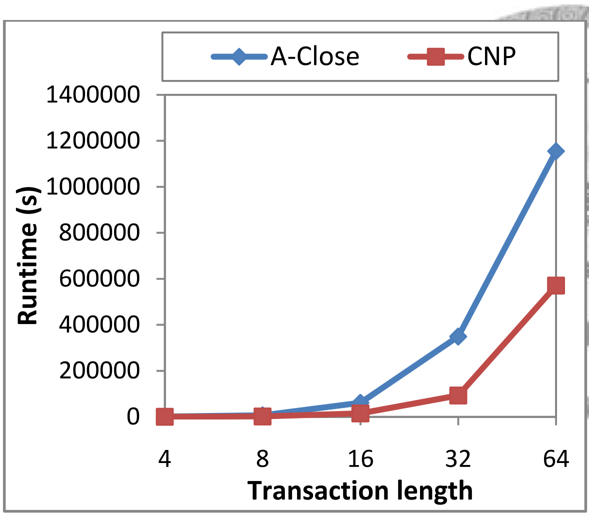


Fig. 6.25. Runtime versus transaction length (CNP).

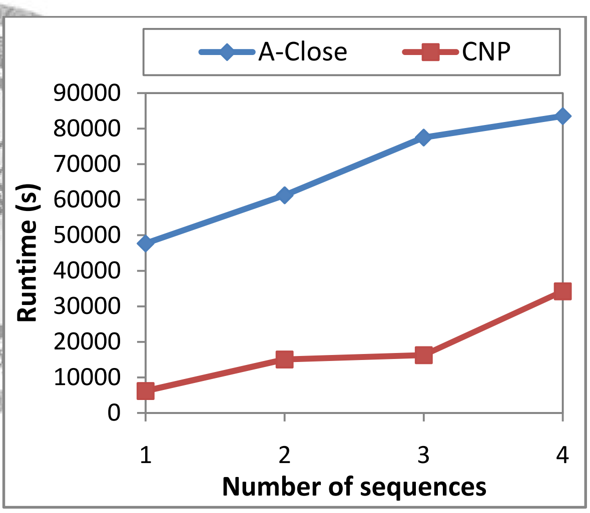


Fig. 6.26. Runtime versus number of sequences (CNP).

In Fig. 6.26, we examine the relationship between the runtime and the number of sequences in a transaction for the CNP and the modified A-Close algorithms. The runtime of both algorithms increases as the number of sequences in a transaction increases. This is because more computation is required to check if a pair of multi-sequences is similar. However, the CNP algorithm performs better than the modified A-Close algorithm.

In Fig. 6.27, we evaluate the performance on the CNP and the modified A-Close algorithms with different distance thresholds (ε). When the distance threshold increases, the runtime of both algorithms also increases. This is because more closed patterns can be generated for the larger distance threshold. However, the CNP algorithm runs about 2-17 times faster than the modified A-Close algorithm. It is due to that our pruning strategies avoid generating unnecessary patterns and the projected databases can be used to reduce the search space of finding similar patterns, which speed up the mining process.

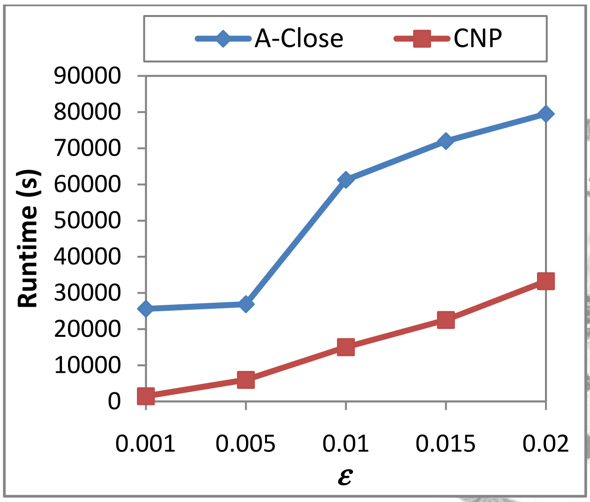


Fig. 6.27. Runtime versus distance threshold (CNP).

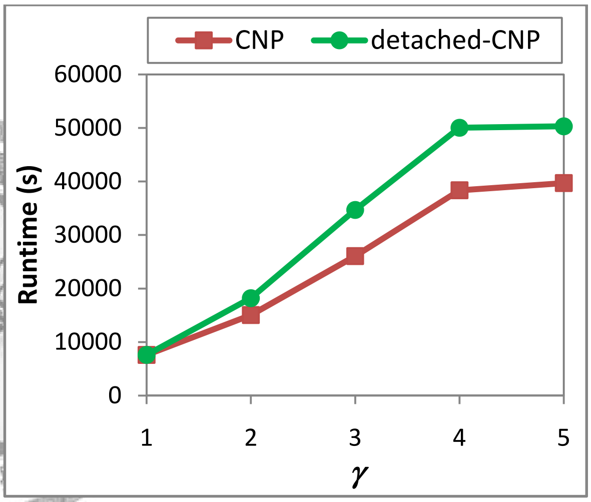


Fig. 6.28. Runtime versus number of resolutions (CNP).

Fig. 6.28 demonstrates the effect of the number of resolutions (γ) on the runtime of the CNP and the detached-CNP algorithms, where the detached-CNP algorithm contains no restoring operation to restore patterns from lower resolutions to higher resolutions; instead, it mines patterns from different resolution levels separately. As the number of resolutions increases, the runtime of both algorithms also increases. The major cost of the CNP algorithm is to grow closed patterns in the low resolution and restore them to the high resolution in the mining process. The more the Haar wavelet transforms are applied, the smoother the numerical sequences are and they have a better chance to be similar. The CNP algorithm thus produces more closed patterns in the low resolution and requires more effort on the restoring operations. On the other hand, the cost of the detached-CNP algorithm is to mine patterns from different resolution levels. As the level of resolution is lower, more closed patterns can be mined and more runtime is required, whereas fewer patterns are generated with higher level of resolutions and less runtime is required. However, the CNP algorithm outperforms the detached-CNP algorithm by 20% on average. Notice that the runtime increases moderately when γ varies from 4 to 5. This is because fewer frequent patterns can be found in the low resolution as the length of the transformed time-series is short. That is, when γ= 4, the length of each transformed time-series becomes 2, whereas the length of each transformed time-series becomes 1 when γ= 5. In both cases, the time required to find frequent patterns in the low resolution is not that different from each other. Therefore, the overall runtime slightly increases when γvaries from 4 to 5.

