國立交通大學

資訊工程系

碩 士 論 文

多領域同時性循序序列模式探勘

Multi-domain Simultaneous Sequential Pattern Mining

研 究 生：胡星垣

指導教授：彭文志　教授

中 華 民 國 九 十 四 年 八 月

多領域同時性循序序列模式探勘
Multi-domain Simultaneous Sequential Pattern Mining

研 究 生：胡星垣　　Student：Hsing-Yuan Hu

指導教授：彭文志　　Advisor：Wen-Chih Peng

國 立 交 通 大 學
資 訊 工 程 系
碩 士 論 文

A Thesis

Submitted to Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science and Information Engineering

August 2005

Hsinchu, Taiwan, Republic of China

中華民國九十四年八月

多領域同時性循序序列模式探勘

學生：胡星垣　　　　　　　　　　　　　　　　　　　　　　　　指導教授：彭文志 教授

國立交通大學資訊工程系（研究所）碩士班

摘　　　要

循序序列模式探勘(Sequential pattern mining)是一個資料探勘研究中相當重要的研究課題，循序序列模式探勘的問題在於:在循序序列資料庫中探勘頻繁序列模式。先前關於循序序列探勘的研究，只探討關於一個領域(domain)的循序序列模式，例如：找出購買習慣、網頁瀏覽行為或是頻繁移動模式。然在不同領域的循序序列模式，如果發生在相同的時間，則這些循序序列模式可以形成一個多領域同時性循序序列模式(Multi-domain simultaneous sequential pattern)。相較於傳統的單一領域循序序列模式，多領域同時性循序序列模式可以完整的反應出一個使用者的行為模式，因此，探勘多領域同時性循序序列模式有其必要性。在本論文中，我們提出一個以模式傳遞(pattern-propagation)為基礎的演算法 PropagatedMine，並利用該演算法有效率的探勘多領域同時性循序序列模式。藉由起始領域(starting domain)開始傳遞循序序列模式發生的時間到其它的領域，我們所提出的演算法可以明顯的降低探勘的空間，也因此大量的減少探勘多領域同時性循序序列模式的成本。此外，執行 PropagatedMine 演算法的成本和傳遞循序序列模式到不同領域的順序有很大的相關性，在本研究中，我們進一步最佳化傳遞循序序列模式之順序。實驗結果顯示 PropagatedMine 演算法可以有效率的探勘多領域同時性循序序列模式，針對傳遞順序做最佳化的 PropagatedMine 演算法可以更進一步的提高資料探勘的效能。

Multi-domain Simultaneous Sequential Pattern Mining

student：Hsing-Yuan Hu Advisors：Dr. Wen-Chih Peng

Department of Computer Science and Information Engineering
National Chiao Tung University

ABSTRACT

Sequential pattern mining has attracted a significant amount of research efforts recently. The problem of sequential pattern mining is that discovering frequent sequences with their occurrence counts being larger than or equal to the user-specified number, *min_support*, among a set of sequences. Most of the previously sequential pattern mining methods only explore mining sequential patterns in one domain, such as buy behavior, Web browsing, and moving patterns. In reality, sequential patterns may exist in multiple sequence databases and for these sequential patterns in each sequence database, if the occurrences of these sequential patterns appear at the same time, these sequential patterns are able to form a multi-domain simultaneous sequential pattern. Note that mining multi-domain simultaneous sequential patterns is very important in that simultaneous sequential patterns reflect the complete behavior of users. In this paper, we propose a propagation-based approach (referred to as algorithm PropogatedMine) for efficient mining of multi-domain sequential patterns. By propagating patterns with their occurrences of time from one starting domain to other domains, our proposed approach is able to significantly reduce the mining space, which improves the performance of mining multi-domain sequential patterns. Note that the cost of performing PropagatedMine is greatly affected by the propagation order. Thus, in this paper, we further develop a novel method to determine the optimized propagation order. A comprehensive performance study is conducted and experimental results show that algorithm PropagatedMine is able to efficiently mine multi-domain sequential patterns. Moreover, algorithm PropagatedMine with an optimized propagation order is able to further improve the performance in mining multi-domain sequential patterns and the performance of the optimized propagation order determined by our proposed method is very close to that of the optimal one resulted by selecting the minimal cost among all possible propagation orders.

誌　　　謝

感謝兩年來指導教授彭文志老師在各方面的悉心指導和照顧，在這篇論文的寫作過程中，帶領我走過完成一項研究所必經的過程。以及吳秀陽與李官陵老師，不僅是我大學時代的老師，這次也遠從東華大學趕來當我的口試委員，並給予我論文繼續發展的寶貴建議。

感謝張舜理學長，在這篇論文架構的初期，時常耐心的討論我所遭遇到的研究瓶頸，這些討論使我得到許多啟發。並在研究後期的實驗階段也熱心的提供協助，讓我能夠順利完成這篇論文。

感謝實驗室的伙伴們，我會記得大家一起在 621 實驗室熬夜準備考試，以及一同歡樂的時光，特別感謝洪智傑同學無私地提供研究經驗的分享。

最後要感謝我的家人們對我完全的支持，使我能夠心無旁騖的努力於研究。僅將這篇論文獻給最親愛的他們。

Contents

List of Figures

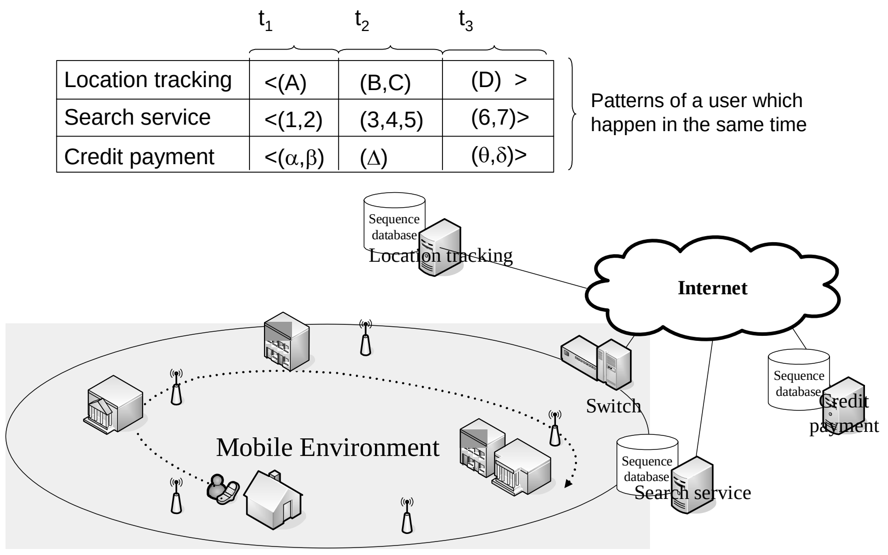
List of Tables

Chapter 1

Introduction

Sequential pattern mining has attracted a significant amount of research efforts recently [2][10][11][12][13]. The problem of sequential pattern mining is that discovering frequent sequences with their occurrence counts being larger than or equal to the user-specified number, *min_support*, among a set of sequences. Sequential pattern mining can be applied on business and marketing analysis, web page browsing behavior, symptomatic pattern of a disease, DNA sequence, hacker invasion detecting, and so on. Due to the importance of sequential patterns mining, many efficient sequential pattern mining methods have been proposed [3][4][5][9][15].

Traditional sequential pattern mining methods are mostly performed in only one domain sequence database. For example, sequential patterns mining method is applied in a moving log sequence database to find out user moving patterns or in a Web page browsing sequence database to mine Web browsing behavior. In reality, sequential patterns may exist in multiple sequence databases. Mining simultaneous sequential patterns is very important in that these simultaneous sequential patterns reflect the complete behavior of users. An illustrative example of multi-domain sequential patterns is shown in Figure 1.1, where a mobile user can obtain a variety of services via mobile devices. In Figure 1.1, there are three services available: a location tracking service, a location dependent search service, and a credit payment service. Each service has its own sequence database that records the access data and the corresponding



t1 t2 t3
Location tracking <(A) (B,C) (D) >
Search service <(1,2) (3,4,5) (6,7)>
Credit payment <(α,β) (Δ) (θ,δ)>
Patterns of a user which happen in the same time
Sequence database
Location tracking
Internet
Switch
Sequence database
Credit payment
Mobile Environment
Sequence database
Search service

Figure 1.1: An illustrative example for the usage of multi-domain sequential patterns

access time of each user. Note that each service is viewed as an individual "**domain**". In the example, each row means a sequential pattern of a domain, e.g., p_1: <(A)(B,C)(D)> is a moving pattern of an user and this pattern happens in the same time with search pattern p_2: <(1,2),(3,4,5)(6,7)> and payment pattern p_3: $<(\alpha,\beta)(\Delta)(\theta,\delta)>$ of that user. It can be seen that patterns p_1, p_2, and p_3 are able to form a multi-domain sequential patterns in that (A), (1,2), and (α,β) are all frequent in time t_1, (B,C), (3,4,5), and (Δ) are taken place in time t_2, and (D), (6,7), and (θ,δ) are occurred in time t_3.

Mining multi-domain sequential patterns is very useful in that the relations of these domains can give us some information about user behavior. For example, the illustrative multi-domain sequential pattern shown in Figure 1.1, reflects not only the moving pattern of the user but also the search and buying behavior of this mobile user. Thus, multi-domain sequential patterns give us more useful and diversified information than only one domain sequential patterns. In this paper, we address the problem of mining multi-domain sequential patterns.

Though many sequential pattern mining methods work well in one domain database, these sequential methods suffer from poor performance when being applied in mining multi-domain sequential patterns from multi-domain databases. Intuitively, we can mining sequential patterns in each individual domain and composite multi-domain sequential patterns by examining whether the occurrences of these sequential patterns occur in the same time. For example, in Figure 1.1, mining moving patterns, search patterns, and payment patterns in the corresponding sequence databases. Then, for each pattern mined in these three domains, we examine whether these patterns occur in the same time. As a result, we could obtain multiple sequential patterns as well. However, this method unavoidably leads to the poor performance in terms of efficiency and scalability. Notice that mining all sequential patterns individually in every sequence database may waste efforts since these patterns may not be necessary to be formed as multi-domain sequential patterns. In addition, the overhead of this native method also includes the extra-effort to check whether these sequential patterns occur in the same time or not.

To our best knowledge, mining multi-domain sequential patterns is a new research topic. In this paper, we propose an efficient algorithm named as *PropagatedMine* to mine *multi-domain sequential patterns (to be referred to as MDSSP)* efficiently. The concept of PropagatedMine is to propagate patterns with their occurrences of time from the selected starting domain to all the other domains with an optimized order, and therefore the mining space is significantly reduced. Note that the cost of performing PropagatedMine may greatly different with different propagation orders, so finding a good propagation order is important to PropagatedMine. In this work a cost estimation method is also proposed in PropagatedMine to determine an optimized propagation order. Our performance study shows that our proposed PropagatedMine is able to efficiently mine multi-domain sequential patterns. The contributions of this paper are three folds: (1) Exploiting a novel and useful sequential patterns (i.e., multi-domain sequential patterns), (2) devising algorithm PropagatedMine to efficiently mine multi-domain sequential patterns, and (3) determining a optimized propagation order to further reduce the multi-domain sequential pattern mining cost significantly.

The remaining of the paper is organized as follows. In Chapter 2, problem description and some preliminaries are presented. Algorithm PropagatedMine is developed in Chapter 3. Performance studies are conducted in Chapter 4. This paper concludes with Chapter 5.

Chapter 2

Preliminary

2.1 Basic Terminologies

Let $I = \{x_1, x_2, ..., x_n\}$ be a set of all items. An **itemset** is a non-empty subset of I and is denoted as $X = (x_1, x_2, ..., x_r)$, where x_j is an item of I for every j, $1 \leq j \leq r$. A **sequence** s is an ordered list of itemsets, and is represented as $s =< X_1, X_2, ..., X_l >$, where X_j is an itemset (i.e., $X_j \subseteq I$ for $1 \leq j \leq l$). For the brevity purpose, X_j is called an **element** of sequence s. Note that an item can appear only once in one element but it can appear in different elements of a sequence. Sequence $s_a =< a_1, a_2, ..., a_n >$ is contained in sequence $s_b =< b_1, b_2, ..., b_m >$, denoted as $s_a \sqsubseteq s_b$, if and only if there exists integers $1 \leq i_1 \leq i_2 \leq ... \leq i_n \leq m$ such that $a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, ..., a_n \subseteq b_{i_n}$. Sequence s_a is a **subsequence** of sequence s_b, and s_b is a **super sequence** of s_a. For example, sequence <(3)(1,2)(5)> is contained in sequence <(1,3)(1,2,4)(4,5)(3)> due to that (3)⊆(1,3), (1,2)⊆(1,2,4), and (5)⊆(4,5).

A **period** P is a section of time, and can be fully divided into several continuous, non-overlapping, and equal range **time slots.** A period with n time slots is denoted as $P|_n = \{t_1, t_2, ..., t_n\}$, where t_j is a time slot and $1 \leq j \leq n$. Since each time slot represents an interval of time and time slots are non-overlapping, a total ordering $t_1 > t_2 > ... > t_n$ are specified within a period. A **time sequence** $ts = PID :< t_1, t_2, ..., t_m >$ is an ordered list of time slots in one period, where PID is an identifier used to indicate which period. In addition, the ordering of time slots in time sequence ts must satisfies $t_1 > t_2 > ... > t_m$, and n needs to be greater than or equal to m. A time sequence $ts_a = PID_a :< t_1, t_2, ..., t_n >$ is called a sub time sequence of $ts_b = PID_b :< t_1, t_2, ..., t_m >$ if PID_a equals to PID_b and $< t_1, t_2, ..., t_n >$ is a sub sequence of $< t_1, t_2, ..., t_m >$.

A **time related sequence database** denoted as $TRSDB$ is comprised by a set of tuples, where each tuple in the sequence database is a **time related sequence**. Table 2.1 shows an illustrative example of a time related sequence database. SID is the identifier of every time related sequence, $TIME\_SEQ$ is a time sequence while $CONTEXT\_SEQ$ is a sequence. Note that for each time related sequence trs, the occurrence of time for each element X_i in $CONTEXT\_SEQ$ is uniquely mapped to the time slot t_i. The time related sequence database in Table 2.1 can be illustrated by Figure 2.1. Time related sequence trs_a is a **sub time related sequence** of trs_b if $trs_a.TIME\_SEQ \sqsubseteq trs_b.TIME\_SEQ$ and $trs_a.CONTEXT\_SEQ \sqsubseteq trs_b.CONTEXT\_SEQ$.

The **support** for a sequence s in a time related sequence database $TRSDB$ is defined as the number of time related sequences which contain s in their $CONTEXT\_SEQ$. Formally speaking, the support of sequence s is defined as $Support(s) = |\{x|x \in TRSDB$, and $s \sqsubseteq x.CONTEXE\_SEQ\}|$. For example, assume that sequence s=<(a)(b,c)>. The support of s in $TRSDB_1$ shown in Table 2.1 is 3 since three time related sequences contain s in their corresponding $CONTEXT\_SEQ$. Note that the support of a sequence s in a time related sequence database is the number of tuples that contain s in its $CONTEXT\_SEQ$. Given a minimum support threshold, $min\_support$, and a time related sequence database $TRSDB$, a sequence s is **frequent** if and only if $Support(s) \geq min\_support$.

