國立臺灣大學管理學院資訊管理學研究所

碩士論文

Graduate Institute of Information Management
College of Management
National Taiwan University

Master Thesis

點集合資料庫中封閉性樣式之資料探勘

Mining Closed Patterns in Pointset Databases

陳柏吟

Chen, Po-Yin

指導教授：李瑞庭 博士

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

中華民國 97 年 6 月

June, 2008

時間序列資料庫中封閉性多序列樣式之資料探勘

Mining Closed Patterns in Pointset Databases

本論文係提交國立台灣大學

資訊管理學研究所作為完成碩士

學位所須條件之一部分

研究生：陳柏吟 撰

中華民國九十七年六月

國立臺灣大學碩士學位論文
口試委員會審定書

點集合資料庫中封閉性樣式之資料探勘

Mining Closed Patterns in Pointset Databases

本論文係陳柏吟 君（學號 R95725010）在國立臺灣大學資訊管理學系、所完成之碩士學位論文，於民國 97 年 6 月 4 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：

劉敦仁　　

陳彥良　　李瑞庭

所　　長：　陳靜枝

謝辭

在台大兩年的日子轉眼間過去了，在校園中留下的不只是這篇論文，還有許多美好的回憶與感謝。從論文初稿的提出，寫程式，實驗的進行，到最後一個月的口試與修改，這一路最要感謝的是我的指導教授李瑞庭老師。老師不僅擁有豐富的學識涵養，也樂於和學生分享生活經驗，總能在我遇到問題時，給予最及時的建議與指導。研究所兩年間，感受至深的是老師嚴謹、細心的研究態度，讓我瞭解對於學術論文的每個環節，都要謹慎小心。此外，特別感謝劉敦仁老師、陳彥良老師在口試時給予的寶貴意見，使本篇論文更加完備，在此謹致最誠摯的謝意。

兩年前，懷著忐忑不安的心情踏進這個陌生的環境，貼心的韻茹就像大姊姊一樣帶著我熟悉實驗室與同學，讓我覺得很溫暖。在奕伃、春宏、韋廷、怡安、惠雯及學弟妹們的陪伴下，渡過了充實的兩年研究所生活。亦師亦友的惠雯學姊，不僅在我論文遇到困難時，幫著我一起解決，也陪我一起走過快樂和不開心的日子。還要特別感謝政大資科系鄭仲強學長在程式上的協助與建議，實驗才能順利完成。

謝謝父母從小到大對我的栽培，總能讓我無後顧之憂地在外地念書，也因為他們的支持，才能堅持到最後。也恭喜哥哥今年順利的在英國拿到碩士學位，這段期間哥哥偶而透過 MSN 問候近況，雖然簡短，卻也覺得特別窩心。

僅以此文獻給我最愛的親人、師長與朋友們，感謝你們這些年來陪伴我成長。

陳柏吟 謹識

于台大資訊管理研究所

中華民國九十七年六月

論文摘要

論文題目：點集合資料庫中封閉性樣式之資料探勘

作者：陳柏吟　　九十七年六月

指導教授：李瑞庭　博士

隨著行動運算及定位技術的進步，位置定位服務（Location Based Services，LBS）也跟著蓬勃發展。透過定位裝置，我們可以將大量點座標資料搜集到點集合資料庫中，而點集合是由一些點座標所形成的集合。藉由資料探勘的技術，可以幫助我們在點集合資料庫中發現物體移動的經常路徑或行為。因此，這篇論文將探討如何在點集合資料庫中尋找經常出現的樣式或路徑，我們提出一個有效率的探勘演算法叫「PCP-Miner」，用來找尋點集合資料庫中的封閉性樣式。演算法主要可分為兩個階段。第一階段，我們產生出所有長度為 2 的頻繁樣式。第二階段，我們利用頻繁樣式樹以深先搜尋法的方式遞迴產生所有的頻繁樣式。在產生的過程中，我們會對這些樣式做檢查，檢查它們是否為封閉的。由於 PCP-Miner 只需掃描資料庫一次，且能避免產生不必要的候選樣式，實驗結果顯示，不管在合成資料或真實資料中，我們所提出的方法皆比改良式的 Apriori 演算法有效率。

關鍵詞：資料探勘、點集合資料庫、封閉性樣式

THESIS ABSTRACT

Mining Closed Patterns in Pointset Databases

By Chen, Po-Yin

DEPARTMENT OF INFORMATION MANAGEMENT

NATIONAL TAIWAN UNIVERSITY

June, 2008

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

With advance in mobile computing and positioning technologies, location-based services (LBS) have gained significant progress. By using these technologies, a large amount of pointsets can be collected in an LBS database where a pointset contains a set of points. Mining frequent pointset in a pointset database can help us understand the movement patterns of objects. In this thesis, we proposed a novel algorithm, PCP-Miner (Pointset Closed Pattern Miner), to mine frequent closed pointset patterns. Our proposed algorithm consists of two phases. First, we find all frequent patterns of length two in the database. Second, for each pattern found in the first phase, we recursively generate frequent patterns by a frequent pattern tree in a depth-first search manner. During the process of pattern generation, we check whether the frequent patterns are closed or not. Since the PCP-Miner only needs to scan the database once and doesn't generate unnecessary candidates, it is more efficient than the modified Apriori algorithm. The experiment results show that the PCP-Miner outperforms the modified Apriori by one order of magnitude in both synthetic and real data.

Keywords: data mining, pointset databases, closed patterns.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

With advance in mobile computing and positioning technologies, location-based services have gained significant progress. These technologies can be applied to many daily life applications, such as emergency assistance system, car navigation system, and GIS applications. For example, if we want to know more about customers' behaviors, we may want to find if there exist any customer groups which have similar shopping routes. Then, we can apply a marketing segmentation strategy on them. By using GPS, RFID or other positioning devices to record customers' locations, we can indirectly infer what types of products or stores their paid more attention on, and then understand their preferences and needs.

Since a customer may go forwards and backwards, and repeat the route again and again, we do not consider the orders of locations on the shopping route but consider the locations where he/she has visited. Since a location can be represented by a point in a map, a shopping route can be represented by a set of points (or a pointset). Based on such data, we can find frequently-occurring customers' shopping routes.

Moreover, let us take an online game to be another example, where there are several maps in each scenario or stage in the online game. In order to finish the mission, players may have to go forwards and backwards to get enough information, beat enemies to reach the higher level, or buy necessary items (or devices) to arm themselves. Based on such data, we can analyze player's frequently-moving routes and explain why they prefer these routes.

Nowadays, many frequent itemset mining methods have been proposed [1][9] [11][23][27][34][38]. An itemset is frequent if its support is not less than the user-specified minimum support threshold. The support of an itemset is defined as the number of transactions containing the itemset in the database. Agrawal et al. [1] first proposed the Apriori method [1] to mine frequent itemsets. The Apriori method needs

to generate and test all candidate itemsets to find all frequent itemsets in a database. Compared with Apriori [1], FP-growth [11] finds frequent itemsets without candidate generation by using special data structure called FP-tree. Grahne et al. [9] presented a novel FP-array technique that greatly reduces the need to traverse FP-trees and significantly improves performance for FP-tree-based algorithms. Palshikar et al. [23] proposed the concept of heavy itemset and a greedy algorithm to generate a collection of disjoint heavy itemsets in a transaction database. An itemset is heavy if all possible association rules generated from the items in the itemset satisfy the support and confidence thresholds. Rozenberg et al. [27] dealt with the problem of association rule mining from distributed vertically partitioned data with the goal of preserving the confidentiality of each database. Yao et al. [34] considered the utility constraint to satisfy user's preferred quality. The utility constraint is used to overcome the limitation that a frequent itemset only reflects the statistical correlation between items, but does not reflect its semantic significance. Zaki [38] proposed a vertical data representation, called *Diffset*, that keeps track of the differences between the tids of a candidate pattern and those of the frequent patterns from which it is generated, to reduce the memory requirement of storing intermediate results.

In addition to the frequent itemsets mining, sequential pattern mining was introduced by Srikant et al. [29], where frequently occurring events or subsequences are mined as patterns. Many Apriori-based methods have been proposed such as GSP [29], SPADE [36], SPAM [3], GO-SPADE [20], WSpan [35], SSM [7] and a tree-like sequential patterns mining algorithm [4]. GSP [29] applies the Apriori property to prune candidates generated by the self-join of the sequential patterns found in the previous stage. SPADE [36] uses a vertical data format and a divide-and-conquer strategy to reduce the search space and the number of database scans. SPAM [3] exploits a vertical bitmap representation to generate candidates and count supports efficiently. GO-SPADE [20] extends SPADE [36] to incorporate generalized occurrences to improve mining efficiency. Instead of generating a large number of

candidates and multiple database scans, Han et al. [11] designed the FP-tree structure to mine frequent itemset without candidate generation. Then, the FP-growth method is applied to mine frequent sequential patterns without candidates generation [12][26]. FreeSpan [12] mines sequential patterns by recursively partitioning the search space and projecting the sequence subdatabases. PrefixSpan [26] uses the FP-tree [11] to mine frequent sequential patterns without generating candidates where the prefix projected databases are used to reduce memory usage. Yun et al. [35] designed a weighted sequential pattern mining algorithm, called WSpan, to generate fewer but important weighted sequential patterns. Ezeife et al. [7] proposed an algorithm, called SSM [7], for mining frequent sequential patterns in data streams. Chen et al. [4] presented a dynamic programming-based data mining approach for exploring hierarchical tree structures, named tree-like patterns, representing the relationships for a pair of items in a sequence that can be identified in terms of cause and effect.

Many real-world applications can be model as graphs such as protein structures, objects' structures, etc. Therefore, mining frequent subgraphs has attracted increasing attention recently. Many graph mining algorithms [10][15][17][18][19][33] have been proposed. AGM [17] and FSG [19] use a level-wise approach to mine frequent subgraphs, which combines frequent subgraphs mined at the previous level to generate all candidates at the next level. gSpan [33] generates the frequent subgraphs without candidates generation in a depth-first search manner. FFSM [15] exploits an algebraic graph framework called canonical adjacency matrix (CAM) tree to perform join and extension operation to enumerate all frequent subgraphs unambiguously, and completely avoids subgraph isomorphism testing by maintaining an embedding set for each frequent subgraph. Gudes et al. [10] proposed an edge-disjoint path-based algorithm which adds paths to grow up the candidate subsgraphs to reduce the number of iterations. Jin et al. [18] presented a method based on topological minor concept to eliminate some vertices in a path of a graph which can keep topology structure in the graph. A topological minor of a graph is an abstraction that focuses on its structural

information. So the algorithm can reduce the search space and increase the mining efficiency.

Recently, some data mining methods which take spatial and temporal attributes into account have been proposed. Tsoukatos et al. [30] used the lattice-theoretic approach to decompose the original search space and proposed a depth-first search manner algorithm called DFS_MINE to discover frequent spatiotemporal sequences. Chung et al. [6] presented an Apriori-based method to mine the frequent trajectory patterns in a database, where a series of locations of moving objects is generalized to a sequence of itemsets. Nhan et al. [14] proposed two algorithms for mining frequent spatiotemporal patterns and maximal frequent patterns in the mobile environment. Hwang et al. [16] proposed a trajectory-based algorithm to mine a group of moving objects that have similar trajectories, the pattern mined is a set of moving objects which have similar movement patterns. Giannotti et al. [8] introduced trajectory patterns as concise descriptions of frequent behaviors in terms of both space and time, and developed an extension of the sequential pattern mining paradigm that analyzed the trajectories of moving objects.

