國立交通大學

資訊科學與工程研究所

碩 士 論 文

容 許 間 距 之 近 似 重 覆 樣 式 探 勘

Mining Repeating Pattern with Gap Constraint

研 究 生：邱欣怡

指導教授：黃俊龍　教授

中 華 民 國　九 十 八 年 一 月

容許間距之近似重覆樣式探勘
Mining Repeating Pattern with Gap Constraint

研 究 生：邱欣怡　　Student：Shin-Yi Chiu

指導教授：黃俊龍　　Advisor：Jiun-Long Huang

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

Jan. 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年一月

摘要

以往對於 repeating pattern mining 的研究主要著重於從一個由音樂轉成較長的字串中找出經常重覆出現的子字串。舉例來說，A 公司和 B 公司股價上漲，則 C 公司股價則會在 4 天之後上漲。然而，鄧教授所提出的問題給予太多的限制在從一長串 set 中找出 repeating pattern，這使得許多潛在的 frequent patterns 會因為這個限制導致他們的 support 分散進而無法被找出。因此，在我們的論文中定義了一個新的 pattern，它允許二個相鄰 set 之間有 gap 的存在，此外我們也提出了一個演算法，G-Apriori，找出允許 gap 的 pattern。G-Apriori 演算法產生 candidates 且透過掃描 database 來計算 candidates 的 support。然而為了要避免掃描 database 太多次，GwI-Apriori 被提出來解決這個問題。在 GwI-Apriori 中，我們設計了一個 index list，它包含一個開始位置跟一串的結尾位且利用它來紀錄 frequent pattern 的所在位置。 透過這些 index lists，GwI-Apriori 只需要掃瞄 database 一次且利用它們來進行較長 pattern 的 support 的計算。此外，在 GwI-Apriori 中我們也設計了 pruning 策略來加速 support 的計算。實驗的資料是以實際的資料評估，且實驗的結果顯示 GwI-Apriori 優於 G-Apriori。

Abstract

Previous studies on mining repeating patterns focus on discovering sub-strings which appear frequently in a long string, converted from the music. An example of such repeating pattern is ”if the stock price of companies A and B both goes up on day one, the stock price of company C will go up on *exactly* day fifth.” But the problem proposed by Tung gives too much limitation for mining repeating patterns from set sequence, the potential frequent patterns can not be found due to the frequencies distrusted. Hence, in our paper we define a new pattern, which allows the *gap* between two adjacent sets, and propose an algorithm, G-Apriori, to discover the repeating patterns with gap constraint from a set sequence. G-Apriori algorithm generates candidates and counts the frequency of these candidates by scanning the database. In order to avoid scanning the database so many times, the algorithm, GwI-Apriori is proposed to solve the problem. In GwI-Apriori method, it designs an index list, which contains the start position (*SP*) and end position (*EP*) list, for recording the positions of the frequent patterns. Besides, the GwI-Apriori also takes the additional strategy for pruning the searching space among the index lists. By using the index lists, the GwI-Apriori only scans the database once and computes the frequency of frequent patterns through the index lists. The experimental results show that the GwI-Apriori performs much better than G-Apriori.

致謝

首先我最感謝我的指導教授黃俊龍老師，感謝他在碩士班時間內給予我研究上的指導以及論文上寫作的技巧。我也要感謝指導我的博班班邱釷銓學長，每星期特地抽空與我討論，解答我心中的疑惑。此外，也感謝口試委員：台科大資工系戴碧如教授及在美商新思科技公司工作的莊坤達博士在口試過程中提供了許多的寶貴意見及看法，讓我的論文能更加完善。

此外我也要感謝我實驗室的學長，同學及學弟們。很感謝你們在我的碩士班的生活裡給了我幫忙及鼓勵，好讓我在失去鬥志時能夠很快的恢復動力繼續努力做研究。然而，我也很感謝我的大學好友們在這段時間內很有耐心的傾聽我訴苦，真的很謝謝你們。

最後，誠摯感謝父母親及哥哥們的支持及鼓勵，使我能夠在無後顧之憂下專心做研究。在此希望與你們共同分享完成這篇論文的喜悅。

欣怡　2009.01

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In our diary life, we can find that patterns appear repeatedly, such as in DNA sequence, music and video, the behavior of a person, ups and downs relation of the stock price between each company,. . ., so on. We show a example in Fig 1.1. However, if we can find the patterns from these data, we can use these patterns to describe and forecast the future trend or behavior of these data. For example, in music, we can extract music segment, appearing frequently, from the music data and make use of these segment to represent the music. Therefore we can achieve both efficiency and semantic-richness requirements for content-based music data retrieval. Besides, the investors may also interested in the relation of stock prices among the companies, such as ”When stock price of A company rise 10 percent and stock price of B company rises 5 percent, the stock price of C company will fall 4 percent within the following three days.” The investors can make a profitable investment while obtaining these information.

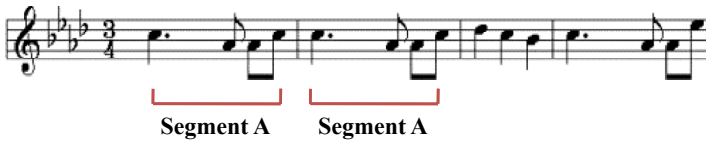


Figure 1.1: A phrase excerpted from Brahms Waltz in A flat

The first application for discovering the pattern appearing repeatedly is in biological field [2]. In this field, we convert the DNA sequence into a string, we want to find the sub-string which is tandem repeat in the converted string. In multimedia area, the indexing and searching techniques for multimedia data are main topic, therefore Chen et al. propose the new problem, repeating pattern mining, to discover the repeating music segment. The repeating pattern mining problem firstly focused on finding exact frequent patterns in music database. [9] was the first work to solve the problem. In this work, the music is converted into a sequence of notes, and a data structure called correlative matrix which integrates associated algorithm is proposed to discovering the repeating

patterns efficiently. Fig 1.1 shows the example, where the sub-string ”C6-Ab5-Ab5-C6” appears two times in the string ”C6-Ab5-Ab5-C6-C6-Ab5-Ab5-C6-D6-C6-B5-C6-A5-A5-E6.” However, music segments with minor difference may be regard as the same segments and also could be important patterns for indexing, shown in Fig 1.2, so Hsu et al. proposed the approximate repeating patterns mining problem in [10].

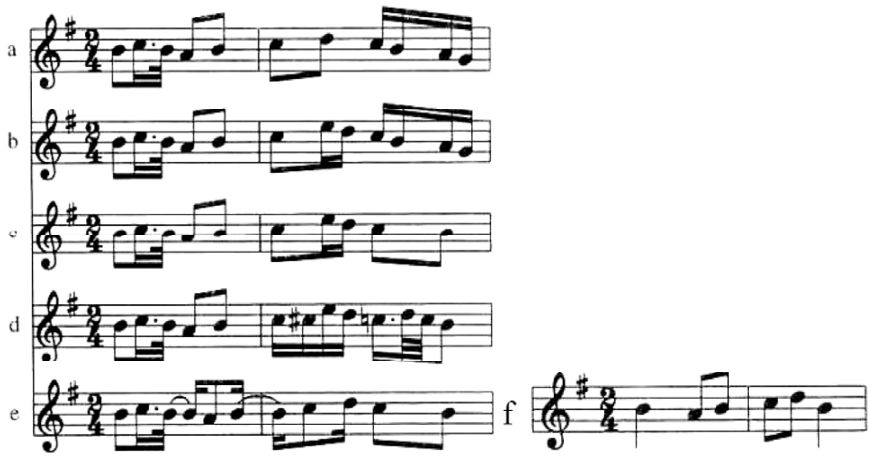


Figure 1.2: Five extracts from Mozarts Piano Sonata K. 311 and a prototypical melody (excerpted from [Self98]).

In [10], it defines the match operator which is used to determine whether the pattern match this music segment, then divides the music into non-overlap music segments and sums up the number of segments satisfying the match operator. Besides, Liu, et al.[14] proposed new definition for approximate repeating patterns in 2005. The method in [14] applies edit distance to find out the approximate repeating patterns in music data. However, in the following year, 2006, Koh, et al. proposed the new problem for mining top-k fault-tolerant repeating patterns in [13]. The work used bit strings to express the string data, and applied this bit strings to perform the ”AND” and ”Shift” operators to obtain the fault-tolerant repeating patterns. Nevertheless, the events may happen on the same time and we may interested in finding the connection of these events which happen on the different time. For example, investors may want to know whether the pattern, the stock price of A company and B company both go up 5%, the following data the stock price of C company goes up 5%, appears frequently. Hence Tung proposed the related work to solve this problem in [20]. In this paper, it views these events appearing on the same time as a set, then connects these sets of different appearing time by order, and finds the relation among all sets.

However, in real world the frequent patterns may be concealed by the noise or delay, therefore the work which proposed by Tung in [20] will give too much limitation for discovering the frequent patterns, the potential frequent patterns may not be found. Therefore, in our study, we loosed the restrictions and viewed these patterns the same by allowing the gaps (delay or noise) between two

adjacent sets. For example, consider the following patterns $< \{A\}, \{B\} >$, $< \{A\}, *, \{B\} >$ and $< \{A\}, *, *, \{B\} >$, we said that frequency of pattern $< \{A\}, \{B\} >$ is 3 from above patterns by giving the $gap = 2$, but the frequency of pattern $< \{A\}, \{B\} >$ is 2 by giving the $gap = 1$. The algorithm, G-Apriori, is proposed to solve the problem. In this algorithm the candidates are generated based on apriori property, and the frequent patterns, which frequency are larger than the user-defined threshold, are obtained from these candidates by scanning the database for counting the frequency. Nevertheless, the G-Apriori algorithm takes to much time to scan database for frequency counting. Hence, we refined the G-Apriori algorithm and proposed GwI-Apriori algorithm to avoid inherently scanning database many times. The first stage of GwI-Apriori is to scan the database and record the index positions for each 1-length frequent pattern. For the patterns which length is larger than 1, we only need to use the index positions, recorded in the first stage, for counting the frequency. Besides, in order to speed up the frequency counting, we also design the pruning technique to reduce the redundant comparison among these index positions.

The remainder of this paper is organized as follows: In Section 2, we present preliminaries, including related work and definitions of our new pattern, the repeating pattern with gap constraint. In Section 3, we propose two methods, G-Apriori method and GwI-Apriori method, to mine repeating patterns with gap constraint efficiently. In Section 4, we prove correctness. In Section 5 shows experimental results. Finally, we will conclude in Section 6.

Chapter 2

Preliminaries

2.1 Related Works

In this section, we review the related work on discovering repeating patterns. Besides, we also discuss inter-transaction association. Section 2.1.1 provides a brief discussion about the existing work on finding repeating patterns. The following section, the algorithm for finding inter-transaction association rule will be offered.

2.1.1 Repeating Pattern

In this subsection, we will talk about three types of repeating pattern, exact, approximate and fault-tolerant, sequentially.

Exact Repeating Pattern

[9] was the first work that proposed the repeating pattern mining problem in music field. In this paper, the music is converted into a sequence of notes, and a data structure called correlative matrix which integrates associated algorithms is proposed to discovering the repeating pattern efficiently, which is a shorter sequence of notes appearing more than once in a music object. Consider the phrase with 12 notes from Brahms Waltz in A flat. Its corresponding sequence of nodes is ”C6-Ab5-Ab5-C6-C6-Ab5-Ab5-C6-Db5-C6-Bb5-C6”. The correlative matrix of this sequence is shown in Fig 2.1. The function of the correlative matrix is to preserve the intermediate results of substring matching.

