

國立交通大學

資訊科學與工程研究所

博士論文

時序資料庫中高效率頻繁樣式探勘演算法之研究

Efficient Algorithms for Mining Frequent Patterns in Temporal
Databases

研究生：朱俊榮

指導教授：梁婷 教授

曾新穆 教授

中華民國九十七年一月

時序資料庫中高效率頻繁樣式探勘演算法之研究
Efficient Algorithms for Mining Frequent Patterns in Temporal
Databases

研究生：朱俊榮

Student : Chun-Jung Chu

指導教授：梁 婷 博士

Advisor : Dr. Tyne Liang

曾新穆 博士

Dr. Vincent Shin-Mu Tseng

國立交通大學

資訊科學與工程研究所



A Dissertation Submitted to

Department of Computer Science

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Computer Science

January 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年一月

時序資料庫中高效率頻繁樣式探勘演算法之研究

學生：朱俊榮

指導教授：梁 婷 博士
曾新穆 博士

國立交通大學 資訊科學與工程研究所

摘 要

在資料探勘的領域中，從大型資料庫中尋找資料項目間的頻繁樣式是相當重要之議題。因為資料項目間的頻繁樣式在現實生活中可應用在許多領域裡，譬如超級市場裡商品之間的販售關係。然而頻繁樣式所涵蓋的項目廣泛，其中包含了高頻率項目集、新興高頻率項目集以及高利益項目集等。近年來，由於經濟的關係，市場獲利的需求增加，新興高頻率項目集與高利益項目集成為主要探討的兩個議題。所謂的新興高頻率項目集是指項目集在舊有的資料庫中是低頻率項目集，然而在資料庫新增資料後，這些項目集變成高頻率項目集。至於高利益項目集的探勘，則針對項目集所能得到利益的高低來探討。過去的研究多著重於如何在大型資料庫中快速且正確地尋找這些項目集，減少候選項目集的產生、搜尋資料庫的次數以及記憶體的使用等。然而，傳統的演算法無法直接找尋在時序性資料庫中的頻繁樣式。因此，本論文主旨在研發從時序性資料庫中挖掘出新興高頻率的項目集以及高利益的項目集的高效率探勘方法。

在新興高頻率項目集的探勘上，我們提出了一個有效率的演算法 EFI (Emerging Frequent Itemsets)-Mine。此演算法利用移動視窗交集法來預測高頻率項目集，因此不需要像以往的演算法須找出所有的高頻率項目集，以至減少高頻率項目集搜尋所需花費的時間。實驗的結果，我們所提出的演算法較 Apriori 演算法在以 IBM 資料產生器中所產

生的資料情形下，減少 90%的執行時間。

此外，在本論文我們也提出一個高利益項目集的演算法 THUI (Temporal High Utility Itemsets)-Mine。此演算法利用分割移動視窗法與交易權重項目集來減少候選項目集的產生，因此不需要像以往的演算法須找出所有的候選項目集，以至減少搜尋高利益項目集與資料庫次數所需花費的時間。在以 IBM 資料產生器中所產生的資料下，我們所提出的演算法較 Two-Phase 演算法，執行時間的改善率平均高達 67%。

從利益探勘的特性中，我們可以觀察到資料庫存有許多未滿足門檻值的利益項目集，有些是值得參考的。我們提出了兩個珍貴的利益項目集之演算法 TP-RUI (Two-Phase Rare Utility Itemsets) -Mine 和 TRUI (Temporal Rare Utility Itemsets) -Mine。尤其是 TRUI-Mine，此演算法利用相對性的門檻值來尋找珍貴的利益項目集，並利用分割移動視窗法與交易權重項目集來減少候選項目集的產生，以至減少尋找珍貴的利益項目集所需花費的時間。

由於過去所探討的高利益項目集都是針對正利益的項目來研究，於是我們提出一個從大型的資料庫中尋找包含負利益的高利益項目集的演算法 HUIINV (High Utility Itemsets with Negative Item Values) -Mine。此演算法利用去除負利益項目的交易權重項目集方法來減少候選項目集產生。實驗的結果，我們所提出的演算法較 MEU 演算法在以 IBM 資料產生器中所產生的資料情形下，減少 99%的候選項目集。