Instead of mining all frequent itemsets, Pasquier et al. [24] introduced a new concept to mine the frequent closed itemsets. A frequent itemset X is closed if there does not exist any super-itemset of X with the same support. It is obvious that the number of frequent closed itemsets is smaller than that of frequent itemsets. Moreover, frequent closed itemsets mined can be used to generate a complete set of frequent itemsets [24]. Generally speaking, the algorithm of mining frequent closed itemsets is usually more efficient than that of mining frequent itemsets. A-CLOSE [24], an Apriori-based algorithm, uses a closure mechanism to find frequent closed itemsets. In order to avoid generating all candidates and keep them in the memory, many algorithms have been proposed, such as CLOSET [25], TFP [13], CLOSET+ [32], CHARM [37], DCI_CLOSED [21]. Both CLOSET [25] and CLOSET+ [32] uses the FP-tree as a compact data structure and mines frequent closed itemsets with projected

databases [11]. TFP [13] uses a hybrid mining strategy to mine top-*k* frequent closed patterns without using the minimum support threshold. CHARM [37] uses an itemset-tidset search tree as its data structure and then applies a diffset technique and hash function to increase its performance. DCI_CLOSED [21], based on a bitwise vertical representation of datasets and a divide-and-conquer approach, introduced the equivalence class and order-preserving generator to avoid keeping previously mined closed itemsets in the memory to perform subsumption checking.

Singh et al. [28] designed CloseMiner algorithm to mine closed patterns where they considered the frequent closed pattern mining problem as the problem of clustering the complete set of itemsets with closed tidsets. Uno et al. [31] proposed LCM algorithm, based on the prefix-preserving closure extension of closed patterns, to search all frequent closed patterns in a depth-first search manner so that it does not need any memory storage to store patterns mined. Moonesinghe et al. [22] presented the PGMiner algorithm to construct a prefix graph structure and decomposed the database into bit vectors of variable lengths. Then, they use inter-node and intra-node pruning strategies to substantially reduce the combinatorial search space to mine closed patterns. Cheng et al. [5] proposed an algorithm to mine δ-tolerance closed frequent itemsets (δ-TCFIs) in order to reduce the number of closed itemsets.

The itemset mining algorithms are usually used to find frequent itemsets, but they cannot maintain the relationship between items (i.e., the points in a pointset database form a continuous route). The sequential pattern mining algorithms are often used to mine frequent sequential patterns, where the patterns are ordered by the temporal attribute such as timestamp. However, the spatial attribute is not considered in the sequential pattern mining algorithms. The graph mining algorithms are used to mine frequent subgraphs in a database, where the spatial neighborhood is not considered. The trajectory mining algorithms consider both spatial and temporal dimensions. However, customers may go forwards and backwards, and repeat the route again and again. Consequently, the trajectory mining algorithms generate a large

number of patterns in such a database. Since the patterns mined by the trajectory mining algorithms are too detailed, many of them are redundant. Thus, the itemset mining, sequential pattern mining, graph mining and trajectory mining methods are not suitable for finding frequent patterns in a pointset database.

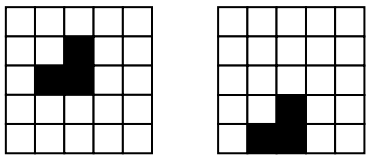
Therefore, in this thesis, based on CHARM [37] and PrefixSpan [26], we propose an efficient algorithm, called PCP-Miner, to mine frequent closed patterns in a pointset database. Our proposed algorithm consists of two phases. First, we mine all frequent patterns of length two (2-patterns) in the database and generate the projected database for each frequent 2-pattern. Second, we recursively generate frequent patterns of length k (k-patterns) by joining (k-1)-patterns in a joinable class in a depth-first search manner, where $k > 2$. During the enumeration process, we apply the pruning properties to prune impossible candidates, and remove the frequent but non-closed patterns. By using the projected database, we scan the database only once and localize the support counting, candidate pruning, and pattern joining in the projected database. Thus, our proposed algorithm can efficiently mine frequent closed patterns in a pointset database.

In summary, the contributions of this thesis are listed as follows. First, we introduce a novel concept for mining frequent closed patterns in pointset databases. Next, during the mining processes, we develop the pruning strategies to reduce the number of candidate patterns and eliminate non-closed patterns. Finally, the experimental results show that our proposed algorithm is efficient and scalable, and outperforms the modified Apriori algorithm.

The rest of this thesis is organized as follows. Chapter 2 introduces the preliminary concepts and problem definitions. Chapter 3 describes our proposed algorithm in detail and presents an example to demonstrate how it works. Chapter 4 shows the experiment setup and performance evaluation. Finally, the conclusions and future work are made in Chapter 5.

Chapter 2 Problem Definition

Consider a data space of m by m in a 2-dementsional space. A database $D = \{t_1, t_2, \ldots, t_n\}$, where t_i is a transaction in a form of $<tid, R>$, tid is a transaction identifier, and R is a route containing a set of points. For example, Figure 1 illustrates a database containing two routes in a data space of 5×5. The route on Figure 1(a) contains a set of points {(1, 2), (2, 1), (2, 2)}, where the rows and columns are numbered from 0, and (1, 2) stands for the point at the first row and the second column.



(a) (b)

Figure 1. A database containing two routes.

A pattern is defined as $P = \{(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)\}$, where the points in P are sorted by x-coordinates and then by y-coordinates, k is the number of points in P, and $x_1 = 0$. A pattern with k points is called a k-pattern. Note that we record a pattern in a relative coordinate system where the point at the upper left corner of the pattern is treated as the reference point of the pattern.

For example, the pattern in Figure 1(a) is denoted by {(0, 1), (1, 0), (1, 1)}, where (1, 1) is the reference point of the pattern. Since it contains 3 points, it is a 3-pattern. Also, we can say that the pattern in Figure 1(b) is the same pattern as that in Figure 1(a).

Definition 1. Let (x_i, y_i) and (x_j, y_j) be two points. $(x_i, y_i) < (x_j, y_j)$ if 1) $x_i < x_j$, or 2) $x_i = x_j$ and $y_i < y_j$. Moreover, $(x_i, y_i) = (x_j, y_j)$ if $x_i = x_j$ and $y_i = y_j$.

Definition 2. Let $P = \{(x_1, y_1), (x_2, y_2), \ldots, (x_p, y_p)\}$ and $Q = \{(x'_1, y'_1), (x'_2, y'_2), \ldots, (x'_q, y'_q)\}$ be two patterns, where $2 \le p$ and $2 \le q$. $Q > P$ if 1) $(x'_1, y'_1) > (x_1, y_1)$, or 2) there exists a positive integer s, such that $(x_i, y_i) = (x'_i, y'_i)$, $i = 1, 2, \ldots, s-1$, and $(x'_s, y'_s) > (x_s, y_s)$.

For example, (1, 2) < (2, 1), (2, 1) < (2, 3), {(0, 0), (1, 0)} < {(0, 0), (1, 2)}, and {(0, 0), (0, 1)} < {(0, 1), (1, 0)}. All the points in a pattern are sorted in ascending order.

Definition 3. Let $P = \{(x_1, y_1), (x_2, y_2), \ldots, (x_p, y_p)\}$ and $P' = \{(x'_1, y'_1), (x'_2, y'_2), \ldots, (x'_q, y'_q)\}$ be two patterns, where $1 \le p \le q$. P is a subpattern of P' if we can find p points in P' and two integers s and t such that $x_i = x'_{ji} - s$, $y_i = y'_{ji} - t$, $1 \le i \le p$, and $1 \le j_1 < j_2 < \ldots < j_p \le q$. We can also say that P' is a super-pattern of P, or P' contains P, denoted as $P \subset P'$.

Definition 4. The *support* of a pattern P, denoted as $sup(P)$, is defined as the number of transactions containing P in the database.

Definition 5. A pattern P is *frequent* if $sup(P)$ is not less than a user-specified minimum support threshold, *min_sup*.

Definition 6. A frequent pattern P is *closed* if there does not exist any super-pattern of P with the same support.

Figure 2 illustrates an example database D which contains four transactions: t_1, t_2, t_3, and t_4. Note that all transactions are also represented in a relative coordinate system. t_1 = {(0, 0), (0, 1), (0, 2), (1, 2), (2, 3), (3, 3), (4, 3)}, t_2 = {(0, 2), (1, 0), (1, 1), (1, 2), (2, 2)}, t_3 = {(0, 2), (1, 0), (1, 1), (1, 2), (1, 3), (2, 2), (3, 2)}, and t_4 = {(0, 2), (1, 0), (1, 1), (1, 2)}. And we can find that $t_4 \subset t_2$, and $t_4 \subset t_3$.

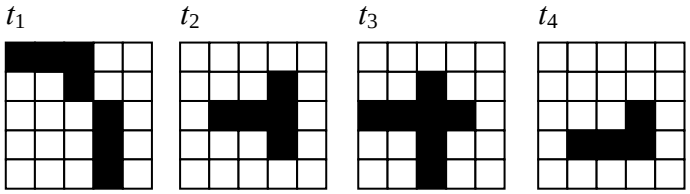


Figure 2. An example database.

Definition 7. Let P be a frequent pattern in a database D. The P-projected database, denoted as $D|_P$, contains a set of transactions in D containing P. $D|_P = \{t(x, y) \mid t$ is a transaction containing P in D, and the starting point of P occurs at (x, y) in $t.\}$ We can also say that $t(x, y)$ is a projected transaction of t. (x, y) is called the starting point

of $t(x, y)$. Note that there may be more than one projected transaction of t containing P. In this case, we will record all those projected transactions in $D|_P$.

Let's consider the database shown in Figure 2 again. Assume that *min_sup* = 3, Since P = {(0, 0), (1, 1)} is contained by t_1, t_2 and t_3, its support is 3. Thus, it is frequent. The projected database of P is $D|_P$ = {t_1(0, 1), t_1(1, 2), t_2(1, 1), t_3(0, 2), t_3(1, 1)}. Note that there are more than one projected transaction of t_1 and t_3 containing P. Let's consider another pattern Q = {(0, 0), (0, 1), (1, 1)} which is contained by t_1, t_2 and t_3. Its support is 3. Thus, it is frequent. Since Q contains P, Q is a super-pattern of P and $sup(Q) = sup(P)$. Therefore, P is not closed.

Definition 8. $D|_P \cap D|_Q$ is defined as the intersection of two projected databases, where $D|_P$ and $D|_Q$ are the projected databases of patterns P and Q, respectively.

For example, let $D|_P$ = {t_1(0, 0), t_1(0, 1), t_2(1, 0), t_2(1, 1), t_3(1, 0), t_3(1, 1), t_3(1, 2), t_4(1, 0), t_4(1, 1)} and $D|_Q$ = {t_1(0, 1), t_1(1, 2), t_2(1, 1), t_3(0, 2), t_3(1, 1)} be the projected databases for patterns P and Q, respectively. $D|_P \cap D|_Q$ = {t_1(0, 1), t_2(1, 1), t_3(1, 1)}.

The objective of mining frequent closed patterns is to find all frequent closed patterns in a pointset database with respect to the user-specified minimum support threshold.

Chapter 3 Our Proposed Method

In this chapter, we propose an algorithm, called PCP-Miner (Pointset Closed Pattern Miner), for mining frequent closed patterns in a pointset database. Our proposed algorithm consists of two phases. First, we find all frequent 2-patterns in the database and generate the projected database for each frequent 2-patterns found. Second, we recursively generate frequent (k+1)-patterns by joining k-patterns in a joinable class (defined in Section 3.1.2) in a depth-first search (DFS) manner, $k \geq 2$. During the enumeration process, we apply the pruning properties described in Section 3.2 to prune impossible candidates and apply the subsumption check to remove frequent but non-closed patterns.