The first step for finding the repeating pattern is to initialize the matrix, and it use $T_{i,j}$ to indicate the element of the i-th row and j-th column in the matrix T. The upper-triangle slots in the matrix will be filled up based on the following principles: For any two notes S_i and S_j ($i \neq j$ and $i, j > 1$) in

	C6	Ab5	Ab5	C6	C6	Ab5	Ab5	C6	Db5	C6	Bb5	C6
C6	—			1	1			1		1		1
Ab5		—	1			2	1					
Ab5			—			1	3					
C6				—	1			4		1		1
C6					—			1		1		1
Ab5						—	1					
Ab5							—					
C6								—		1		1
Db5									—			
C6										—		1
Bb5											—	
C6												—

Figure 2.1: Correlative Matrix

the music string S, if $S_i = S_j$, we set $T_{i,j} = T_{i-1,j-1} + 1$. If the value stored in $T_{i,j}$ is n, it indicates a repeating pattern of length n appearing in the positions (j V n + 1) to j in S. Fig 2.1 shows the result after all notes are processed. After filling up the correlative matrix, the following step is to find all repeating patterns and their repeating frequencies. In this step, it uses a set called the candidate set (denoted CS) to record the repeating patterns and their repeating frequencies. However, there are only four cases which can be put into the CS.The cases are (1). $T_{i,j} = 1$ and $T_{i+1,j+1} = 0$, (2). $T_{i,j} = 1$ and $T_{i+1,j+1} \neq 0$, (3). $T_{i,j} > 1$ and $T_{i+1,j+1} \neq 0$ and (4). $T_{i,j} > 1$ and $T_{i+1,j+1} = 0$. After finding all candidate sets, there are extra two steps to implement. The first is the pruning step. If a repeating pattern satisfies the pruning principle, it will be removed from the candidate set. The principle is that for any repeating patterns in CS, if it is a substring of another repeating pattern and they have the same frequencies, it will be pruned from the CS. The second step is that compute the real repeating frequency for each repeating pattern based on the formulas: $f = \frac{1+\sqrt{1+8\times rep\_count}}{2}$.

Approximate Repeating Pattern

However, there exists another repeating pattern which loosens the condition for finding the repeating pattern. [10] was the first proposed by Jia-Lien Hsu, et al. to solve this problem. In [10], it defines the match operator which is used to determine whether the pattern match this music segment. The match operator is stated as follow: Given $P = (p_1, p_2, \ldots, p_m)$ and $LL = (s_1, s_2, \ldots, s_n)$, where $n >$

m. $long\_leng\_match(P,LL) = 1$, if $p_i = s_{b_i}, fori = 1,2,\ldots,m, where 1 = b_1 < b_2 < \ldots < b_m = n$, otherwise, $long\_leng\_match(P,LL) = 0$. Based on the match operator, Jia-Lien Hsu also proposed the method for compute the repeating frequency of a pattern P. The main idea of frequency counting for a pattern P is dividing the music into non-overlap music segments and sum up the number of segments which satisfies the match operator. In order to find the longer length pattern, Jia-Lien Hsu uses the *pattern_join* operator in level-wise approach.

Besides, Jia-Ling Koh, et al. [13] also proposed another new definition for approximate repeating patterns, which allows insertiong/deletion errors occurring.

In [13], Jia-Ling Koh proposed two definitions of counting frequency for a pattern, IFT-contain and DFT-contain. Given a data sequence $DSeq = D_1D_2\ldots D_n$ and a pattern $P = P_1P_2P_3\ldots P_m$, we said that DSeq is FT-contain pattern P on position i with ε insertion errors iff there exist an integer $1 \leq i \leq n$, such that $D_i = P1$, $D_{(i+m-1)+\varepsilon} = P_m$, and P is a sub-sequence of $D_iD_{i+1}\ldots D_{(i+m-1)+\varepsilon}$ and DSeq is said to IFT-contain pattern P under fault tolerance ε_I, iff DSeq FT-contain P with ε insertion errors and $\varepsilon \leq \varepsilon_I$. The DSeq is FT-contain pattern P on position i with ε deletion errors iff there exists an integer $1 \leq i \leq n$, such that $D_iD_{i+1}\ldots D_{(i+m-1)-\varepsilon}$ is a sub-sequence of P, and we call DSeq is DFT-contain pattern P by giving a fault tolerance ε_D, iff DSeq FT-contains P on position with ε deletion errors, where $D_i = P_1$. and $\varepsilon \leq \varepsilon_D$. Consequently, the fault tolerant frequency for a pattern P in DSeq is the number of different positions in DSeq where DSeq IFT/DFT-contains P. The example is provided as follows.

Example Consider DSeq=ABCDCABA, $\varepsilon_I = 2$ and $\varepsilon_D = 3$. Given patterns P_1=ABCC, P_2=BCDC, P_3=ACAB,P_4=AEF, P_5=BCFC. DSeq FT-contains P_1 on position 1 with 1 insertion error and DSeq also FT-contains P_2 on position 2 with 0 insertion error. Hence, DSeq IFT-contains P_1 and P_2. However, DSeq doesn't IFT-contains P_3 for the insertion error of it is larger than ε_I. With regard to P_4 and P_5, DSeq DFT-contains both as they all satisfied the DFT-contain definition.

In order to speed up the time of frequency counting, the bit sequence representation of data item and *shift* and *and* operation on bit sequence [5] are incorporated into two algorithms, named TFTRP-Mine and RE-TFTRP-Mine which were proposed in [13]. Fig 2.2 shows the bit sequence of each data item in data sequence, "ABCDABCACDEEABCCDEACD" .

In fig 2.2, the bit sequence for each data item N is denoted as $Appear_N$ and the length of bit sequence for each data item is equal to the length of the data sequence. The numbers, 1 and 0, appearing in bit sequence represent whether some data item appears on the ith position of the data sequence respectively. Therefore, we can obtain the frequency of data item by accumulating the all none zero number in the bit sequence.

However, two recursive functions of counting Insertion/Deletion fault tolerance frequency for a

Sequence Data	ABCDABCACDEEABCCDEACD
Data Item	Bit Sequence
A	10001001000010000100
B	01000100000001000000
C	00100010100000110010
D	00010000010000001001
E	00000000011000001000

Figure 2.2: Bit Sequence Representation

pattern are proposed. The functions are shows as follows,

(1) Recursive functions of getting $Appear_P^+(E)$:Suppose a pattern $P = P_1P_2\ldots P_m$ is given. Let $P^{'} = P_1P_2\ldots P_{m-1}$ and X denote P_m. $Appear_P^+$(E) is obtained from the following function for $0 \leq E \leq \varepsilon_I$.

IF $|P| = 1$, **then** $Appear_P^+$(E)=$Appear_p$; $\forall 1 \leq E \leq \varepsilon_I, Appear_P^+(E) = 0$;

Else If E=0, **then** $temp_1(E) = Appear_{p}^{+}(0); temp_2(E) = L\_shift(Appear_x, |P|-1)$;

Else $temp_1(E) = temp_1(E-1) \vee Appear_{P'}^{+}(E)$;

$temp_2(E) = L\_shift(temp_2(E-1),1); Appear_P^+(E) = temp_1(E) \wedge temp_2(E)$.

(2)Recursive functions of getting $Appear_P^-(E)$: Suppose a pattern $P = P_1P_2\ldots P_m$ is given, where $P_i(i = 1,\ldots,m)$ is a data item. Let Y denote P_1, $P_{''}$ denote $P_2P_3\ldots P_m$, Q denote $P_2P_3\ldots P_{m-1}$, and X denote P_m. When deletion fault tolerance E is given, $Appear_P^-(E)$ is obtained from the following recursive function.

IF $|P| \leq E+1$, **then** $Appear_P^-(E) = Appear_Y$;

Else $temp_p^{'}(E-1)$=$Appear_Q \vee (Appear_Q^-(E-1) \wedge L\_shift(Appear_X, |P^{''}|-E, 0))$;

$temp_p^{'} = temp_Q(E-1) \vee (Appear_Q^-(E) \wedge L\_shift(Appear_X, |P^{''}-E-1, 0))$;

$Appear_P^-(E) = Appear_Y \wedge L\_shift(temp_p"'(E), 1, 0)$.

In TFTRP-Mine algorithm, all fault tolerant patterns, denoted as FT-RPs, are obtain by using the recursive functions defined in the previous paragraph. But in TFTRP-Mine algorithm, the top-k non-trivial FT-RPs was extracted from all results which are found first, RE-TFTRP-Mine algorithm was designed to improve TFTR-Mine algorithm. In RE-TFTRP-Mine method, the FT-RPs which are not possible the top-k non-trivial FT-RPs are removed in advance by increasing the min_freq during the mining process, hence the FT-RPs with the fault-tolerant frequencies less than the min_freq will not be employed in the following mining process. Besides, it also gives the priorities for the found FT-RPs, the higher fault-tolerant frequency the patterns have, the higher the priorities the patterns have. Then, the FT-RPs with higher frequencies are selected to generate the new candidates.

However, Ning-Han Liu, et al. [14] also proposed another new method to find the approximate repeating pattern. The first step of this method is converting the pitch string of music into the interval string, and then divide the interval string into the interval segments according to *max_len* and *min_len* constraints, which used to filter out unimportant music patterns. Then we regard these segments as candidates ARP. Then, for each candidate, the edit distance was adopted to measure the similarity degree between two music segments. Finally, according to the number of similar music segments and how they overlap each other, we decide whether the candidate ARP has qualification for being an ARP. In order to speed up the execution time, it also modifies the R*-tree to remove impossible candidates before computing the edit distances.

2.1.2 Inter-transaction Association

There are several kinds of inter-transaction association mining problem, such as sequential pattern mining [4], frequent episodes mining [17] [16], periodic patterns mining [7] [22] [24] [6] and frequent continuities mining [21] [20] [11] [12] [15]. We will give a shorter introduction for sequential pattern, frequent episodes, and periodic pattern mining, but give a detailed explanation for frequent continuities mining which is most resemble to our work.

Sequential Pattern Mining

The sequential pattern mining problem was first introduced in [4] by Agrawal and Srikant. In order to improve the speed for algorithm proposed in [4], there were a lot of methods are designed, such as PrefixSpan [19], SPADE [25], SPAM [5], FreeSpan [8] ... and so on. Since the sequential pattern mining may generate many redundant patterns, it will decrease not only effectiveness but also efficiency of mining. Therefore, closed pattern mining problem was gradually noticed by our. The famous algorithms for it are Clospan [23] and BIDE [1].

Frequent Episodes

Different from the sequential pattern, the data for frequent episodes is a sequence of event sets where the events are sampled regularly. An episodes is defined as a collection of events in a user-defined windows interval that appear relatively close to each other in a given partial order [17]. In [17], Mannila et at. defined three classes of episodes: serial, parallel and combination of serial and parallel. Serial episodes consider order for patterns in the sequence, while parallel episodes do not have constraints on the relative order of event sets. Fig 2.3 shows the three kinds of episodes.

Moreover, Mannila, et al. also proposed a new approach, WINEPI, for discovering the all frequent serial/parallel episodes in [17]. For finding the exact relation among episodes, Mannila et al.

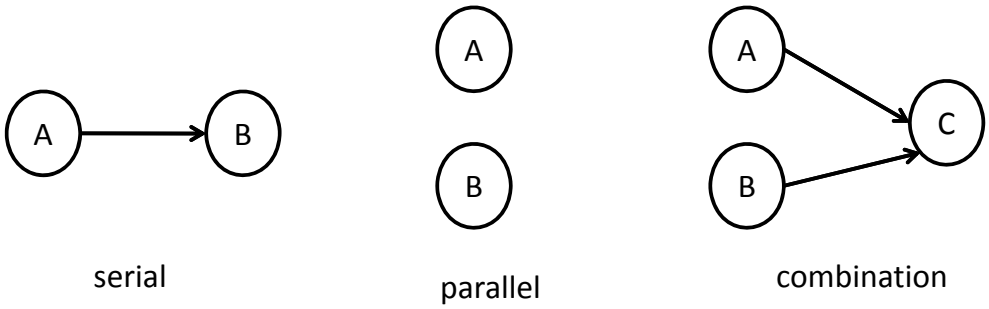


Figure 2.3: Episodes Class

also specify another classes of generalized epiosdes in [16] and designed an algorithm, MINEPI, for discovering the frequent episodes based on minimal occurrences of episodes.