關鍵字： 頻繁樣式；關聯法則；時序性資料庫；利益探勘；

Efficient Algorithms for Mining Frequent Patterns in Temporal Databases

Student: Chun-Jung Chu

Advisors: Dr. Tyng Liang
Dr. Vincent Shin-Mu Tseng

Department of Computer Science
National Chiao Tung University

ABSTRACT

Mining frequent patterns for discovering the relationship among data items from large databases is an important topic in data mining since frequent patterns can be applied to wide applications. There exist various kinds of frequent patterns like frequent itemsets, emerging frequent itemsets, high utility itemsets and so on. Recently, emerging pattern mining and utility mining become very popular topics because of the rising of economic applications. Emerging frequent itemsets are those who are infrequent in the current database and then become frequent in the new database temporally added with new data transactions. High utility itemsets reveal the utility of an itemset, which can be measured in terms of cost, profit or other expressions of user preferences. In the past, most studies focus on developing efficient and effective methods for finding frequent itemsets from large database by reducing candidate itemsets, database scans and memory space. However, most methods designed for the traditional databases cannot be directly applied for mining temporal patterns in temporal databases because of the high complexity. Hence, we investigate efficient methods for mining emerging frequent itemsets and high utility itemsets in temporal databases in this dissertation.

In emerging pattern mining, a novel approach named *EFI (Emerging Frequent Itemsets)-Mine* is presented to effectively identify the potential emerging itemsets by crossing sliding windows to predict frequent itemsets such that the execution time can be reduced

substantially in mining all frequent itemsets in temporal databases. The experimental results show that *EFI* delivers 90.6% of improvement over the well-known method *Apriori* in terms of execution time on various kinds of synthetic datasets.

Besides, we also propose a novel method, namely *THUI (Temporal High Utility Itemsets)-Mine*, for mining temporal high utility itemsets from temporal databases efficiently and effectively. *THUI-Mine* can effectively identify the temporal high utility itemsets by partitioning sliding window and using transaction-weighted utilization itemsets to generate fewer candidate itemsets such that the execution time and number of database scans can be reduced substantially in mining high utility itemsets from temporal databases. The experimental results show that *THUI-Mine* delivers 67% of improvement over the well-known method *Two-Phase* in terms of execution time on various kinds of synthetic datasets.

According to the characters of utility mining, we could observe some utility itemsets are those itemsets which appear infrequently in the current time window of large databases but are highly associated with specific data. Hence, we proposed two novel algorithms, namely *TP-RUI (Two-Phase Rare Utility Itemsets) -Mine* and *TRUI (Temporal Rare Utility Itemsets) -Mine*, which can effectively identify the temporal rare utility itemsets by using relative utility threshold. The temporal rare utility itemsets are discovered by partitioning sliding window and using transaction-weighted utilization itemsets to generate fewer candidate itemsets such that the execution time and database scan can be reduced substantially in mining all high and rare utility itemsets in temporal databases.

The past studies on high utility itemsets mining considered only positive item values and ignored the cases of negative item values that may occur in real-life applications. Therefore, we propose a novel method, namely *HUINIV (High Utility Itemsets with Negative Item Values) -Mine*, for mining high utility itemsets from large databases with consideration of negative item values. *HUINIV-Mine* can effectively identify high utility itemsets by using transaction

utility without negative item values to generate fewer candidate itemsets. The experimental results show that *HUINIV-Mine* delivers 99% of improvement over the well-known method *MEU* in terms of candidate itemsets on various kinds of synthetic datasets.

Keywords: *Frequent patterns mining, Association rules, Temporal databases, Utility mining*



ACKNOWLEDGEMENT

(誌 謝)

首先，我最感謝的是我的指導教授梁婷老師以及共同指導教授曾新穆老師，感謝他們在這四年的研究所生涯中啟發我對學術研究的興趣，並且指導我對論文寫作上的技巧，以及生活的幫助。在跟隨著梁老師以及曾老師研究的過程中，梁老師以及曾老師對研究的執著以及熱忱，一直是我學習的榜樣。此外，還要感謝李素瑛教授和彭文志教授在計畫書口試、校內口試以及校外口試時提供許多寶貴的意見。