3.1. Frequent pattern enumeration

We use a frequent pattern tree to enumerate frequent patterns in the following way. The root of tree is labeled by ∅. Next, we scan the database and find all frequent 2-patterns in the database and add these 2-patterns to the level 1 of the frequent pattern tree. Then, we recursively extend a frequent k-pattern ($k \geq 2$) at level (k-1) to get its frequent super-pattern (k+1)-patterns at the next level k.

Definition 9. Every node in a frequent pattern tree consists of a pattern P and its projected database $D|_P$, denoted as $P<D|_P>$. Let $P_1<D|_{P_1}>$, $P_2<D|_{P_2}>$, …, $P_m<D|_{P_m}>$ be the m children of $P<D|_P>$ in the frequent pattern tree. Then, we have 1) $P \subset P_1$, $P \subset P_2$, …, $P \subset P_m$, and 2) $P < P_1 < P_2 < \ldots < P_m$.

3.1.1. Generating frequent 2-patterns

Consider a database D where all transactions in the database are in a data space of $m \times m$. In order to generate all candidate 2-patterns, we can divide the candidate 2-patterns into m groups, where $G_1 = \{\{(0, 0), (x, y)\} | \ 0 \leq x \leq m-1, \ 0 \leq y \leq m-1,$ and $(x, y) > (0, 0)\}$, $G_2 = \{\{(0, 1), (x, 0)\} | \ 1 \leq x \leq m-1\}$, $G_3 = \{\{(0, 2), (x, 0)\} | \ 1 \leq x \leq$

$m-1\}$, $G_4 = \{\{(0, 3), (x, 0)\}| 1 \le x \le m-1\}$, …, $G_m = \{\{(0, m-1), (x, 0)\}| 1 \le x \le m-1\}$.

Let us consider the example shown in Figure 3 to illustrate how we generate all candidate 2-patterns. Assume that all transactions in the database D are in a data space of 5×5. The black block in Figure 3 is the fixed point in every group, and the gray areas are the points that can be combined with the fixed point to generate a candidate 2-pattern.

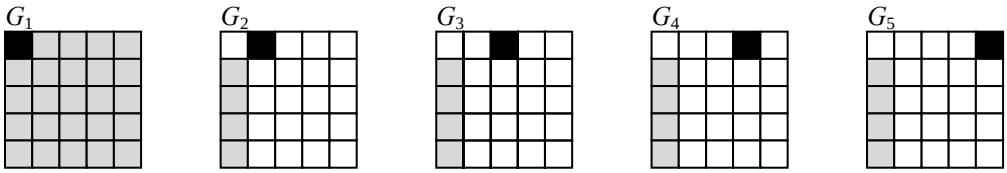


Figure 3. Generating all candidate 2-ptterns.

In G_1, we choose (0, 0) as the fixed point, and combine it with each point (x, y) in the gray area of G_1, where all $(x, y) > (0, 0)$. Therefore, we can derive twenty four 2-patterns in ascending order: {(0, 0), (0, 1)}, {(0, 0), (0, 2)}, {(0, 0), (0, 3)}, {(0, 0), (0, 4)}, {(0, 0), (1, 0)}, {(0, 0), (1, 1)}, …, {(0, 0), (1, 4)}, {(0, 0), (2, 0)}, …, {(0, 0), (2, 4)}, {(0, 0), (3, 0)}, …, {(0, 0), (3, 4)}, {(0, 0), (4, 0)}, …, {(0, 0), (4, 4)}. In G_2, we pick (0, 1) as the fixed point, so we can generate another four 2-patterns in ascending order: {(0, 1), (1, 0)}, {(0, 1), (2, 0)}, {(0, 1), (3, 0)}, {(0, 1), (4, 0)}. In G_3, we pick (0, 2) as the fixed point, we derive another four 2-patterns: {(0, 2), (1, 0)}, {(0, 2), (2, 0)}, {(0, 2), (3, 0)}, {(0, 2), (4, 0)}. Similarly, we pick (0, 3) and (0, 4) as the fixed points to generate remaining eight 2-patterns. Therefore, there are forty candidate 2-patterns in total in the data space of 5×5.

Lemma 1. *There are $2m^2 - 2m$ candidate 2-patterns in a data space of $m \times m$.*

Proof. In a data space of $m \times m$, we can get (m^2-1) candidate 2-patterns in G_1. Also, we can get $(m-1)$ candidate 2-patterns for each group from G_2 to G_m. So there are $(m^2-1)+(m-1)^2 = (2m^2-2m)$ candidate 2-patterns in a data space of $m \times m$. ■

After generating all candidate 2-patterns, we scan the database once to count the

support for each 2-pattern. Then, we record all frequent 2-patterns in the frequent pattern tree and generate their projected databases.

For the database shown in Figure 2, we can find all frequent patterns and their projected databases as shown in Figure 4, where *min_sup* = 3, and the join operation (×) is defined in Section 3.2. The root of the frequent pattern tree is a null pattern {∅} and eight frequent 2-patterns are listed in ascending order at level 1, namely, P_1, P_2, P_3, P_4, P_5, P_6, P_7, and P_8. For P_4 = {(0, 0), (1, 1)}, its projected database is $D|_{P_4}$ ={t_1(0, 1), t_1(1, 2), t_2(1, 1) t_3(0, 2), t_3(1, 1)}. In Figure 4, P_3={(0, 0), (1, 0)} has three children $P_{3\times6}$={(0, 0), (1, 0), (2, 0)}, $P_{3\times7}$={(0, 1), (1, 0), (1, 1)}, and , $P_{3\times8}$={(0, 2), (1, 0), (1, 2)}, where $P_3 \subset P_{3\times6}$, $P_3 \subset P_{3\times7}$, and $P_3 \subset P_{3\times8}$, so $P_{3\times6}$, $P_{3\times7}$ and $P_{3\times8}$ are the super-pattern of P_3. Moreover, the patterns of the children of P_3 are all greater than P_3, that is, $P_3 < P_{3\times6} < P_{3\times7} < P_{3\times8}$.

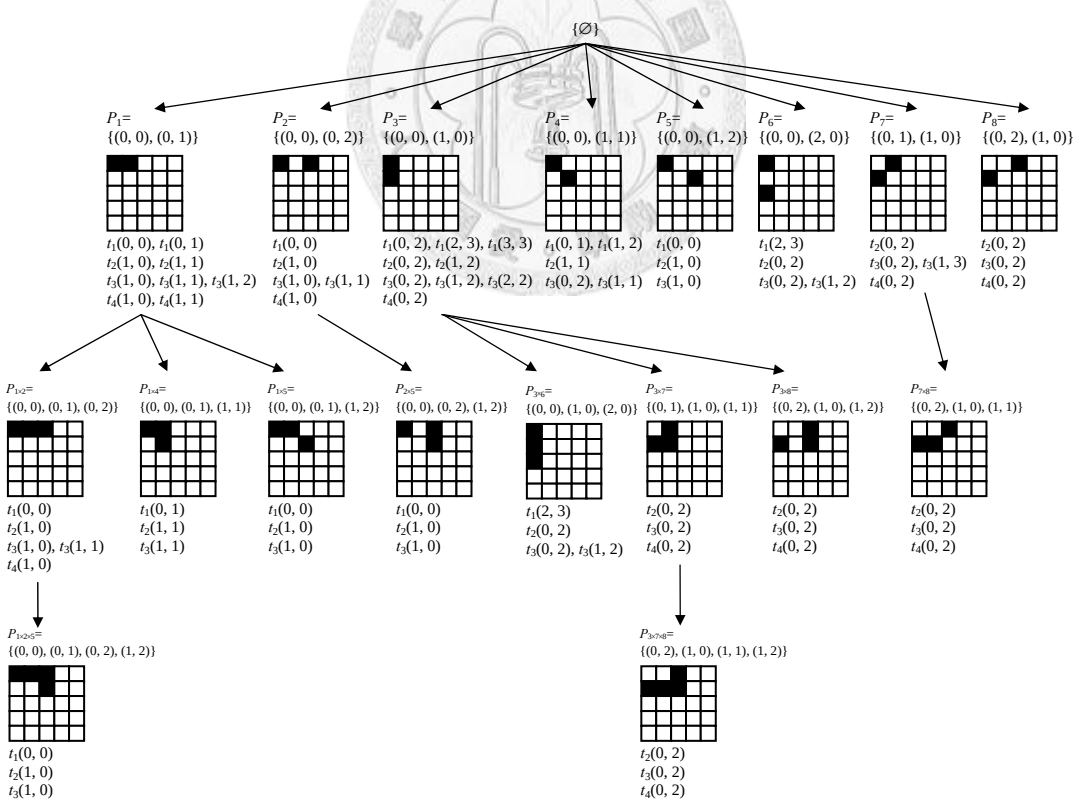


Figure 4. A frequent pattern tree.

3.1.2. Generating frequent *k*-patterns

Next, we recursively extend a frequent *k*-pattern ($k \geq 2$) at level (*k*-1) to get its frequent super (*k*+1)-patterns at level *k* in a depth-first search manner. The extending process relies on the concept of the joinable class.

Definition 10. Let $P<D|_P>$ be a node of a frequent tree. A joinable class, [*P*], contains a set of nodes, each of which is $P<D|_P>$'s child in the frequent pattern tree. Note that any two patterns in a joinable class are joinable, and [{∅}] contains all frequent 2-patterns.

For example, as shown in Figure 4, the root class [{∅}] contains all frequent 2-patterns. For node $P_3<D|_{P_3}>$, its joinable class contains 3 nodes, i.e., $P_{3\times6}<D|_{P_{3\times6}}>$ = {(0, 0), (1, 0), (2, 0)}<{t_1(2, 3), t_2(0, 2), t_3(0, 2), t_3(1, 2)}>, $P_{3\times7}<D|_{P_{3\times7}}>$ = {(0, 1), (1, 0), (1, 1)}<t_2(0, 2), t_3(0, 2), t_4(0, 2)>, and $P_{3\times8}<D|_{P_{3\times8}}>$ = {(0, 2), (1, 0), (1, 2)}<t_2(0, 2), t_3(0, 2), t_4(0, 2)>.

Consider two joinable patterns $P = \{(x_1, y_1), (x_2, y_2), \ldots, (x_p, y_p)\}$ and $Q = \{(x'_1, y'_1), (x'_2, y'_2), \ldots, (x'_q, y'_q)\}$ in an $m \times m$ data space, $2 \leq p \leq q$, $P < Q$. To join *P* and *Q*, if $(x_1, y_1) \neq (x'_1, y'_1)$, we have to shift (x_1, y_1) in *P* to match (x'_1, y'_1) in *Q* to avoid generating duplicate patterns. If any point in the shifted *P* is greater than $(m-1, m-1)$, we say that shifted *P* is not a valid pattern. So the join operation is not allowed.

Definition 11. Let $P = \{(x_1, y_1), (x_2, y_2), \ldots, (x_p, y_p)\}$ and $Q = \{(x'_1, y'_1), (x'_2, y'_2), \ldots, (x'_q, y'_q)\}$ be two joinable patterns in an $m \times m$ data space, where $2 \leq p \leq q$, and $P < Q$. *P* join *Q*, denoted as $P \times Q$, is defined as $P \times Q = P \cup Q$, if $(x_1, y_1) = (x'_1, y'_1)$; otherwise, $P \times Q = P' \cup Q$, where *P'* is the shifted. The projected database of the new pattern $P \times Q$ is $D|_{P\times Q} = D|_P \cap D|_Q$.