Periodic Patterns

Periodic pattern is defined as the pattern appears in the same time periodically. In last decades, there exist many studies for finding periodic patterns. However, in these studies many definitions of periodic pattern are proposed to apply in different situation which more conforms to real life. For example, in early days, cyclic association rules mining was first proposed by Banu Özden, et al. in [18], and the following is partial periodic patterns defined by Jiawei Han, et al. in [7] [6] to loose the constraints on whether every point in time contributes to the periodicity. For example, Bob eats breakfast from 8:00 to 9:00 every day, but do other things which is not regular at other times. Moreover in order to solve the problem that the periodic pattern may occurs asynchronous, the asynchronous periodic pattern is designed by Jiong Yang, et al. in [24] [22]. Take the previous example, Bob may eats breakfast from 9:00 to 10:00, which also contribute to the periodic pattern.

Frequent Continuities

The name continuity pattern was coined by Huang in [11] which used to substitute the name inter-transaction association rule defined by Anthony K.H. Tung in [20]. The continuity pattern, also called inter-transaction association rule, is defined as the pattern that considers the occurring order of each itemset in the pattern. Hence, we can also refer this patter as a looser constraint of periodic pattern which has limitation on contiguous and disjoint match. An algorithm, FITI [21], was proposed to solve this problem efficiently. FITI [21] have three stages:

(1) Mining and Storing Frequent Intra-transaction Itemsets, (2) Database Transformation, and (3) Mining Frequent Inter-transaction Itemsets. Nevertheless, it also takes to much time to find the results, hence Huang in [11] designed PROWL algorithm to mine results efficiently. The central

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A C	E	B D	A C	B D E	B D	A C	A C E	B D	E	A C	B D	D E	A C	E	B D

(a) Temporal Transaction Database

Event	TIDLists
A	1,4,7,8,11,14
B	3,5,6,9,12,16
C	1,4,7,8,11,14,15
D	3,5,6,9,12,13,16
E	2,5,8,10,13,15

(b) Vertical database

Code	Eventset	TIDLists	Note
#1	{C}	1,4,7,8,11,14,15	C.F.I
#2	{D}	3,5,6,9,12,13,16	C.F.I
#3	{E}	2,5,8,10,13,15	C.F.I
#4	{A,C}	1,4,7,8,11,14	C.F.I
#5	{B,D}	3,5,6,9,12,16	C.F.I

(c) Encoding table for F.I (minsup=4)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
#1 #4	#3	#2 #5	#1 #4	#2 #3 #5	#2 #5	#1 #4	#1 #3 #4	#2 #5	#3	#1 #4	#2 #5	#2 #5	#1 #4	#1 #3	#2 #5

(d) Encoded Horizontal Database

Figure 2.4: Data Convert Process

thought of PROWL is to use the memory for both the event sequence and the indices in the mining process.

Huang also integrates prune hash table into PROWL algorithm to design the algorithm, Closed-PROWL [12]. In ClosedPROWL [12], there are three phrases for discovering the results. The first phase is to find all 1-size closed frequent itemsets, called C.F.E.. Then in the second phase, encode these C.F.E and construct them into a encoded horizontal database. Fig 2.4 shows the each step from converting the temporal database into encoded horizontal database.

In the third phase, refined PROWL [11] algorithm was utilized to find all closed frequent continuities. The mining process of refined PROWL is described as follows:

(1) First we find the 1-offset projected window list, denoted as PWL, of each encoded eventset P, also called closed frequent continuity.

(2) Find all eventsets P which support surpass the minimum support, and record these eventsets and their PWL into Prune Hash Table, denoted as PruneHT, through the hash function. Moreover, using the pruning strategy to prune the redundant eventsets.

(3) For each eventsets X which are not removed after step 2, we connect it with P to generate the new continuity,and then perform step (1), (2) and (3) recursively to produce the larger closed

frequent continuities until the length of the patter is larger the maxwin or its support is smaller than minimum support.

(4) Find all possible closed frequent continuities, then use the Closed Continuity Checking Table, denoted as CCCT, to filter the duplicated closed frequent continuities.

Finally we give the comparison among these patterns. Fig 2.5 shows the table.

	Order	Temporal	Pattern type	Window constraint	Input
Repeating pattern					
Exact	Y	N	ABC	N	A string
Approximate	Y	N	ABC	Y	A string
Inter-transaction					
Sequential pattern	Y	Y	<(A)(B)(C)>	N	A customer sequence
Episode pattern	Y	Y	A->B (A,B) (A,B)->C	Y	A sequence
Periodic pattern	Y	Y	<A,*,*>	N	A sequence
Frequent continuity	Y	Y	<A,*,*>	Y	A sequence

Figure 2.5: Comparison Table

2.2 Definition

In this section, we present essential preliminaries.

Definition 1 *(**Set Sequence Database**) Let* $\mathrm{I} = \{i_1, i_2, \ldots, i_m\}$ *be a set of elements. Let* S_i *be a subset of* I*, where* $S_i = (s_1, s_2, \ldots, s_n)$ *is a set of elements such that* $s_k \in I$ *for* $1 \leq k \leq n$ *and each element in* S_i *is distinct. The* **set sequence database SD** *is defined as an order sets of* S_i*, i.e* $\mathrm{SD} =< S_1, S_2, \ldots, S_n >$.

Definition 2 *(**GC_k-contain instance**) Given two set sequence* $\mathrm{SD} =< S_1, S_2, \ldots, S_n >$*, and* $P =< p_1, p_2, \ldots, p_m >$*, where* $n \gg m$*, we say that SD* GC_k*-contains P at position k iff there exists an integer* $1 \leq k \leq n$*, such that* $p_1 \subseteq S_{i_0}$*,where* $i_0 = k$*,* $p_2 \subseteq S_{i_1}$*, ...,*$p_m \subseteq S_{i_{m-1}}$ *and* $i_j - i_{j-1} \leq GC + 1$ *for* $1 \leq j \leq m$*. The GC (abbreviated from gap constraint and denoted as* γ*) is a user-defined upper bound number of gaps between two adjacent set of P in SD.*

Example *Consider SD =*$< \{A,B,C,D\},\{A,C\},\{A,B,C\},\{A\},\{A,C,D,E\},\{A\},\{B,C,E,F\},\{B,D\},\{A,C\},\{E\} >$ *and GC=1, we say that SD has two* GC_1*-contain instances for pattern P1 =*$< \{D\},\{C\} >$ *i.e. m1 and m4 in Fig 2.6.*

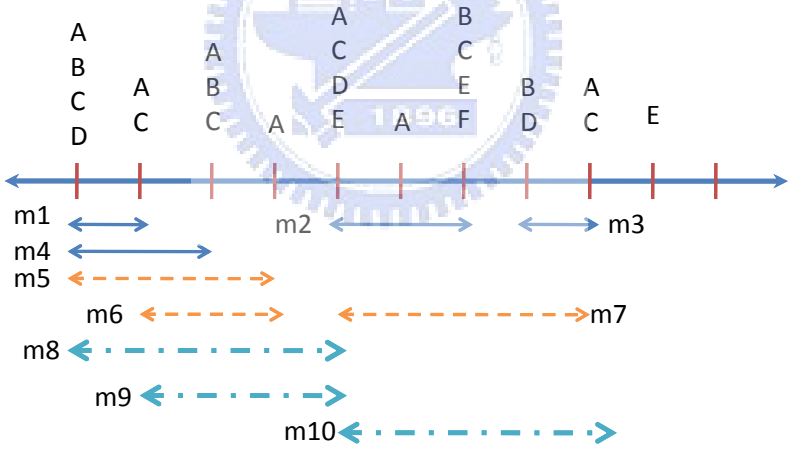


Figure 2.6: Illustrative Example I

Definition 3 *(**Length and Size of a pattern**) Given a pattern* $P =< p_1, p_2, \ldots, p_m >$*, then size of P is defined as the number of sets in P, denoted as size(P). The length* $\boldsymbol{l}$ *of P is defined by the* $\boldsymbol{l} = \sum_{i=1}^{m} |P_i|$.

Example *Given* $P =< \{A,B\},\{C\},\{A\},\{C\} >$*, size of P is 4, length of P is 5.*

Definition 4 *(**kth Position** GC_k**-Contain Set, abbreviated as KPCS**) Given a SD* $=< S_1, S_2, \ldots, S_n >$*, a pattern* $P =< p_1, p_2, \ldots, p_m >$*, where* $size(SD) \geq size(P)$ *and the GC, the kth Position* GC_k*-Contain Set consists of a starting position, k, and different ending positions which are the ending positions of distinct* GC_k*-Contain instances for P in SD under GC is given. we use* $\{k, (n_1, n_2, \ldots, n_j)\}$

to record all positions, where n_i means the ending position for a pattern P for $1 \leq i \leq j$. We also use $< k, n_i >$ to mean one instance of kth Position GC_k-Contain Set, where k is a starting position and n_i is an ending position. i.e. Given a KPCS $\{1,(3,4,5,7)\}$, the instances of this KPCS are $< 1,3 >$, $< 1,4 >$, $< 1,5 >$ and $< 1,7 >$.

Example *Given SD =$< \{A,B,C,D\},\{A,C\},\{A,B,C\},\{A\},\{A,C,D,E\},\{A\},\{B,C,E,F\},\{B,D\}, \{A,C\},\{E\} >$ and GC=1, we say that P1 =$< \{D\},\{C\} >$ has two GC_1-contain instances in SD, i.e. m1 and m4 in Fig 2.6, where KPCS is $\{1,(2,3)\}$. Besides the P1 also has a GC_5-contain instance, i.e m2 and a GC_8-contain instance, i.e m3.*

Definition 5 ***(Repeating Pattern with Gap Constraint, abbreviated as RPGC)*** *Given a set sequence SD =$< S_1, S_2, \ldots, S_n >$ and all distinct kth Position GC_k-Contain Sets of a pattern P, the frequency of a pattern P, denoted as freq(P,SD), is the maximum number of non-overlap instances of all distinct kth Position GC_k-Contain Sets of a pattern P. If P is called a RPGC, then $freq(P,SD) \geq \delta$, where δ is a minimum support defined by user.*

Example *Consider SD =$< \{A,B,C,D\},\{A,C\},\{A,B,C\},\{A\},\{A,C,D,E\},\{A\},\{B,C,E,F\},\{B,D\}, \{A,C\},\{E\} >$, GC=1 and a pattern P1 =$< \{A,C\},\{B\},\{A\} >$ and P2 =$< \{A,C\},\{B\},\{C\} >$. The freq(P1,SD)=2 and freq(P2,SD)=1 ,shown in Fig 2.6. Hence, P1 is a RPGC, but P2 is not a RPGC.*

Finally, we define the repeating pattern with gap constraint problem as follows,

Definition 6 ***(RPGC discovery problem)*** *Given a set sequence SD and GC, find all RPGC P in SD, where $freq(P,SD) \geq \delta$.*

In order to give a clear explanation for the correctness of our algorithms, we define the following definitions.

Definition 7 ***(Counting Basis Set, abbreviated as CBS)*** *Given all distinct kth Position GC_k-Contain Sets of pattern P in SD under GC = m, the counting basis set is defined as a set of instances, i.e. $\{< k_0, n_0 >, < k_1, n_2 >, \ldots, < k_e, n_e >\}$ obtained from KPCSs of different k and satisfied the following condition :*

1) For any pair of $< k_i, n_i >$ and $< k_j, n_j >$, $< k_i, n_i >$ can not overlaps $< k_j, n_j >$ for $i \neq j$.