感謝口試委員：東華資管系楊維邦教授、中央資工系洪炯宗教授、逢甲資工系楊東麟教授、清大資工系陳宜欣教授在口試過程中提供許多寶貴的建議，讓我的論文能更趨完善。諸位口試委員都是我在學術研究的道路上的最佳學習典範。資訊擷取實驗室的同學和學弟妹是我博士班研究生涯的好伙伴，謝謝大家也祝福學弟妹們早日收穫豐富的研究成果，再次謝謝你們在這段過程中對我的幫忙以及鼓勵。

最後要感謝的是一直支持我的家人，有你們的支持與鼓勵，讓我在求學的過程中能無憂無慮且專心的在學術研究上，謝謝你們。

要感謝的人太多，僅在此對所有曾經幫助過我的朋友們，致上我真切的謝意。

僅以此論文，獻給我最親愛的家人。

TABLE OF CONTENTS

ABSTRACT	iii
ACKNOWLEDGEMENT	vi
TABLE OF CONTENTS	vii
LIST OF FIGURES	viii
LIST OF TABLES	ix
Chapter 1 Introduction	1
1.1 Motivation and Research Goal	2
1.2 Related Work	3
1.3 Organization of Thesis.....	6
Chapter 2 Mining Temporal Emerging Itemsets from Temporal Databases.....	8
2.1 Problem Definition	8
2.2 Mining Temporal Emerging Itemsets.....	13
2.3 Experiments and Analysis	20
Chapter 3 Mining Temporal High Utility Itemsets from Temporal Databases.....	26
3.1 Problem Definition	26
3.2 Proposed Method.....	30
3.3 Experiments and Analysis	42
Chapter 4 Mining Temporal Rare Utility Itemsets in Large Databases Using Relative Utility Thresholds	51
4.1 Problem Definition	51
4.2 Proposed Methods	55
4.3 Experiments and Analysis	69
Chapter 5 Mining High Utility Itemsets with Negative Item Values.....	77
5.1 Problem Definition	77
5.2 Proposed Method.....	81
5.3 Experiments and Analysis	88
Chapter 6 Conclusions and Future Work.....	98
Bibliography.....	102

LIST OF FIGURES

Figure 2-1. An example of online transaction flows.	9
Figure 2-2. Potentially emerging frequent itemsets in DB ₂₃₄₅	13
Figure 2-3. Emerging frequent itemsets.	15
Figure 2-4. Potentially emerging frequent itemsets.	16
Figure 2-5. Potentially emerging frequent itemsets for temporal patterns.	18
Figure 2-6. Algorithm of EFI-Mine.	19
Figure 2-7. Accuracy under different support values (N100T5C1000, W=10).	22
Figure 2-8. Execution time with w=10.	23
Figure 2-9. Accuracy under different window sizes.	24
Figure 2-10. Accuracy under different numbers of items per transaction with w=10.	24
Figure 2-11. Accuracy under different numbers of items.	25
Figure 3-1. An example of online transaction flows.	27
Figure 3-2. Temporal high utility itemsets generated by THUI-Mine.	35
Figure 3-3. Preprocessing procedure of THUI-Mine.	40
Figure 3-4. Incremental procedure of THUI-Mine.	42
Figure 3-5. Utility value distribution in utility table.	44
Figure 3-6. Execution time for Two-Phase and THUI on T20.I6.D100K.d10K.	47
Figure 3-7. Execution time for Two-Phase and THUI on T10.I6.D100K.d10K.	48
Figure 3-8. Scale-up performance results for THUI vs. Two-Phase.	49
Figure 3-9. Execution time for Two-Phase and THUI on gazelle dataset.	50
Figure 4-1. Temporal rare utility itemsets generated by TRUI-Mine.	61
Figure 4-2. Preprocessing procedure of TRUI-Mine.	67
Figure 4-3. Incremental procedure of TRUI-Mine.	69
Figure 4-4. Utility value distribution in utility table.	71
Figure 4-5. Execution time on T10.I6.D100K.d10K.	74
Figure 4-6. Execution time on T20.I6.D100K.d10K.	74
Figure 4-7. Execution time on T10.I4.D100K.d10K.	75
Figure 4-8. Scale-up performance results.	75
Figure 5-1. High utility itemsets generated from large databases by HUINIV-Mine.	86
Figure 5-2. Procedure used by HUINIV-Mine.	88
Figure 5-3. Utility value with negative distribution in utility table.	90
Figure 5-4. Execution time for HUINIV on T20.I6.D100K and T10.I6.D100K.	94
Figure 5-5. Execution time for HUINIV on T20.I4.D100K and T10.I4.D100K.	95
Figure 5-6. Execution time for HUINIV on BMS-POS.	97