In summary, the experimental results show that the CNP algorithm is scalable and efficient on all the parameter settings. The CNP algorithm spends much less runtime because it uses projected databases to localize the pattern extension and support counting, and employs two pruning strategies to prune the unnecessary extensions. In contrast, the modified A-Close algorithm requires multiple scans of the database and spends much time on computing the closures of the frequent patterns. Therefore, the CNP algorithm significantly outperforms the modified A-Close algorithm.

6.4.2 Evaluations on real data

To further evaluate the performance of the CNP algorithm on real-world datasets, we collect a stock dataset from Yahoo Finance [65] which provides daily closing prices for S&P (Standard and Poor's) 500 companies [55]. The data is collected from the period of Jan 1st 2009 to July 7th 2009. There are 128 trading days in total. Moreover, this period is characterized as the turnover point creeping up from the recession to the recovery period.

Next, we apply the 50-day short-term moving average (MA50) and 200-day long-term moving average (MA200) to the time-series sequences of each company in the collected stock dataset. Generally, there are 500 companies in S&P but five of them are removed due to incomplete data. As a result, there are 495 transactions in the database and each of which contains two sequences of length 128. Since the stock price scale is different for each of the S&P 500 companies, we normalize the dataset by the min-max normalization [19].

The performance of the CNP and the modified A-Close algorithms on the real dataset is similar to that on the synthetic datasets. Fig. 6.29 shows the runtime versus the minimum support threshold. For smaller minimum support thresholds, the runtime for mining closed patterns becomes significant for both algorithms. When the minimum support threshold is larger than 60%, the runtime of both algorithms remains almost constant. This is because no pattern can be discovered after the minimum support threshold exceeds 60%. Nevertheless, an exception can be noticed for the case where the performance of the CNP algorithm degrades a little when the minimum support threshold increases from 95% to 100%. This is caused by how we generate frequent 1-patterns. Recall that we use the lower and upper bounds of the indices in an array to find candidates that are similar to a pivot with respect to the first dimension. For each candidate, we further check if it is similar to the pivot with respect to the second dimension. When the minimum support threshold is 95%, most pivots are potential to be frequent 1-patterns since there are sufficient candidates that are similar to the pivots with respect to the first dimension. Thus, additional runtime is required to check whether sufficient candidates are similar to those pivots with respect to the second dimension. On the other hand, when the minimum support threshold is 100%, most pivots cannot find sufficient candidates with respect to the first dimension, and hence it is unnecessary to check the second dimension and no additional runtime is required.

However, the performance of the CNP algorithm is always better than that of the modified A-Close algorithm because the latter consumes much time in generating large number of candidates and calculating the supports and closures of the patterns.

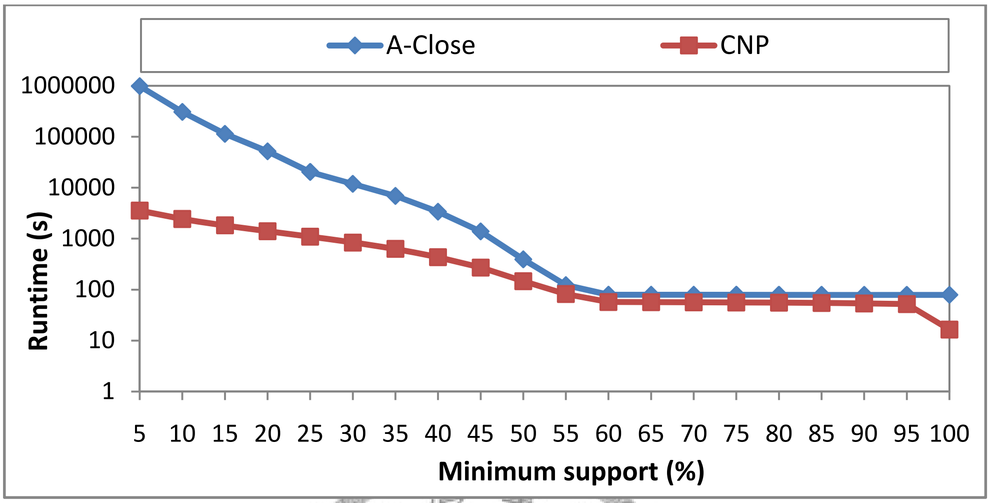


Fig. 6.29. Runtime versus minimum support for S&P 500 stock.