2) we select the instances, $< k_j, n_j >$, from all distinct kth Position GC_k-Contain Sets, where j starts from 1 to e and $n_j - k_j$ is the minimum value.[1].

Example *Given SD =$< \{A,B,C\},\{A,C\},\{A,B\},\{A,C\},\{A,C,D,E\},\{A,C\},\{B,C,E,F\},\{B,D\}, \{A,C\},\{E\} >$, GC=1 and a pattern P1 =$< \{A,C\},\{A\},\{C\} >$, We say that CBS for pattern P1 =$< \{A,C\},\{A\},\{C\} >$ are $< 1,4 >$ and $< 5,7 >$, which means m1 and m8 in Fig 2.7.*

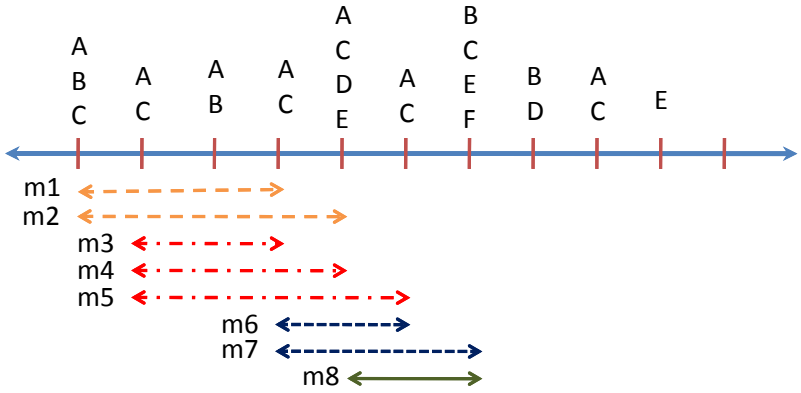


Figure 2.7: Illustrative Example II

Definition 8 *(**Unit Counting Set, abbreviated as UCS**) Given all distinct k KPCS of a pattern P in SD under GC = m and a starting position S and ending position E, denoted as* $UCS_P(S,E)$. *The Unit Counting Set is defined as a set of instances obtained from the KPCS, where ending position of each instance in UCS is equal to E, and starting position of each instance in UCS is large or equal to S. We regard all sets in Unit Counting Set being 1 of frequency count for we only need to compute the non overlap instance.*

Example *In Fig 2.7, Given SD, S=1 and E=4 and GC=1, then the* $UCS_P(1,4)$ *for pattern* $P =< \{A,C\},\{A\},\{C\} >$ *is* $< 1,4 >$ *and* $< 2,4 >$*, which mean m1 and m3.*

Definition 9 *(**Frequency Counting Set Group, abbreviated as FCSG**) Given the CBS for a pattern P under GC = m, we classified distinct kth Position GC-Contain Sets of P into a group of UCS according to the CBS of a pattern P. FCSG contains these UCS, where each UCS* $\neq \emptyset$ *and the freq(P,SD) is equal to the number of UCS in FCSG.*

Example *Consider SD=*$< \{A,B,C\},\{A,C\},\{A,B\},\{A,C\},\{A,C,D,E\},\{A,C\},\{B,C,E,F\},\{B,D\},\{A,C\},\{E\} >$*, GC=1 and a pattern* $P1 =< \{A,C\},\{A\},\{C\} >$*. The CBS for P is (1,4) and (5,7), which mean m1 and m8. The UCS according to the CBS of P is* $UCS_{P_{(1,4)}}\{(< 1,4 >,< 2,4 >\}$ *and* $UCS_{P_{(5,7)}}\{< 5,7 >\}$*. Because non of UCS is* $\emptyset$*, the FCS contain these UCS and the freq(P,SD)=2. Fig 2.7 shows the example.*

Chapter 3

The Proposed Algorithms

In this section, we propose a algorithm, G-Apriori, to find the RPGC. Besides, we also design an index list which is incorporated into G-Apriori algorithm to generate a refined method, GwI-Apriori, for efficiency.

3.1 G-Apriori Algorithm

According to anti-monotonic property, any length-(n-1) pattern of length-n RPGC must be RPGC. Hence, G-Apriori algorithm employs the property to generate the candidates C_k from L_{k-1} then finds L_k by scanning the set sequence and counting the support of C_k,where C_k is a set of length k candidate for RPGC and L_k is a set of length-k RPGC. Here, we also name a RPGC as a frequent pattern. The process is as follows: 1) Scan the set sequence to find all length 1 frequent patterns L_1. 2) Generate all length k candidates C_k from length k-1 frequent patterns L_{k-1} by pattern-grow method. 3) Scan the set sequence to count support of C_k and find L_k from C_k which support is larger than the minimum support. 4) Go to step 2) until L_k is empty. However, the method of generating candidates for a RPGC is different from the method of generating candidates for association rule [3] because each set in a RPGC has its order. Hence, we propose the pattern-grow method to generate all possible candidates C_k from L_{k-1}. Based on anti-monotonic principle, we can know that for every length ***l*** frequent pattern, all its ***l-1*** length patterns are frequent. The pattern-grow method is designed as follows. Given two patterns p_1 and p_2 in L_{k-1}, then we delete the first element from p_1 to obtain p_1' and delete the last element from p_2 to obtain p_2'. First, we need to check p_1' and p_2'. If p_1' and p_2' are both $\emptyset$, two possible length-2 patterns will be generated by two methods, appending set of p_2 to set of p_1 and adding element in set of p_2 into set of p_1. Otherwise, both p_1' and p_2' are not $\emptyset$, length k pattern will be generated by combining p_1 and p_2, which means if the length of the last set of p_1 equals to 1, we append the last set of p_2 to p_1, otherwise we add the deleted element

of p_2 into the last set of p_1. The detailed steps of G-Apriori is described in ***Algorithm*** 1.

Algorithm 1 G-Apriori Algorithm

Input:A set sequence SD, threshold δ, Gap Constraint γ
Output: All large RPGC

1: $L_1 = \{i | i \in I, freq(SD, i) \geq \delta\}$
2: **for** $k = 2; L_{k-1} \neq \emptyset$;k++ **do**
3: C_k =Candidate-Generate(L_{k-1});
4: **for all** patten c in C_k **do**
5: count = freq(c,SD,γ)
6: $L_k = \{c \in C_k | freq(c, SD, \gamma) \geq \delta\}$
7: $RSPSet = \bigcup_k L_k$

Algorithm 2 Candidate-Generate Algorithm

Input:L_{k-1}
Output:C_k

1: **for** each pair(cs_i,cs_j) where cs_i and $cs_j \in L_{k-1}$ **do**
2: cs'_i = delete first element of cs_i
3: cs'_j = delete last element of cs_j
4: **if** equal(cs'_i,$\emptyset$) and equal(cs'_j,$\emptyset$) **then**
5: cs_{k_1} = append the last set of cs_j to cs_i
6: cs_{k_2} = add the last element of cs_j to cs_i last set
7: **if** length(cs_{k_1})=i+1 **then**
8: add cs_{k_1} to C_k
9: **if** length(cs_{k_2})=i+1 **then**
10: add cs_{k_2} to C_k
11: **else**
12: **if** equal(cs'_i,cs'_j) **then**
13: **if** last element of cs'_i is the set of length 1 then **then**
14: cs_k = append the last set of cs_j to cs_i
15: **else**
16: cs_k = add the last element of cs_j to cs_i last set
17: **if** length(cs_k)=i+1 **then**
18: add c_k to C_k
19: return C_k

We give an example to show the merging process for *Candidate_Generate* step. Let L_3 be $\{<\{A\},\{B,C\}>,<\{B,C\},\{D\}>,<\{D\},\{A\},\{B\}>,<\{B,C,F\}>\}$. For patterns $<\{A\},\{B,C\}>$ and $<\{B,C\},\{D\}>$, after the *Candidate_Generate* step, C_4 will be $\{<\{A\},\{B,C\},\{D\}>,<\{A\},\{B,C,F\}>\}$ and $\{<\{D\},\{A\},\{B,C\}>\}$.

Example

Given a set sequence $sd = <\{B,C\},\{D\},\{A\},\{B,C\},\{E,G\},\{A,B,C\},\{C\},\{A,F\},\{A,C\},\{H\}>$, shown in fig 3.2, where min-support=2 and GC=1. L_1 is obtained by scanning the set sequence and checking frequency for each item, and then we use L_1 to generate C_2, line 3 of ***Algorithm*** 1. After frequency counting for each candidate in C_2, line 5 in ***Algorithm*** 1, L_2 is computed by removing

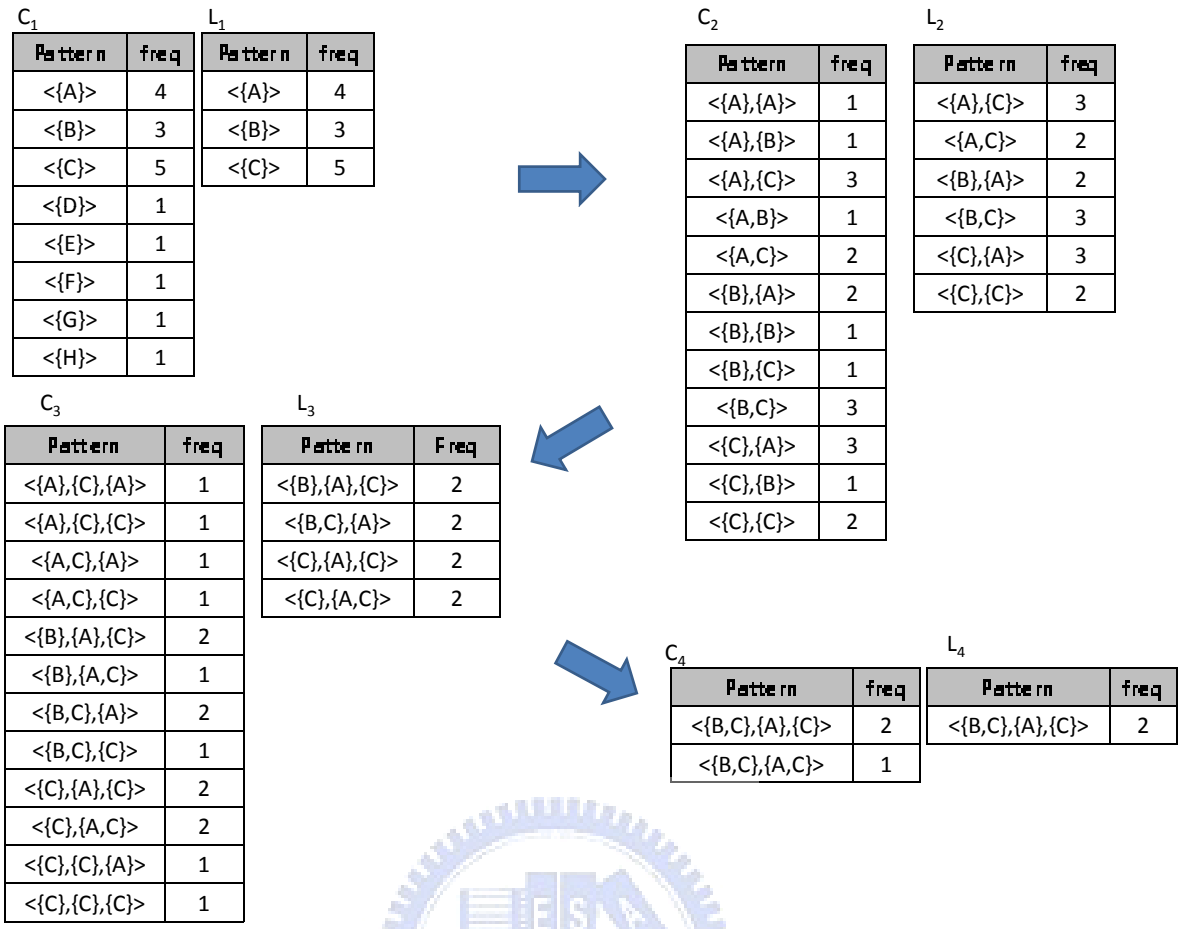


Figure 3.1: Apriori Based Example

Index Position	1	2	3	4	5	6	7	8	9	10
Sets	B C	D	A	B C	E G	A B C	C	A F	A C	H

Figure 3.2: Set Sequence Data

those patterns in C_2 which support is under the threshold. Whole process will terminate when no large pattern is derived. In this example, since L_6 is empty set, the process will stop. The frequent patterns are in L_i, where $I = \{1,2,3,4\}$. Fig 3.1 shows the process for discovering all L_k.