LIST OF TABLES

Table 2-1. The support values of the inter-transaction itemset $\{c, g\}$	11
Table 2-2. Parameters of the synthetic datasets.....	21
Table 2-3. Parameter settings of synthetic datasets.....	21
Table 3-1. A transaction database and its utility table.....	28
Table 3-2. Transaction utility of the transaction database.....	34
Table 3-3. Meanings of symbols used.....	38
Table 3-4. The number of candidate itemsets generated on database T10.I6.D100K.d10K....	45
Table 3-5. The number of candidate itemsets generated on database T20.I6.D100K.d10K....	46
Table 4-1. A transaction database and its utility table.....	53
Table 4-2. Transaction utility of the database.....	57
Table 4-4. Number of candidate itemsets, temporal high utility itemsets and temporal rare utility itemsets generated on dataset T10.I6.D100K.d10K.....	72
Table 5-1. A transaction database and its utility table.....	78
Table 5-2. Transaction utility of the transaction database.....	84
Table 5-3. Transaction utility without negative item values of the transaction database.....	85
Table 5-4. Meanings of symbols used.....	87
Table 5-5. The number of candidate itemsets and high utility itemsets generated from database T10.I4.D100K.....	92
Table 5-6. The number of candidate itemsets and high utility itemsets generated from database T10.I6.D100K.....	92
Table 5-7. The number of candidate itemsets and high utility itemsets generated from database T20.I4.D100K.....	93
Table 5-8. The number of candidate itemsets and high utility itemsets generated from database T20.I6.D100K.....	93
Table 5-9. Database BMS-POS characteristics.....	96
Table 5-10. The number of candidate itemsets and high utility itemsets generated on database BMS-POS.....	97

Chapter 1

Introduction

The mining of association rules for finding the relationship between data items in large databases is a well studied technique in data mining field with representative methods like *Apriori* [1][2][7]. An important research issue extended from the association rules mining is the discovery of temporal association patterns in temporal databases due to the wide applications on various domains. Temporal data mining can be defined as the activity of looking for interesting correlations or patterns in large sets of temporal data accumulated for other purposes [6]. For a database with a specified transaction window size, we may use the algorithm like *Apriori* to obtain frequent itemsets from the database.

For time-variant temporal databases, there is a strong demand to develop an efficient and effective method to mine various temporal patterns [13]. However, most methods designed for the traditional databases cannot be directly applied for mining temporal patterns in temporal databases because of the high complexity.

In many applications, the frequency of an itemset may not be a sufficient indicator of interestingness, because it only reflects the number of transactions in the database that contain the itemset. It does not reveal the *utility* of an itemset, which can be measured in terms of cost, profit or other expressions of user preferences. On the other hand, frequent itemsets may only contribute a small portion of the overall profit, whereas non-frequent itemsets may contribute a large portion of the profit. In reality, a retail business may be interested in identifying its most valuable customers (customers who contribute a major fraction of the profits to the company). Hence, frequency is not sufficient to answer questions such as whether an itemset is highly profitable, or whether an itemset has a strong impact. Utility mining is thus useful in a wide range of practical applications and was recently studied in [8][21][35].

1.1 Motivation and Research Goal

The research objective of this dissertation is to investigate algorithms for mining emerging frequent itemsets and high utility itemsets efficiently and effectively.

The first research issue of this dissertation is mining temporal emerging frequent itemsets from temporal databases efficiently and effectively, we propose a new method, namely *EFI (Emerging Frequent Itemsets)-Mine*. The temporal emerging frequent itemsets are those that are infrequent in current time window of database but have high potential to become frequent in the subsequent time windows. Discovery of emerging frequent itemsets is an important process for mining interesting patterns like association rules from temporal databases. The novel contribution of *EFI-Mine* is that it can effectively identify the potential emerging itemsets such that the execution time can be reduced substantially in mining all frequent itemsets in temporal databases. This meets the critical requirements of time and space efficiency for mining temporal databases.