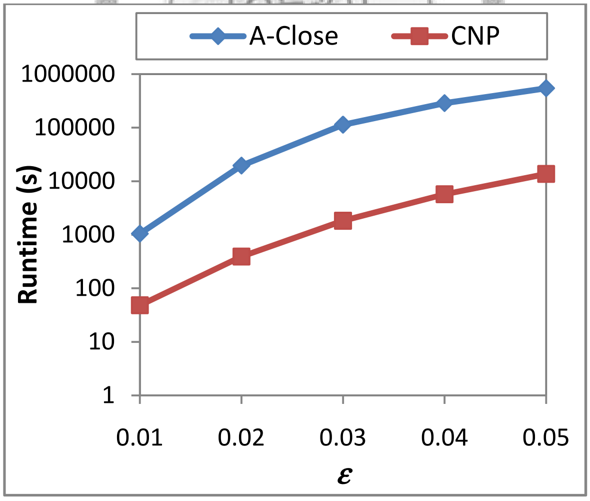
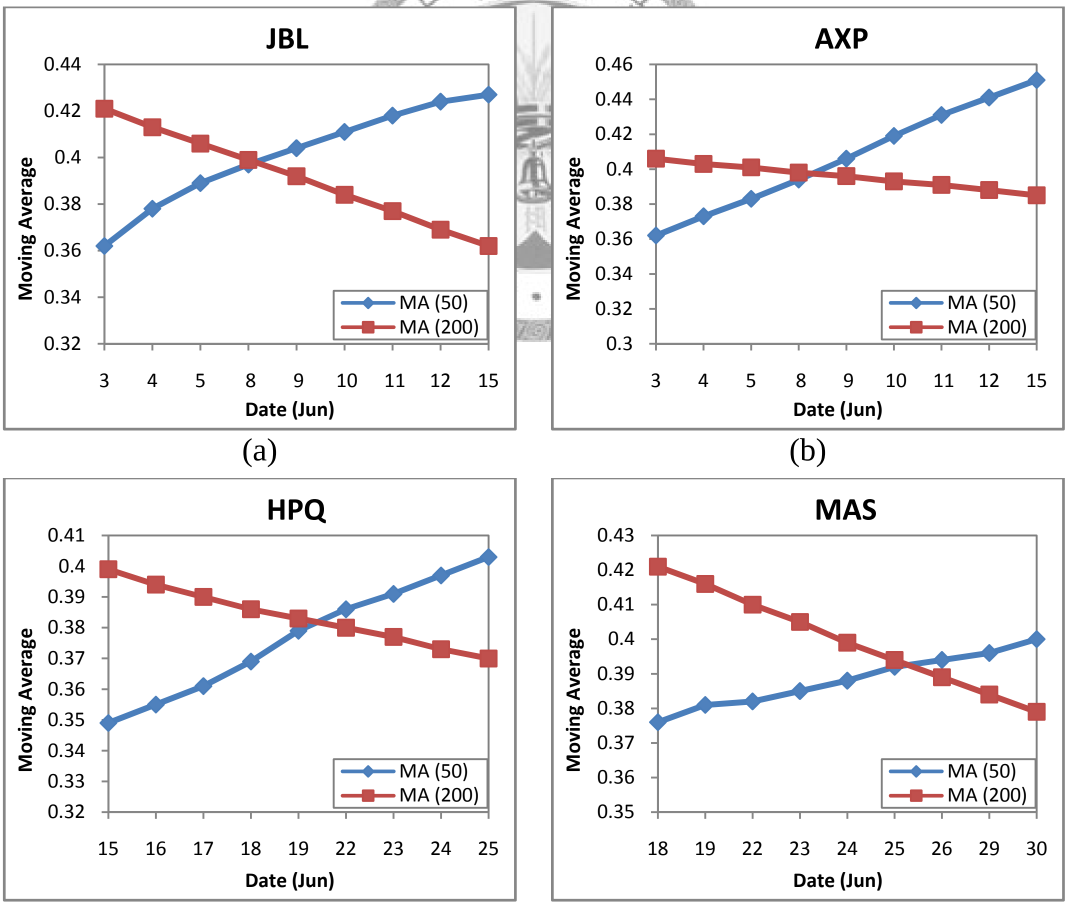


Fig. 6.30. Runtime versus distance threshold for S&P 500 stock.

Fig. 6.30 illustrates the runtime versus the distance threshold. We observe a significant performance improvement as the distance threshold drops to 0.01. With smaller distance threshold, more closed patterns fail to satisfy the similarity constraint and thus fewer closed patterns can be mined from the data. However, the CNP algorithm consistently outperforms the modified A-Close algorithm in all cases.

Mining the stock dataset, we could find some interesting patterns. For example, we find a pattern that represents a golden cross in Fig. 6.31a, where a golden cross is a signal of bullish markets and occurs when the line of the short-term moving average passes through the long-term moving average from the lower side to the upper side [12][56]. There are 26 companies in the projected database having the golden cross pattern and they are categorized in the following sectors according to S&P official website [55]: information technology (7), financial (5), industrials (5), materials (3), consumer discretionary (3), healthcare (2), and consumer staples (1). Note that the number in the bracket after a sector name denotes the number of companies found in the sector.



(a) (b)

(c) (d)

Fig. 6.31. The golden crosses of JBL, AXP, HPQ, and MAS.

The pattern shows a continuing upward trend in stock price in the recovery period and implies that numerous companies gradually revive as the global economy progressively recovers from the recession. Moreover, we clearly see that the golden cross pattern is taken place in June. We further check the patterns in the projected database and find that the golden crosses for those patterns shift slightly. Here we show some of those patterns in Figs. 6.31a, 6.31b, 6.31c, and 6.31d which represent the companies JBL, AXP, HPQ, and MAS, respectively. These shifts are caused by the tolerance of distance threshold. Since the tolerance of distance threshold may cause a pair of multi-sequences to be similar but a slightly different, the golden cross may occur at a different point. In addition, we shift the patterns of these companies to the same time interval to show that they are indeed similar to each other as illustrated in Fig. 6.32. Moreover, we verify that each stock of these companies has a positive outlook since its golden cross occurs. Interestingly, this phenomenon has confirmed that a golden cross is indeed a buy point for traders. Besides, the result demonstrates that the sectors of information technology, financial, and industrials are likely to revive faster than other sectors.