For example, consider two joinable patterns P_7 = {(0, 1), (1, 0)} and P_8 = {(0, 2), (1, 0)} in [{∅}], since their first points are not the same, we have to match their first points before a join operation. Therefore, we shift the *y*-coordinates of all points in P_7 by 1 and obtain P'_7={(0, 2), (1, 1)}. Then $P_7 \times P_8 = P'_7 \cup P_8$ ={(0, 2), (1, 1)}∪ {(0, 2),

(1, 0)}={(0, 2), (1, 0), (1, 1)}. The new pattern $P_7 \times P_8$ is denoted as $P_{7\times8}$.

Let's consider the example database shown in Figure 2 again. Its frequent pattern tree is shown in Figure 4, where $P_2 < D|_{P_2} > = \{(0, 0), (0, 2)\} < \{t_1(0, 0), t_2(1, 0), t_3(1, 0), t_3(1, 1), t_4(1, 0)\} >$ and $P_5 < D|_{P_5} > = \{(0, 0), (1, 2)\} < \{t_1(0, 0), t_2(1, 0), t_3(1, 0)\} >$ are two joinable patterns of $[\{\varnothing\}]$, and $P_5 > P_2$. Since both 2-patterns, P_2 and P_5, share the same first point, $P_2 \times P_5 = P_2 \cup P_5 = \{(0, 0), (0, 2), (1, 2)\}$. The projected database of $P_2 \times P_5$ is $D|_{P_{2\times5}} = D|_{P_2} \cap D|_{P_5} = \{t_1(0, 0), t_2(1, 0), t_3(1, 0), t_3(1, 1), t_4(1, 0)\} \cap \{t_1(0, 0), t_2(1, 0), t_3(1, 0)\} = \{t_1(0, 0), t_2(1, 0), t_3(1, 0)\}$. The step of joining P_2 and P_5 to generate a frequent 3-pattern is shown in Figure 5, where *min_sup*=3.

Similarly, we can join two frequent 3-patterns in a joinable class to generate a 4-pattern. For example, $\{(0, 1), (1, 0), (1, 1)\} < \{t_2(0, 2), t_3(0, 2), t_4(0, 2)\} >$ and $\{(0, 2), (1, 0), (1, 2)\} < \{t_2(0, 2), t_3(0, 2), t_4(0, 2)\} >$ are in the joinable class of $[\{(0, 0), (1, 0)\}]$. Since the first points of both patterns are different, we shift the *y*-coordinates of all points in the former pattern by 1 and obtain the shifted pattern, $\{(0, 2), (1, 1), (1, 2)\}$. By joining both patterns, we can obtain a 4-pattern $\{(0, 2), (1, 1), (1,2)\} \cup \{(0, 2), (1, 0), (1, 2)\} = \{(0, 2), (1, 0), (1, 1), (1, 2)\}$.

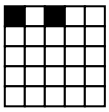
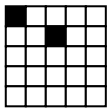
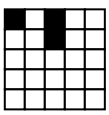
2-pattern	$P_2 = \{(0, 0), (0, 2)\}$ 	$P_5 = \{(0, 0), (1, 2)\}$ 
Projected database	$D\|_{P_2} =$ $\{t_1(0, 0), t_2(1, 0), t_3(1, 0), t_3(1, 1), t_4(1, 0)\}$	$D\|_{P_5} =$ $\{t_1(0, 0), t_2(1, 0), t_3(1, 0)\}$
3-pattern	 $P_2 \times P_5 = P_2 \cup P_5 = \{(0, 0), (0, 2), (1, 2)\}$ $D\|_{P_{2\times5}} = D\|_{P_2} \cap D\|_{P_5} = \{t_1(0, 0), t_2(1, 0), t_3(1, 0)\}$ Thus $sup(P_2 \times P_5) = 3$	

Figure 5. Joining P_2 to P_5 to generate a frequent 3-pattern.

3.2. The closure checking and pruning strategies

Based on the method described in Section 3.1, we can generate all frequent patterns. To find all frequent closed patterns, we apply the similar concept used in

CHARM [37] for closure checking. Thus, we have the following four pruning strategies to mine frequent closed patterns, where P_1, P_2 are frequent pattern in a joinable class [C], and $P_3 = P_1 \times P_2$.

1. If $D|_{P_1} = D|_{P_2}$, $D|_{P_3} = D|_{P_1} \cap D|_{P_2} = D|_{P_1} = D|_{P_2}$. Thus, we can simply replace every occurrence of P_1 with P_3, and remove $P_2 < D|_{P_2} >$ from the frequent pattern tree (i.e., [C]), since it is not closed.
2. If $D|_{P_1} \subset D|_{P_2}$, $D|_{P_3} = D|_{P_1} \cap D|_{P_2} = D|_{P_1}$. We replace every occurrence of P_1 with P_3.
3. If $D|_{P_1} \supset D|_{P_2}$, $D|_{P_3} = D|_{P_1} \cap D|_{P_2} = D|_{P_2}$. In this case, we add $P_3 < D|_{P_3} >$ to [P_1] and remove $P_2 < D|_{P_2} >$ from [C], since P_3 occurs in wherever P_2 occurs.
4. If $D|_{P_1} \neq D|_{P_2}$, we cannot eliminate any pattern of both P_1 and P_2, just add $P_3 < D|_{P_3} >$ to [P_1].

The pruning strategies can be summarized as follows. If both projected databases of P_1 and P_2 are equal, one of them is pruned (Property 1). If the projected database of one pattern is a subset of the projected database of another, the former pattern is replaced by the pattern joined by P_1 and P_2 (Property 2 or 3). Finally, if both projected databases of P_1 and P_2 are not equal, we cannot eliminate any pattern.

Next, we demonstrate how the four pruning strategies are applied to the frequent pattern tree as show in Figure 6, where the root node has four children, {(0, 0), (1, 0)} <t_1(0, 2), t_1(2, 3), t_1(3, 3), t_2(0, 2), t_2(1, 2), t_3(0, 2), t_3(1, 2), t_3(2, 2), t_4(0, 2)>, {(0, 0), (0, 2)}< t_1(2, 3), t_2(0, 2), t_3(0, 2), t_3(1, 2)>, {(0, 1), (1, 0)}<t_2(0, 2), t_3(0, 2), t_3(1, 3), t_4(0, 2)>, and {(0, 2), (1, 0)}<t_2(0, 2), t_3(0, 2), t_4(0, 2)>. Assume that *min_sup*=3.

First, when joining P_3 and P_6, we find that $D|_{P_3} \supset D|_{P_6}$. Therefore, property 3 can be applied, $P_{3\times6} < D|_{P_{3\times6}} >$ is added to [P_3], and P_6<P_6> is removed from [{∅}]. Next, we join P_3 and P_7. By joining P_3 and P_7, we find $D|_{P_3} \neq D|_{P_7}$ (i.e., property 4), so we cannot eliminate any pattern. Therefore, we add $P_{3\times7} < D|_{P_{3\times7}} >$ to [P_3]. Next, by joining P_3 and P_8, we find that $D|_{P_3} \supset D|_{P_8}$. Thus, we add $P_{3\times8} < D|_{P_{3\times8}} >$ to [P_3]

and prune P_8<P_8> from [{∅}]. Now, there are three children in [P_3]. We start processing [P_3] by joining $P_{3\times6}$ and $P_{3\times7}$, and then joining $P_{3\times6}$ and $P_{3\times8}$, but their supports are less than the minimum support. Finally, we join $P_{3\times7}$ and $P_{3\times8}$, and find that $D|_{P_{3\times7}} = D|_{P_{3\times8}}$ (i.e., property 1). Thus, we replace $P_{3\times7} < D|_{P_{3\times7}} >$ with $P_{3\times7\times8} < D|_{P_{3\times7\times8}} >$ and remove $P_{3\times8} < D|_{P_{3\times8}} >$ from [P_3]. Then adding $P_{3\times7\times8} < D|_{P_{3\times7\times8}} >$ to $P_{3\times7} < D|_{P_{3\times7}} >$.

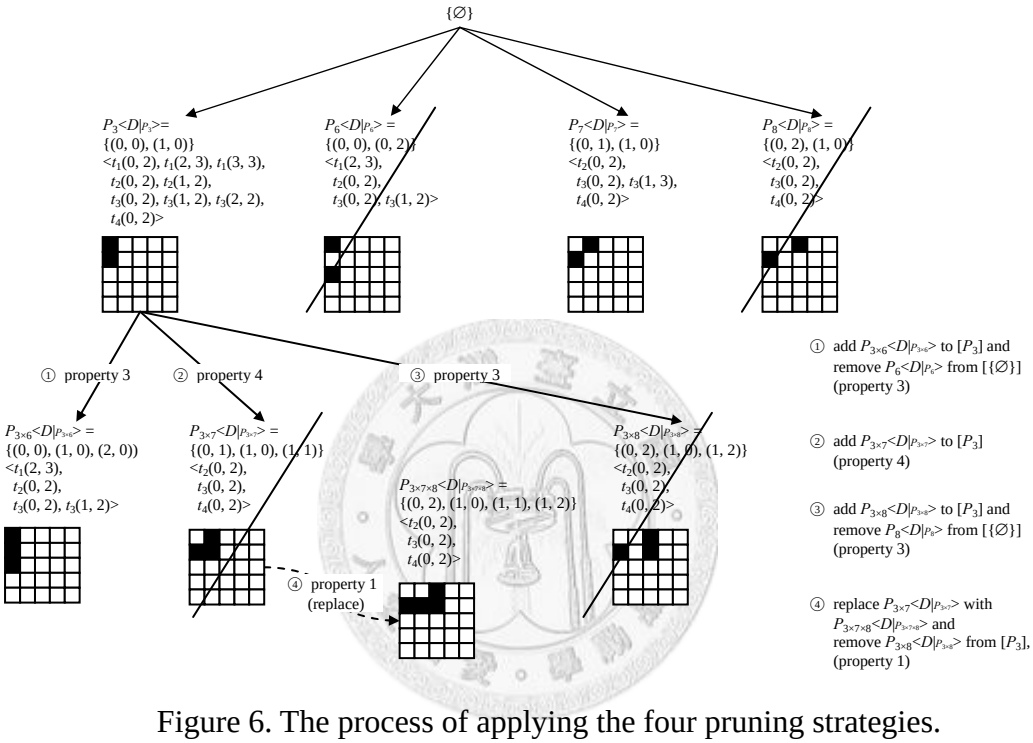


Figure 6. The process of applying the four pruning strategies.

3.3. The PCP-Miner algorithm

The PCP-Miner algorithm is shown in Figure 7, which contains a procedure, called PCP-Growth. The procedure is shown in Figure 8.

In step 1 of the PCP-Miner algorithm, we scan the database once to find all frequent 2-patterns and generate the projected database for each frequent 2-pattern found. Then, we add those frequent 2-patterns and their projected databases to the frequent pattern tree in step 2. In step 4, starting from the root joinable class [{∅}], we recursively call the PCP-Growth procedure to enumerate all closed patterns in a depth-first search manner.

Algorithm: PCP-Miner

Input: an input database *D*, and a minimum support threshold *min_sup*.

Output: all closed patterns *CP*.

1 Scan the database *D* to find all frequent 2-patterns and generate the projected database for each frequent 2-pattern found.

2 Add those frequent 2-patterns and their projected databases (i.e., $P<D|_P>$) to the frequent pattern tree.

3 Let *CP* be ∅;

4 Call *PCP-Growth* ([{∅}], *CP*, *min_sup*) ;

5 **return** *CP*; //all closed patterns

Figure 7. The PCP -Miner algorithm.