Given sequence $s =< X_1, X_2, ..., X_n >$ and a time related sequence $trs = \{PID :< t_1, t_2, ..., t_m >, < Y_1, Y_2, ..., Y_m >\}$ of time related sequence database $TRSDB$, if $s \sqsubseteq trs.CONTEXE\_SEQ$ is satisfied then we can find integers $1 \leq i_1 < i_2 < ... < i_n \leq m$

SID	TIME_SEQ	CONTEXE_SEQ
1	1:<1,2,3,4>	<(a)(b,c)(b,c,d)(e)>
2	2:<2,3,4>	<(a,b)(b,c)(c,e)>
3	3:<1,2,3>	<(a,e)(h)(g,j)>
4	4:<1,2,3,4>	<(a,b,f)(d)(b,c)(e,f)>

Table 2.1: $TRSDB_1$

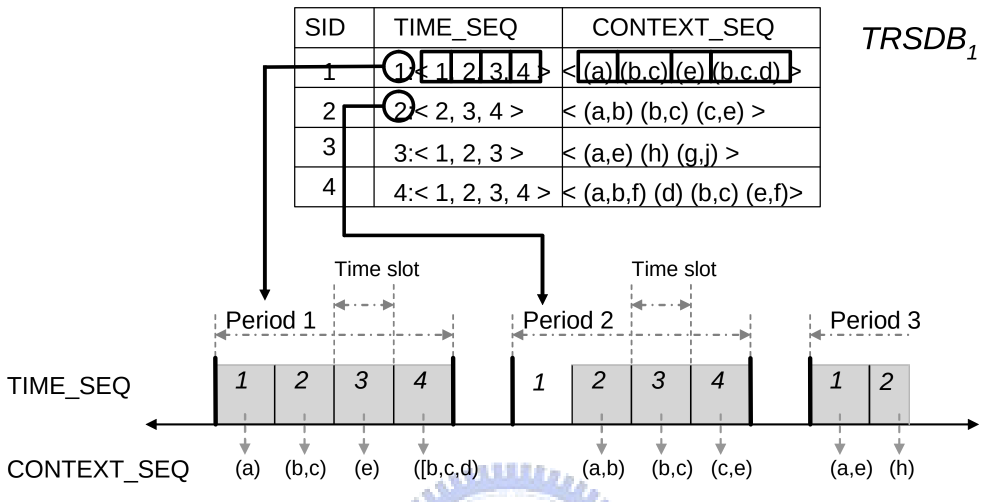


Figure 2.1: An illustrative example of time related sequence database

such that $X_1 \subseteq Y_{i_1}, X_2 \subseteq Y_{i_2}, ..., X_n \subseteq Y_{i_n}$ and $PID :< t_{i_1}, t_{i_2}, ...t_{i_n} >$ is called a **time instance** of the sequence s. Note that a time related sequence trs can contain more than one time instance of sequence s because s can appear more than once in $trs.CONTEXT\_SEQ$. Gathering all time instances of sequence s in $TRSDB$ forms a **time instance set** represented as $TIS_s^{TRSDB} = \{x|x$ is a time instance of s in $TRSDB\}$. For example, in Table 2.1, there are three time related sequences containing sequence <(a)(b,c)>. Accordingly, we could have the time instance set of sequence <(a)(b,c)> to be {1:<1,2>, 1:<1,4>, 2:<2,3>, 4:<1,3>}.

2.2 Problem Formulation

We elaborate on the concept of multi-domain simultaneous sequential patterns and formulate the problem of mining multi-domain simultaneous sequential pattern in this section. The term **simultaneous** means that if the periods in all domains have the same PID, these periods

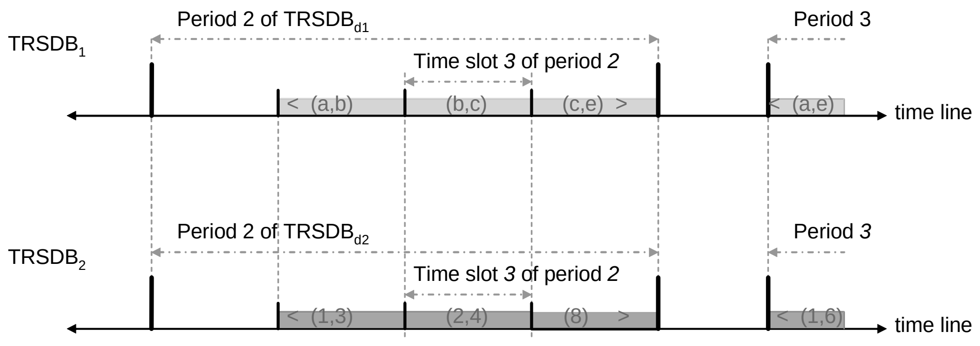


Figure 2.2: Simultaneity of periods and time slots in different domains

SID	TIME_SEQ	CONTEXT_SEQ
1	1:<1,2,3,4>	<(1,2)(2,3)(6)(4,5)>
2	2:<2,3,4>	<(1,3)(2,4)(8)>
3	3:<1,2,3>	<(1,6)(5)(9,10)>
4	4:<1,2,3,4>	<(1,2,5)(7)(2,3)(4,5,6)>

Table 2.2: $TRSDB_2$

represent the simultaneous time. In Figure 2.2, the alignment of two period 2 indicates that these two periods represent exactly the same time section in $TRSDB_1$ and $TRSDB_2$. Next, we specify a section of time which is a time slot. Similar to periods in different domains, the alignment of time slots in different domains represents they are same time sections.

Definition 1 (Simultaneous time instances) Based on the definition of time sequence, a time instance is also a time sequence. Given time instances: $ti_1 = PID_1 :< t_{11}, t_{12}, ..., t_{1l_1} >$ $, ti_2 = PID_2 :< t_{21}, t_{22}, ..., t_{2l_2} >, ..., ti_r = PID_r :< t_{r1}, t_{r2}, ..., t_{rl_r} >$, these time instances are simultaneous if and only if $PID_1 = PID_2 = ... = PID_r$, and $l_1 = l_2 = ... = l_r = l$, and for every j, $1 \leq j \leq l$, $t_{1j} = t_{2j} = ... = t_{rj}$.

Definition 2 (Simultaneous sequences) Given r sequences $s_1, s_2, ..., s_r$ in time related sequences database $TRSDB_1, TRSDB_2, ..., andTRSDB_r$ respectively and their corresponding time instance set $TIS|_{s_1}^{TRSDB_1}, TIS|_{s_2}^{TRSDB_2}, ..., TIS|_{s_r}^{TRSDB_r}$. Sequence s_1, s_2, ..., and s_r are simultaneous if there exist time instances $ti_1, ti_2, ..., ti_r$ such that $ti_1, ti_2, ..., ti_r$ are simultaneous time instances, where ti_j is a time instance in $TIS|_{s_j}^{TRSDB_j}$, $1 \leq j \leq r$.

For example, sequences s_1= <(a)(b,c,d)(e)> in Table 2.1 and s_2=<(1)(6)(5)> in Table 2.2 are simultaneous sequence because $TIS_{s_1}^{TRSDB_1}$ = {1:<1,3,4>} and $TIS_{s_2}^{TRSDB_2}$ = {1:<1,3,4>}.

Definition 3 (Simultaneous sequential patterns) Given a minimum threshold, $min\_support$, and sequential patterns $s_1, s_2, ..., s_n$ in time related sequence database $TRSDB_1, TRSDB_2, ..., TRSDB_n$

respectively and their corresponding time instance set $TIS|_{s_1}^{TRSDB_1}, TIS|_{s_2}^{TRSDB_2}, ..., TIS|_{s_r}^{TRSDB_r}$. Time instances $(ti_1, ti_2, ..., ti_r)$ is called a **match** if $ti_1, ti_2, ..., trs_r$ are simultaneous, where $ti_j \in TIS|_{s_j}^{TRSDB_j}, 1 \leq j \leq r$. If $(ti_1, ti_2, ..., ti_r)$ is a match, then $PID_1 = PID_2 = ... = PID_r$ must be true. Let $Macthed\_PID = PID_1 = PID_2 = ... = PID_r$, and patterns s_1, $s_2, ..., s_n$ are simultaneous if and only if the number of different $Macthed\_PID$ in $TIS|_{s_1}^{TRSDB_1}, TIS|_{s_2}^{TRSDB_2}, ..., TIS|_{s_r}^{TRSDB_r}$ is equal to or larger than $min\_support$.

For example, given support threshold $min\_support = 2$ and two time related sequence database $TRSDB_1, TRSDB_2$, sequence s_1=<(a)(b,c)> and s_2=<(1)(2,3)> are sequential patterns in $TRSDB_1$ and $TRSDB_2$ respectively because both their support value larger than $min\_support$. Since $TIS|_{s_1}^{TRSDB_1}$ ={1:<1,2>, 1<1,3>, 2:<2,3>, 4:<1,3>} and $TIS|_{s_2}^{TRSDB_2}$ ={1:<1,2>, 4:<1,3>}, we can find two matches (1:<1,2>, 1:<1,2>), and (4:<1,3>, 4:<1,3>) and these two matches have two different $Mactched\_PID$: 1 and 2. Therefore, <(a)(b,c)> and <(1)(2,3)> are simultaneous sequential patterns. A counter example is <(a)(b,c,d)(e)> and <(1)(6)(5)> in the example of definition 2. Since there is only one match (1:<1,3,4>, 1<1,3,4>) can be found in time instance sets of <(a)(b,c)(e)> and <(1)(2,3)(5)>, sequences <(a)(b,c)(e)> and <(1)(2,3)(5)> are not simultaneous sequential patterns, though they are simultaneous sequences and also sequential patterns in $TRSDB_1$ and $TRSDB_2$ respectively.

If we have k time related sequence databases: $TRSDB_1, TRSDB_2, ..., TRSDB_k$ and each related to one domain. A **M**ulti-**D**omain **S**imultaneous **S**equential **P**attern ($MDSSP$) about these k domains can be expressed by a matrix as:

$$MDSSP = \begin{bmatrix} X_{11} & X_{12} & ... & X_{1l} \\ X_{21} & X_{22} & ... & X_{2l} \\ ... & ... & ... & ... \\ X_{k1} & X_{k2} & ... & X_{kl} \end{bmatrix}$$

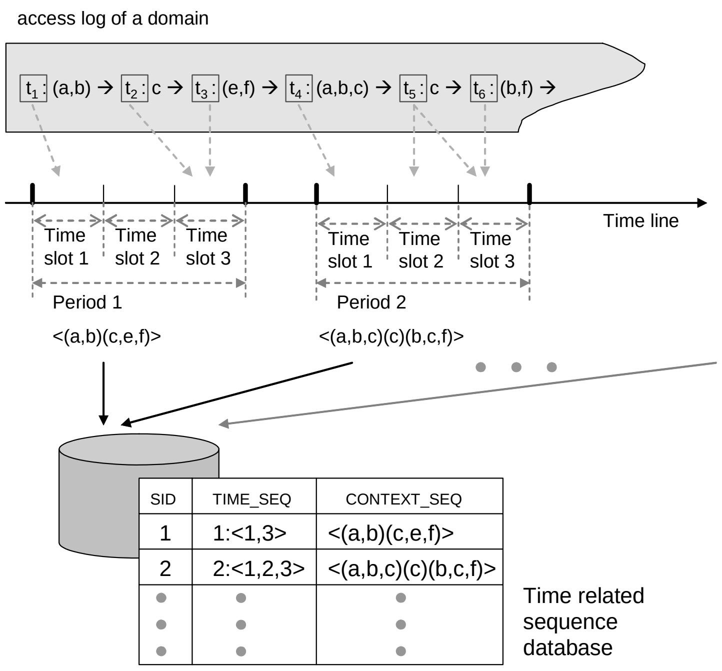
X_{ij} is the $j-th$ element of sequence $s_i =< X_{i1}, X_{i2}, ..., X_{il} >$, where s_i is a sequential pattern in $TRSDB_i$. Each row in $MDSSP$ is called a **component pattern**, i.e., for each i, $1 \leq i \leq k$, $< X_{i1}, X_{i2}, ..., X_{il} >$ is a component pattern of $MDSSP$. Number k clarify

$MDSSP$ related to k time related sequence database or said k domains, and l is the number of columns in $MDSSP$.

$\begin{bmatrix} X_{11} & X_{12} & ... & X_{1l} \\ X_{21} & X_{22} & ... & X_{2l} \\ ... & ... & ... & ... \\ X_{k1} & X_{k2} & ... & X_{kl} \end{bmatrix}$ is a multi-domain simultaneous sequential pattern of $TRSDB_1$, $TRSDB_2, ...,$ and $TRSDB_k$, where (1) each component pattern is a sequential pattern: for every i, $1 \leq i \leq k$, the component pattern $< X_{i1}, X_{i2}, ..., X_{il} >$ is a sequential pattern of $TRSDB_i$ and (2) all component patterns are simultaneous sequential patterns (i.e., sequential patterns $< X_{11}, X_{12}, ..., X_{1l} >$, $< X_{21}, X_{22}, ..., X_{2l} >$,..., and $< X_{k1}, X_{k2}, ..., X_{kl} >$ are simultaneous sequential patterns).

2.3 Data Transformation

Since our MDSSP is defined on time related sequence database, we need to transform access log into a time related sequence database. As mentioned in Chapter 1, assume that the access log contains both accessed data and accessed time information. Accessed data is an itemset and accessed time could be time instant (e.g., 10:35 AM) or a time range (e.g., 10:30AM~11:00AM). An example of access log is shown in Figure 2.3. Each accessed data in an access log can be mapped to one or several time slots of a certain period according to the accessed time of the accessed data (if the accessed time t_i of a accessed data x_i is an time instant, x_i belongs to one time slot; otherwise t_i is a time range, and then x_i may be mapped to multiple time slots whose time ranges have overlap with the time range of t_i.) After mapping, we merge accessed data in a time slot into an element which is a minimum super set of these accessed data. Elements in a period forms a CONTEXT_SEQ, and time slots which have been mapped in a period plus PID of the period forms a TIME_SEQ. Therefore, combining TIME_SEQ and CONTEXT_SEQ of a period becomes a time related sequence. The whole



SID	TIME_SEQ	CONTEXT_SEQ
1	1:<1,3>	<(a,b)(c,e,f)>
2	2:<1,2,3>	<(a,b,c)(c)(b,c,f)>
⋮	⋮	⋮

Figure 2.3: Build time related sequence database by access log

process of data transformation can be illustrated with Figure 2.3.

Chapter 3

Multi-domain Simultaneous Sequential Pattern Mining

In this section, we first present the algorithm *IndividualMine* in Chapter 3.1 and then an efficient algorithm, *PropagatedMine*, is proposed in Chapter 3.2. Furthermore, we develop a solution procedure that can determine a propagation order for efficient multi-domain simultaneous sequential pattern mining in Chapter 3.3.

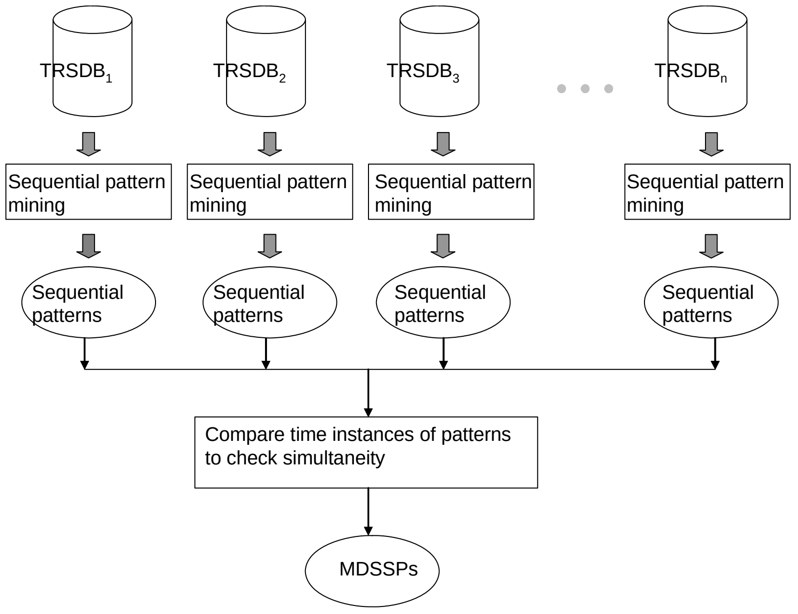
3.1 IndividualMine

MDSSPs can be found by mining sequential patterns in individual time related sequence database and then check whether these patterns mined in each domain have overlapped time or not. Figure 3.1 shows the flow of algorithm IndividualMine. In Figure 3.1, after performing sequential pattern mining method on every domain, we get a set of sequential patterns which contains all sequential patterns found in this domain, and then we compare the simultaneity of patterns in different domain for mining multi-domain simultaneous sequential patterns. Checking the simultaneity of patterns Checking the simultaneity of patterns is as follows. First, we select two domains $TRSDB_i$ and $TRSDB_j$, $1 \leq i, j \leq n$ and $i \neq j$, and for each pattern α in $TRSDB_i$ each pattern β in $TRSDB_j$, we determine whether α and β are simultaneous by examining their time instance sets. If the time instance sets for these two patterns are the same, patterns α and β is able to form a simultaneous pattern. After forming finding MDSSPs of $TRSDB_i$ and $TRSDB_j$, these MDSSPs are sent to the next domain $TRSDB_k$ to further form MDSSPs with three domains. Follow this procedure, we could easily identify multi-domain simultaneous sequential patterns

```
Algorithm: IndividualMine
Input: Time related sequence databases: $TRSDB_1, TRSDB_2, ..., TRSDB_n$,
    and the minimum support threshold $min\_support$.
Output: Simultaneous sequential patterns with the number of domains being $n$
    $(TRSDB_1, TRSDB_2, ..., TRSDB_n)$.
BEGIN
    1. Apply PrefixSpan on each time related sequence database $TRSDB_i$, $1 \leq i \leq n$.
    2. For each pattern $p$ mined in $TRSDB_1$, call $CheckSimultaneous(p, d_2)$
END

Subroutine: $CheckSimultaneous(pattern, domainID)$
/* $pattern$ is a candidate simultaneous sequential pattern, and $domainID$
  is an integer used to indicate which domain.*/
BEGIN
    1. For every sequential pattern $p'$ in $TRSDB_{domainID}$, we examine if $\begin{bmatrix} p \\ p' \end{bmatrix}$ is a
       MDSSP of $TRSDB_{domainID-1}$ and $TRSDB_{domainID}$ by checking time instance sets
       $TIS_p^{TRSDB_{domainID-1}}$ and $TIS_{p'}^{TRSDB_{domainID}}$.
    2. IF $domainID \neq d_n$
           $CheckSimultaneous(\begin{bmatrix} p \\ p' \end{bmatrix}, domainID + 1)$.
       ELSE
           Output $\begin{bmatrix} p \\ p' \end{bmatrix}$ is a MDSSP of $TRSDB_1, TRSDB_2, ..., TRSDB_n$.
END
```

Although mining MDSSP by IndividualMine is a possible solution, IndividualMine is inefficient when the size of time related sequence database is large. IndividualMine needs to find all sequential patterns in every time related sequence database, but many of these patterns may not be simultaneous patterns, so the effort spent on mining these useless patterns is a waste. In addition to mining many useless patterns, algorithm *IndividualMine* spends expensive costs on comparing time instance sets to examine the simultaneity of patterns in different domains. The wasted cost becomes even more serious when the size and the number



TRSDB$_1$
TRSDB$_2$
TRSDB$_3$
TRSDB$_n$
Sequential pattern mining
Sequential pattern mining
Sequential pattern mining
Sequential pattern mining
Sequential patterns
Sequential patterns
Sequential patterns
Sequential patterns
Compare time instances of patterns to check simultaneity
MDSSPs

Figure 3.1: Concept of IndividualMine

of sequence databases increase.