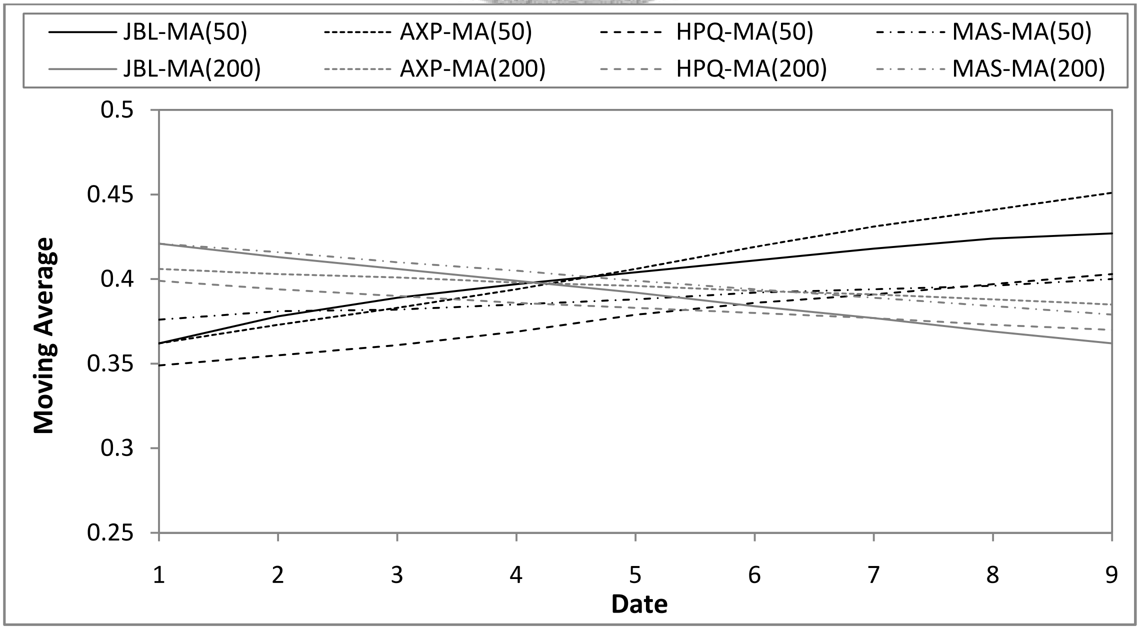
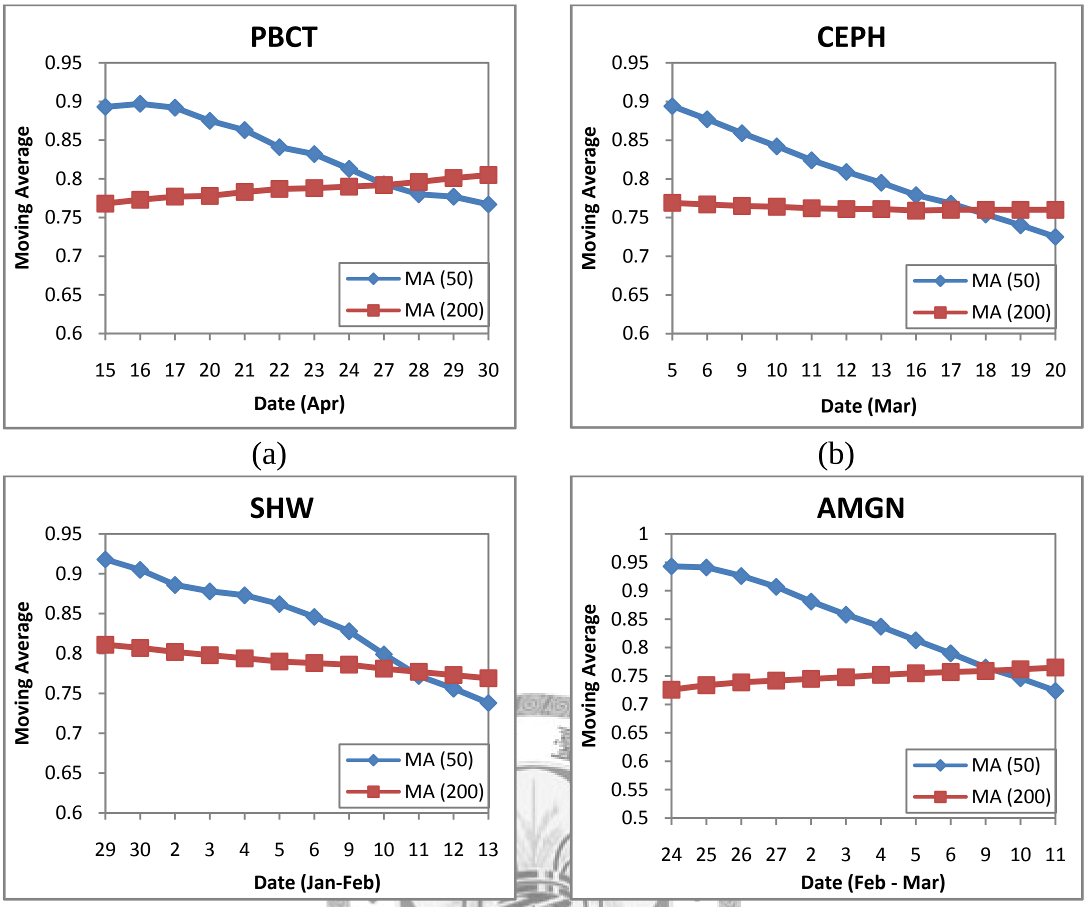


Fig. 6.32. The golden crosses of JBL, AXP, HPQ, and MAS in the same time interval.



(a) (b)

(c) (d)

Fig. 6.33. The death crosses of PBCT, CEPH, SHW, and AMGN.

Another interesting pattern found is shown in Fig. 6.33a. It is a death cross which is a signal of bearish markets and occurs when the line of long-term moving average passes through the short-term moving average from the lower side to the upper side [12][56]. The pattern reflects that there are numerous companies suffering hard times during the recession and they have not yet recovered. The pattern can be projected onto the companies, such as PBCT, CEPH, SHW, and AMGN as shown in Figs. 6.33a, 6.33b, 6.33c, and 6.33d, respectively. The occurrences of the death crosses for these companies spread out from January to April. This signifies that many stocks have suffered depreciation at different time points during the recession. These four companies can be categorized into the sectors of financial, healthcare, and consumer discretionary. Moreover, we shift the patterns of these companies to the same time interval to show that they are indeed similar to each other as illustrated in Fig. 6.34.