In the PCP-Growth procedure as shown in Figure 8, for each pattern in class [*C*], we generate its frequent super-pattern by joining P_i to P_j in steps 1-3, where P_j is a joinable pattern of P_i in [*C*], and $P_j > P_i$. If newly generated pattern $P_{i\times j}$ is frequent, we apply the pruning strategies described in Section 3.2 to prune non-closed patterns in steps 4 and 5. By applying those pruning strategies, class [*C*] may be modified and the newly generated frequent super-patterns are added to [P_i].

In step 7, after joining P_i to all joinable patterns in [*C*], we apply the subsumption check on P_i to see if P_i is closed when adding P_i into the closed pattern set *CP*. The step of subsumption check is to see if there exists any super-pattern of P_i with the same support in *CP*. If *CP* contains any super-pattern of P_i with the same support, P_i is not added into *CP*; otherwise, it is added. Also, we need to check if *CP* has any subpatterns of P_i with the same support. If this is the case, we have to remove these subpatterns from *CP*.

In step 8, we recursively call the PCP-Growth procedure to generate the frequent super-patterns of P_i in a depth-first search manner until no more frequent patterns can be found.

Procedure: PCP-Growth ([*C*], *CP*, *min_sup*)

Input: the joinable class [*C*] to be processed, the set of the closed patterns *CP*, and a minimum support threshold *min_sup*.

Output: a set of closed patterns *CP*.

1 **for each** pattern P_i in [*C*] **do**

2 **for each** pattern P_j in [*C*] with $P_j > P_i$ **do**

3 $P_{i\times j} = P_i \times P_j$; // P_i join P_j

4 **if** $sup(P_{i\times j}) \geq min\_sup$ **then**

5 Apply the pruning strategies described in Section 3.2 to update class [*C*] and add the newly generated super-patterns to $[P_i]$.

6 **end for**

7 Apply the subsumption check on P_i. If P_i is closed, add P_i into *CP* and remove non-closed patterns from *CP*.

8 **if** $[P_i] \neq \varnothing$ **then** call PCP-Growth($[P_i]$, *CP*, *min_sup*);

9 **end for**

Figure 8. The PCP-Growth procedure.

Lemma 2. *Every pattern obtained by the PCP-Miner algorithm is frequent and closed.*

Proof. Because 1) by step 1 of Figure 7, every 2-pattern must be frequent; 2) by using the join operation in step 3 and the examination operation in step 4 of Figure 8**,** every *k*-pattern must be frequent, where $k \geq 2$. Therefore, by the above analyses, we can conclude that every pattern generated by the PCP-Miner must be frequent. Meanwhile, we apply the subsumption checking in step 7 of Figure 8**.** Thus, all the frequent patterns kept in *CP* must be closed. Therefore, we can conclude that every pattern mined by the PCP-Miner algorithm must be frequent and closed. ■

Lemma 3. *The PCP-Miner algorithm finds every frequent closed pattern in the database.*

Proof. We prove this by considering the patterns of various lengths mined by the algorithm. First, by scanning the database in step 1 of Figure 7, we can find all frequent 2-patterns. Next, from the Definition 10 and Definition 11, the join operations of k-patterns in step 3 of Figure 8 can be used to find all frequent (k+1)-patterns, where $k \geq 2$. By the closure checking and pruning strategies stated in Section 3.2, we can prune the nodes in a frequent pattern tree whose patterns are not closed without missing any closed frequent patterns. In addition, we apply the subsumption check in step 7 of Figure 8 to remove the non-closed patterns in *CP*. Thus, all patterns kept in *CP* must be frequent and closed. Therefore, we can conclude that the PCP-Miner algorithm finds every frequent closed pattern in the database. ■

Theorem 1. *The PCP-Miner algorithm can enumerate all frequent closed patterns.*

Proof. Based on the lemmas 2 and 3, we can conclude that the PCP-Miner algorithm can find every closed frequent pattern in the database and every pattern found by the algorithm is frequent and closed. ■

Besides mining the frequent closed patterns in a relative coordinate system, the PCP-Miner algorithm can be applied to finding the frequent closed patterns in an absolute coordinate system. To do so, the transactions in the database are recorded in an absolute coordinate system. Then, we generate candidate 2-patterns in a different way. In a data space of $m \times m$, there are $C_2^{m^2}$ candidate 2-patterns instead of $2m^2$-$2m$. We can divide the candidate 2-patterns into (m-1) groups as shown in Figure 9, where $G_1 = \{(0, 0), (x, y) \mid 0 \leq x \leq m-1, 0 \leq y \leq m-1, \text{ and } (x, y) > (0, 0)\}$, $G_2 = \{(0, 1), (x, y) \mid 0 \leq x \leq m-1, 0 \leq y \leq m-1, \text{ and } (x, y) > (0, 1)\}$, $G_3 = \{(0, 2), (x, y) \mid 0 \leq x \leq m-1, 0 \leq y \leq m-1, \text{ and } (x, y) > (0, 2)\}$, …, $G_{m\text{-}1} = \{(m\text{-}1, m\text{-}2), (x, y) \mid 0 \leq x \leq m-1, 0 \leq y \leq m-1, \text{ and } (x, y) > (m\text{-}1, m\text{-}2)\}$.

After generating all candidate 2-patterns, we scan the database once to count the support for each 2-pattern. Then, we record all frequent 2-patterns in the frequent pattern tree and generate their projected database. However, the definition of the

projected database should be modified slightly as follows: the projected database of pattern P, $D|_P$ = {$t(x, y)$ | t is a transaction containing P in D, and the first point of P, occurs at (x, y) in t.} Let's consider the database shown in Figure 2 again, but using the absolute coordinate system. The projected database of P = {(2, 2), (3, 3)} is $D|_P$ = {$t_2(2, 2)$, $t_3(2, 2)$}.

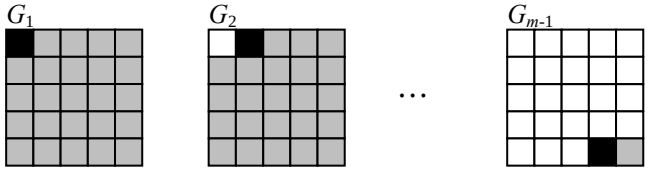


Figure 9. Generating all candidate 2-pattenrs in an absolute coordinate system.

After generating all frequent 2-patterns, the procedures of pattern join, candidate pruning and subsumption check used in the relative coordinate system can be applied to mining frequent closed patterns in an absolute coordinate system.

3.4. An example

In this section, we use the database shown in Figure 2 to illustrate how the PCP-Miner algorithm works in a relative coordinate system. Assume that *min_sup* = 3. The corresponding frequent pattern tree is shown in Figure 10, where a pattern marked with a slash means that the pattern is pruned during the mining process, a pattern marked with a dotted slash means that the pattern is non-closed, an arrow from node A to node B means that the pattern of A is extended from that of B, and a dotted arrow from node A to node B means that the pattern of node A is replaced by that of B.

First, we generate all candidate 2-patterns and scan the database once to count the support for each 2-pattern. We can find eight frequent 2-patterns as follows: {(0, 0), (0, 1)}, {(0, 0), (0, 2)}, {(0, 0), (1, 0)}, {(0, 0), (1, 1)}, {(0, 0), (1, 2)}, {(0, 0), (2, 0)}, {(0, 1), (1, 0)}, and {(0, 2), (1, 0)}. All these patterns with their projected databases are added to the first level of the frequent pattern tree as shown in Figure 10. That is, [{∅}] contains all frequent 2-patterns and their projected databases.

Starting from the first node of [{∅}], $P_1 < D|_{P_1} >$, we join P_1 to the rest patterns in [{∅}]. When joining P_1 and P_2, we obtain a frequent 3-pattern $P_{1\times2}$= {(0, 0), (0, 1), (0, 2)}. Since $D|_{P_1} \supset D|_{P_2}$ (property 3), we add $P_{1\times2} < D|_{P_{1\times2}} >$ to [P_1] and remove $P_2 < D|_{P_2} >$ from [{∅}]. Then, we join P_1 and P_3, but we find that newly generated patterns are not frequent. When joining P_1 to P_4, we obtain a frequent 3-pattern $P_{1\times4}$= {(0, 0), (0, 1), (1, 1)} and add $P_{1\times4} < D|_{P_{1\times4}} >$ to [P_1]. Since $D|_{P_1} \neq D|_{P_4}$ (property 4), we cannot eliminate any pattern of both P_1 and P_4. Next, by joining P_1 to P_5, we obtain another frequent 3-pattern $P_{1\times5}$ = {(0, 0), (0, 1), (1, 2)}. Since $D|_{P_1} \supset D|_{P_5}$ (property 3), we add $P_{1\times5} < D|_{P_{1\times5}} >$ to [P_1] and remove $P_5 < D|_{P_5} >$ from [{∅}]. Next, when joining P_1 to P_6, P_7, and P_8, no frequent 3-patterns can be generated. After processing the first node of [{∅}], we apply the subsumption check to remove non-closed patterns. Because *CP* is empty, we just add P_1 into *CP*.

Now, [P_1] has three children. We next start processing the first node in [P_1]. By joining $P_{1\times2}$ and $P_{1\times4}$, the support of $P_{1\times2\times4}$ is less than the minimum support threshold. By joining $P_{1\times2}$ to $P_{1\times5}$, we obtain a frequent 4-pattern $P_{1\times2\times5}$ ={(0, 0), (0, 1), (0, 2), (1, 2)}. Since $D|_{P_{1\times2}} \supset D|_{P_{1\times5}}$ (property 3), we add $P_{1\times2\times5} < D|_{P_{1\times2\times5}} >$ to [$P_{1\times2}$] and remove $P_{1\times5} < D|_{P_{1\times5}} >$ from [P_1]. Having processed the first node in [P_1], we apply the subsumption check to see if all patterns in *CP* are closed when we are going to add $P_{1\times2}$ into the *CP*. Then, we find that $P_{1\times2}$ is P_1's super-pattern and both have the same support (i.e., $sup(P_{1\times2}) = sup(P_1) = 4$), so we remove P_1 from *CP* and add $P_{1\times2}$ to *CP*.

Next, we recursively process the first and the only one node in [$P_{1\times2}$]. We can directly perform the subsumption check because there are no nodes can be joined to $P_{1\times2\times5}$. After the subsumption check, we add $P_{1\times2\times5}$ into *CP*.

Next, we process the second node in [P_1] and find there is no other frequent patterns can be joined to $P_{1\times4}$ because $P_{1\times5} < D|_{P_{1\times5}} >$ has been pruned. Thus, we directly perform the subsumption check on $P_{1\times4}$ and find that $P_{1\times4}$ is $P_{1\times2\times5}$'s subpattern and both have the same support (i.e., $sup(P_{1\times2\times5}) = sup(P_{1\times4}) = 3$).

Therefore, $P_{1\times4}$ will not be added into *CP*. So far, we have finished processing the branch of $P_1 < D|_{P_1} >$.

Similarly, we can apply the PCP-Growth procedure to the rest nodes in [{∅}] and the frequent closed patterns found are shown in Figure 10. Finally, we obtain five closed patterns as follows: {(0, 0), (1, 0)}, {(0, 0), (0, 1), (0, 2)}, {(0, 0), (1, 0), (2, 0)}, {(0, 0), (0, 1), (0, 2), (1, 2)}, and {(0, 2), (1, 0), (1, 1), (1, 2)}.