3.2 PropagatedMine

By continuously propagating currently mined patterns and their time instance sets from current domain (**propagator domain**) to the next domain (**propagated domain**), PropagatedMine is able to efficiently mine multi-domain sequential patterns. When propagating to other domains, each pattern in the current domain will construct a propagated database in the propagated domain. The propagated database constructed by pattern p in propagator domain denoted by $TRSDB_{propagator}$ is built by matching time related sequences with the time instance set of pattern p, $TIS_p^{TRSDB_{propagator}}$. After constructing propagated databases, we can find simultaneous sequential patterns in propagated databases by SSM (slot-by-slot sequential pattern mining). We will define propagated database formally in Chapter 3.2.2 and elaborate SSM in Chapter 3.2.3. The concept of PropagatedMine is illustrated in Figure 3.2.

PropagatedMine is more efficient than IndividualMine in that (1) propagated database is usually much smaller than the original time related sequence database and (2) for each

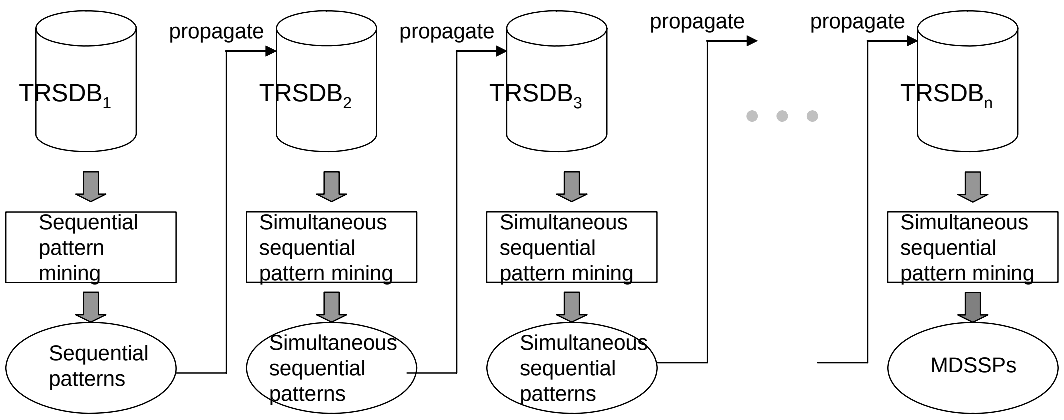

TRSDB1
propagate
TRSDB2
propagate
TRSDB3
propagate
propagate
TRSDBn
Sequential pattern mining
Simultaneous sequential pattern mining
Simultaneous sequential pattern mining
Simultaneous sequential pattern mining
Sequential patterns
Simultaneous sequential patterns
Simultaneous sequential patterns
MDSSPs

Figure 3.2: Concept of PropagatedMine

mined pattern in the propagated database, we do not need to compare it with patterns of the propagator domain to mine simultaneous patterns, because frequent patterns mined in propagated database are exactly simultaneous patterns.

Given k time related sequence databases: $TRSDB_1, TRSDB_2, ..., TRSDB_k$, suppose that the propagation order is $TRSDB_1 \rightarrow TRSDB_2 \rightarrow TRSDB_3 ... \rightarrow TRSDB_k$ when performing PropagatedMine, and then $TRSDB_1$ is called the starting domain and $TRSDB_2$ is called first propagated domain and $TRSDB_3$ is called the second propagated domain. Note that the propagation order is important in algorithm PropagatedMine. With a good propagation order, the mining space is able to significantly reduced. The detail for the generation propagation order will be elaborated in Chapter 3.3

3.2.1 Sequential pattern Mining in the Starting Domain

Sequential patterns in the starting domain (i.e., time related sequence database $TRSDB_1$) is able to be mined by performing PrefixSpan[9]. The illustrative example is given below.

Example 1. Given time related sequence database $TRSDB_1$ in Table 3.1 and $min\_support =$ 2, $TRSDB_1$ can be mined by PrefixSpan in the following steps.

1: Find frequent single-item sequences. By scanning $TRSDB_1$ once, we can get all frequent single-item sequences (i.e., <(a)>:4, <(b)>:3, <(c)>:3, <(d)>:2, and <(e)>:4, where

"<(item)>:count" represents the frequent single-item sequence and its associated support).

2: Divide search space and recursively find subsets of sequential patterns. Split the complete pattern search space into five subsets, according to the five **prefixes**: <(a)>, <(b)>, <(c)>, <(d)>, and <(e)>. Each subset is mined by constructing its corresponding projected database and recursively mining it. A **projected database** contains postfix sequences and postfix time sequences. A **postfix sequence** consists of all those items that follow the first appearance of a given prefix in any sequence in CONTEXT_SEQ part of a time related sequence database. If the first postfix item is in the same element as the last prefix item, it is represented as (_item) in order to avoid ambiguity. **A postfix time sequence** is a time sequence associated with a postfix sequence. Note that the projected database is a time related sequence database, and $\alpha-$projected database of time related sequence database $TRSDB_i$ is denoted by $TRSDB_i|_\alpha$, where α is a prefix.

Consider an example with perfix <(a)>. In order to mine sequential patterns with prefix <(a)>, $TRSDB_1|_{<(a)>}$ is constructed and shown in Table 3.1. After scanning $TRSDB_1|_{<(a)>}$, we find single-item frequent sequences which are <(b)>:3, <(c)>:3, <(d)>:2, <(e)>:3, and <(_b)>:2. All sequential patterns having prefix <(a)> can be partitioned into five subsets: (1) patterns prefixed with <(a)(b)>, (2) patterns prefixed with <(a)(c)>,.(3) patterns prefixed with <(a)(d)>, (4) patterns prefixed with <(a)(e)>, and (5) patterns prefixed with <(a,b)>. These subsets can be mined by constructing corresponding projected databases and finding frequent item in each recursively.

After completing the processing of mining <(a)>-projected database, we can mine frequent sequential patterns with prefix <(b)>, <(c)>, <(d)>, and <(e)> respectively by constructing corresponding projected databases and recursively find frequent items on projected databases. The complete set of all sequential patterns in $TRSDB_1$ are constructed by collecting sequential patterns found in <a>-, <b>-, <c>-, <d>-, and <e>-projected databases. The sequential patterns found in <a>-, <b>-, <c>-, <d>-, and <e>-projected databases are shown in Table

SID	TIME_SEQ	CONTEXE_SEQ
1	1:<1,2,3,4>	<(a)(b,c)(b,c,d)(e)>
2	2:<2,3,4>	<(a,b)(b,c)(c,e)>
3	3:<1,2,3>	<(a,e)(h)(g,j)>
4	4:<1,2,3,4>	<(a,b,f)(d)(b,c)(e,f)>

$TRSDB_1$

SID	TIME_SEQ	CONTEXT_SEQ
1	1:<2,3,4>	<(b,c)(e)(b,c,d)>
2	2:<2,3,4>	<(_b)(b,c)(c,e)>
3	3:<1,2,3>	<(_e)(h)(g,j)>
4	4:<1,2,3,4>	<(_b,f)(d)(b,c)(e,f)>

$TRSDB_1|_{<a>}$

(a)

Table 3.1: An example of one projected database

Prefix	Sequential Patterns
<(a)>	<(a)>:3, <(a)(b)>:3, <(a)(c)>:3, <(a)(d)>:2, <(a)(e)>:3, <(ab)>:2, <(a)(bc)>,<(a)(b)(c)>, <(a)(b)(e)>, <(a)(b,c)(e)> <(a)(c)(e)>:2, <(a)(c)(c)>:2, <(a,b)(b)>:2, <(a,b)(b,c)>:2, <(a,b)(b,c)(e)>:2, <(a,b)(c)>:2, <(a,b)(c)(e)>:2, <(a,b)(e)>:2
<(b)>	<(b)>:3, <(b,c)>:3, <(b)(e)>:3, <(b)(b)>:3, <(b)(c)>:3, <(b)(d)>:2, <(b,c)(e)>:3, <(b,c)(c)>:2,<(b)(b,c)>:3, <(b)(b,c)(e)>:2,<(b)(b)(e)>:2
<(c)>	<(b)(c)(e)>:2, <(c)>:3, <(c)(e)>:3, <(c)(c)>:2
<(d)>	<(d)>:2
<(e)>	<(e)>:4

Table 3.2: All sequential patterns found in $TRSDB_1$

3.2 respectively.

3.2.2 Pattern Propagation

After mining patterns in a time related database, we propagate these patterns to other domains. Since the only relationship between two time related sequence databases is TIME_SEQ attribute, not only patterns but also their corresponding time instance sets are propagated.

The time instance set of each pattern α in time related sequence database $TRSDB$ could be found easily, if we have a set PID_{α}^{TRSDB} for every α-projected database when performing PrefixSpan. PID_{α}^{TRSDB} contains period identifiers when α occurs in $TRSDB$.

For example, given time related sequence database $TRSDB_1$ in Table 3.1. It is easy to obtain periods contain <(a)(b)> when constructing $TRSDB_1|_{<(a)(b)>}$. Since periods 1, 2, and 4 contain <(a)(b)>, $PID_{<(a)(b)>}^{TRSDB_1} = \{1, 2, 4\}$ is generated. The time instance set of <(a)(b)>

is determined by scanning those time related sequences with period identifiers belonging to $PID^{TRSDB_1}_{<(a)(b)>}$, and extracting time instances of <(a)(b)>. Time instances 1:<1,2> and 1:<1,3> are found in the time related sequence with $PID = 1$ and its own CONTEXT_SEQ having (a)(b). In time related sequences with period identifiers being 1 and 2, we find time instance 2:<2,3> and 4:<1,3> respectively. Therefore, the complete time instance set of pattern <(a)(b)> in $TRSDB_1$ equals to {1:<1,2>, 1:<1,3>, 2:<2,3>, 4:<1,3>}.

After finding time instance set of every pattern in propagator domain, mined patterns with their corresponding time instance sets are propagated to propagated domains so as to construct propagated databases.

Definition 5 (propagated database) Let $TRSDB_u$ be a time related sequence database, and α is a sequential pattern in time related sequence database $TRSDB_v$ with time instance set $TIS^{TRSDB_v}_{\alpha} =< ti_1, ti_2, ..., ti_m >$. The propagated database constructed by α in $TRSDB_v$ is denoted as $TRSDB_v||_{\alpha} = \{strs_1, strs_2, ..., strs_n\}$ where $strs_i, 1 \leq i \leq n$, is time related sequence and $n \leq m$. $TRSDB_v||_{\alpha}$ is constructed as following: For each j, $1 \leq j \leq m$, if there exist a time related sequence trs in $TRSDB_v$ such that $ti_j \subseteq trs.TIME\_SEQ$, then we add sub time related sequence of trs $\{PID :< t_1, t_2, ...t_k >, < X_1, X_2, ...X_k >\}$ to $TRSDB_v||_{\alpha}$ where $PID :< t_1, t_2, ..., t_k >= ti_j$ and for each r, $1 \leq r \leq k$, X_r is the element in $trs.CONTEXT\_SEQ$ mapped by time slot t_r.

When propagating α form $TRSDB_u$ to $TRSDB_v$, $TRSDB_u$ is referred to as a **propagator domain,** and $TRSDB_v$ is view as a **propagated domain**, and α is the **propagator pattern** of **propagated database** $TRSDB_v||_{\alpha}$.

Assume that sequential pattern <(a)(b,c)> is in $TRSDB_1$. Given time related sequence database $TRSDB_1$ in Table 3.1 with sequential pattern <(a)(b,c)> and $TRSDB_2$ in Table 3.3, propagated database $TRSDB_2||_{<(a)(b,c)>}$ is constructed as follows:

Since $TIS^{TRSDB_1}_{<(a)(b,c)>}$ is {1:<1,2>, 1:<1,3>, 2:<2,3>, 4:<1,3>}, for time instance 1:<1,2> in $TIS^{TRSDB_1}_{<(a)(b,c)>}$ we extract sub-time related sequence {1:<1,2>, <(1,2)(2,3)>} from $TRSDB_2$ and add it into $TRSDB_2||_{<(a)(b,c)>}$. Similarly, time related sequences {1:<1,3>, <(1,2)(6)>}, {2:<2,3>, <(1,3)(2,4)>}, and {4:<1,3>, <(1,2,5)(2,3)>} are added to $TRSDB_2||_{<(a)(b,c)>}$. The entire $TRSDB_2||_{<(a)(b,c)>}$ is shown in Table 3.3.

Note that $TIME\_SEQs$ in a propagated database may have repeated period identifiers,

SID	TIME_SEQ	CONTEXT_SEQ
1	1:<1,2,3,4>	<(1,2)(2,3)(6)(4,5)>
2	2:<2,3,4>	<(1,3)(2,4)(8)>
3	3:<1,2,3>	<(1,6)(5)(9,10)>
4	4:<1,2,3,4>	<(1,2,5)(7)(2,3)(4,5,6)>

$TRSDB_2$

(a)

SID	TIME_SEQ	CONTEXT_SEQ
1	1:<1,2>	<(1,2)(2,3)>
2	1:<1,3>	<(1,2)(6)>
3	2:<2,3>	<(1,3)(2,4)>
4	4:<1,3>	<(1,2,5)(2,3)>

$TRSDB_2||_{<(a)(b,c)>}$

(b)

Table 3.3: $TRSDB_2$ and $TRSDB_2||_{<(a)(b,c)>}$

because a propagator pattern can have multiple time instances in one period. For example, propagator pattern <(a)(b,c)> has two time instances in period 1 (i.e., 1:<1,2> and 1:<1,3>). Thus, in $TRSDB_2||_{<(a)(b,c)>}$, we have $SIDs$ 1 and 2 having the same period identifier.

3.2.3 Sequential Pattern Mining in the Propagated Domain

After constructing propagated databases in the propagated domain, we describe how to mine simultaneous sequential patterns in the propagated databases. Observing a propagated database $TRSDB_{prop}$, we find that: the numbers of time slots and elements in every time related sequence are exactly the same with that of the propagator pattern of $TRSDB_{prop}$. If a sequential pattern β in the propagated database and propagator pattern α are simultaneous, β has the same number of elements as that in α. Accordingly, we propose a simultaneous sequential pattern mining method referred to as **SSM** (standing for **S**lot-by-**S**lot sequential pattern **M**ining). The concept of SSM is that since sequential pattern β has exactly the same number of elements as propagator pattern α, we can collect elements that map to time slots which have the same rank in $TIME\_SEQs$ to form an element set and mining these element sets step by step. If no frequent itemset is found, sequential pattern β and propagator pattern α are not simultaneous. An example of mining simultaneous patterns by SSM is described as follows.

Example 2 (SSM) Given $min\_sup = 2$, $\alpha =$<(a)(b,c)(e)>, which is a sequential pattern in $TRSDB_1$with $TIS_{\alpha}^{TRSDB_1}$ ={1:<1,2,4>, 1:<1,3,4>, 2:<2,3,4>, 4:<1,3,4>} and propagated database $TRSDB_2||_{\alpha}$ in Table 3.4, the process of SSM on $TRSDB_2||_{\alpha}$ is described as the following steps.