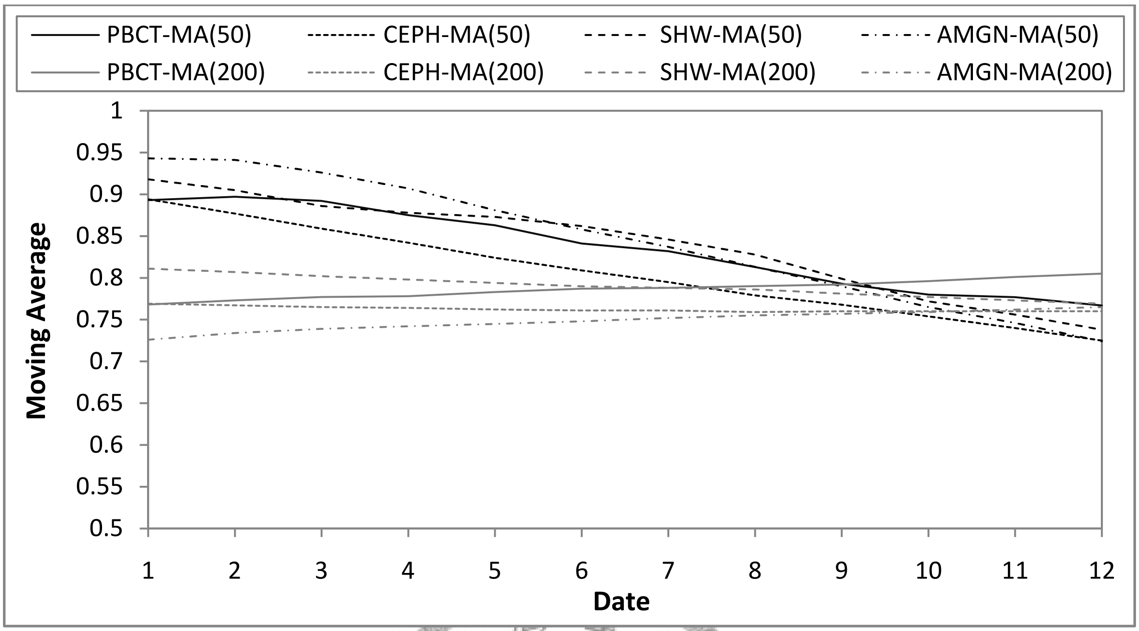


Fig. 6.34. The death crosses of PBCT, CEPH, SHW, and AMGN in the same time interval.

In summary, since the CNP algorithm uses the projected databases to grow patterns and compute the supports, and applies two pruning strategies to stop growing the unnecessary patterns, it achieves much better performance in comparison to the modified A-Close algorithm in all cases.

Chapter 7 Conclusions and Future Work

In the dissertation, we have proposed three algorithms. The first algorithm, called CMP, is designed to mine closed patterns from a time-series database, where each transaction contains multiple time-series sequences. The second algorithm, called CFP, integrates the idea of allowing flexible gaps between items in a pattern and is capable of discovering closed flexible patterns in a time-series database. The third algorithm, called CNP, mines multi-resolution closed patterns directly from raw numerical data without any transformation from numerical sequences to symbolic sequences as performed in the CMP and CFP algorithms. It aims at providing analyzers different views on data by various resolutions.

The CMP algorithm consists of three phases. First, we transform each time-series sequence into a symbolic sequence. Second, we scan the transformed database to find all frequent 1-patterns, and build a projected database for each frequent 1-pattern. Third, we recursively use a frequent *k*-pattern and its projected database to generate its frequent super-patterns at the next level in the frequent pattern tree, where $k \geq 1$. The CMP algorithm adopts a DFS manner and uses the projected database to localize support counting, candidate pruning, and closure checking. Therefore, it can efficiently mine closed patterns. The experimental results show that the CMP algorithm outperforms the modified Apriori and BIDE algorithms by one or two orders of magnitude.

The CFP algorithm takes into consideration the flexible gaps between items in a pattern to mine closed flexible patterns in a time-series database. The problem of mining closed flexible patterns is solved by initially transforming a time-series database into a symbolic database, and then identifying frequent 1-patterns within the transformed database, and recursively mining closed flexible patterns in a DFS manner. Since the CFP algorithm localizes the pattern extension in a small number of projected databases and eliminates unnecessary patterns by two pruning strategies and the closure checking scheme, it is more efficient and scalable than the modified Apriori algorithm. The experimental results show that the CFP algorithm outperforms the modified Apriori algorithm by an order of magnitude.

Transforming time-series databases into symbolic databases may change the context in which the values may be seen and some valuable information may still be uncovered. Moreover, a notable disadvantage for the symbolic sequence analysis is that the number of symbols and breakpoints must be supplied. Therefore, the CNP algorithm is designed to address with these issues. It mines multi-resolution closed numerical patterns in a multi-sequence time-series database. Initially, the Haar wavelet transform is applied to convert each time-series in the database into a sequence in the low resolution, and then all frequent 1-patterns are identified from the transformed database. Subsequently, each frequent *k*-pattern is recursively extended to find frequent super-patterns in a DFS manner. For each frequent *k*-pattern found in the low resolution, it is restored back to the high resolution. As a result, all closed numerical patterns can be mined in the low and high resolutions. Since the CNP algorithm employs the projected database to localize the pattern growth and takes the benefits of pruning strategies to speed up the mining process, the CNP algorithm has demonstrated a significant runtime improvement in comparison to the modified A-Close algorithm.