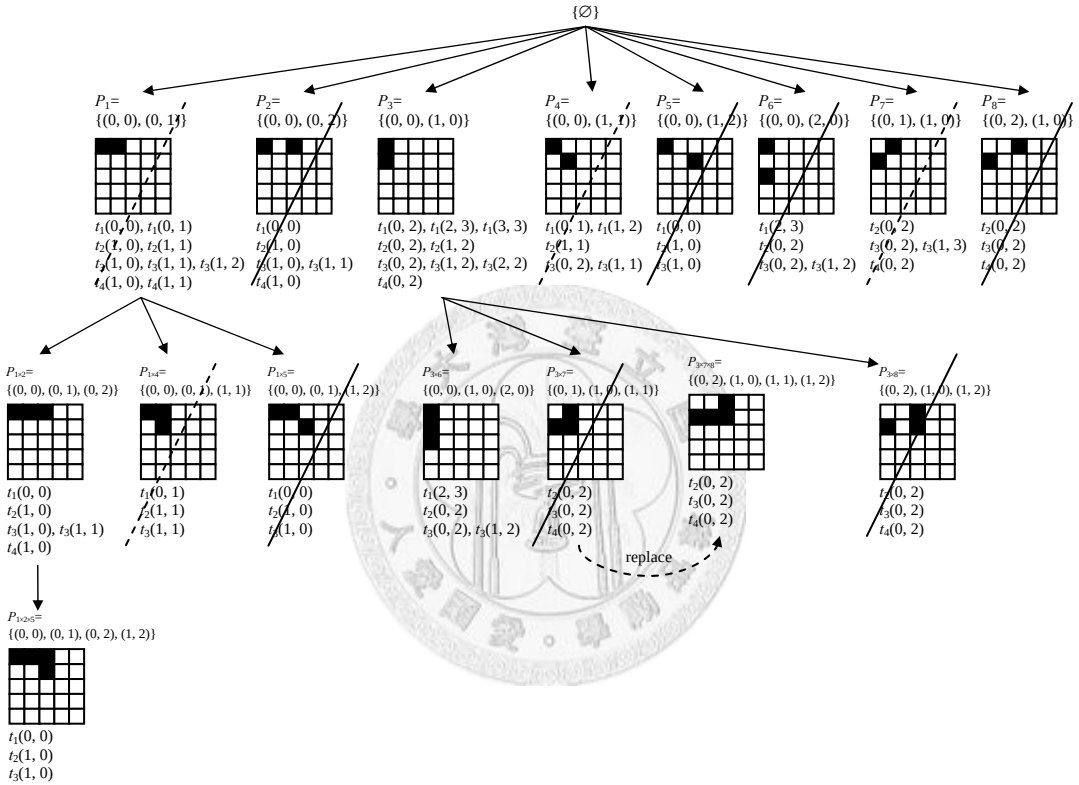


Figure 10. The frequent pattern tree for the database shown in Figure 2.

Chapter 4 Performance Analysis

In this chapter, we conducted the experiments by using both synthetic and real datasets and compared our proposed method with the modified Apriori algorithm [6]. Both algorithms were implemented by Microsoft Visual C++ 2005. All of experiments were performed on an IBM Compatible PC with Intel Core 2 Quad CPU Q6600 @ 2.40GHz, 2.0GB main memory, running on Windows XP professional.

The modified Apriori algorithm generates frequent patterns level by level in a breadth-first search manner. At each level, it combines two frequent *k*-patterns to generate a candidate (*k+1*)-pattern. For each candidate (*k+1*)-pattern, we scan the database to count its support and check if it is frequent. The process is repeated until no more frequent patterns can be generated. The modified Apriori algorithm uses only the anti-monotone property to prune the impossible candidates.

To compare with the modified Apriori algorithm, the PCP-Miner will generate all frequent patterns from the closed patterns mined. Note that the support of a pattern is defined as the fraction of transactions containing the pattern in the database in the experiment section.

4.1. Synthetic datasets

The synthetic data generator is similar to the one used in Agrawal et al. [2] with some modifications since our transaction contains a set of points, not a sequence of itemsets. We fix the data space to be a square of 100×100. Table 1 lists the parameters and the default settings used in the synthetic data generator.

Table 1. Parameters used to generate the synthetic datasets.

Parameter	Meaning	Default settings
$\|D\|$	The number of transactions in the database	50,000
L	The average length of transactions	12
P	The number of potential patterns	1,000
PL	The average length of potential patterns	8

The synthetic data generator contains two steps. First, we generate potential patterns. We use a uniform distribution to choose a starting point (x, y), where $1 \le x, y \le 100$. Then the next point is chosen from eight neighboring points around the previous point. This process is repeated until the number of points in the potential pattern is equal to the predefined potential pattern length, PL. The length of potential pattern is determined by a Poisson distribution whose mean is PL.

Second, from the potential patterns generated in the first step, we generate a synthetic dataset where the length of each transaction in the dataset is also determined by a Poisson distribution whose mean is L, which means the average transaction length is L. We randomly pick one point from a potential pattern, and then choose the next point from eight neighboring points around the previous point to extend the pattern until the number of points of this pattern is equal to the length generated by the Poisson distribution.

4.2. Performance evaluation on synthetic datasets

In this section, we compare the PCP-Miner algorithm with modified the Apriori algorithm by varying one of parameters, and keeping the others at the default values as shown in Table 1.

The first experiment is conducted by varying minimum support threshold, where the number of transaction in the dataset is 50,000, and the average length of transactions is 12. Figure 11 shows the runtime versus the minimum support threshold, where the minimum support threshold varies from 0.1% to 5%. The PCP-Miner runs

about 2-217 times faster than the modified Apriori. As the minimum support gets smaller, the runtime of the modified Apriori increases sharply; however, the runtime of the PCP-Miner increases smoothly. In other words, the modified Apriori is more sensitive to the minimum support threshold than the PCP-Miner. Since the PCP-Miner requires only one database scan and can localize support counting, pattern joining and candidate pruning in a projected database, it is more efficient than the modified Apriori.

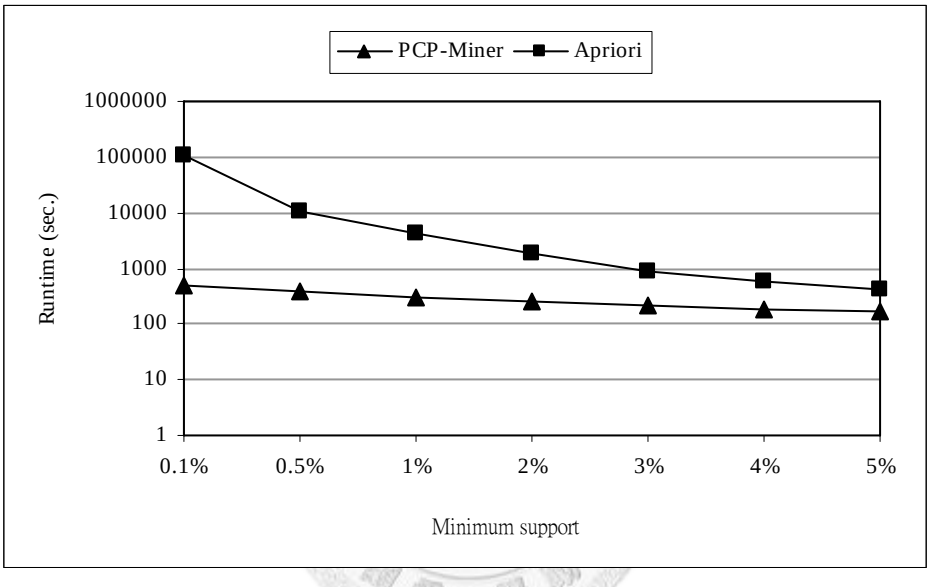


Figure 11. Runtime versus minimum support.

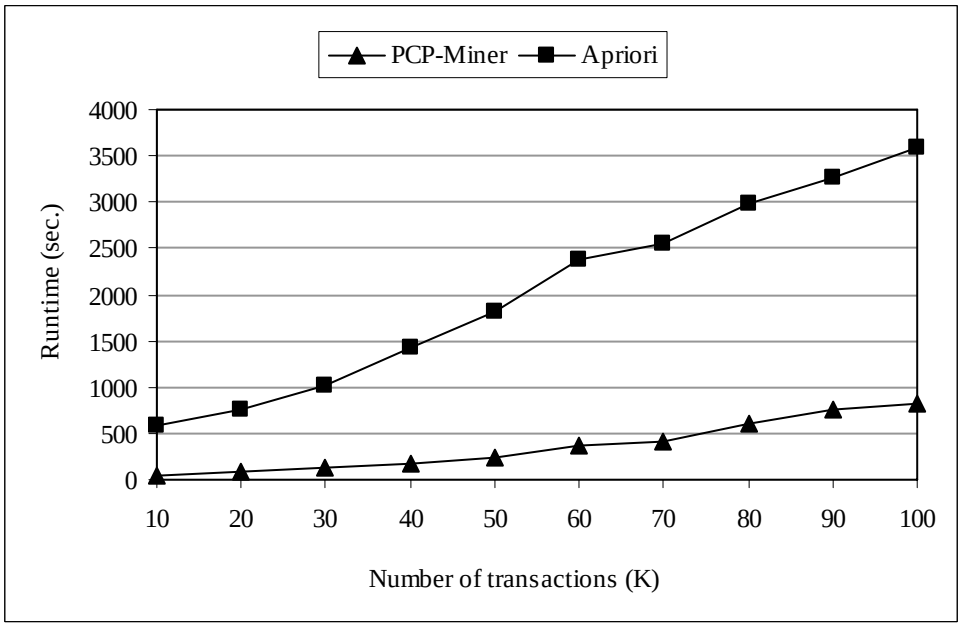


Figure 12. Runtime versus number of transactions.

Figure 12 depicts the runtime versus the number of transactions, where the number of transactions varies from 10,000 to 100,000, the average length of transaction is 12, and the minimum support threshold is 2%. The runtime of both algorithms increases linearly as the number of transactions increases.

Figure 13 shows the runtime versus the average length of transactions where the average length of transactions varies from 9 to 15, the number of transactions is 50,000, and the minimum support threshold is 2%. As the average transaction length increases, the number of frequent patterns increases. Thus, the runtime of both algorithms increases. However, the PCP-Miner is more efficient than the modified Apriori. Compared to the modified Apriori, the runtime of the PCP-Miner grows much slower when the average length of transactions is getting larger.

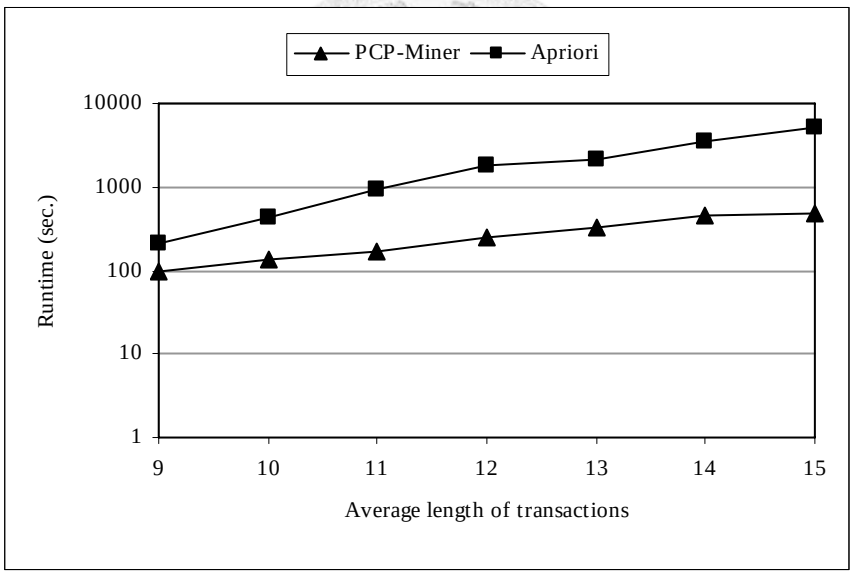


Figure 13. Runtime versus average length of transactions.