Step 1. Mine frequent itemsets occurred in the first time slot: Extracting the elements occurred in the first time slot of each $TIME\_SEQ$ in $TRSDB_2||_{\alpha}$ into one element set, which can be treated as a transaction database, denoted by DB_{1st_slot}. In this example, DB_{1st_slot} of $TRSDB_2||_{\alpha}$ is {1:(1,2), 1:(1,2), 2:(1,3), 4:(1,2,5)}, where the number before an itemset indicates the period in which this itemset occurs. Then we can use traditional frequent itemset mining method [1],[7],[14] to find frequent itemsets in DB_{1st_slot}. Note that the support of an itemset is counted one for the same period. Consequently, the supports of (1), (2), and (1,2) in DB_{1st_slot} are 3, 2, and 2, respectively.

Step 2. Divide search space: For every frequent itemset X found in DB_{1st_slot}, we construct the <X>-projected database in $TRSDB_2||_{\alpha}$. Projected databases in SSM is somewhat different with that in PrefixSpan, because we have mined all frequent itemsets in DB_{1st_slot}, and therefore items within the postfix and the first element is useless for every $CONTEXT\_SEQ$ in $TRSDB_2||_{\alpha}$. Consequently, items within the postfix and the first element are ignored when <X>-projected database is constructed. For example, the <(1)>-projected database in $TRSDB_2||_{\alpha}$ should contain postfixes <(_2)(2,3)(4,5)>, <(_2)(6)(4,5)>, <(_3)(2,4)(8)>, and <(_2,5)(2,3)(4,5,6)>. After ignoring items, we get new postfixes: <(2,3)(4,5)>, <(6)(4,5)>, <(2,4)(8)>, and <(2,3)(4,5,6)>. Projected database $(TRSDB_2||_{\alpha})|_{<(1)>}$ is shown in Table 3.4. Similarly, $(TRSDB_2||_{\alpha})|_{<(2)>}$ and $(TRSDB_2||_{\alpha})|_{<(1,2)>}$ are also constructed. We only list $(TRSDB_2||_{\alpha})|_{<(1,2)>}$ in Table 3.4.

Step 3. Mine frequent itemsets in the 1st_element set of split search space recursively: For every projected database found in the previous step, we mine frequent itemsets in currently DB_{1st_slot} and divide search space recursively. Consider $(TRSDB||_{\alpha})|_{<(1,2)>}$ as an example. We extract the first element in every $CONTEXT\_SEQ$ to form cur-

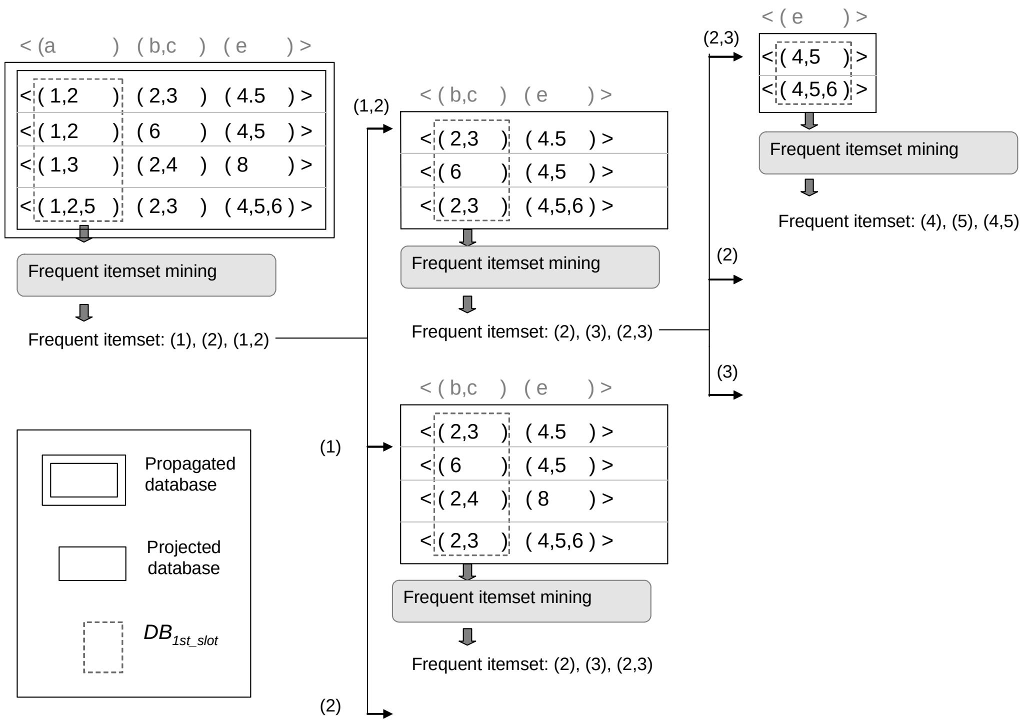


Figure 3.3: Illustrate the process of SSM in example 2

rent DB_{1st_slot}={1:(2,3), 2:(6), 4:(2,3)}. After performing frequent itemset mining method on DB_{1st_slot} of $(TRSDB||_{\alpha})|_{<(1,2)>}$, we can find frequent itemsets (2), (3) and (2,3) with the support value to be 2. Therefore, $(TRSDB_2||_{\alpha})|_{<(1,2)(2)>}$, $(TRSDB_2||_{\alpha})|_{<(1,2)(3)>}$, and $(TRSDB_2||_{\alpha})|_{<(1,2)(2,3)>}$ are constructed and frequent items are mined recursively. Since there is no element in <(1,2)(2,3)(4)>-, <(1,2)(2,3)(5)>-, and <(1,2)(2,3)(5)>-projected databases, the process of SSM on <(1,2)(2,3)>-projected database stops and returns three sequential patterns: <(1,2)(2,3)(4)>, <(1,2)(2,3)(5)>, and <(1,2)(2,3)(4,5)>. Therefore, we get three simultaneous sequential patterns for propagator pattern <(a)(b,c)(e)>, i.e., $\begin{bmatrix} (a) & (b,c) & (e) \\ (1,2) & (2,3) & (4) \end{bmatrix}$, $\begin{bmatrix} (a) & (b,c) & (e) \\ (1,2) & (2,3) & (5) \end{bmatrix}$, and $\begin{bmatrix} (a) & (b,c) & (e) \\ (1,2) & (2,3) & (4,5) \end{bmatrix}$.

Following the above procedure, we could mine all simultaneous patterns in divided space. We use Figure 3.3 to illustrate the processing of SSM with the profile given in Example 2. Note that in Figure 3.3, we omit $TIMES\_SEQ$ part in propagated and projected databases and focus on the operation performed in $CONTEXT\_SEQ$.

SID	TIME_SEQ	CONTEXE_SEQ
1	1:<1,2,3,4>	<(a)(b,c)(b,c,d)(e)>
2	2:<2,3,4>	<(a,b)(b,c)(c,e)>
3	3:<1,2,3>	<(a,e)(h)(g,j)>
4	4:<1,2,3,4>	<(a,b,f)(d)(b,c)(e,f)>

$TRSDB_1$

SID	TIME_SEQ	CONTEXT_SEQ
1	1:<1,2,3,4>	<(1,2)(2,3)(6)(4,5)>
2	2:<2,3,4>	<(1,3)(2,4)(8)>
3	3:<1,2,3>	<(1,6)(5)(9,10)>
4	4:<1,2,3,4>	<(1,2,5)(7)(2,3)(4,5,6)>

$TRSDB_2$

SID	TIME_SEQ	CONTEXT_SEQ
1	1:<1,2,4>	<(1,2)(2,3)(4,5)>
2	1:<1,3,4>	<(1,2)(6)(4,5)>
3	2:<2,3,4>	<(1,3)(2,4)(8)>
4	4:<1,3,4>	<(1,2,5)(2,3)(4,5,6)>

$TRSDB_2||_\alpha$

(a)

SID	TIME_SEQ	CONTEXT_SEQ
1	1:<2,4>	<(2,3)(4,5)>
2	1:<3,4>	<(6)(4,5)>
3	2:<3,4>	<(2,4)(8)>
4	4:<3,4>	<(2,3)(4,5,6)>

$(TRSDB_2||_\alpha)|_{<(1)>}$

(b)

SID	TIME_SEQ	CONTEXT_SEQ
1	1:<2,4>	<(2,3)(4,5)>
2	1:<3,4>	<(6)(4,5)>
3	4:<3,4>	<(2,3)(4,5,6)>

$(TRSDB_2||_\alpha)|_{<(1,2)>}$

(c)

SID	TIME_SEQ	CONTEXT_SEQ
1	1:<4>	<(4,5)>
2	4:<4>	<(4,5,6)>

$(TRSDB_2||_\alpha)|_{<(1,2)(2,3)>}$

(d)

Table 3.4: Projected databases used in Example 2

Algorithm: Slot-by-slot simultaneous sequential pattern mining (SSM.)
Input: propagator pattern α and the propagated database $TRSDB_v||\alpha$ and minimum support threshold $min\_support$.
Output: The complete MDSSPs which can be found in $TRSDB_v||\alpha$.

```
    1. call $SSM(TRSDB_v||\alpha)$.
    2. For every pattern $p$ return from previous step, if the number of elements
       in $p$ equals to the number of elements in $\alpha$, we output that $\begin{bmatrix}\alpha\\p\end{bmatrix}$ is a MDSSP.
```

```
Function $SSM(TRSDB)$
/*$TRSDB$ is a time related sequence database.*/
IF every $CONTEXT\_SEQ$ in $TRSDB$ is empty
    RETURN
ELSE BEGIN
    1. Collect every first element which is mapped by the
       first time slot in every $CONTEST\_SEQ$ of $TRSDB$
       to form the transaction database $DB_{1st\_slot}$.
    2. Mine frequent itemset in $DB_{1st\_slot}$.
    3. IF no frequent itemset can be found in $DB_{1st\_slot}$
           RETURN
       ELSE BEGIN
           1. For every found frequent itemset $X_i$, we construct
              projected database $TRSDB|_{<X_i>}$.
           2. For every $TRSDB|_{<X_i>}$, we call $SSM(TRSDB|_{<X_i>})$.
           3. For every returned pattern $p$ from previous step,
              we insert itemset $X_i$ into $p$ to form a new pattern $p'$
```

```
            such that $X_i$ is the first element in $p'$.
        4. RETURN every $p'$.
    END
END
```

When performing SSM on propagated databases in propagated domain $TRSDB_{target}$, we can ignore some propagated databases and do not need to apply SSM on them because of the following property:

Properity 1 (Reducible Propagation) Assume that both α and β are propagator patterns which are represented as $\begin{bmatrix} c_1 & c_2 & ... & c_m \end{bmatrix}$ and $\begin{bmatrix} c'_1 & c'_2 & ... & c'_n \end{bmatrix}$ respectively where c_i is a column in α and c'_j is a column is β such that $1 \leq i \leq m$, $1 \leq j \leq n$, $n > m$ and $c_1 = c'_1, c_2 = c'_2, ..., c_m = c'_m$. If no simultaneous sequential patterns can be found in $TRSDB_{traget}||\alpha$, also no simultaneous sequential patterns can be found in $TRSDB_{traget}||\beta$.

Based on the property above, we can do simultaneous sequential pattern mining more efficiently: first we perform SSM on propagated databases constructed by propagator patterns having only one column. We record which propagator patterns can not find no simultaneous sequential pattern in $TRSDB_{target}$ and then prune propagator patterns which are prefixed with these recorded propagator patterns. After doing propagation pruning, we perform SSM on propagated databases constructed by propagator patterns having two columns, and then similarly prune propagator patterns according to the result of performing SSM. Repeat this process until all propagator patterns have been propagated or pruned.

Based on the above discussion, the algorithm of PropagatedMine is presented as follows:

```
Algorithm : PropagatedMine
Input: Time related sequence databases: $TRSDB_1, TRSDB_2, ..., TRSDB_n$,
    and the minimum support threshold $min\_support$.
Output: The complete MDSSPs of $TRSDB_1, TRSDB_2, ...,$and $TRSDB_n$.
BEGIN
    1. Perform PrefixSpan on $TRSDB_1$.
    2. Construct propagated database $TRSDB_2||\alpha$ and call $SSM(\alpha, TRSDB_2||\alpha)$
       for every mined sequential pattern $\alpha$ in step 1 if $\alpha$ cannot be pruned.
    3. For every MDSSP $p$ returned form step 2, call $Propagation(p, d_3)$.
END

Subroutine: $Propagation(propagator, domainID)$
/* $propagator$ is the propagator pattern, and $domainID$
is a identifier used to indicate which domain.*/
BEGIN
    1. call $SSM(propagator, TRSDB_{domainID}||propagator)$ if $propagator$
```

```
        cannot be pruned.
    2. IF $domainID$ equals to $d_n$ BEGIN
            OUTPUT every MDSSP $p$ returned from step 1 is a MDSSP of
            $TRSDB_1, TRSDB_2, ...,$and $TRSDB_n$
        END
        ELSE BEGIN
            call $Propagation(p, domainID + 1)$ for every MDSSP $p$
            returned from step 1.
        END
END
```

3.3 Optimized Propagation Order

In the pervious section, algorithm PropagatedMine is performed with a propagation order given. In fact, multi-domain simultaneous sequential patterns can be mined by PropagatedMine with a variety of propagation orders. These propagation orders will not have any influences on the result of patterns mined.

Theorem: *Given n time related sequence database ($TRSDB_1$, $TRSDB_2$, ..., $TRSDB_n$), multi-domain simultaneous sequential patterns mined by PropagatedMine with different order are exactly the same.*

Proof: Every propagation order mines the same MDSSPs means that any MDSSP α found with propagation order $O_i = (i_1, i_2, ..., i_n)$ can ecactly maps to a MDSSP β found with propagation $O_j = (j_1, j_2, ..., j_n)$ such that α equals to β, where i_k and j_k are domain identifiers and $1 \leq i_k, j_k \leq n$. If not all propagation order find the same MDSSPs, then there must exist a MDSSP $\alpha = [r_{i_1}, r_{i_2}, ..., r_{i_n}]^T$ which having no MDSSP to map, where r_{i_k} is a component pattern,$1 \leq k \leq n$. Let $TIS = TIS|_{r_{i_1}}^{TRSDB_{i_1}} \cap TIS|_{r_{i_2}}^{TRSDB_{i_2}} \cap ... \cap TIS|_{r_{i_n}}^{TRSDB_{i_n}}$. Since α is a MDSSP, the numebr of different period identifiers in TIS must be greater than or equal to minimum support threshold. Therefore, if PropagatedMine is processed with O_j, r_{j_1} can be mined in $TRSDB_{j_1}$, because $TIS|_{r_{j_1}}^{TRSDB_{j_1}} \subseteq TIS$ and r_{j_2} can be mined by propagating TIS to $TRSDBr_{j_2}$ since $TIS|_{r_{j_1}}^{TRSDB_{j_1}} \subseteq TIS$. Similarly, propagate TIS to $TRSDBr_{j_3}, TRSDBr_{j_4}, ...,$ and $TRSDBr_{j_n}$ one by one, and we can find simultaneous sequential patterns $r_{j_3}, r_{j_4}, ...,$ and r_{j_n}, repsectively. Accordingly, $\alpha = [r_{j_1}, r_{j_2}, ..., r_{j_n}]^T$ can be mined by PropgatedMine with O_j, showing a contradiction. Q.E.D.

Though different propagation order will not affect the mining result, the costs of applying PropagatedMine with different propagation orders are not the same. This phenomenon is obvious shown in Figure 3.4. There are three rectangles that overlap with each other, and these three rectangles represent sequential patterns contained in $TRSDB_1$, $TRSDB_2$, and $TRSDB_3$, respectively. The overlapped areas represent that patterns in the overlapping area are simultaneous. Therefore, the overlapping area resulted by three rectangles contains MDSSPs that are required in this paper. Based on PropagatedMine, the central overlapping area can be mined by propagating patterns in overlapping area of $TRSDB_1$ and $TRSDB_2$ to $TRSDB_3$, or by propagating overlapping area of $TRSDB_2$ and $TRSDB_3$ to $TRSDB_1$. The former propagation order is more expensive than the latter one because more patterns are needed to be propagated in the former propagation order. Consider an illustrated example in Table 3.5, where there are three time related sequence database: $TRSDB_1$, $TRSDB_2$, and $TRSDB_3$ shown in Table 3.5(a), 3.5(b), and 3.5(c) respectively. Two propagation orders: $O_1 = (1, 2, 3)$ and $O_2 = (3, 2, 1)$ are used in algorithm PropagatedMine. The result is listed in Table 3.5(d), where "1st propagated domain" means simultaneous patterns mined in the first propagated domain, and similarly "2nd propagated domain" means patterns mined in the second propagated domain. By observing the result, propagation order O_1 has 48 sequential patterns in the starting domain and 111 simultaneous sequential patterns in the first propagated domain, which means that if we pick O_1 as our propagation order, we will need to perform 159 (48+111) times pattern propagation and SSM to find the final 6 MDSSPs. However, if we select O_2, only 44 (25+19) times pattern propagation and algorithm SSM are performed. Therefore, applying PropagatedMine with propagation order O_2 has smaller cost than applying PropagatedMine with propagation order O_1.