At present, our work on the CMP algorithm has been published on Data and Knowledge Engineering Journal [32] and the work on the CFP algorithm has been published on Expert Systems with Applications Journal [64]. The significant contribution of this dissertation is that we have designed three efficient algorithms which are able to solve real-world problems. Specifically, we have presented a novel concept of mining closed multi-sequence patterns in a time-series database and designed the CMP algorithm to mine closed multi-sequence patterns. Moreover, we have removed the limitation of exact sequence alignments and incorporated the idea of flexible-range of consecutive gaps to discover patterns. We have proposed the CFP algorithm to mine closed flexible patterns in a time-series database. In addition, we have used the Haar wavelet transform to view a time-series database in multiple resolutions and designed a novel algorithm, CNP, to mine closed numerical multi-sequence patterns in a time-series database. We have devised effective closure checking schemes and pruning strategies with respect to each proposed algorithm to avoid generating redundant candidates, and hence each results in less execution time. All the proposed algorithms are evaluated with both synthetic and real datasets. The experimental results show that the CMP algorithm outperforms the modified Apriori and BIDE algorithms by one or two orders of magnitude; the CFP algorithm outperforms the modified Apriori algorithm by an order of magnitude; and the CNP algorithm outperforms the modified A-Close algorithm by one or two orders of magnitude.

The limitations of the proposed algorithms are addressed as follows. First, since we use an existing data discretization method, such as SAX representation, to transform time-series sequences into symbolic sequences in the initial phase of the CMP and CFP algorithms, we may not know whether the breakpoints are best determined and what is the effect of the discretization. Second, the CNP algorithm cannot cope with multiple sequences that have different lengths or missing values in a transaction since no gap symbol is taken into consideration. Finally, the CNP algorithm is unable to mine closed patterns in a database that contains non-aligned time-series because the concept of gap symbols is not integrated in the mining process.

The present dissertation has illustrated that the CMP, CFP, and CNP algorithms can efficiently mine closed patterns from both synthetic and real-world datasets; however, in the future, subsequent studies can be conducted in the following directions:

1. We may modify the CMP algorithm to mine closed patterns in one-sequence databases.
2. We may combine the essence of the CMP and CFP algorithms and develop a novel algorithm to address the problem of mining closed flexible patterns in

multi-sequence time-series databases.

3. We may allow a user-specified gap interval, instead of a user-specified maximum gap threshold, in the CFP algorithm to find specific patterns in which a user is interested.
4. It is worth extending the CNP algorithm further to mine closed patterns with some complicated constraints. For instance, a gap constraint may be pushed into the mining process.
5. We may modify the CNP algorithm to mine multi-resolution closed numerical patterns in one-sequence databases.
6. In the CNP algorithm, instead of measuring whether a pair of multi-sequences is similar by calculating the distance between each two numerical points in the sequences, other mechanisms may be adopted to improve the efficiency of this task.
7. The CNP algorithm uses the already mined patterns to check if a newly found pattern is closed. This is a time-consuming task; therefore, it is helpful to design effective closure checking schemes in order to speed up the algorithm.
8. Without generalization, too many patterns may be mined and they may be too detailed. By generalizing with a concept hierarchy, we may be able to obtain patterns that are more abstract and meaningful.
9. We have implemented the memory-based algorithms for the CMP, CFP, and CNP algorithms. It will be worth further study on implementing the disk-based algorithms for a very large database.
10. We may further apply the proposed methods to analyze other real-world applications, such as bioinformatics, medical diagnosis, hurricane forecasts, etc.

References

[1] R. Agrawal, K. Lin, H. S. Sawhney, K. Shim, Fast similarity search in the presence of noise, scaling, and translation in time-series databases, in: Proceedings of the 21th International Conference on Very Large Data Bases, 1995, pp. 490-501.

[2] C. D. Ahrens, Meteorology today: an introduction to weather, climate, and the environment (8th ed.), Thomson Brooks/Cole, Belmont, 2007.

[3] D. Alter, Liver-function testing, MLO: Medical Laboratory Observer 40 (12) (2008) 10-17.

[4] J. Ayres, J. Gehrke, T. Yiu, J. Flannick, Sequential pattern mining using a bitmap representation, in: Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining, 2002, pp. 429-435.

[5] BBC News, <http://news.bbc.co.uk/2/hi/business/7073131.stm>.

[6] D. J. Berndt, J. Clifford, Finding patterns in time series: a dynamic programming approach, Advances in Knowledge Discovery and Data Mining (1st ed.), American Association for Artificial Intelligence, 1996, pp. 229-248.

[7] Central Weather Bureau, <http://www.cwb.gov.tw/>.

[8] L. Chang, T. Wang, D. Yang, H. Luan, SeqStream: mining closed sequential patterns over stream sliding windows, in: Proceedings of the 8th IEEE International Conference on Data Mining, 2008, pp. 83-92.

[9] H. Chen, W. Chung, J. J. Xu, G. Wang, Y. Qin, M. Chau, Crime data mining: a general framework and some examples, IEEE Computer 37 (4) (2004) 50-56.

[10] T. S. Chen, S. C. Hsu, Mining frequent tree-like patterns in large datasets, Data and Knowledge Engineering 62 (1) (2007) 65-83.

[11] Y. L. Chen, T. C. K. Huang, A novel knowledge discovering model for mining fuzzy multi-level sequential patterns in sequence databases, Data and Knowledge Engineering 66 (3) (2008) 349-367.

[12] Y. Chen, S. Mabu, K. Shimada, and K. Hirasawa, A genetic network programming

with learning approach for enhanced stock trading model, Expert Systems with Applications 36 (10) (2009) 12537-12546.

[13] C. J. Chu, V. S. Tseng, T. Liang, Efficient mining of temporal emerging itemsets from data streams, Expert Systems with Applications 36 (1) (2009) 885-893.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to algorithms (2nd ed.), The MIT Press, Cambridge, 2003.

[15] G. Das, K. Lin, H. Mannila, Rule discovery from time series, in: Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining, 1998, pp. 16-22.

[16] Data Bank for Atmospheric Research, <http://dbar.as.ntu.edu.tw/>.