3.2 GwI-Apriori Algorithm

Since G-Apriori algorithm takes too much time to scan database for counting support of the patterns. Hence, we propose GwI-Apriori (abbreviated from Gap with Index Apriori) algorithm to solve this problem. The GwI-Apriori algorithm also bases on G-Apriori algorithm to generate the candidates, but only scans database once and records the positions information of L_1, then uses the positions

information for further counting the support of the patterns. Moreover, we devise an index list, which consists a S_P and E_P list, to record a start and end positions where the pattern may appear in the set sequence, i.e KPCS of the pattern in kth position. Here, we need to notice that for each pattern, it has a set of index lists where every index list stands for KPCS of the pattern at distinct S_P position. Besides, we also design a pruning strategy to speed up the execution time.

As the G-Apriori algorithm, we scan the set sequence and construct the index lists to record the positions where L_1 locate. However, We modify *Candidate_Generated* in G-Apriori algorithm to derive the *Merge_Check* shown in ***Algorithm*** 3 for generating the candidates. In *Merge_Check*, the candidates and their corresponding extended type and extended pattern are return. After finding the L_1, C_2 are generated from L_1 by calling the *Merge_Check*. For C_2, distinct strategies to construct index lists are adopted according to different extended types, sequence-extended (S-Step) and itemset-extended (I-Step) [5]. For example, given a set sequence s=$< \{A\},\{B\} >$, then the S-Step for the s is $< \{A\},\{B\},\{C\} >$ and the I-Step for the s is $< \{A\},\{B,C\} >$. When a pattern c_i in L_k can merge c_j in L_k under $\gamma = n$. According to the returned extend type and extend pattern, two different range check are applied to construct the index lists for merged pattern. The steps for constructing the index lists for merged pattern is stated as follows, we takes each value EV in E_P list from each index list of pattern c_i to check which values in S_P from all index lists of last element of pattern c_j are contained in corresponding range, where range (EV,$EV+n+1$] is for S-Step but range [S_P,S_P] is for I-Step, then we construct the index list which S_P equals to S_P in current checked index list of pattern c_i and record the values which stratified the corresponding range into the E_P list. This process will continue until all index lists of pattern C_i are checked. However, the patterns which supports are less than the δ are removed from the C_k. To be mentioned that we count the frequency while constructing the result index lists. For instance, consider the the following L_2, pattern $< \{A\},\{C\} >$, which index lists are $\{1,(2,3,4)\}$, $\{3,(4,5)\}$, $\{4,(5)\}$, pattern $< \{C\},\{B\} >$ which index lists of pattern $< \{B\} >$ are $\{2,(2)\}$, $\{3,(3)\}$, $\{5,(5)\}$, $\{6,(6)\}$, $\{10,(10)\}$ and pattern $< \{C,D\} >$ which index lists of pattern $< \{D\} >$ are $\{4,(4)\}$, $\{5,(5)\}$, $\{9,(9)\}$. C_3 are $< \{A\},\{C\},\{B\} >$ and $< \{A\},\{C,D\} >$ where the index list, $\{1,(2,3,5,6)\}$, for the pattern $< \{A\},\{C\},\{B\} >$ under $\gamma = 2$ is generated by taking each value in E_P list of $\{1,(2,3,4)\}$ to do S-Step range check in index lists of pattern $< \{B\} >$, hence the times of comparison are 3*5. Besides $\{3,(5,6)\}$ and $\{4,(6)\}$ are also the index lists for the pattern $< \{A\},\{C\},\{B\} >$. And the index lists for pattern $< \{A\},\{C,D\} >$ are $\{1,(4)\}$, $\{3,(4,5)\}$ and $\{4,(5)\}$ by applying the I-Step range check. The pattern mining process will terminate until L_k is $\emptyset$.

An example for RwI-Apriori Algorithm

Let us consider the example as shown in Fig 3.2, where $\delta = 2$ and $\gamma = 1$. We scan the sequence

Algorithm 3 *Merger_Check*

Input: L_{k-1}

Output: All merged pattern C_k and corresponding extended type C_k.T and extended pattern $C_k.L\_E$

1: **for** each pair(cs_i,cs_j) where cs_i and $cs_j \in L_{k-1}$ **do**
2: cs_i' = delete first element of cs_i
3: cs_j' = delete last element of cs_j
4: **if** equal(cs_i',$\emptyset$) and equal(cs_j',$\emptyset$) **then**
5: cs_{k_1} = append the last set of cs_j to cs_i
6: cs_{k_2} = add the last element of cs_j to cs_i last set
7: **if** length(cs_{k_1})=i+1 **then**
8: add 1 to C_k.T
9: add cs_{k_1} to C_k
10: **if** length(cs_{k_2})=i+1 **then**
11: add 0 to C_k.T
12: add cs_{k_2} to C_k
13: add last element of c_j to $C_k.L\_E$
14: **else**
15: **if** equal(cs_i',cs_j') **then**
16: **if** last element of cs_i' is the set of length 1 then **then**
17: add 1 to C_k.T
18: cs_k = append the last set of cs_j to cs_i
19: **else**
20: add 0 to C_k.T
21: cs_k = add the last element of cs_j to cs_i last set
22: **if** length(cs_k)=i+1 **then**
23: add c_k to C_k
24: add last element of c_j to $C_k.L\_E$
25: return C_k, C_k.T and $C_k.L\_E$

data and construct the index lists for L_1, shown in Fig 3.3 (a), where the pattern $\{A\}$ occurs at positions of 3, 6, 8, 9 in set sequence sd. C_2 are generated by $L1$, where one of pattern $< \{B,C\},\{A\} >$ is generated by combining pattern pattern $< \{B,C\} >$ and $< \{C\},\{A\} >$. The index lists for pattern $< \{B,C\},\{A\} >$ are generated by taking each index lists of pattern $< \{B,C\} >$, which are $\{1,(1)\}$ $\{4,(4)\}$ and $\{6,(6)\}$, to do range check in index lists of pattern $< \{A\} >$, which are $\{3,(3)\}$, $\{6,(6)\}$, $\{8,(8)\}$, $\{9,(9)\}$. Fig 3.3 shows the index lists for L_1, L_2, L_3 and L_4, respectively, where the S_P underlined means that it is contributed to the frequency counting.

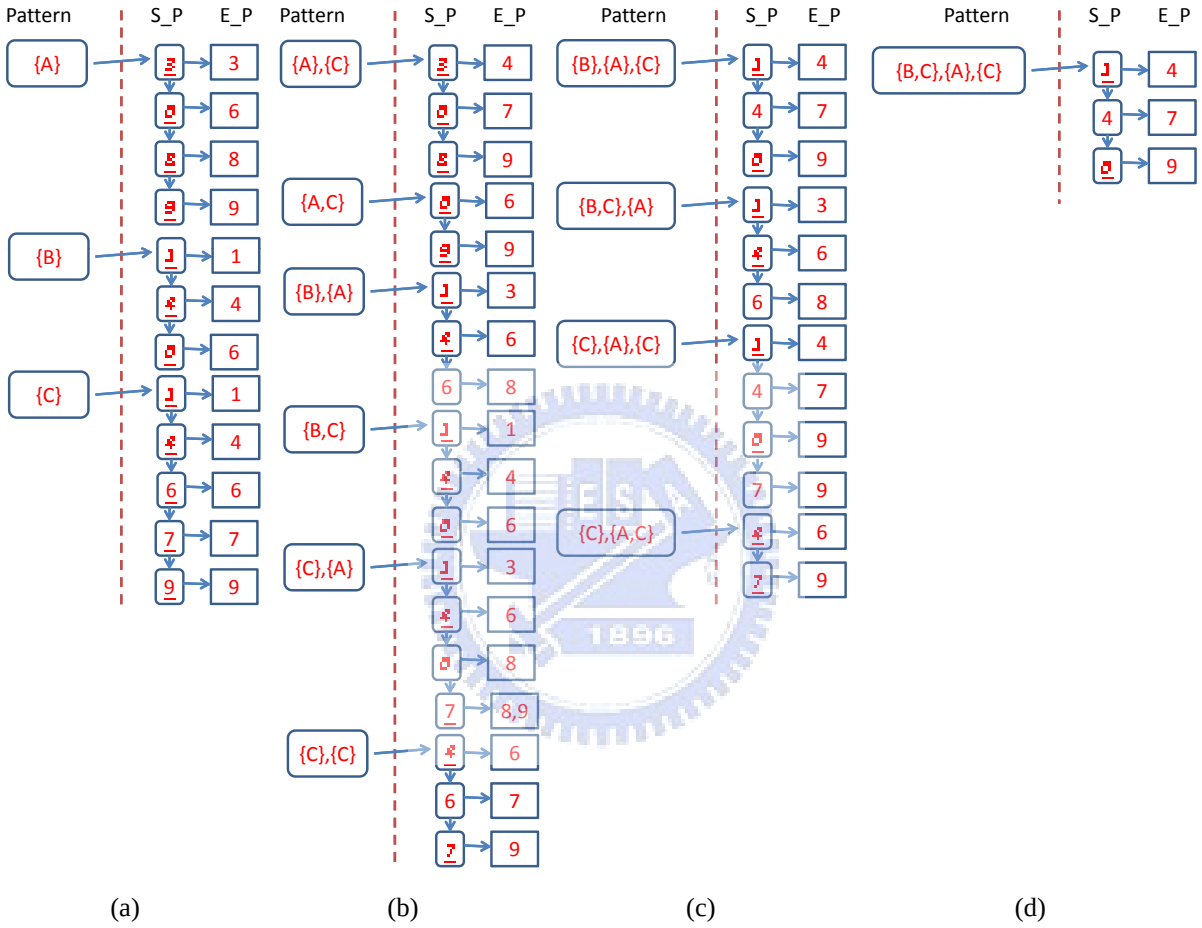


Figure 3.3: The example for GwI-Apriori Algorithm

3.2.1 Pruning Strategies

Because times of comparisons between index lists take much time, in order to resolve this problem, we design the pruning strategies, based on order of index lists for a pattern, to speed up the mining process. The pruning strategies is stated as follows.

(1)**Range pruning:** The general concept is that for a pattern P1, which S_P=ps1 and E_P=pe1, if we want to extend a pattern P2 of size 1 under γ=n, then based on distinct extended type, the the position of pattern P2 must locate at the range (pe1,pe+$n+1$] or [pe1,pe1]. For detailed description, if $IL.E\_P[i]+\gamma+1 < PL[j]$, then we check if $IL.E\_P[i+1]+\gamma+1 < PL[j]$, if *TRUE* then check

$IL.E\_P[i+2]$, if *FALSE* then scan the PL from position j. The reason is that *True* means we have scanned the previous E_P and put result positions into the result index list.