In addition to the number of simultaneous sequential patterns, the size of propagated databases is taken into consideration when estimating the cost of algorithm PropagatedMine. Given a propagation order O, O=(d_1, d_2, ..., d_n), where d_i is the identifier of a domain, $1 \leq i \leq n$, the cost of performing PropagatedMine can be formulated as:

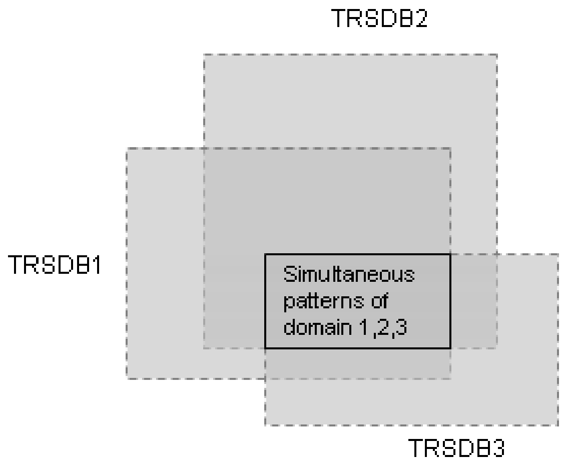


Figure 3.4: Simultaneous relation of sequential patterns in domain 1, 2, and 3

SID	TIME_SEQ	CONTEXT_SEQ
1	1:<1,2,3,4>	<(a,b)(k)(c,d)(e)>
2	2:<1,2,3,4>	<(k)(f,d)](h)(i,j)>
3	3:<2,3,4>	<(a,b)(c,d)(e)>
4	4:<1,2,3,4>	<(f)(m)(h)(i,j)>
5	5:<1,2,4>	<(a,b)(c,d)(e)>

$TRSDB_1$
(a)

SID	TIME_SEQ	CONTEXT_SEQ
1	1:<1,2,3,4>	<(1,2)(8)(3,5)(4,10)>
2	2:<1,2,3,4>	<(1,8)(5)(6,7)(9,10)>
3	3:<2,3,4>	<(1,2)(3)(4)>
4	4:<1,3,4>	<(5)(6,7)(9)>
5	5:<1,2,3,4>	<(1,2)(3)(11)(4)>

$TRSDB_2$
(b)

SID	TIME_SEQ	CONTEXT_SEQ
1	1:<2,3,4>	$<(\alpha)(\beta)(\gamma,\theta)>$
2	2:<1,2,3,4>	$<(\alpha)(\beta)(\lambda,\omega)(\gamma,\theta)>$
3	3:<1,2,3,4>	$<(\beta)(\alpha)(\phi,\rho)(\alpha)>$
4	4:<1,2,3,4>	$<(\delta)(\theta,\beta)(\phi,\beta)(\alpha,\varepsilon)>$
5	5:<2,3,4>	$<(\alpha,\beta)(\phi)(\varepsilon,\theta)>$

$TRSDB_3$
(c)

	O_1=(1,2,3)	O_2=(3,2,1)
Starting domain	48 patterns	25 patterns
1st propagated domain	111 patterns	19 patterns
2nd propagated domain	6 patterns	6 patterns

Patterns found by PropagatedMine with O_1 or O_2
(d)

Table 3.5: Time related sequence databases used to explain propagation order determining

$$Cost(PropagatedMine(O)) = Cost(PrefixSpan(TRSDB_{d_1})) + \sum_{i=2}^{n}\sum_{j=1}^{|P_{i-1}|} Cost(SSM(TRSDB_{d_i}||_{\alpha_{i-1,j}}))$$

For each i, $2 \le i \le n$, P_{i-1} is the set which contains all simultaneous sequential patterns found in time related sequence database $TRSDB_{d_{i-1}}$ with the propagation order: $(d_1, d_2, ..., d_{i-1})$. $|P_{i-1}|$ denotes the number of patterns in P_{i-1}. $\alpha_{i-1,j}$ represents a simultaneous sequential pattern contained in P_{i-1} (i.e., $\alpha_{i-1,j} \in P_{i-1}$,where $2 \le i \le n$ and $1 \le j \le |P_{i-1}|$). The cost of performing PrefixSpan or SSM on a time related sequence database includes frequent items/itemsets mining, projected databases construction, and time instances search. Note that the cost of PropagatedMine is mainly dependent on total amount of database scans. Therefore, the number of database scans is viewed as the main cost of applying PrefixSpan or SSM.

Since different propagation orders have different mining costs, a propagation order with the minimal cost is desired. Unfortunately, estimating the costs of PrefixSpan and SSM are difficult because patterns are not found in advance. Thus, an efficient method to approximately estimate the costs of PrefixSpan and SSM is proposed.

We develop a greedy method to progressively generate the optimized propagation order so as to improve the performance of PropagatedMine. Judiciously selecting the starting domain and the propagated domain is able to reduce the amount of mining spaces required. Note that instead of generating full propagation order once at a time, the optimized propagation order selection is embedded in algorithm PropagatedMine and the cost estimation method will determine how to efficiently select the next propagated domain when needing to perform a propagation in algorithm PropagatedMine. Optimized propagation order method can be divided into two phases: (1) the selections of the starting domain and the first propagated domain and (2) the selection of the next propagated domain. These two phases will be described in the following sections.

3.3.1 Selecting the Starting Domain and the First Propagated Domain

The starting domain and the first propagated domain can be determined as following: In the beginning, we perform PrefixSpan cost estimation method to estimate the cost of PrefixSpan in each domain $TRSDB_i$, and pick the domain with the minimal PrefixSpan cost as the starting domain (denoted by $TRSDB_{start}$). Furthermore, for every other domains $TRSDB_i$, $1 \leq i \leq n$ and $TRSDB_i \neq TRSDB_{start}$, we estimate the cost of performing SSM on all propagated databases in $TRSDB_i$. Similarly, we select the domain with the minimum SSM cost as our first propagated domain (denoted as $TRSDB_{1st\_propagated}$).

Synopsises Construction

To evaluate the costs of PrefixSpan and SSM, we scan each time related sequence database and compute some statistic data for every time related sequence database as follows.

(1). Average length of context sequences: the average length of context sequences of a time related sequence database $TRSDB$ (denoted as seq_length_{TRSDB}.) is defined as:

$$seq\_length_{TRSDB} = \frac{\sum_{i=1}^{|TRSDB|} |trs_i.CONTEXT\_SEQ|}{|TRSDB|}$$

$|TRSDB|$ represents the number of time related sequences in $TRSDB$. trs_i is a time related sequence in $TRSDB$ with $trs_i.SID = i$, and $|trs_i.CONTEXT\_SEQ|$ represents the length of the associated context sequence $trs_i.CONTEXT\_SEQ$, $1 \leq i \leq |TRSDB|$. Consider an example of a time related sequence database $TRSDB_{ex}$ in Table 3.6(a), $seq\_length_{TRSDB_{ex}} =$

$\frac{7+7+5+7+6}{5} = 6.4$.

(2). Average length of elements: the average length of elements of a time related sequence database $TRSDB$ (denoted as $elem\_length_{TRSDB}$) is defined as:

$$elem\_length_{TRSDB} = \frac{\sum_{i=1}^{|TRSDB|} \frac{|trs_i.CONTEXT\_SEQ|}{|trs_i.TIME\_SEQ|}}{|TRSDB|}$$

Since time slots in $trs_i.CONTEXT\_SEQ$ and elements in $trs_i.TIME\_SEQ$ are one to one mapped, the number of elements in $trs_i.CONTEXT\_SEQ$ equals to the length of $trs_i.TIME\_SEQ$. For example in Table 3.6(a), $elem\_length_{TRSDB_{ex}} = \frac{\frac{7}{5}+\frac{7}{5}+\frac{5}{4}+\frac{7}{4}+\frac{6}{4}}{5} = 1.38$.

(3). Period Synopsis structure: for every length one sequential pattern x_i in time related sequence database $TRSDB$, we build a period synopsis structure denoted by $PID\_Synopsis_{x_i}$. Actually, $PID\_Synopsis_{x_i}$ is a table having two attributes: PID and COUNT, where PID indicates item x_i appeared in which periods and COUNT represents the number of counts, where x_i appears in that period. If $PID\_Synopsis_{x_i}$ has k tuples, they are expressed by $pst_1, pst_2, ..., pst_k$. For example, given a time related sequence database $TRSDB_{ex}$ in Table 3.6(a), $PID\_Synopsis_a$ for frequent item a of $TRSDB_{ex}$ is shown in Table 3.6(b).

(4). Position Synopsis structure: we build a position synopsis structure denoted as $POS\_Synopsis_{x_i}$ for every length one sequential patterns x_i in $TRSDB$. $POS\_Synopsis_{x_i}$ is a table with three attributes: RANK, POSITION, SUPPORT. Given a time related sequence database $TRSDB_{ex}$ in Table 3.6(a), we can build $POS\_Synopsis_a$ for frequent item a of $TRSDB_{ex}$. $POS\_Synopsis_a$ is shown in Table 3.6(b). The first tuple {RANK=1, POSITION=1.8, SUPPORT=5} means that there are 5 periods (PID=1, 2, 3, 4, and 5) in $TRSDB_{ex}$ contain the first appearance of "a" (RANK=1) and the average position of these five "a" is 1.8 because their positions in periods are element 2, 1, 2, 4, 1, respectively, and thus, the average position is 1.8 ($\frac{2+1+1+4+1}{5}$). Similarly, The second tuple {RANK=2, POSITION=2, SUPPORT=3} means that there are 3 periods (with PID=2, 3, and 5) in $TRSDB_{ex}$

SID	TIME_SEQ	CONTEXT_SEQ
1	1:<1,2,3,4,5>	<(c)(a)(e)(b,c)(g,h)>
2	2:<1,2,3,4,5>	<(a,f)(a)(b)(f)(a,b)>
3	3:<2,3,4,5>	<(a)(a)(e)(a,h)>
4	4:<1,2,3,4>	<(c,f)(c)(f)(a,b,c)>
5	5:<1,3,4,5>	<(a,c)(a)(g)(c,g)>

$TRSDB_{ex}$
(a)

PID	COUNT
1	1
2	3
3	3
4	1
5	2

$PID\_Synopsis_a$
(b)

RANK	POSITION	SUPPORT
1	1.8	5
2	2	3
3	4.5	2

$POS\_Synopsis_a$
(c)

PID	COUNT
1	1
2	3
3	3
5	2

Revised $PID\_Synopsis_a$
(d)

RANK	POSITION	SUPPORT
1	1.25	4
2	2	3
3	4.5	2

Revised $POS\_Synopsis_a$
(e)

Table 3.6: Synposis examples

contain the second appearance of "a" (RANK=2) and the average position of these "a" is 2, since their position in period 2, 3, and 5 are 2, 2, 2,respectively. As a result, the average position of the second appearance of "a" is 2 ($\frac{2+2+2}{3}$). Assume that $POS\_Synopsis_{x_i}$ has k tuples, and these tuples are denoted as $sst_1, sst_2, sst_3, ..., sst_k$.

Note that we can build $PID\_Synopsis_{x_i}$ and $POS\_Synopsis_{x_i}$ for every frequent item x_i in time related sequence database $TRSDB$ by scanning $TRSDB$ once. However we need to scan $TRSDB$ once more for revising every $PID\_Synopsis_{x_i}$ and $POS\_Synopsis_{x_i}$. The revising operation is to ignore positions which are far away from the average position and the revising operation makes revised average position more representative. The revising operation is presented as following: for every tuple sst_j in $POS\_Synopsis_{x_i}$, suppose that $sst_j.RANK = k$, we recompute the value of POSITION and SUPPORT by exclude those

k-th x_i whose distance between x_i and $sst_j.POSTION$ is larger than the standard deviation of all k-th x_i derived in the first disk scan. Then, $PID\_Synopsis_{x_i}$ is updated accordingly. Also, if the revised value of SUPPORT in sst_j is less than $min\_support$, we remove sst_j from $POS\_Synopsis_{x_i}$ and $PID\_Synopsis_{x_i}$. For example, the standard deviation of all first "a" in $TRSDB_{ex}$ equals to 1.166, and the first "a" in time related sequence with SID=4 satisfies that the distance between its position and the average position ($|4-1.8| = 2.2$) is larger than the standard deviation 1.166. Therefore, we exclude this tuple. The final revised $PID\_Synopsis_a$ and $POS\_Synopsis_a$ are in Table 3.6(d) and 3.6(e), respectively.

Cost of PrefixSpan

The total scan space of performing PrefixSpan on a time related sequence database $TRSDB$ can be calculated as:

$$Cost(PrefixSpan(TRSDB)) = Space(TRSDB) + \sum_i Space(TRSDB|_{\alpha_i})$$

where $\sum_i Space(TRSDB|_{\alpha_i})$ is the summation of space of projected databases constructed by sequential pattern α_i and function $Space(TRSDB) = \sum_{i=1}^{|TRSDB|} |trs_i.CONTEXT\_SEQ|$. Based on the definition of seq_length_{TRSDB}, $Space(TRSDB)$ equals to: $Space(TRSDB) = |TRSDB| \times seq\_length_{TRSDB}$. Therefore, the definition of PrefixSpan cost can be rewritten as following:

$$\begin{aligned} Cost(PrefixSpan(TRSDB)) &= seq\_length_{TRSDB} \times |TRSDB| + \\ &\sum_i (|(TRSDB|_{\alpha_i})| \times seq\_length_{TRSDB|_{\alpha_i}}) \end{aligned} \quad (3.1)$$

Formula (3.1) is illustrated in Figure 3.5. Every rectangle in the figure is a database whose space can be evaluated by multiply the number of sequences with the average sequence length of that database.

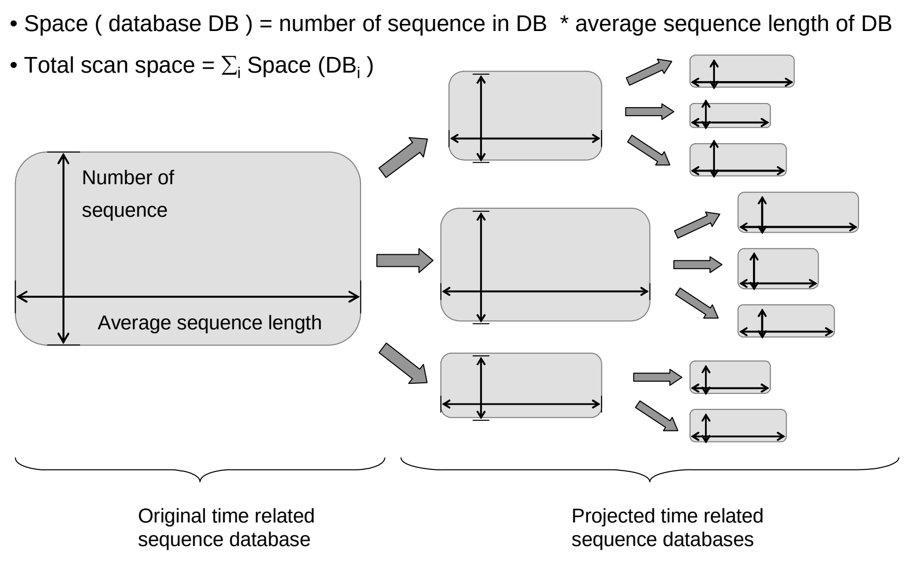

• Space (database DB) = number of sequence in DB * average sequence length of DB
• Total scan space = Σ_i Space (DB$_i$)
Number of sequence
Average sequence length
Original time related sequence database
Projected time related sequence databases

Figure 3.5: Illustrate PrefixSpan cost formula

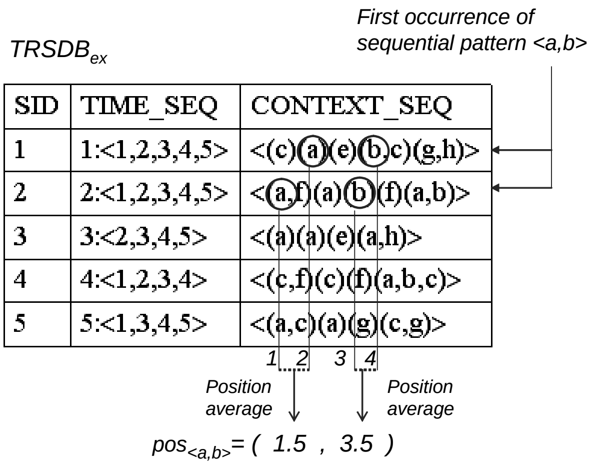
Based on formula (3.1), the method of PrefixSpan cost estimation can be designed by add the scan space of every projected database constructed by some **representative sequential patterns** together. The concept of representative sequential pattern proposed here helps estimating cost efficiently, since representative sequential patterns have two properties: (1) every element in the sequential pattern contains only one item, and (2) the sequential pattern often occurs in similar time slots in different periods. Synopsises of a time related sequence database are used to generate representative sequential patterns in PrefixSpan cost estimation method.