In summary, by using the projected database, the PCP-Miner algorithm requires only one database scan and can localize support counting, pattern join, and candidate pruning in a projected database, it is more efficient and scalable than the modified Apriori in all cases, especially when the minimum support threshold is low or the average length of transactions is large.

4.3. Performance evaluation on the real dataset

In this section, we use two real datasets to evaluate the performance of both algorithms. One is the maze dataset, and the other is the typhoon data in the west Pacific[1].

4.3.1. Players' movements in the maze dataset

The maze dataset contains 300 transactions, and each transaction contains a set of points. All 300 transactions are collected from a maze game designed by ourselves, and the interface of the maze game[2] is shown in Figure 14.

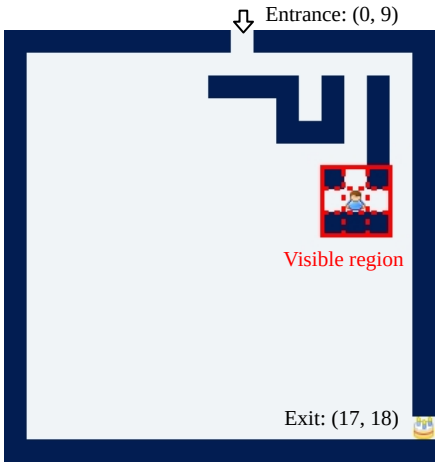


Figure 14. The interface of the maze.

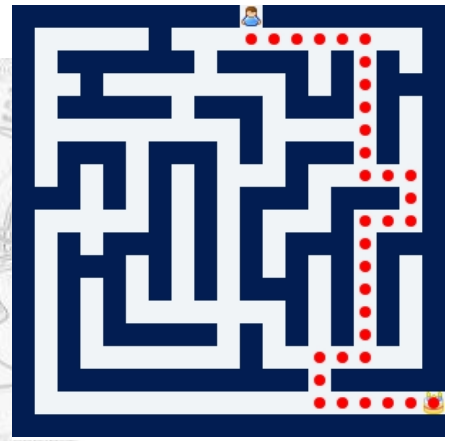


Figure 15. The whole view of the maze.

The size of the maze is 21 both in height and width. When starting the game, a player can only see the entrance, exit, and bounding blocks around the player. By moving forward, the player can see the surroundings around him/her gradually. The visible region, the bounding blocks is confined to the eight blocks around the user as shown in Figure 14. When a player plays the game, we simultaneously record each step (position) that the user just visited. The average length of transactions is 52.59. That is, in average, user takes about 53 steps from the entrance to the exit.

[1] http://weather.unisys.com/hurricane/index.html
[2] http://iis.im.ntu.edu.tw/Maze/index.html

Figure 16 illustrates the runtime versus the minimum support threshold for the real dataset where the minimum support threshold varies from 60% to 90%. The PCP-Miner runs 3-77 times faster than the modified Apriori.

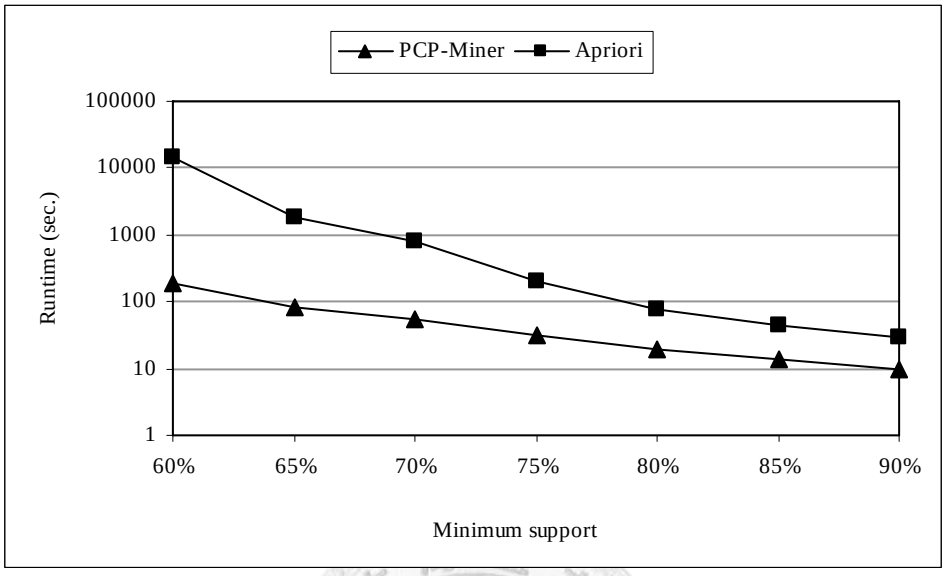


Figure 16. Runtime versus minimum support for the Maze dataset.

The PCP-Miner outperforms the modified Apriori because the latter generates a large number of candidates, especially when the minimum support threshold is low. In addition, the modified Apriori needs to scan the whole database repeatedly to count the supports. In contrast, the PCP-Miner only scans the database once and localizes the support counting, pattern joining, and candidate pruning in the projected database. Therefore, the PCP-Miner outperforms the modified Apriori.

Mining the maze dataset, we could find some interesting patterns as shown in Figure 15, where a 34-pattern with the support equal to 43% was found. Because a player cannot see the whole maze in the very beginning, the only clue for the player is the location of the exit. Thus, most (43%) of the players intuitively keep moving rightwards or downwards toward the exit, and find the way to the exit. The shortest path from the entrance to the exit of the maze is a 32-pattern with the support equal to 9.3% as shown in Figure 17. Note that most players do not exactly take 34 (or 32) steps to get to the exit, they may get stuck somewhere and go forwards and backwards

in the maze.

Figure 17. The shortest path of the maze.

4.3.2. Typhoon data in the west Pacific

We collected the dataset of West Pacific typhoons from 1996 to 2007. Every typhoon was tracked every six hours including its position in latitude and longitude, its maximum sustained winds in knots, and its central pressure in millibars. The typhoon dataset has 400 transactions and its average length of transactions is 25.5.

Figure 18 shows the runtime versus the minimum support threshold for the typhoon dataset where the minimum support threshold varies from 1% to 10%. The PCP-Miner runs 6-124 times faster than the modified Apriori.

When the minimum support threshold is getting lower, the modified Apriori generates a large amount of candidates. The support counting procedure of the modified Apriori also needs many database scans. While the PCP-Miner only scans the database once and can localizes the support counting, pattern joining, and candidate pruning in the projected database. That is why the PCP-Miner outperforms the modified Apriori.

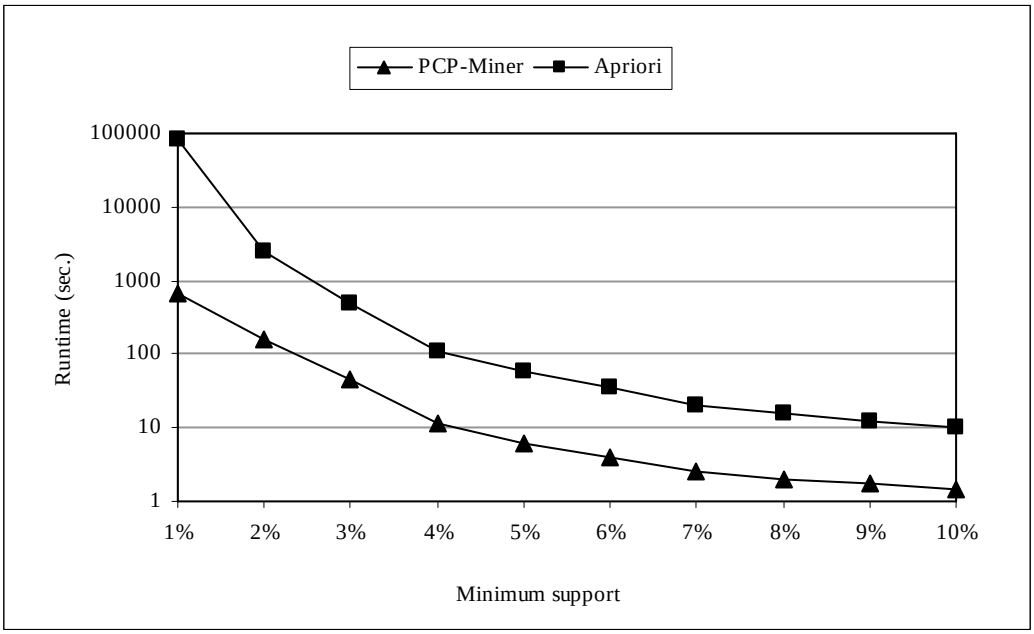


Figure 18. Runtime versus minimum support for the typhoon dataset.

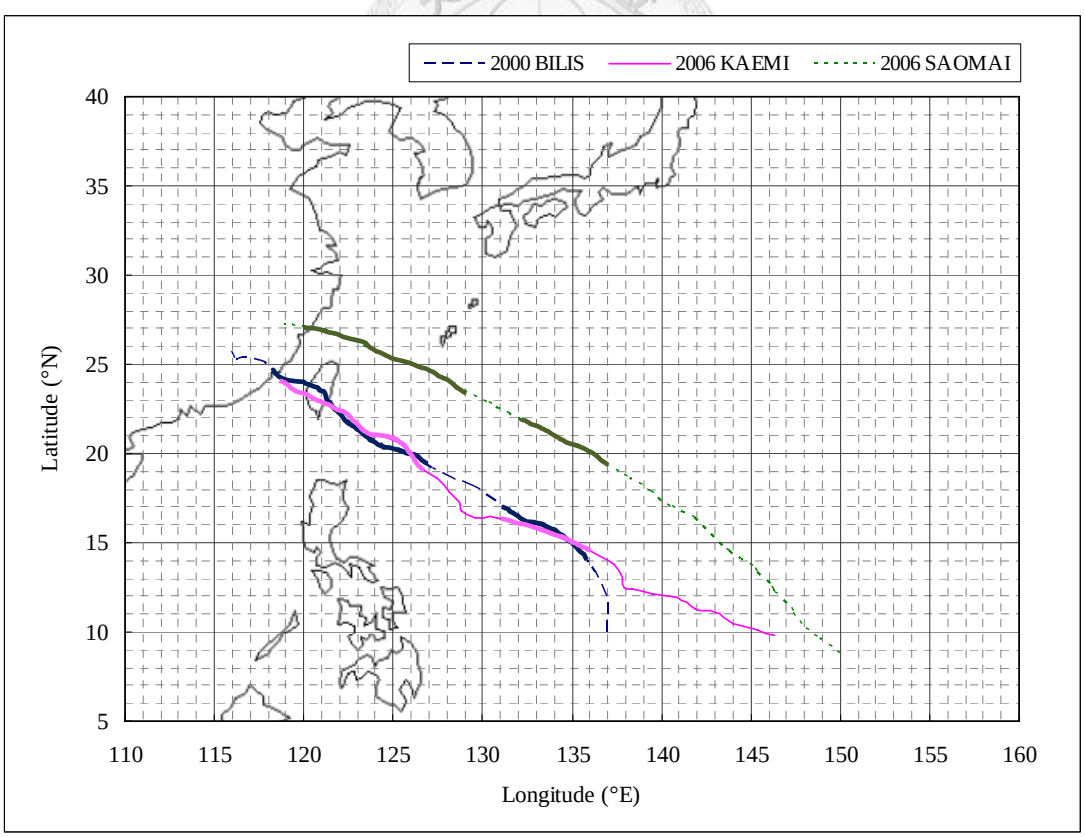
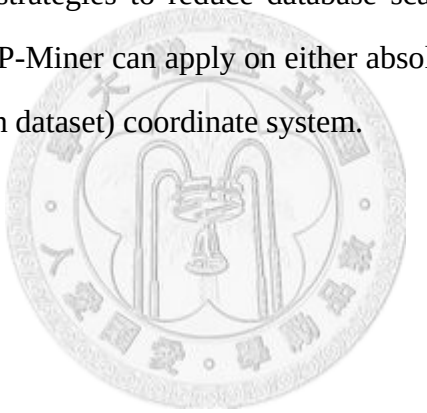


Figure 19. Frequent patterns for the typhoon dataset in the west Pacific.