(2)**Last value pruning:** The general concept is that for a pattern P1, which S_P=ps1 and E_P=pe1, if we want to extended a pattern P2 of size 1 under γ=n, then the position of pattern P2 must larger than pe1. For detailed description, if the $PL[i].E\_P[0] \geq$ the last value in PL then stop comparison. The reason is that $PL[i].E\_P[0] \geq$ last value in PL means the S_P of following index lists must $\geq$ last value in PL, so we do not need to scan the following index lists.

Algorithm 4 shows the pruning strategies which is integrated into frequency counting process, where *next_range* is used to range contained pruning and *cur_L_element* is used to last value pruning. Besides Fig 3.4 shows the flow chart for the pruning strategies.

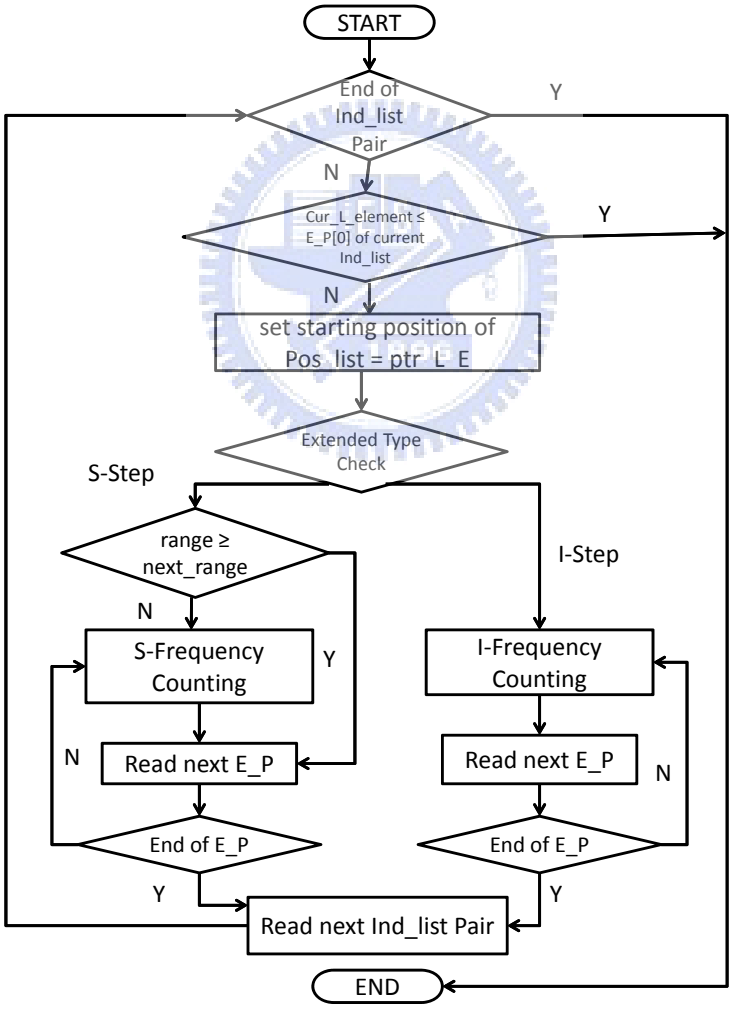


Figure 3.4: Flow Chart for Pruning Strategies

Algorithm 4 *freq_count*

Input: extended type T, index lists of pattern c_i (Ind_list), position list of pattern c_j (Pos_list), threshold δ, Gap Constraint γ

Output: result index lists and count

initialize *next_range*, count, *cur_L_element*, *ptr_L_E*, *L_E_C*=0
{type=0 means S-Step, type=1 means I-Step}
for each index list of pattern c_i **do**
  if $cur\_L\_element \leq Ind\_list.E\_P[0]$ **then**
    break
  i=0
  j=*ptr_L_E*
  for each *E_P* of current *Ind_list* pair;i++ **do**
    $L\_E\_List = NULL$
    if T=0 **then**
      range = $Ind\_list.E\_P[i]$+GC+1
      if range $\geq$ *next_range* **then**
        for each index position ,*Pos_list*, of pattern c_j;j++ **do**
          if $c_j.Pos\_list[j] \leq range \& c_j.Pos\_list[j] > Ind\_list.E\_P[i]$ **then**
            if $Ind\_list.E\_P[i] > L\_E\_C$ & $Ind\_list.S\_P > L\_E\_C$ **then**
              count++;
              $L\_E\_C = c_j.Pos\_list[j]$
              ptr_L_E=j
            Add $c_j.Pos\_list[j]$ to $L\_E\_List.E\_P$
            if $c_j.Pos\_list[j+1]$ = NULL **then**
              $cur\_L\_element = c_j.Pos\_list[j]$
          else if $c_j.Pos\_list[j] >$ range **then**
            $next\_range = c_j.Pos\_list[j]$
            break
    else if T=1 **then**
      $j = ptr\_L\_E$
      for each index position ,*Pos_list*, of pattern c_j;j++ **do**
        if $c_j.Pos\_list[j] = Ind\_list.E\_P[i]$ **then**
          if $Ind\_list.E\_P[i] > L\_E\_C$ & $Ind\_list.S\_P > L\_E\_C$ **then**
            count++;
            $L\_E\_C = c_j.Pos\_list[j]$
            ptr_L_E=j
          Add $c_j.Pos\_list[j]$ to $L\_E\_List.E\_P$
          if $c_j.Pos\_list[j+1]$ = NULL **then**
            $cur\_L\_element = c_j.Pos\_list[j]$
        else if $Ind\_list.E\_P[i]$ ¡ $c_j.Pos\_list[j]$ **then**
          break
        else if $c_j.Pos\_list[j+1]$ = NULL & $Ind\_list.E\_P[i] \neq c_j.Pos\_list[j]$ **then**
          $cur\_L\_element = c_j.Pos\_list[j]$
          break
  $L\_E\_List.S\_P$=$Ind\_list.S\_P$
  Add L_E_List to R_Ind_list
return $[R\_Ind\_list, count]$

An example for pruning strategies

Fig 3.5 shows the example of pruning process for S-Step, where $\gamma = 4$. We start in the first index list of pattern $< \{A\}, \{B\} >$, and scan the index list of pattern $< \{C\} >$. First, we start to scan the first value of E_P list, and sequentially to scan the values in the position list of pattern $< \{C\} >$. When we find the position value which is first satisfied the range contained condition, we record the position value to be following used, then we start to find next value in E_P list. This process terminates until the position value that is not satisfied the range condition, then we add the position value, which is satisfied the range condition, into temporary E_P list. Because the position value, 10, is the last value in index list of pattern $< \{C\} >$, we need to record this position value for future 2nd pruning strategy used. Then we check next value, 3, in E_P list and base on 1st pruning strategy to do pruning process. In this example, $\gamma + 1$ to 3, denoted 8, is not larger than 10, so we start the next value in E_P list. We end this process until all values in E_P list of this index list are all been check. We record S_P and corresponding satisfied position values for E_P list to the result index list, Fig 3.5 (d) shows the result. We start to check next index list. Similarly, we compare the first value in E_P list of this index list to 10, last value in index list for pattern $< \{C\} >$. According to the 2nd pruning strategy, because 3 is smaller than 10, we start the same process that we have explained previous, here we must pay attention that when we start to check next index list of pattern $< \{A\}, \{B\} >$, the first checked value in index lists of pattern $< \{C\} >$ is started in the value which is the last E_P value contributed to the last frequency count. Fig Fig 3.6 (a) shows the last E_P value equals to 3. However, when we check the index list, which S_P is equal to 10, we check the first value in E_P list of this index list, if the position value is larger than the last value in index list of pattern $< \{C\} >$, then we start to check next index list of pattern $< \{A\}, \{B\} >$. In Fig 3.6 (d), the first value in E_P list of this index list, 11, is smaller than 10. Hence based on 2nd pruning strategy, we do not need to check the following index lists.