To facilitate the presentation of this paper, some terminologies and functions are defined as follows.

(1).**Length**() takes a real number r as its parameter, and $Length(r)$ is defined as: $Length(r) = r$, if $r \geq 1$; $Length(r) = 1$, if $r < 1$.

(2) $\mathbf{PID_{\alpha}^{TRSDB}}$ is a set that contains period identifiers. For each period identifier pid in PID_{α}^{TRSDB}, it represents that there will be a time related sequence trs existing in $TRSDB$ such that $trs.TIME\_SEQ.PID = pid$ and $trs.CONTEXT$ contains sequential pattern α.

(3) $\mathbf{POS_{\alpha}^{TRSDB}}$ is a list used to record the average position of all first occurring α in



SID	TIME_SEQ	CONTEXT_SEQ
1	1:<1,2,3,4,5>	<(c)(a)(e)(b,c)(g,h)>
2	2:<1,2,3,4,5>	<(a,f)(a)(b)(f)(a,b)>
3	3:<2,3,4,5>	<(a)(a)(e)(a,h)>
4	4:<1,2,3,4>	<(c,f)(c)(f)(a,b,c)>
5	5:<1,3,4,5>	<(a,c)(a)(g)(c,g)>

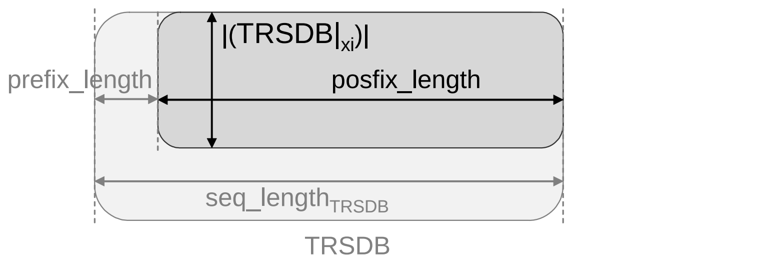
Figure 3.6: Average position of sequential pattern <a,b> in $TRSDB_{ex}$

time related sequence database $TRSDB$, where α is a sequential pattern in $TRSDB$. Figure 3.6 is used to illustrate $POS_{<a,b>}^{TRSDB_{ex}}$ in time related sequence database $TRSDB_{ex}$: there are two context sequences having $< a, b >$ in $TRSDB_{ex}$ (i.e., context sequences with SID equal to 1 and 2) and the average position of the first "a" and the first "b" in these two sequence are $\frac{2+1}{2} = 1.5$ and $\frac{4+3}{2} = 3.5$. Given POS_{α}^{TRSDB} and $\alpha =< a_1, a_2, .., a_k >$, function $\mathbf{pos}(\mathbf{POS_{\boldsymbol{\alpha}}^{TRSDB}}, \mathbf{a_i})$ will return the position of a_i in POS_{α}^{TRSDB}, where $a_i \in \{a_1, a_2, .., a_k\}$. $\mathbf{pos}(\mathbf{POS_{\boldsymbol{\alpha}}^{TRSDB}})$ will return the last value in POS_{α}^{TRSDB}.

The Cost Estimation Method of PrefixSpan

We use GSP[11] like method to generate candidate sequential patterns (i.e., length k sequential patterns are generated from length $k-1$ sequential patterns), and utilize synopsises to make sure whether a candidate sequential pattern is a representative sequential pattern or not. We also use the synopsises to estimate the databases constructed by representative sequential patterns. For conciseness, when saying sequential patterns in PrefixSpan cost estimation method we indicate representative sequential patterns actually. The cost of performing PrefixSpan on a domain can be estimated by the following two steps and we add all scan space computed by these two steps together as the whole PrefixSpan cost.

Step 1. projected databases of length 1 representative sequential patterns:



Scan space = the number of sequences * posfix_length

=|(TRSDB|$_{xi}$)| * (seq_length$_{TRSDB}$ – prefix_length$_{TRSDB}$)

=|(TRSDB|$_{xi}$)| * (seq_length$_{TRSDB}$ – pos(POS$_{xi}^{TRSDB}$)*elem_length$_{TRSDB}$)

Figure 3.7: Illustrate how to calculate scan space of step 1 in the PrefixSpan cost estimation method

In this step, we calculate scan space of projected databases constructed by length 1 sequential patterns. Suppose we have n length-1 sequential patterns, $x_1, x_2, ..., x_n$, in $TRSDB$. The number of sequences contained in projected database of x_i is $|(TRSDB|_{x_i})|$, $1 \leq i \leq n$, which will equal to $|PID_{x_i}^{TRSDB}|$. Moreover, we use $Length(seq\_length_{TRSDB} - POS_{x_i}^{TRSDB} \times elem\_length_{TRSDB})$ to estimate $seq\_length_{TRSDB|_{x_i}}$. Therefore, according to scan space formula, the scan space of projected database constructed by frequent item $x_i, 1 \leq i \leq n$, is as follows and illustrated in Figure 3.7.

$$|PID_{<x_i>}^{TRSDB}| \times Length(seq\_length_{TRSDB} - pos(POS_{<x_i>}^{TRSDB}) \times elem\_length_{TRSDB})$$

Selecting sst form $POS\_Synopsis_{x_i}$ with $sst.RANK = 1$, and then the formula above can be computed by letting (1). $|PID_{x_i}^{TRSDB}| = sst.SUPPORT$ and

(2). $POS_{x_i}^{TRSDB} = sst.POSITION.$

Step 2. Projected databases of length k representative sequential patterns where k above 1:

We generate length k sequential patterns from length $k-1$ sequential patterns. Rules of pattern generation and pruning are similar to those in GSP[11] (i.e., a candidate length k sequential pattern p is generated by two length $k-1$ sequential patterns where their first $k-2$

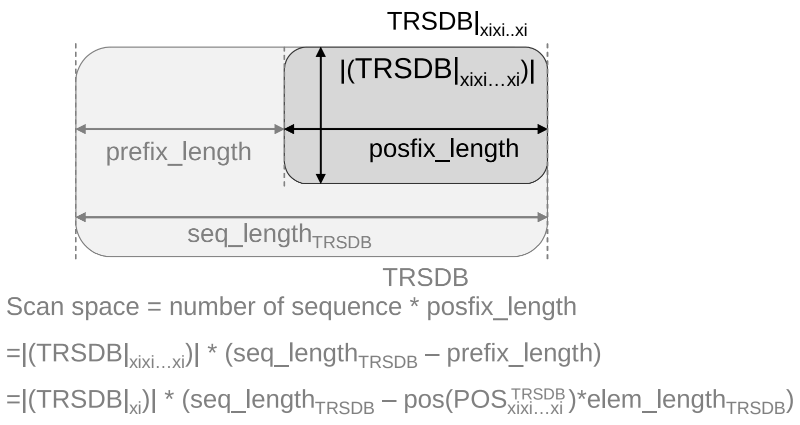

TRSDB|xixi..xi
|(TRSDB|xixi...xi)|
prefix_length
posfix_length
seq_lengthTRSDB
TRSDB
Scan space = number of sequence * posfix_length
=|(TRSDB|xixi...xi)| * (seq_lengthTRSDB – prefix_length)
=|(TRSDB|xi)| * (seq_lengthTRSDB – pos(POS TRSDB xixi...xi)*elem_lengthTRSDB)

Figure 3.8: Illustrate how to calculate scan space of the case 1 in step 2 of the PrefixSpan cost estimation method

items are the same and every length $k-1$ subsequence of p must be a sequential pattern). Length k sequential patterns could be categorized into two kinds:

Case 1: (Homogeneous sequential patterns)

A length k sequential pattern of this kind will have the form like $x_ix_i...x_i$ and $|x_ix_i...x_i| = k$, $1 \leq i \leq n$. Scan space of the projected database constructed by $x_ix_i...x_i$ is as follows and illustrated in Figure3.8.

$$|PID^{TRSDB}_{x_ix_i...x_i}| \times Length(seq\_length_{TRSDB} - pos(POS^{TRSDB}_{x_ix_i...x_i}) \times elem\_length_{TRSDB})$$

If there exists a tuple sst_j in $POS\_Synopsis_{x_i}$ such that $sst_j.RANK = k$, then the formula above can be computed by letting (1). $|PID^{TRSDB}_{x_ix_i...x_i}| = sst_j.SUPPORT$ and (2). $pos(POS^{TRSDB}_{x_ix_i...x_i}) = sst_j.POSITION$

Though we can compute $|PID^{TRSDB}_{x_ix_i...x_i}|$ by $POS\_Synopsis_{x_i}$ directly, we still need to build $PID_{x_ix_i...x_i}$ which will be used in estimating scan space of projected databases constructed by length $k+1$ sequential patterns. $PID^{TRSDB}_{x_ix_i...x_i}$ can be obtained by scanning $PID\_Synopsis_{x_i}$ once. When scanning $PID\_Synopsis_{x_i}$, we add $pst_j.PID$ into $PID^{TRSDB}_{x_ix_i...x_i}$ if $pst_j.COUNT \geq k$.

Case 2: (Heterogeneous sequential patterns)

Given a candidate length k sequential pattern $\alpha = a_1a_2...a_k$, $a_i \in \{x_1, x_2, ..., x_n\}$ for $1 \leq i \leq k$ and α is a homogeneous sequence, α is a sequential pattern in $TRSDB$ if α satisfies two conditions: (Condition 1) $pos(POS^{TRSDB}_{a_1a_2,...a_k}) - pos(POS^{TRSDB}_{a_1a_2,...a_{k-1}}) \geq 1$ and (Condition 2) $|PID^{TRSDB}_{a_1a_2...a_k}| \geq min\_support$.

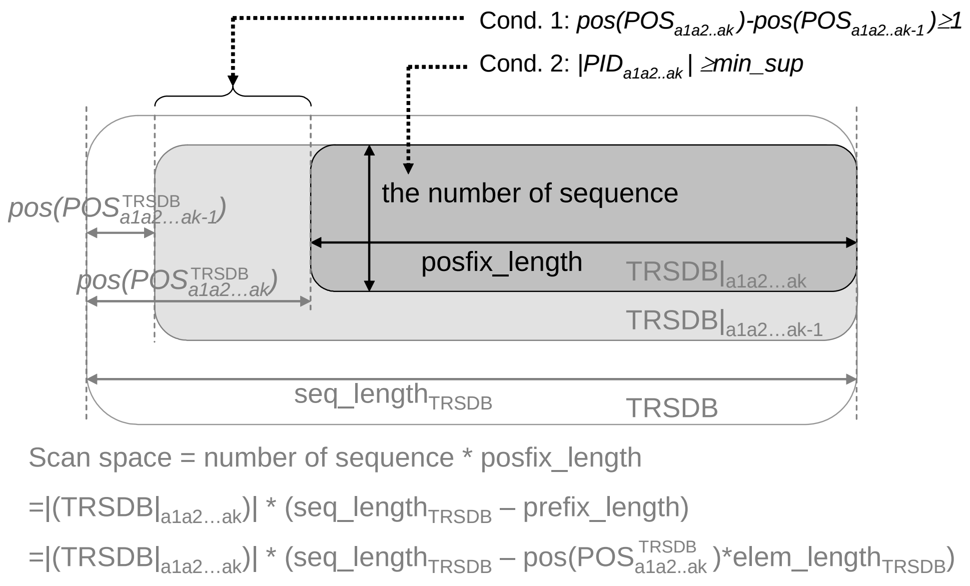
(1) **Condition 1:** Examining $POS\_Synopsis_{a_k}$ to see if there exist a tuple sst in $POS\_Synopsis_{a_k}$ such that $sst.POSITION - pos(POS^{TRSDB}_{a_1a_2,...a_{k-1}}) \geq 1$. If condition 1 can be satisfied, then $POS^{TRSDB}_{a_1a_2...a_k}$ equals to $POS^{TRSDB}_{a_1a_2,...a_{k-1}} + sst.POSITION$, where "+" means append $sst.POSITION$ to $POS^{TRSDB}_{a_1a_2,...a_{k-1}}$ such that $sst.POSITION$ is the minimal value satisfying condition 1 in $POS\_Synopsis_{a_k}$. Note that $POS^{TRSDB}_{a_1a_2,...a_{k-1}}$ has already been computed in the generation of length $k-1$ sequential patterns.

(2) **Condition 2:** $|PID^{TRSDB}_{a_1a_2...a_k}|$ can be estimated as follows. Suppose variable $rank$ equals to $sst.RANK$ where sst is the synopsis tuple we found in condition 1. We can build $PID^{TRSDB}_{rank\_th\_a_k}$ by scanning $PID\_Synopsis_{a_k}$. When scanning $PID\_Synopsis_{a_k}$, we add $pst.PID$ into $PID^{TRSDK}_{rank\_th\_a_k}$ if $pst.COUNT \geq rank$. After $PID^{TRSDB}_{rank\_th\_a_k}$ has been built, $PID^{TRSDB}_{a_1a_2...a_k}$ equals to $PID^{TRSDB}_{a_1a_2,...a_{k-1}} \cap PID^{TRSDB}_{rank\_th\_a_k}$, and then we can check if $|PID^{TRSDB}_{a_1a_2...a_k}| \geq min\_support$. Note that since $\alpha = a_1a_2...a_k$ is a candidate sequential pattern, $a_1a_2, ...a_{k-1}$ must be a sequential pattern. Therefore, $PID^{TRSDB}_{a_1a_2,...a_{k-1}}$ has already been built when generating length $k-1$ sequential patterns.

If $\alpha = a_1a_2...a_k$ satisfies condition 1 and 2, then scan space of the projected database constructed by α can be formulated as following and illustrated with Figure 3.9.

$$|PID^{TRSDB}_{a_1a_2...a_k}| \times Length(seq\_length_{TRSDB} - pos(POS^{TRSDB}_{a_1a_2...a_k}) \times elem\_length_{TRSDB})$$

The formula above can be computed by substituting the value of $|PID^{TRSDB}_{a_1a_2...a_k}|$ and $pos(POS^{TRSDB}_{a_1a_2...a_k})$ which can be found in judging condition 1 and condition 2.



Scan space = number of sequence * posfix_length

$=|(TRSDB|_{a1a2...ak})| * (seq\_length_{TRSDB} - prefix\_length)$

$=|(TRSDB|_{a1a2...ak})| * (seq\_length_{TRSDB} - pos(POS^{TRSDB}_{a1a2..ak}) * elem\_length_{TRSDB})$

Figure 3.9: Illustrate how to calculate scan space of the case 2 in step 2 of the PrefixSpan cost estimation method

Cost of SSM

The cost of performing SSM on the propagated domain is estimated by finding out the scan space of propagated databases in the propagated domain. As illustrated in Figure 3.10, SSM is performed in propagated databases constructed by frequent patterns in $TRSDB_u$.

Given a pair $(TRSDB_u, TRSDB_v)$, where $TRSDB_u$ is the propagator domain having n length 1 sequential patterns (i.e., $\{x_1, x_2, ..., x_n\}$), and $TRSDB_v$ is the propagated domain having m length 1 sequential patterns (i.e., $\{y_1, y_2, ..., y_m\}$), we will use $PID_\alpha^{TRSDB_u}$ and $POS_\alpha^{TRSDB_u}$ to estimate scan space of performing SSM on $TRSDB_v||_\alpha$ for each representative sequential pattern $\alpha = a_1a_2...a_k$ found in $TRSDB_u$,where $a_i \in \{x_1, x_2, ..., x_n\}$ and $1 \leq i \leq k$. The cost of performing SSM in propagated database $TRSDB_v||_\alpha$ can be estimated as:

$$\begin{aligned} Cost(SSM(TRSDB_v||_\alpha)) &= |(TRSDB_v||_\alpha)| \times elem\_length_{TRSDB_v} + \\ &\quad ssm(TRSDB_v||_\alpha, prefix, c) \end{aligned} \tag{3.2}$$

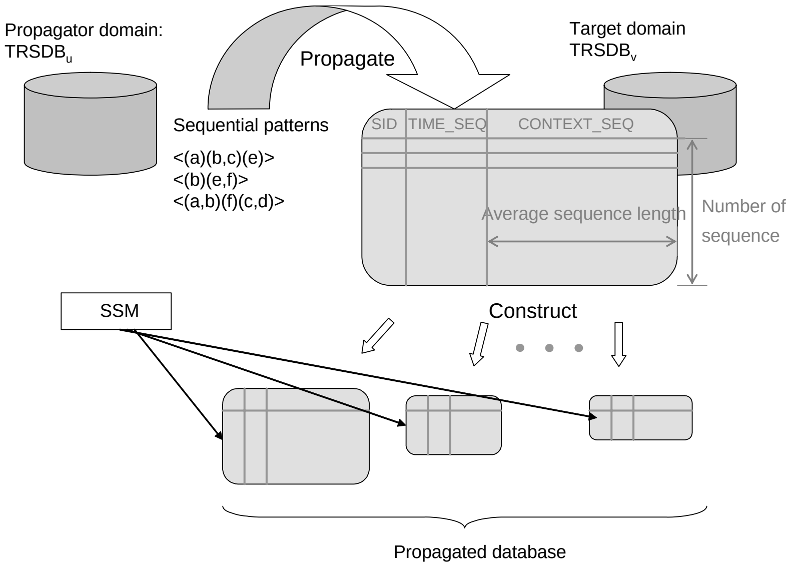


Figure 3.10: Illustrate SSM cost estimation

where $ssm(TRSDB_v||_\alpha, prefix, c)$ is defined as:

$$\begin{aligned} ssm(TRSDB_v||_\alpha, prefix, c) &= \sum_j |PID_{y_j}^{DB_{1st\_slot}}| \times elem\_length_{TRSDB_v} + \\ & ssm(TRSDB_v||_\alpha, prefix + y_j, c+1) \end{aligned} \qquad (3.3)$$

DB_{1st_slot} represents the first element set of $(TRSDB_v||_\alpha)|_{prefix}$. $prefix$ is a representative sequential pattern in $TRSDB_v$, and it is a empty sequence initially. According to formula (3.3), every time we append a frequent item y_j found in current DB_{1st_slot} to $prefix$ and call $ssm()$ recursively. Parameter c is a variable used to count the projection depth where c equals to 1 in the beginning.