[17] C. Faloutsos, M. Ranganathan, Y. Manolopoulos, Fast subsequence matching in time-series databases, ACM SIGMOD Record 23 (2) (1994) 419-429.

[18] J. Han, G. Dong, Y. Yin, Efficient mining of partial periodic patterns in time series database, in: Proceedings of the 15th International Conference on Data Engineering, 1999, pp. 106-115.

[19] J. Han, M. Kamber, Data mining: concepts and techniques (2nd ed.), Morgan Kaufmann, San Francisco, 2006.

[20] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, M. Hsu, FreeSpan: frequent pattern-projected sequential pattern mining, in: Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2000, pp. 355-359.

[21] J. Han, J. Wang, Y. Lu, P. Tzvetkov, Mining top-k frequent closed patterns without minimum support, in: Proceedings of the 2002 IEEE International Conference on Data Mining, 2002, pp. 211-218.

[22] J. W. Huang, C. Y. Tseng, J. C. Ou, M. S. Chen, A general model for sequential pattern mining with a progressive database, IEEE Transactions on Knowledge and Data Engineering 20 (9) (2008) 1153-1167.

[23] Y. Huang, L. Zhang, P. Zhang, A framework for mining sequential patterns from spatio-temporal event data sets, IEEE Transactions on Knowledge and Data Engineering 20 (4) (2008) 433-448.

[24] L. Ji, K. L. Tan, K. H. Tung, Compressed hierarchical mining of frequent closed patterns from dense data sets, IEEE Transactions on Knowledge and Data Engineering 19 (9) (2007) 1175-1187.

[25] E. Keogh, Fast similarity search in the presence of longitudinal scaling in time series database, in: Proceedings of the Ninth International Conference on Tools with Artificial Intelligence, 1997, pp. 578-584.

[26] E. Keogh, S. Kasetty, On the need for time series data mining benchmarks: a survey and empirical demonstration, in: Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2002, pp. 102-111.

[27] C. Kim, J. Lim, R. T. Ng, K. Shim, SQUIRE: sequential pattern mining with quantities, The Journal of Systems and Software 80 (10) (2007) 1726-1745.

[28] M. Kontaki, A. N. Papadopoulos, Y. Manolopoulos, Adaptive similarity search in streaming time series with sliding windows, Data and Knowledge Engineering 63 (2) (2007) 478-502.

[29] R. J. Larsen, M. L. Marx, An introduction to mathematical statistics and its applications (3rd ed.), Prentice Hall, New Jersey, 2001.

[30] A. J. T. Lee, C. S. Wang, W. Y. Wang, Y. A. C, H. W. Wu, An efficient algorithm for mining closed inter-transaction itemsets, Data and Knowledge Engineering 66 (1) (2008) 68-91.

[31] A. J. T. Lee, Y. T. Wang, Efficient data mining for calling path patterns in GSM networks, Information Systems 28 (8) (2003) 929-948.

[32] A. J. T. Lee, H. W. Wu, T. Y. Lee, Y. H. Liu, K. T. Chen, Mining closed patterns in multi-sequence time-series databases, Data and Knowledge Engineering 68 (10)

(2009) 1071-1090.

[33] C. H. L. Lee, A. Liu, W. S. Chen, Pattern discovery of fuzzy time-series for financial prediction, IEEE Transactions on Knowledge and Data Engineering 18 (5) (2006) 613-625.

[34] T. H. Lee, R. Kim, J. T. Benson, T. M. Therneau, L. J. Melton III, Serum aminotransferase activity and mortality risk in a United States community, Hepatology 47 (3) (2008) 880-887.

[35] Y. S. Lee, S. J. Yen, Incremental and interactive mining of web traversal patterns, Information Sciences 178 (2) (2008) 287-306.

[36] H. F. Li, C. C. Ho, S. Y. Lee, Incremental updates of closed frequent itemsets over continuous data streams, Expert Systems with Applications 36 (2) (2009) 2451-2458.

[37] J. Lin, E. Keogh, S. Lonardi, B. Chiu, A symbolic representation of time series, with implications for streaming algorithms, in: Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 2003, pp. 2-11.

[38] M. Y. Lin, S. C. Hsueh, C. W. Chang, Fast discovery of sequential patterns in large databases using effective time-indexing, Information Sciences 178 (22) (2008) 4228-4245.

[39] F. Masseglia, P. Poncelet, M. Teisseire, Efficient mining of sequential patterns with time constraints: reducing the combinations, Expert Systems with Applications 36 (2) (2009) 2677-2690.

[40] F. Masseglia, P. Poncelet, M. Teisseire, Incremental mining of sequential patterns in large databases, Data and Knowledge Engineering 46 (1) (2003) 97-121.

[41] F. Mörchen, A. Ultsch, Optimizing time series discretization for knowledge discovery, in: Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2005, pp. 660-665.

[42] Y. Nishi, R. Doering, Handbook of semiconductor manufacturing technology (1st ed.), Marcel Dekker Inc., New York, 2000.

[43] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal, Discovering frequent closed itemsets for association rules, in: Proceeding of the 7th International Conference on Database Theory, 1999, pp. 398-416.

[44] J. Pei, J. Han, R. Mao, CLOSET: an efficient algorithm for mining frequent closed itemsets, in: Proceedings of the 5th ACM-SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 2000, pp. 21-30.

[45] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth, in: Proceedings of the 17th International Conference on Data Engineering, 2001, pp. 215-224.

[46] W. C. Peng, Z. X. Liao, Mining sequential patterns across multiple sequence databases, Data and Knowledge Engineering 68 (10) (2009) 1014-1033.

[47] D. Perera, J. Kay, I. Koprinska, K. Yacef, O. R. Zaiane, Clustering and sequential pattern mining of online collaborative learning data, IEEE Transactions on Knowledge and Data Engineering 21 (6) (2009) 759-772.