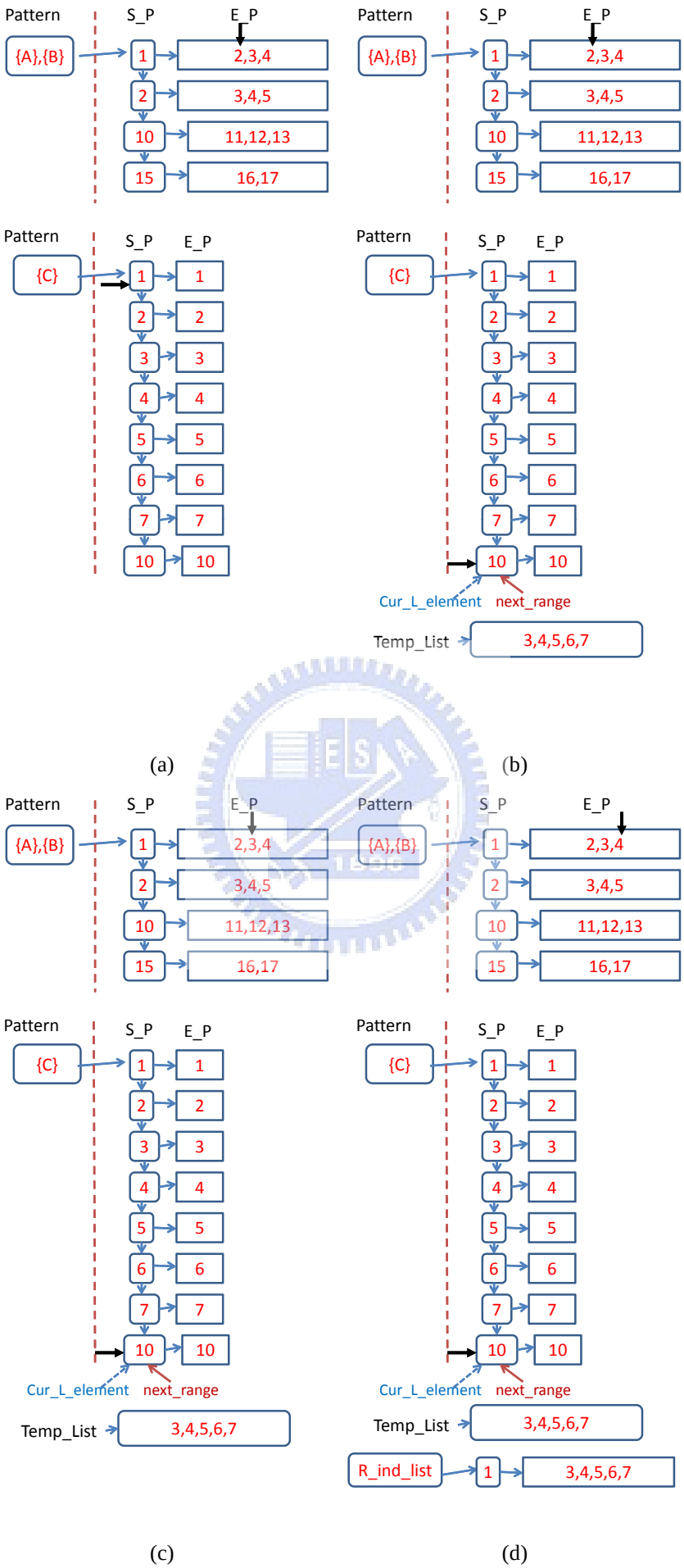


Figure 3.5: The example for Pruning Technique Part I

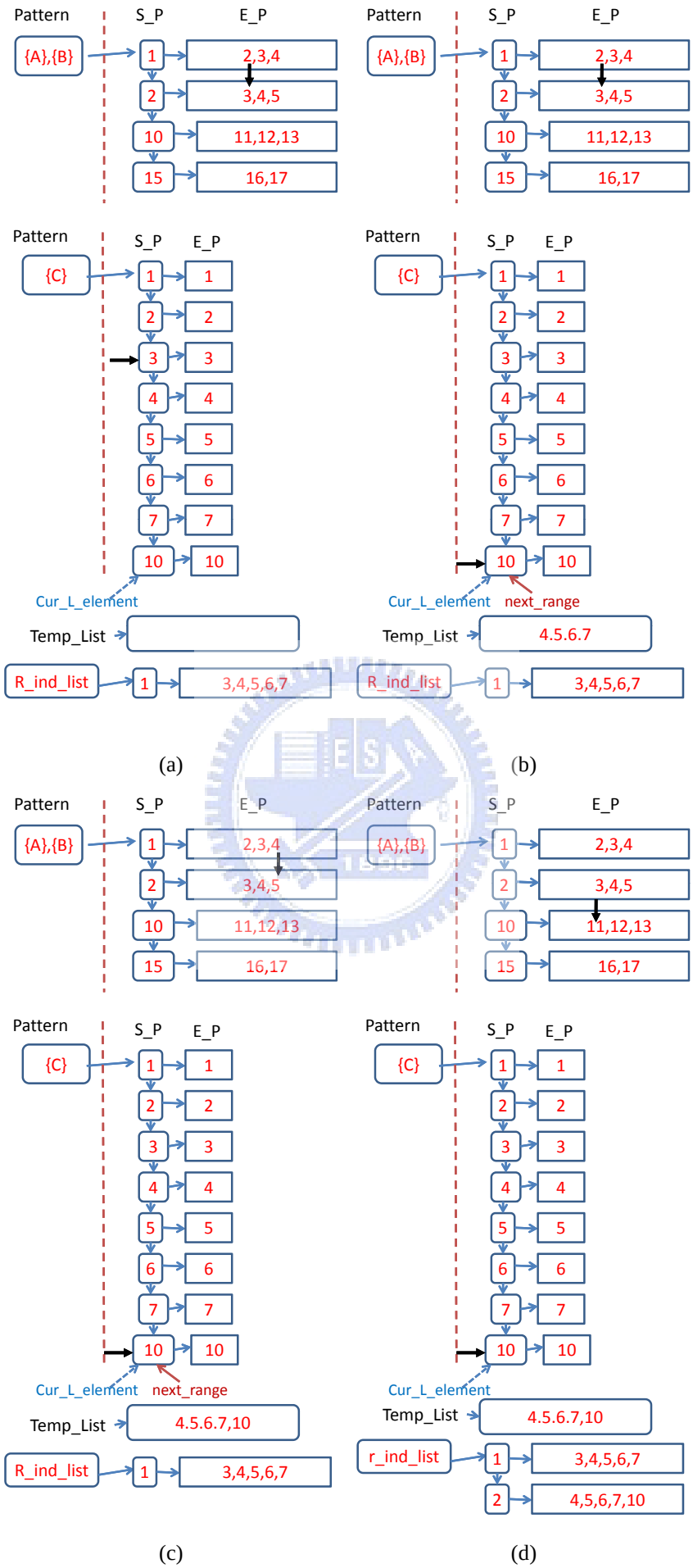


Figure 3.6: The example for Pruning Technique Part II

Chapter 4

Correctness

We also prove the correctness of the G-Apriori algorithm in the following.

Lemma 1: For each pattern P_i in L_2, the each 1 size pattern of P_i is in L_1.

Theorem 1: For each pattern P_i in L_k under GC=m, the k-1 size pattern of P_i is also in L_{k-1} under GC=m.

Proof: Given $P_i =< p_1, p_2, \ldots, p_n >$, where $p_j \subseteq I for 1 \leq j \leq n$ and frequency counting sets under GC=m for P_i. While we delete the first element for P_i to obtain $P\_D\_F =< p\_d\_f_1, p_2, \ldots, p_n >$, we can know FCSG(P) $\subseteq$ FCSG(P_D_F). We delete the last element for P_i to obtain $P\_D\_L =< p_1, p_2, \ldots, p\_d\_l_1 >$, $FCS(P_i) \subseteq$ FCSG(P_D_L). We can clear know that for each pattern P_i in L_k, the k-1 size pattern of P_i is in L_{k-1}.

Lemma 2: Based on the conditions for finding the CBS among the KPCS for pattern P in SD under $GC = m$, the number of instances in CBS is the maximum non overlap instances.

Proof: In here, we need to prove two ideas: 1) Greedy choice property and 2) Optimal substructure property.

(1) Let $S = \{< k_0, n_0 >, < k_1, n_1 >, \ldots, < k_i, n_i >\}$ be a set of instances obtained from the KPCS of P. The instances in S are first sorted by ending positions, after the first stage we have made, if the ending positions of the instances are the same then we need to sort these instances by starting positions progressively. It implies that instances $< k_0, n_0 >$ has the earliest starting position and ending position. Suppose, AS is a subset of S and is an optimal solution then let instances in AS are first ordered by ending positions then ordered by starting position. Suppose, the first instance in AS is $< k_j, n_j >$. If $< k_j, n_j >=< k_0, n_0 >$, then AS begins with greedy choice and we are done. If $< k_j, n_j > \neq < k_0, n_0 >$, we want to show that there is another solution BS that begins with greedy choice, instance $< k_0, n_0 >$. Let $BS = AS - \{< k_j, n_j >\} U \{< k_0, n_0 >\}$. Because $< k_0, n_0 > \leq < k_j, n_j >$, the instances in BS are disjoint and since BS has same number of instances as AS, i.e., $|AS| = |BS|$, BS is also optimal.

(2) Now we prove optimal substructure. If AS is an optimal solution to S, the $AS^{'} = S - \{< k_0, n_0 >\}$ is an optimal solution for $S^{'} = \{< k_p, n_p > \in S | k_p \geq n_0\}$. Therefore, after each greedy choice we are left with an optimization problem of the same from as the original. Induction on the number of choices, the greedy strategy produces an optimal solution.

Theorem 2: The GwI-Apriori algorithm can find maximum frequency for a pattern.

Proof: The frequency counting strategy of GwI-Apriori algorithm for a pattern P based on greedy choice. According the Theorem 2, we can assure that the GwI-Apriori algorithm can find the maximum number of non overlap instances for a pattern, which means the maximum frequency for a pattern.

Chapter 5

Experiment

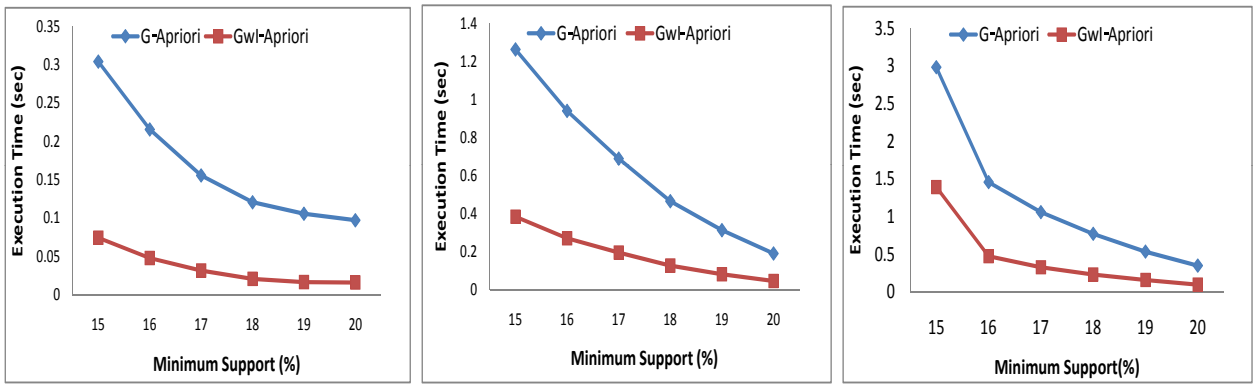
In this section, we present the experiment results of both G-Apriori and GwI-Apriori algorithms. All programs were implemented in Microsoft Visual C++ 6.0. All experiments are performed on Intel Pentium4 CPU 3.20GHz with 1 Gigabytes main memory, running on Linux. For our experimental evaluation we used real data.

We perform our algorithms on real world data, stock data, to get the useful pattern in a set sequence. Stock data are collected form eight companies from Tawian Stock Exchange Daily Official list from January 1, 1995 to December 31, 2007 using Perl and the number of trading days are 3388. We discretize the stock price go-up/go-down into five categories: (1) Up-High(UH): $\leq$ 3.5%, Up-Low(UL): $<$ 3.5% and $>$ 0%, Unbiased(UN): 0%, Down-Low(DL): $>$ -3.5% and $<$ 0%, Down-High(DH): $\leq$ -3.5%. Hence, we have 40 different elements. The average size of the transactions are 8. Table 5.1 shows the companies.

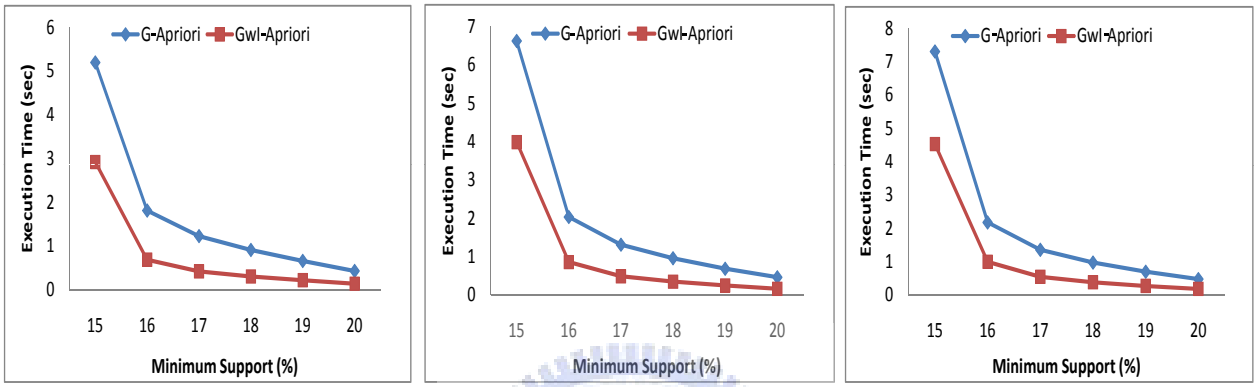
Stock Number	Company Name
2330	TSMC [1]
2308	AELTA [2]
2317	Foxconn [3]
2324	Compal [4]
2311	ASE [5]
2321	TECOM [6]
2312	Kinpo [7]
2313	Compeq [8]

Table 5.1: Stock number and name for companies

[1] http://www.tsmc.com/chinese/default.htm
[2] http://www.delta.com.tw/ch/index.asp
[3] http://www.foxconn.com.tw/
[4] http://www.compal.com/index_En.htm
[5] http://www.asetwn.com.tw/
[6] http://www1.tecom.com.tw/
[7] http://www.kinpo.com.tw/ChineseT/index.htm
[8] http://www.compeq.com.tw/home.htm