The Cost Estimation Method of SSM

If $\alpha = a_1a_2...a_k$ is a length k sequential pattern in $TRSDB_u$, we can compute the total scan space of performing SSM on $TRSDB_v||_\alpha$ by formulas (3.2) and (3.3). Since the former half in formula (3.2) is a constant (i.e., $|(TRSDB_v||_\alpha)| \times elem\_length_{TRSDB_v}$), we focus on explain the later half which is defined in formula (3.3). Formula (3.3) can divide into non-recursive and recursive parts, so we first compute non-recursive part, $\sum_j |PID_{y_j}^{DB_{1st\_slot}}| \times$ $elem\_length_{TRSDB_v}$, in every $ssm()$ function call. $\sum_j |PID_{y_j}^{DB_{1st\_slot}}|$ is estimated by the following steps:

Step 1. find all possible frequent item y_j:

We scan $POS\_Spnopsis_{y_j}$ for each frequent item y_j in $TRSDB_v$ ($y_j \in \{y_1, y_2, ..., y_m\}$). If there exist a tuple sst in $POS\_Spnopsis_{y_j}$ such that $|sst.POSITION - pos(POS_{\alpha}^{TRSDB_u}, a_c)| < 1$, then y_j is a possible frequent item in DB_{1st_slot}. Note that c is the projection depth counter and c equals to 1 in initial $ssm()$ function call.

Step 2. prune y_j:

We further examine whether $prefix + y_j$ is a representative sequential pattern of $TRSDB_v$ (i.e. $prefix + y_j$ must equals to one of sequential patterns found in estimating PrefixSpan cost of $TRSDB_v$.) If $prefix + y_j$ is not a sequential pattern of $TRSDB_v$, we prune y_j. Note that $prefix$ equals to empty sequence in initial $ssm()$ function call.

Step 3. compute $\sum_j |PID_{y_j}^{DB_{1st\_slot}}|$:

For every found possible frequent item y_j, if $|PID_{y_j}^{DB_{1st\_slot}}| \geq min\_support$ we add these $|PID_{y_j}^{DB_{1st\_slot}}|$ together and get $\sum_j |PID_{y_j}^{DB_{1st\_slot}}|$. $|PID_{y_j}^{DB_{1st\_slot}}|$ equals to $|PID_{k\_th\_y_j}^{TRSDB_v} \cap PID_{prefix}^{TRSDB_v}|$ (or $|PID_{k\_th\_y_j}^{TRSDB_v} \cap PID_{\alpha}^{TRSDB_u}|$ in the initial $ssm()$ function call) where $PID_{prefix}^{TRSDB_v}$ has been known in previous $ssm()$ function call and $PID_{k\_th\_y_j}^{TRSDB_v} = \{pst.PID | pst \in PID\_Synopsis_{y_j}, pst.COUNT = k\}$ where k equals to the $RANK$ value of the position synopsis tuple sst found in Step 1. In other words, k equals to $sst.RANK$ where $sst \in PID\_Synopsis_{y_j}$ and $|sst.POSITION - pos(POS_{a}^{TRSDB_u}, a_c)| < 1$.

After computing the non-recursive part in $ssm()$ function call, we further compute the recursive part by calling $ssm(TRSDB_v||_{\alpha}, prefix + y_j, c+1)$ for each y_j satisfying $|PID_{y_j}^{DB_{1st\_slot}}| \geq min\_support$ which can be found in Step 3. Note that the set $PID_{y_j}^{DB_{1st\_slot}}$ we found in step 3 currently will become $PID_{prefix}^{TRSDB_v}$ in the next $ssm()$ function call whose projected database is prefixed with current $prefix$ appended with y_j. We use Figure 3.11 to illustrate the process

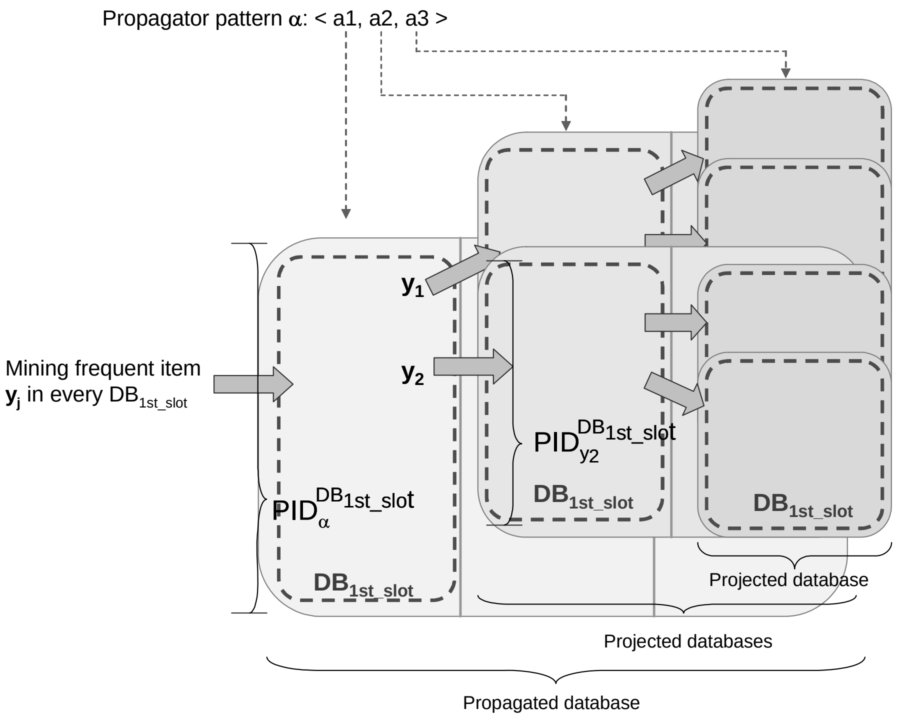


Figure 3.11: Illustrate the process of SSM cost estimation method

of SSM cost estimation method.

3.3.2 Continuously Selecting the Next Propagated Domain

As mentioned above, instead of finding the optimized propagation order at once, our method is able to progressively generate the propagation order. After the starting and the first propagated domains have been selected (e.g., $TRSDB_{start}$ and $TRSDB_{1st\_propagated}$, respectively), we will perform PrefixSpan on $TRSDB_{start}$ and then propagate all mined sequential patterns to $TRSDB_{1st\_propagated}$ and apply SSM on propagated databases to find out all MDSSPs among $TRSDB_{start}$ and $TRSDB_{1st\_propagated}$.

After MDSSPs of $TRSDB_{start}$ and $TRSDB_{1st\_propagated}$ have mined, we need to select the next propagated domain from remaining domains. However, instead of recomputing the SSM costs for remaining domains, the costs of SSM is able to incrementally computed. The incremental update of these costs can be done if information, $\{(propagator\_pattern_1 : cost_1), (propagator\_pattern_2 : cost_2), ...\}$, are recorded in each domain when estimating the cost of SSM for selecting the first propagated domain. The update can be done by subtracting scan space of propagator patterns which are not component patterns of MDSSPs found in $TRSDB_{1st\_propagated}$ after performing SSM and mined MDSSPs of $TRSDB_{1st\_propagated}$. Once the cost costs of SSM are updated, we select the domain having the minimal updated cost as our next propagated domain (i.e., second propagated domain $TRSDB_{2nd\_propagated}$). Similarly, if wanting to select the third propagated domain, we update SSM cost table for every remaining domains by subtracting $(propagator\_pattern_i, cost_i)$ from cost table if $propagator\_pattern_i$ is not a component patterns of any MDSSPs found in $TRSDB_{2nd\_propagated}$. For example, in Figure 3.12, there are four time related sequence databases $TRSDB_1$, $TRSDB_2$, $TRSDB_3$, and $TRSDB_4$, and suppose that {a,b,c,...,l} are representative sequential patterns in $TRSDB_1$. If $TRSDB_1$ is selected as starting domain, we can build a SSM cost table shown in Table 3.7 after estimating the cost of performing SSM on $TRSDB_2$, $TRSDB_3$, and $TRSDB_4$. Since $TRSDB_2$ has the minimal SSM cost, $TRSDB_2$ is chosen as the first propagated domain. Similarly, after mining MDSSPs of $TRSDB_1$ and $TRSDB_2$, we update SSM cost table of $TRSDB_3$ and $TRSDB_4$ by subtracting tuple $(propagator\_pattern_i : cost_i)$ from SSM cost table, where $propagator\_pattern_i$ is not a component pattern of MDSSPs of $TRSDB_1$ and $TRSDB_2$. Assume that all representative patterns {a,b,c,e,d,k,j} in SSM cost table of $TRSDB_2$ are component patterns of MDSSPs, and then the SSM cost table of $TRSDB_3$ and $TRSDB_4$ can be updated accordingly. The updated SSM cost tables are in Table 3.8.

Optimized PropagatedMine algorithm is listed below and the whole architecture is shown in Figure 3.13.

```
**Algorithm :** *Optimized PropagatedMine*
**Input:** Time related sequence databases: $TRSDB_1, TRSDB_2, ..., TRSDB_n$,
    and the minimum support threshold $min\_support$.
**Output:** The complete MDSSPs of $TRSDB_1, TRSDB_2, ...$,and $TRSDB_n$.
**BEGIN**
    **1.** Perform PrefixSpan cost estimation on each $TRSDB_i$, $1 \leq i \leq n$, and
       select the domain with minimum cost as the starting domain denoted as
       $TRSDB_{start}$.
    **2.** Apply SSM cost estimation on each $TRSDB_i$, $1 \leq i \leq n$ and
```

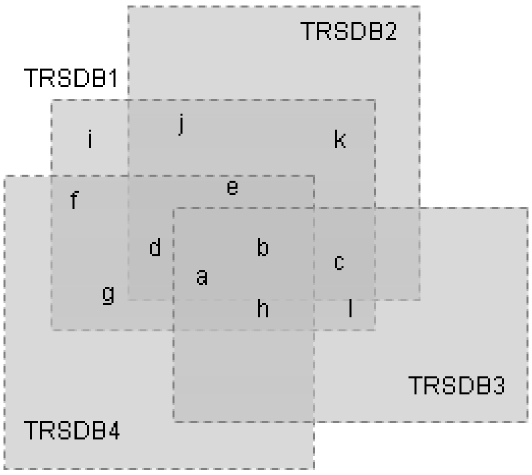


Figure 3.12: Destribution of representative sequential patterns in $TRSDB_1$

propagator	*cost*
a	5
b	10
c	5
d	10
e	10
k	10
j	5
$Total\_cost$	55

$TRSDB_2$
(a)

propagator	*cost*
a	10
b	5
c	10
h	20
i	10
l	15
$Total\_cost$	70

$TRSDB_3$
(b)

propagator	*cost*
a	15
b	5
d	15
e	20
h	15
f	5
g	10
$Total\_cost$	85

$TRSDB_4$
(c)

Table 3.7: SSM cost tables of $TRSDB_3$, $TRSDB_3$, and $TRSDB_4$

propagator	*cost*
a	10
b	5
c	10
$Total\_cost$	25

$TRSDB_3$
(a)

propagator	*cost*
a	15
b	5
d	15
e	20
$Total\_cost$	55

$TRSDB_4$
(b)

Table 3.8: Updated SSM cost tables of $TRSDB_3$ and $TRSDB_4$

```
        $TRSDB_i \neq TRSDB_{start}$. Select the domain has minimum cost as
        the first propagated domain denoted as $TRSDB_{1st\_propagated}$.
    3. call $PrefixSpan(TRSDB_{start})$.
    4. Construct propagated database $TRSDB_{1st\_propagated}||_\alpha$ and call $SSM(\alpha, TRSDB_{1st\_propagated}||_\alpha)$
        for each mined sequential pattern $\alpha$ in step 3 if $\alpha$ cannot be pruned.
    5. For every MDSSP $p$ returned form step 4, call $Propagation(p)$.
END

Subroutine: $Propagation(propagator)$
/* $propagator$ is the propagator pattern.*/
BEGIN
    1. Update SSM cost table for each remaining domain, and select the domain with
        minimum SSM cost as the next propagated domain denoted as $TRSDB_{k\_th\_propagated}$.
    2. IF $propagator$ cannot be pruned BEGIN
            call $SSM(propagator, TRSDB_{k\_th\_target}||_{propagator})$
        END
    3. IF exist no remaining domain BEGIN
            OUTPUT every MDSSP $p$ returned from step 2 is a MDSSP of
            $TRSDB_1, TRSDB_2, ...,$and $TRSDB_n$
        END
        ELSE BEGIN
            call $Propagation(p)$ for every MDSSP $p$ returned from step 2.
        END
END
```

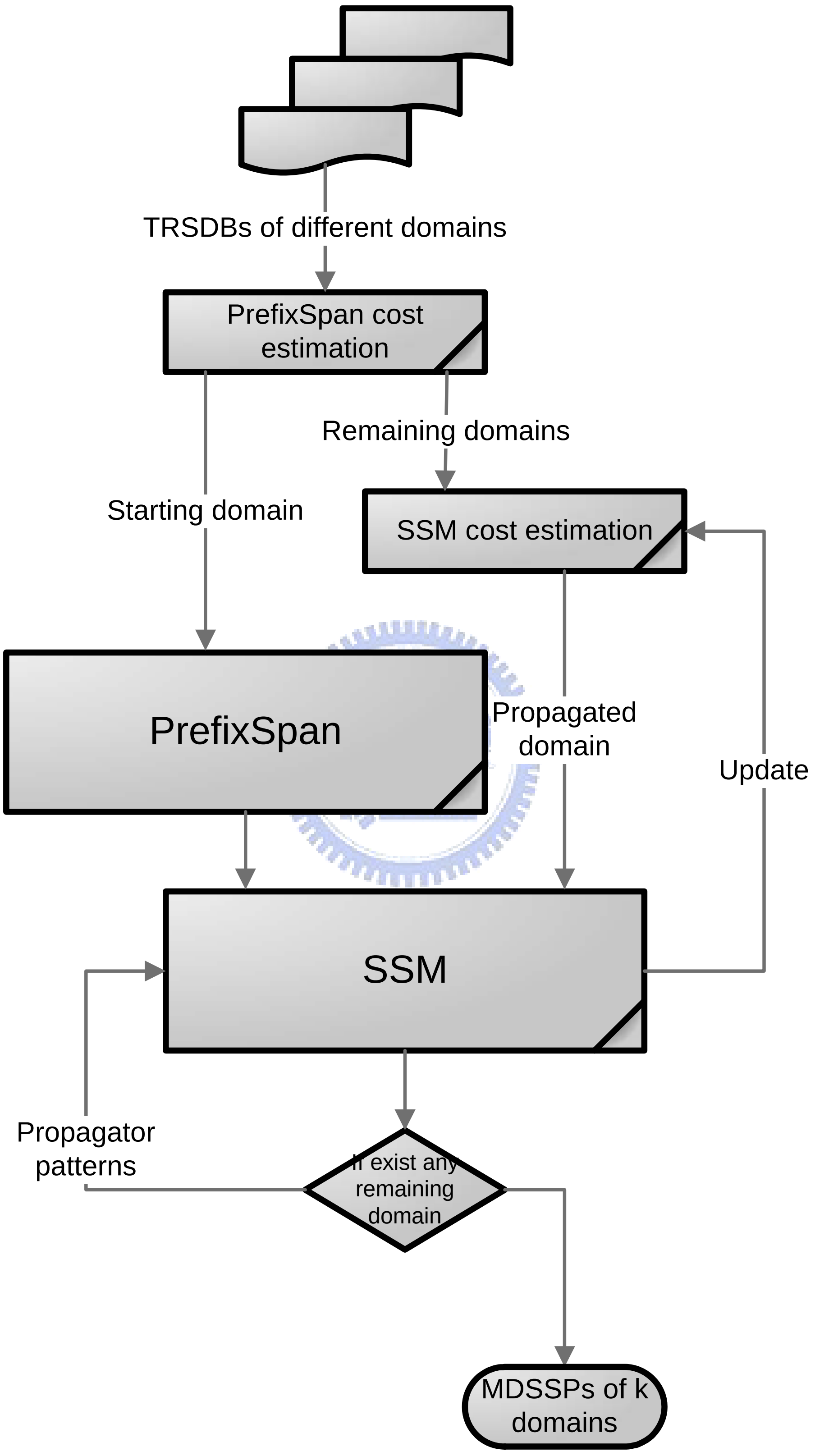


Figure 3.13: Architecture of PropagatedMine with optimization method

Chapter 4

Performance Study

To evaluate the effectiveness and efficiency of algorithm PropagatedMine, we performed an extensive performance study on synthetic data sets with various kinds of sizes and data distributions. The simulation model is described in Chapter 4.1. Chapter 4.2 is devoted to experimental results and comparison of three algorithms: IndividualMine, PropagatedMine and Optimized PropagatedMine.