[48] P. J. Pockros, E. R. Schiff, M. L. Shiffman, J. G. McHutchison, R. G. Gish, N. H. Afdhal, M. Makhviladze, M. Huyghe, D. Hecht, T. Oltersdorf, D. A. Shapiro, Oral IDN-6556, an antiapoptotic caspase inhibitor, may lower aminotransferase activity in patients with chronic hepatitis C, Hepatology 46 (2) (2007) 324-329.

[49] A. H. Ritchie, D. M. Williscroft, Elevated liver enzymes as a predictor of liver injury in stable blunt abdominal trauma patients: case report and systematic review of the literature, Canadian Journal of Rural Medicine 11 (4) (2006) 283-287.

[50] S. Russell, A. Gangopadhyay, V. Yoon, Assisting decision making in the event-driven enterprise using wavelets, Decision Support Systems 46 (1) (2008) 14-28.

[51] S. R. Song, W. Y. Ku, Y. L. Chen, Y. C. Lin, C. M. Liu, L. W. Kuo, T. F. Yang, H. J.

Lo, Groundwater chemical anomaly before and after the Chi-Chi Earthquake in Taiwan, Terrestrial, Atmospheric and Oceanic Sciences 14 (3) (2003) 311-320.

[52] R. Srikant, R. Agrawal, Mining sequential patterns, in: Proceedings of the 11th International Conference on Data Engineering, 1995, pp. 3-14.

[53] R. Srikant, R. Agrawal, Fast algorithms for mining association rules, in: Proceedings of the 20th International Conference Very Large Data Bases, 1994, pp. 487-499.

[54] R. Srikant, R. Agrawal, Mining sequential patterns: generalizations and performance improvements, in: Proceedings of the 5th International Conference on Extending Database Technology, 1996, pp. 3-17.

[55] Standard and Poor's, <http://www2.standardandpoors.com>.

[56] Stocks on Wall Street, <http://stocksonwallstreet.net/2009/07/31/golden-cross-shows-bullish-technical-indicator/>.

[57] Taiwan Stock Exchange Corporation, < http://www.tse.com.tw/ch/index.php>.

[58] J. I. Takeuchi, K. Yamanishi, A unifying framework for detecting outliers and change points from time series, IEEE Transactions on Knowledge and Data Engineering 18 (4) (2006) 482-492.

[59] H. J. Teoh, C. H. Cheng, H. H. Chu, J. S. Chen, Fuzzy time series model based on probabilistic approach and rough set rule induction for empirical research in stock markets, Data and Knowledge Engineering 67 (1) (2008) 103-117.

[60] C. S. Wang, A. J. T. Lee, Mining inter-sequence patterns, Expert Systems with Applications 36 (4) (2009) 8649-8658.

[61] J. Wang, J. Han, BIDE: efficient mining of frequent closed sequences, in: Proceedings of the 20th International Conference on Data Engineering, 2004, pp. 79-90.

[62] J. Wang, J. Han, J. Pei, CLOSET+: searching for the best strategies for mining

frequent closed itemsets, in: Proceedings of the 9th ACM International Conference on Knowledge Discovery and Data Mining, 2003, pp. 236-245.

[63] Y. Wang, E. P. Lim, S. Y. Hwang, Efficient mining of group patterns from user movement data, Data and Knowledge Engineering 57 (3) (2006) 240-282.

[64] H. W. Wu, A. J. T. Lee, Mining closed flexible patterns in time-series databases, Expert Systems with Applications 37 (3) (2010) 2098-2107.

[65] Yahoo Finance, <http://finance.yahoo.com>.

[66] X. Yan, J. Han, R. Afshar, CloSpan: mining closed sequential patterns in large databases, in: Proceedings of the 2003 SIAM International Conference on Data Mining, 2003, pp. 166-177.

[67] T. Q. Yang, A time series data mining based on ARMA and MLFNN model for intrusion detection, Journal of Communication and Computer 3 (7) (2006) 16-22.

[68] D. Yuan, K. Lee, H. Cheng, G. Krishna, Z. Li, X. Ma, Y. Zhou, J. Han, CISpan: comprehensive incremental mining algorithms of closed sequential patterns for multi-versional software mining, in: Proceedings of the 2008 SIAM International Conference on Data Mining, 2008, pp. 84-95.

[69] M. J. Zaki, SPADE: an efficient algorithm for mining frequent sequences, Machine Learning 42 (1) (2001) 31-60.

[70] M. J. Zaki, C. Hsiao, Efficient algorithms for mining closed itemsets and their lattice structure, IEEE Transactions on Knowledge and Data Engineering 17 (4) (2005) 462-478.

簡　歷

姓　　名：吳惠雯

出生地：台灣省台南縣

出生日：中華民國六十九年十一月二十三日

學　　歷：

(1999/9 - 2003/6)

Concordia University, Montréal, P.Q.

Bachelor of Engineering in Computer Engineering

(2003/9 - 2004/6)

University of Chicago, Chicago, IL

Master of Science in Computer Science

(2006/9 - 2010/1)

國立台灣大學資訊管理研究所博士班

著　　作：

A. J. T. Lee, C. S. Wang, W. Y. Weng, Y. A. Chen, H. W. Wu, An efficient algorithm for mining closed inter-transaction itemsets, Data and Knowledge Engineering 66 (1) (2008) 68-91.

A. J. T. Lee, Y. H. Liu, H. M. Tsai, H. H. Lin, H. W. Wu, Mining frequent patterns in image databases with 9D-SPA representation, The Journal of Systems and Software 82 (4) (2009) 603-618.

A. J. T. Lee, H. W. Wu, T. Y. Lee, Y. H. Liu, K. T. Chen, Mining closed patterns in multi-sequence time-series databases, Data and Knowledge Engineering 68 (10) (2009) 1071-1090.

H. W. Wu, A. J. T. Lee, Mining closed flexible patterns in time-series databases, Expert Systems with Applications 37 (3) (2010) 2098-2107.