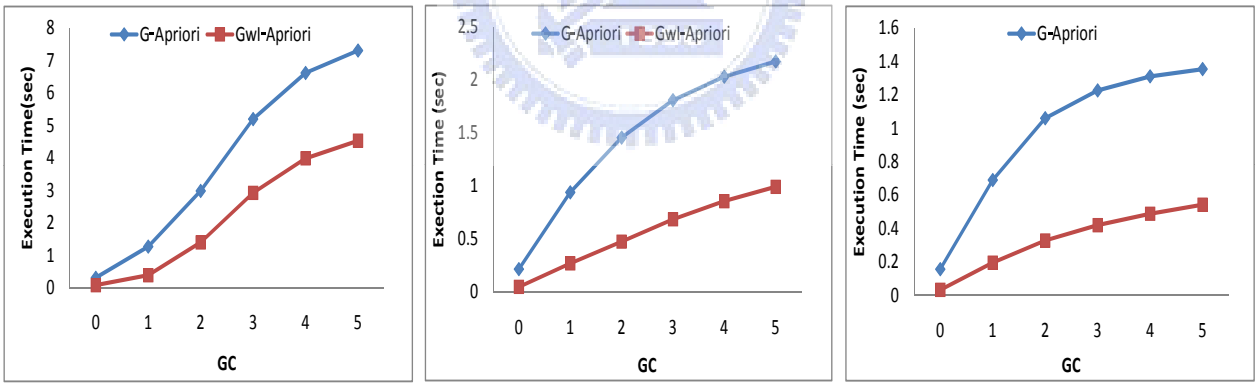


(a) GC=0 (b) GC=1 (c) GC=2

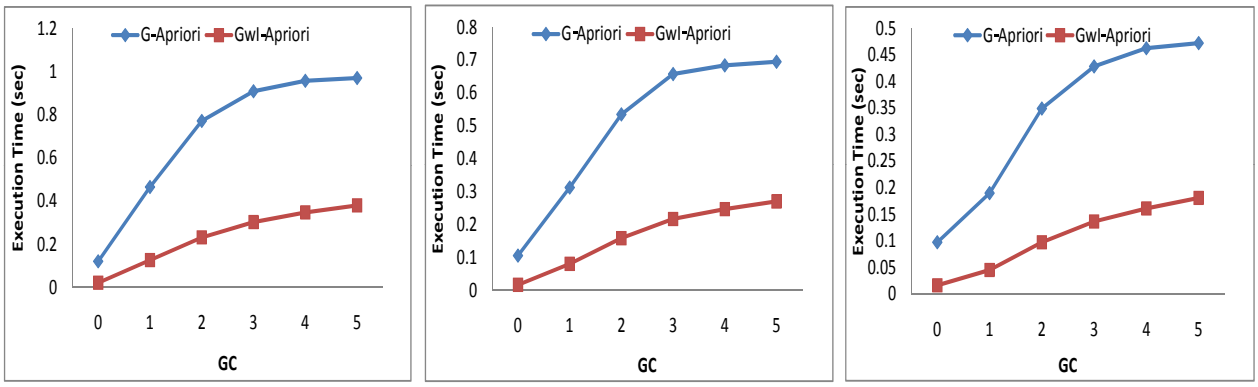


(d) GC=3 (e) GC=4 (f) GC=5

Figure 5.1: Gap constraint versus Execution time



(a) GC=0 (b) GC=1 (c) GC=2

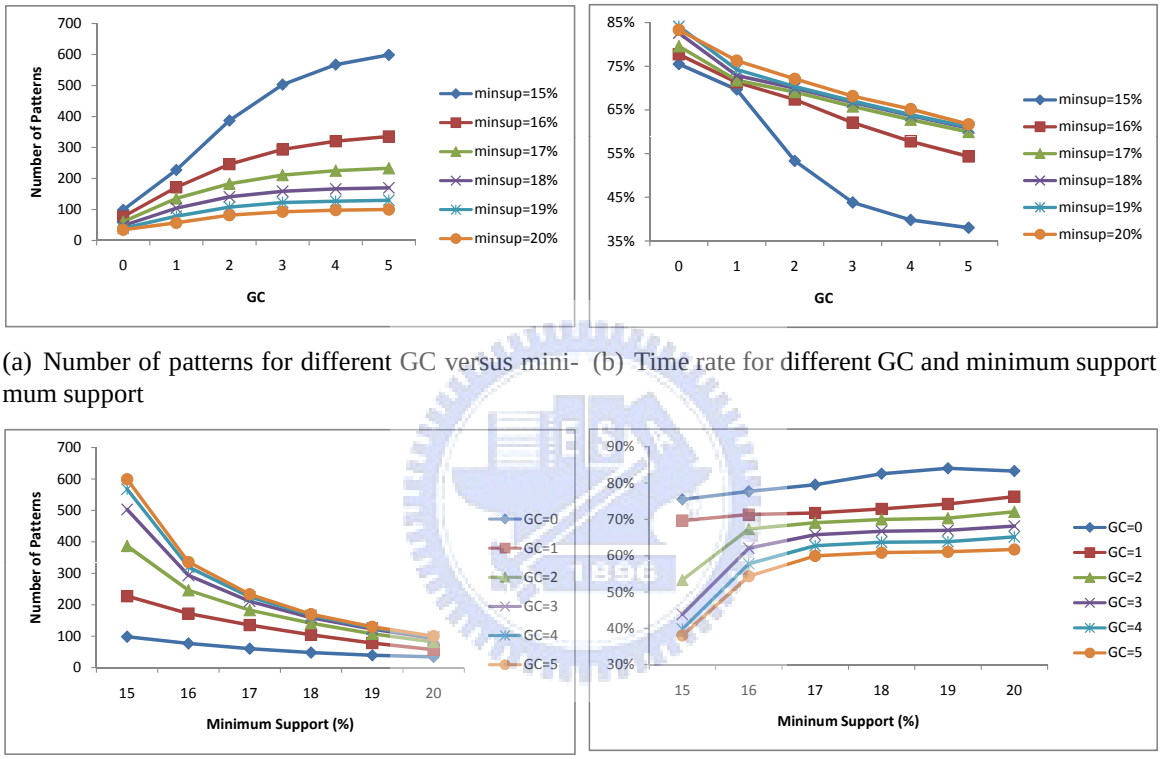


(d) GC=3 (e) GC=4 (f) GC=5

Figure 5.2: Minimum support versus Execution time

The execution time of both algorithms with varying GC are shown in Fig 5.1. From these figures, we can clearly know that when we increase the value of GC, although GwI-Apriori algorithm run faster than the G-Apriori algorithm, the time difference between execution time of them is getting much nearly. The reason is that we may record more index lists needed to compare.

Fig 5.2 shows the execution time of both algorithms with varying minimum support. When the minimum support increase from 15% to 20%, the execution time of both algorithms decrease for the average pattern length being shorter. However, the GwI-Apriori algorithm still performs much better than the G-Apriori as the minimum support increasing.



(a) Number of patterns for different GC versus minimum support

(b) Time rate for different GC and minimum support

(c) Number of patterns for different minimum support versus GC

(d) Time rate for different GC and minimum support

Figure 5.3: Summary Illustration

Fig 5.3 (a) and (c) show the number of frequent patterns for different GC versus minimum support. Besides, in order to clearly show how GwI-Apriori algorithm run faster than G-Apriori algorithm, we define a formula ($\frac{\text{(Execution time of G-Apriori)}-\text{(Execution time of GwI-Apriori)}}{\text{(Execution time of G-Apriori)}}$) and apply this formula to compute the rate of difference of execution time between G-Apriori and GwI-Apriori. Fig 5.3 (b) and (d) show the rate.

Chapter 6

Conclusion

In this paper, we propose a new problem, mining repeating patterns with gap constraint from the set sequence. Besides, we also propose a algorithms, G-Apriori, to mine the repeating pattern with gap constraint. The refined algorithm, GwI-Apriori, is proposed to prevent set sequence from scanning many times to obtain frequent patterns. In GwI-Apriori method, a new data structure is designed to record the appearing start and end positions of a pattern, hence we only need to scan the set sequence once that can save a lot of time to scan database while finding the longer patterns. Besides, the pruning strategies also designed to reduce the comparing times among the index lists. The experimental results show that GwI-Apriori outperforms G-Apriori algorithm. In addition, we can obtain potential repeating patterns while adopting gap constraint.

Bibliography

[1] Bide: Efficient mining of frequent closed sequences.

[2] E. F. Adebiyi, T. Jiang, and M. Kaufmann. An efficient algorithm for finding short approximate non-tandem repeats. *Bioinformatics*, 17(90001):S5–S12, 2001.

[3] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD1993)*, pages 207–216, 1993.

[4] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the 11th International Conference on Data Engineering (ICDE1995)*, pages 3–14, 1995.

[5] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. Sequential pattern mining using a bitmap representation. In *Proceedings of the 8th International Conference on Knowledge Discovery and Data Mining (KDD2002)*, pages 429–435, 2002.

[6] J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. In *Proceedings of the 15th International Conference on Data Engineering (ICDE1999)*, pages 106–115, 1999.

[7] J. Han, W. Gong, and Y. Yin. Mining segment-wise periodic patterns in time-related databases. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD1998)*, pages 214–218, 1998.

[8] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M. Hsu. Freespan: frequent pattern-projected sequential pattern mining. In *Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining (KDD2000)*, pages 355–359, 2000.

[9] J.-L. Hsu and A. L. P. Chen. Efficient repeating pattern finding in music databases. In *Proceedings of Conference on Information and Knowledge Management (CIKM1998)*, pages 281–288, 1998.

[10] J.-L. Hsu, A. L. P. Chen, and H.-C. Chen. Finding approximate repeating patterns from sequence data. In *Proceedings of 5th International Conference on Music Information Retrieval (ISMIR2004)*, 2004.

[11] K.-Y. Huang, C.-H. Chang, and K.-Z. Lin. Prowl: An efficient frequent continuity mining algorithm on event sequences. In *Data Warehousing and Knowledge Discovery, 6th International Conference, (DaWaK 2004)*, pages 351–360, 2004.

[12] K.-Y. Huang, C.-H. Chang, and K.-Z. Lin. Closedprowl: Efficient mining of closed frequent continuities by projected window list technology. In *Proceedings of 5th VLDB Workshop on Secure Data Management (SDM2005)*, 2005.

[13] J.-L. Koh and Y.-T. Kung. An efficient approach for mining top-k fault-tolerant repeating patterns. In *Proceedings of the 11th International Conference on Database Systems for Advanced Applications (DASFAA 2006)*, pages 95–110, 2006.

[14] N.-H. Liu, Y.-H. Wu, and A. L. P. Chen. An efficient approach to extracting approximate repeating patterns in music databases. In *Proceedings of the 10th International Conference on Database Systems for Advanced Applications (DASFAA 2005)*, pages 240–252, 2005.

[15] H. Lu, L. Feng, and J. Han. Beyond intratransaction association analysis: mining multi-dimensional intertransaction association rules. *ACM Transactions on Information Systems (TOIS2000)*, 18(4):423–454, 2000.

[16] H. Mannila and H. Toivonen. Discovering generalized episodes using minimal occurrences. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD1996)*, pages 146–151, 1996.

[17] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. 1(3):210–215, 1995.

[18] B. Özden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. In *Proceedings of the 14th International Conference on Data Engineering (ICDE1998)*, pages 412–421, 1998.

[19] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Transactions on Knowledge and Data Engineering (TKDE2004)*, 16(11):1424–1440, 2004.

[20] A. K. Tung, H. Lu, J. Han, and L. Feng. Breaking the barrier of transactions: mining inter-transaction association rules. In *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining (KDD1999)*, pages 297–301, New York, NY, USA, 1999. ACM.

[21] A. K. H. Tung, H. Lu, J. Han, and L. Feng. Efficient mining of intertransaction association rules. *IEEE Transactions on Knowledge and Data Engineering (TKDE2003)*, 15(1):43–56, 2003.

[22] W. Wang, J. Yang, and P. S. Yu. Mining patterns in long sequential data with noise. *Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD2000)*, 2(2):28–33, 2000.

[23] X. Yan, J. Han, and R. Afshar. Clospan: Mining closed sequential patterns in large databases. In *Proceedings of the 3rd SIAM International Conference on Data Mining (SDM2003)*, 2003.

[24] J. Yang, W. Wang, and P. S. Yu. Mining asynchronous periodic patterns in time series data. *IEEE Transactions on Knowledge and Data Engineering (TKDE2003)*, 15(3):613–628, 2003.

[25] M. J. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31–60, 2001.