4.1 Simulation Model

Our experiments were run on a 1.8GHz Athlon PC with 1 gigabyte main memory, and all methods are implemented using J2SE (version 1.4.2). The synthetic datasets used for our experiments were generated by modifying the source code of "Synthetic Data Generation Code for Associations and Sequential Patterns" which follows the standard procedure described in [2]. This data generator has been used in most studies on sequential pattern mining, like [3][4][6][8][9][11].

Due to the restriction of space, most experimental results we reported are using dataset C10T8S8I8. In this dataset, the number of items is set to 1,000, there are 10,000 sequences in the data set, the average number of items within elements is set to 8, the average number

Notation	Description	value
min_sup	minimum support threshold	various value used
$\|D\|$	Number of domains	various value used
$\|C\|$	Number of sequences in thousands	various value used
$corr$	Correlation of given domains	various value used

Table 4.1: The papameters and measurements used in the simulation

of elements in a sequence is set to 8, and the average length of sequential patterns is set to 8.

Performance of three algorithms: IndividualMine, PropagatedMine, and optimized PropagatedMine is comparatively analyzed. Sensitivity analysis on several parameters, including the number of domains, the number of sequences in TRSDBs, the support threshold, and the correlation of given domains is conducted. Table 4.1 shows the notations of some primary parameters in our model. Assume that there are domains: $TRSDB_1, TRSDB_2, ..., TRSDB_k$. Then, the correlation $corr$ of these k domains equals to:

$$corr = \frac{\sum_{i,j} Corr(TRSDB_i, TRSDB_j)}{C_2^k}, \text{ where } 1 \leq i, j \leq k \text{ and } i \neq j$$

Furthermore, suppose that $TRSDB_i$ has m items and $TRSDB_j$ has n items. $Corr(TRSDB_i, TRSDB_j)$ is formulated as $\frac{\sum_{r,s} Corr(x_r,\, y_s)}{m \times n}$, where for each r, $1 \leq r \leq m$, x_r is an item of domain $TRSDB_i$, and for each s, $1 \leq s \leq n$, y_s is an item of domain $TRSDB_j$. $Corr(x_r, y_s)$ is defined as $\frac{|x_r \cap y_s|}{|x_r| \times |y_s|}$, where $|x_r|$ and $|y_s|$ means the number of slots having item x_r and y_s in domain $TRSDB_i$ and $TRSDB_j$, respectively, and $|x_r \cap y_s|$ represents the number of slots having x_r and y_s.

4.2 Experimental Results

The experimental results of scalability with various support thresholds are shown in Figure 4.1. The number of domains $|D|$ is set to 5 and the dataset of each domain is C10T8S8I8, and the value of $corr$ for these five domains is 1.523. It can be seen in Figure 4.1 that when the support threshold is high there are only a limited number of sequential patterns, and the

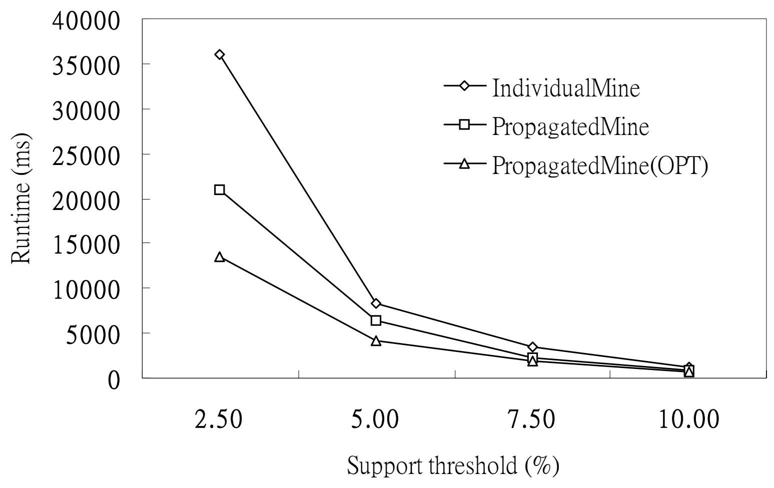


Figure 4.1: Scalability over support threshold

runtime of three method is close. As the support threshold decreases, the gaps become larger. Both PropagatedMine and optimized PropagatedMine are more efficient than IndividualMine, due to that the cost of performing PrefixSpan increases rapidly when the support threshold becomes low and IndividualMine needs to apply PrefixSpan on each domain.

The scalability of these algorithms over the number of domains is next investigated. The dataset of each domain is C10T8S8I8, min_sup is set to 2.5%, and the value of $corr$ is 1.532. the performance is shown in Figure 4.2, two kinds of PropagatedMine show good scalability. From Figure 4.2 both PropagatedMine and optimized PropagatedMine outperform IndividualMine since the number of patterns and the size of propagated database usually decrease. However, IndividualMine needs to mine the whole size of each time related sequence database. Figure 4.2 shows the good scalability of PropagatedMine and optimized PropagatedMine.

The impact of increasing the number of sequences is shown in Figure 4.3. The number of domains $|D|$ is set to 5, min_sup equals to 2.5% and correlation parameter $corr$ of domains is 1.532. Datasets used in this experiment are C1T8S8I8, C4T8S8I8, C7T8S8I8, and C10T8S8I8. As can be seen in Figure 4.3, optimized PropagatedMine has the best scalability among three algorithms, showing the advantage of good propagated orders generated.

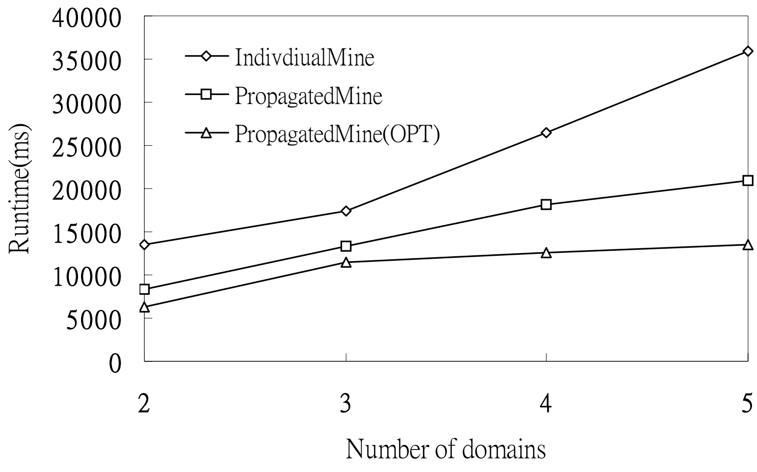


Figure 4.2: Scalability over number of domains

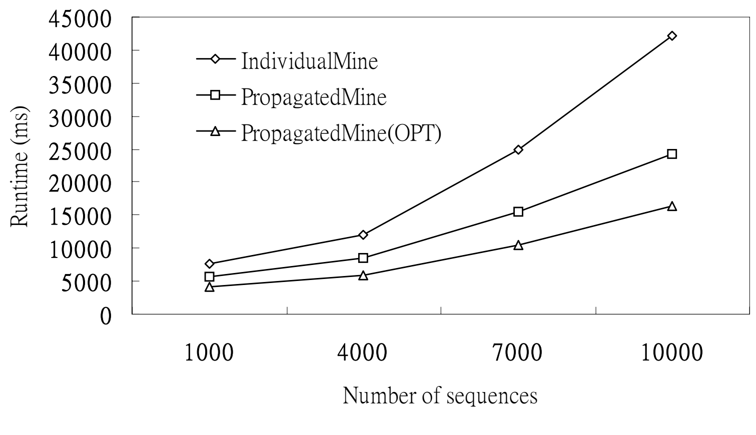


Figure 4.3: Scalability over number of sequences

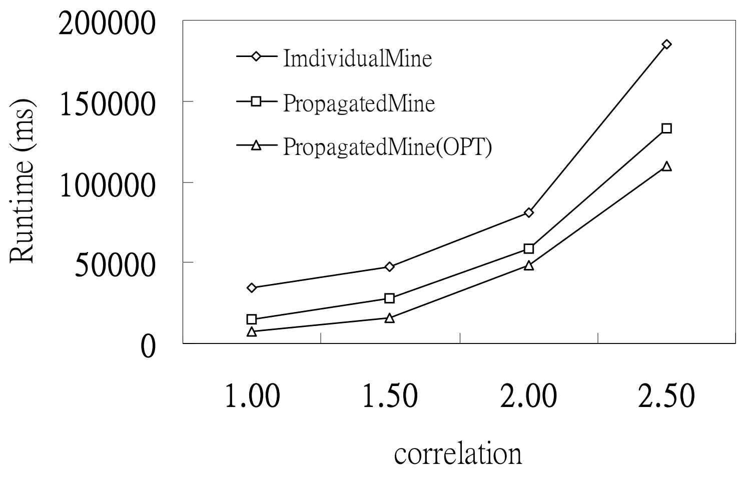


Figure 4.4: Scalability over correlation

Figure 4.4 shows the scalability of methods over correlation. We set $|D| = 5$, $min\_sup = 2.5\%$, and datasets in each domain is C10T8S8I8. The correlation is varying form 1.00 to 2.5. In our experiments, runtime of mining multi-domain sequential patterns is sensitive to the correlation values. As the value of the correlation increase, the run times of these algorithms increase. This is due to that more simultaneous sequential patterns can be generated.

By exploring the feature of propagating, both algorithms PropagatedMine and optimized PropagatedMine outperform IndividualMine. Now, we shall thus focus on comparing the performance of PropagatedMine with optimized PropagatedMine. As shown in Figure 4.5, the process time in each domain of optimized PropagatedMine decreases more rapidly than PropagatedMine since optimized PropagatedMine will choose the propagated domain with the minimal cost for propagation.

In order to investigate how good optimized PropagatedMine can achieve, we compare the runtime of optimized PropagatedMine with the optimal runtime. Note that the optimal runtime is obtained by performing PropagatedMine with all kinds of propagation order and choose the minimum. We set $|D| = 5$, $min\_sup = 2.5\%$, the correlation parameter $corr$=1.532 and the numbers of sequences are 1000, 4000, 7000, and 10000. Figure 4.6 shows that the

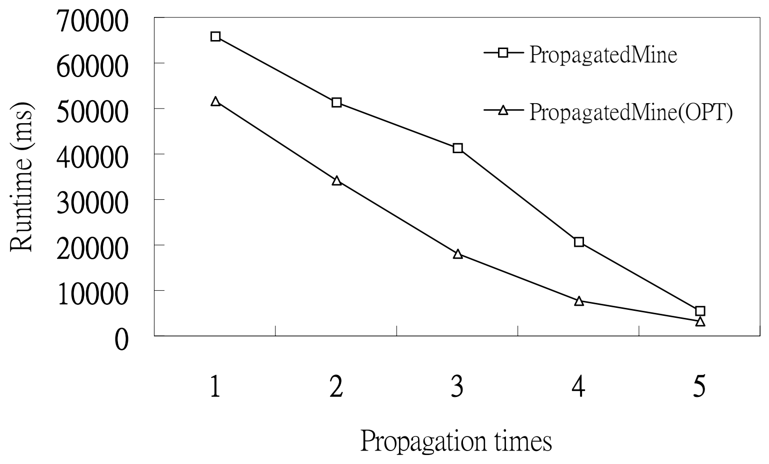


Figure 4.5: Runtime decreasing trend as the process of two kninds of PropagatedMine

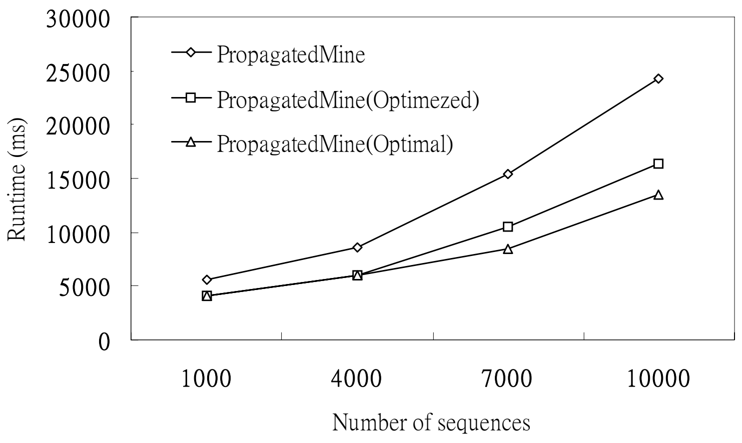


Figure 4.6: Difference of runtimes between optimized PropagatedMine and optimal runtime

runtime of optimized PropagatedMine is very close to the optimal one under different numbers of sequences.

Chapter 5

Conclusions

Multi-domain simultaneous sequential patterns, which are composed of simultaneous sequential patterns in each individual, are of interesting and useful in practice since these patterns clearly reflect the relations of domains hidden in user behavior. In this paper, we proposed a propagation-based approach, algorithm PropagatedMine for efficient mining of multi-domain sequential patterns. By propagating patterns with their occurrences of time from one starting domain to other domains, our proposed approach is able to significantly reduce the mining space, which improves the performance of mining multi-domain sequential patterns. Note that the cost of performing PropagatedMine is greatly affected by the propagation order. Thus, in this paper, we further developed a novel method to determine the optimized propagation order. Performance of the proposed algorithms is comparatively analyzed. Sensitivity analysis on several parameters, including the number of domains, the sizes of sequence databases, and the values of correlations among domains, was conducted. It was shown in our simulation results that by exploiting the feature of propagating, algorithm PropagatedMine is able to efficiently mine multi-domain sequential patterns. Moreover, algorithm PropagatedMine with an optimized propagation order is able to further improve the performance in mining multi-domain sequential patterns and the performance of the optimized propagation order determined by our proposed method is very close to that of the optimal one resulted by selecting the minimal cost among all possible propagation orders.

Bibliography

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of IEEE eleventh International Conference on Data Engineering*, pages 3–14, Taipei, Taiwan, 1995. IEEE Computer Society Press.

[3] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. Sequential pattern mining using a bitmap representation. In *Proceedings of ACM SIGKDD*, pages 429–435, 2002.

[4] A. L. P. C. Ding-Ying Chiu, Yi-Hung Wu. An efficient algorithm for mining frequent sequences by a new strategy without support counting. In *Proceedings of the 20th International Conference on Data Engineering*, pages 375–386, Boston, MA, USA, 2004.

[5] M. N. Garofalakis, R. Rastogi, and K. Shim. SPIRIT: Sequential pattern mining with regular expression constraints. In *The VLDB Journal*, pages 223–234, 1999.

[6] J. Han, J. Pei, and B. Mortazavi-Asl. Freespan: Frequent pattern-projected sequential pattern mining. In *Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining*, pages 20–23, 2000.

[7] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 1–12. ACM Press, 05 2000.

[8] J. H. Hong Cheng, Xifeng Yan. Incspan: incremental mining of sequential patterns in large database. In *Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining*, Seattle, WA, USA, 2004.

[9] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. PrefixSpan mining sequential patterns efficiently by prefix projected pattern growth. In *Proceedings of 2001 International Conference on Data Engineering*, pages 215–226.

[10] J. Pei, J. Han, B. Mortazavi-asl, and H. Zhu. Mining access patterns efficiently from web logs. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 396–407, 2000.

[11] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of 5th International Conference Extending Database Technology, EDBT*, volume 1057, pages 3–17. Springer-Verlag, 25–29 1996.

[12] K. Wang. Discovering patterns from large and dynamic sequential data. *Journal of Intelligent Information Systems*, 9(1):33–56, 1997.

[13] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *Proceedings of International Conference on Data Engineering*, 2002.

[14] M. J. Zaki. Scalable algorithms for association mining. *Knowledge and Data Engineering*, 12(2):372–390, 2000.

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