From the typhoon dataset, we find out three typhoons having similar tracks. The frequent pattern is a 15-pattern, {(0, 0), (1, 1), (1, 2), (2, 3), (2, 4), (3, 4), (3, 5), (4, 6), (4, 7), (5, 8), (8, 13), (8, 14), (9, 15), (9, 16), (10, 17)}. Figure 19 illustrate the tracks of three typhoons BILIS, KAEMI, and SAOMAI. Although these three tracks are not the same, we still find out they have a similar pattern after shifting, where the similar patterns are denoted by the bold segments. Note that the pattern of three typhoons is partitioned into two parts since the KAEMI typhoon takes detours near 17°N, 130°E.

In summary, the PCP-Miner algorithm is more efficient and scalable than the modified Apriori algorithm either on the synthetic or the real dataset. The PCP-Miner can attribute its better performance to adopting the projected database approach and closure-checking pruning strategies to reduce database scans and prune non-closed patterns. Moreover, the PCP-Miner can apply on either absolute (i.e., the maze dataset) or relative (i.e., the typhoon dataset) coordinate system.



Chapter 5 Conclusions and Future Work

In this thesis, we have proposed an efficient algorithm, called PCP-Miner (Pointset Closed Pattern Miner), to mine the closed patterns in a pointset database. The algorithm can apply on either absolute or relative coordinate system. Our proposed algorithm consists of two phases. First, we find all frequent 2-patterns in the database and generate their projected databases. Second, we recursively generate frequent (k+1)-patterns by joining k-patterns in a joinable class in a depth-first search manner, $k \geq 2$. During the enumeration process, we apply the pruning strategies to prune impossible candidates and apply the subsumption check to remove frequent but non-closed patterns. By using the projected database, the PCP-Miner only scans the database once and can localize the support counting, pattern joining, and candidate pruning in a projected database. Thus, it can efficiently mine closed patterns in a pointset database. The experiment results show that the PCP-Miner outperforms the modified Apriori by one order of magnitude.

In addition to the two example applications shown in this thesis, the PCP-Miner can also apply to the binary image applications. The patterns extracted from the binary images can be used to achieve pose/shape recognition or body movement comparison. Moreover, the PCP-Miner could be used to discover the frequent animal migration paths, hurricane track data, or the bus routing routes.

However, the PCP-Miner still has some limitations. The algorithm is a memory-based algorithm. When the datasets or the number of frequent patterns is getting larger and larger, it may not be able to be loaded into the main memory. Thus, how to develop a disk-based mining algorithm is worth further study in the future. Moreover, if the pattern can be represented as a bit string, the cost of pattern joins and subsumption checks can be saved by using the efficient bitwise operators. Thus, it is worth implementing the PCP-Miner by representing patterns and transactions by bit

strings. Also, to improve the flexibility of the algorithm, we can modify the algorithm to tolerate position errors within a specific range by treating the neighboring points as the same location. Furthermore, we can extend our algorithm to take the temporal attribute into consideration. That means there are several sets of points in one transaction ordered by time instead of one set. This extension will significantly improve the usability in detecting frequent consecutive movements in a set of movie clips.

References

[1] R. Agrawal, R. Srikant, Fast algorithms for mining association rules, *Proceedings of the International Conference on Very Large Data Bases*, Santiago, Chile, 1994, pp. 487-499.

[2] R. Agrawal, R. Srikant, Mining sequential patterns, *Proceedings of the International Conference on Data Engineering*, Taipei, Taiwan, 1995, pp. 3-14.

[3] J. Ayres, J.E. Gehrke, T. Yiu, J. Flannick, Sequential pattern mining using a bitmap representation, *Proceedings of the ACM SIGMOD International Conference on Knowledge Discovery in Database*, Edmonton, Canada, 2002, pp. 429-435.

[4] T.S. Chen, S.C. Hsu, Mining frequent tree-like patterns in large datasets, *Data and Knowledge Engineering*, Vol. 62, No. 1, 2007, pp. 65-83.

[5] J. Cheng, Y. Ke, W. Ng, δ-Tolerance Closed Frequent Itemsets, *Proceedings of the IEEE International Conference on Data Mining*, Hong Kong, China, 2006, pp. 139-148.

[6] J.D. Chung, O.H. Paek, J.W. Lee, K.H. Ryu, Temporal moving pattern mining for location-based service, *Proceedings of the International Conference on Database and Expert Systems Applications*, Aix-en-Provence, France, 2002, pp. 331-340.

[7] C.I. Ezeife, M. Monwar, SSM: a frequent sequential data stream patterns miner, *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining*, Honolulu, Hawaii, USA, 2007, pp. 120-126.

[8] F. Giannotti, M. Nanni, F. Pinelli, D. Pedreschi, Trajectory pattern mining, *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Jose, California, USA, 2007, pp. 330-339.

[9] G. Grahne, J. Zhu, Fast algorithms for frequent itemset mining using FP-trees, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17, No. 10, 2005,

pp. 1347-1362.

[10] E. Gudes, E. Shimony N. Vanetik, Discovering frequent graph patterns using disjoint paths, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 18, No. 11, 2006, pp. 1441-1456.

[11] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Dallas, USA, 2000, pp. 1-12.

[12] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, M.C. Hsu, FreeSpan: frequent pattern-projected sequential pattern mining, *Proceedings of International Conference on Knowledge Discovery and Data Mining*, Boston, USA, 2000, pp. 355-359.

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

[14] V.T.H. Nhan, J.H. Chi, K.H. Ryu, Discovery of spatiotemporal patterns in mobile environment, *Proceedings of the Asia-Pacific Web Conference*, Harbin, China, 2006, pp. 949-954.

[15] J. Huan, W. Wang, J. Prins, Efficient mining of frequent subgraphs in the presence of isomorphism, *Proceedings of IEEE International Conference on Data Mining*, Melbourne, Florida, USA, 2003, pp. 549-552.

[16] S.Y. Hwang, Y.H. Liu, J.K. Chiu, E.P. Lim, Mining mobile group patterns: a trajectory-based approach, *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Hanoi, Vietnam, 2005, pp. 713-718.

[17] A. Inokuchi, T. Washio, H.Motoda, An Apriori-based algorithm for mining frequent substructures from graph data, *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, Lyon, France, 2000, pp. 13-23.

[18] R. Jin, C. Wang, D. Polshakov, S. Parthasarathy, G. Agarwal, Discovery frequent

topological structures from graph datasets, *Proceeding of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Chicago, USA, 2005, pp. 606-611.

[19] M. Kuramochi, G. Karypis, Frequent subgraph discovery, *Proceedings of IEEE International Conference on Data Mining*, California, USA, 2001, pp. 313-320.

[20] M. Leleu, C. Rigotti, Jean-Francois Boulicaut, G. Euvrard, GO-SPADE: mining sequential patterns over datasets with consecutive repetitions, *Proceedings of International Conference on Machine Learning and Data Mining*, Leipzig, Germany, 2003, pp. 293-306.

[21] C. Lucchese, S. Orlando, R. Perego, Fast and memory efficient mining of frequent closed itemsets, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 18, No. 1, 2006, pp. 21-36.

[22] H.D.K. Moonesinghe, S. Fodeh, P.N. Tan, Frequent closed itemset mining using prefix graphs with an efficient flow-based pruning strategy, *Proceedings of IEEE International Conference on Data Mining*, Hong Kong, China, 2006, pp. 426-435.

[23] G.K. Palshikar, M.S. Kale, M.M. Apte, Association rules mining using heavy itemsets, *Data and Knowledge Engineering*, Vol. 61, No. 1, 2007, pp. 93-113.

[24] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal, Discovering frequent closed itemsets for association rules, *Proceedings of the 7th International Conference on Database Theory*, Jerusalem, Israel, 1999, pp. 398-416.

[25] J. Pei, J. Han, R. Mao, CLOSET: an efficient algorithm for mining frequent closed itemsets, *Proceedings of the 5th ACM-SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, Dallas, USA, 2000, pp. 11-20.

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

[27] B. Rozenberg, E. Gudes, Association rules mining in vertically partitioned databases, *Data and Knowledge Engineering*, Vol. 59, No. 2, 2006, pp. 378-396.

[28] N.G. Singh, S.R. Singh, A.K. Mahanta, CloseMiner: discovering frequent closed itemsets using frequent closed tidsets, *Proceedings of IEEE International Conference on Data Mining*, Houston, USA, 2005, pp. 633-636.

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

[30] I. Tsoukatos, D. Gunopulos, Efficient mining of spatiotemporal patterns, *Proceedings of the International Symposium on Advances in Spatial and Temporal Databases*, Redondo Beach, CA, USA, 2001, pp. 425-442.

[31] T. Uno, T. Asai, Y. Uchida, H. Arimura, An efficient algorithm for enumerating closed patterns in transaction databases, *Proceedings of the 7th International Conference on Discovery Science*, Padova, Italy, 2004, pp. 16-31.

[32] J. Wang, J. Han, J. Pei, CLOSET+: searching for the best strategies for mining frequent closed itemsets, *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, Washington, D.C., USA, 2003, pp. 236-245.

[33] X. Yan, J. Han, gSpan: graph-based substructure pattern mining, *Proceedings of IEEE International Conference on Data Mining*, Maebashi City, Japan, 2002, pp. 721-724.

[34] H. Yao, H. Hamilton, Mining itemset utilities from transaction databases, *Data and Knowledge Engineering*, Vol. 59, No. 3, 2007, pp. 603-626.

[35] U. Yun, J.J. Leggett, WSpan: weighted sequential pattern mining in large sequence databases, *Proceedings of International IEEE Conference on Intelligent Systems*, London, United Kingdom, 2006, pp. 512-517.

[36] M.J. Zaki, SPADE: an efficient algorithm for mining frequent sequences, *Machine Learning*, Vol. 42, No. 1-2, 2001, pp. 31-60.

[37] M.J. Zaki, C. Hsiao, Efficient algorithms for mining closed itemsets and their lattice structure, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17, No. 4, 2005, pp. 462-478.

[38] M.J. Zaki, K. Gouda, Fast vertical mining using diffsets, *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, D.C., USA, 2003, pp. 326-335.

簡　　歷

姓　　名：陳 柏 吟

出 生 地：台 灣 省 彰 化 縣

出 生 日：中 華 民 國 七 十 三 年 八 月 六 日

學　　歷：九 十 五 年 九 月 至 九 十 七 年 六 月

國 立 台 灣 大 學 資 訊 管 理 研 究 所 碩 士 班

九 十 一 年 九 月 至 九 十 五 年 六 月

國 立 政 治 大 學 資 訊 管 理 學 系