Furthermore, we propose a novel method, namely *THUI (Temporal High Utility Itemsets)-Mine*, for mining temporal high utility itemsets from temporal databases efficiently and effectively. The novel contribution of *THUI-Mine* is that it can effectively identify the temporal high utility itemsets by generating fewer candidate itemsets such that the execution time can be reduced substantially in mining all high utility itemsets in temporal databases. In this way, the process of discovering all temporal high utility itemsets under all time windows of temporal databases can be achieved effectively with less memory space and execution time. This meets the critical requirements on time and space efficiency for mining temporal databases.

According to the characters of utility mining, we could observe some utility itemsets are those itemsets which appear infrequently in the current time window of large databases but are highly associated with specific data. Hence, we proposed two novel algorithms, namely

TP-RUI (Two-Phase Rare Utility Itemsets) -Mine and *TRUI (Temporal Rare Utility Itemsets) -Mine*, which can effectively identify the temporal rare utility itemsets by using relative utility threshold. The novel contribution of *TRUI-Mine* is particularly that it can effectively identify the temporal rare utility itemsets by generating fewer temporal high transaction-weighted utilization 2-itemsets in temporal databases. In this way, the process under all time windows of temporal databases can be achieved effectively with limited memory space, less candidate itemsets and CPU I/O time.

The final issue explored in this thesis is based on the observation that the past studies on high utility itemsets mining considered only positive item values and ignored the cases of negative item values that may occur in real-life applications. Therefore, we propose a novel method, namely *HUINIV (High Utility Itemsets with Negative Item Values) -Mine*, for mining high utility itemsets from large databases with consideration of negative item values. The novel contribution of *HUINIV-Mine* is that it can effectively identify high utility itemsets by generating fewer high transaction-weighted utilization itemsets such that the execution time can be reduced substantially in mining the high utility itemsets. In this way, the process of discovering all high utility itemsets with consideration of negative item values can be accomplished effectively with less requirements on memory space and CPU I/O. This meets the critical requirements of temporal and spatial efficiency for mining high utility itemsets with negative item values.

1.2 Related Work

In association rules mining, Apriori [1], DHP [24], and partition-based ones [20][27] are proposed to find frequent itemsets. Many important applications have called for the need of incremental mining. This is due to the increasing use of the record-based databases whose data are being continuously added. Many algorithms like FUP [24], FUP₂ [11] and UWEP

[4][5] are proposed to solve incremental database for finding frequent itemsets. The FUP algorithm updates the association rules in a database when new transactions are added to the database. Algorithm FUP is based on the framework of Apriori and is designed to discover the new frequent itemsets iteratively. The idea is to store the counts of all the frequent itemsets found in a previous mining operation. Using these stored counts and examining the newly added transactions, the overall count of these candidate itemsets are then obtained by scanning the original database. An extension to the work in [10] was reported in [11] where the authors propose an algorithm FUP_2 for updating the existing association rules when transactions are added to and deleted from the database. UWEP (Update With Early Pruning) is an efficient incremental algorithm, that counts the original database at most once, and the increment exactly once. In addition the number of candidates generated and counted is minimum.

In recent years, processing data from data streams is a very popular topic in data mining. A number of algorithms like Lossy Counting [22], FTP-DS [30] and RAM-DS [31] have been proposed to process data in data streams. Lossy Counting divided incoming stream conceptually into buckets. It uses bucket boundaries and maximal possible error to update or delete the itemsets with frequency for mining frequent itemsets. FTP-DS is a regression-based algorithm to mine frequent temporal patterns for data streams. A wavelet-based algorithm, called algorithm RAM-DS, to perform pattern mining tasks for data streams by exploring both temporal and support count granularities.

Some algorithms like SWF [18] and Moment [12] are proposed to find frequent itemsets over a stream sliding window. By partitioning a transaction database into several partitions, algorithm SWF employs a filtering threshold in each partition to deal with the candidate itemset generation. Moment algorithm use the closed enumeration tree (CET), to maintain a dynamically selected set of itemsets over a sliding window.

Dong and Li define an emerging pattern as an itemset the support of which increases

significantly between two databases. We view emerging frequent itemsets as a special case of the emerging patterns described by Dong and Li. Recently, a new algorithm modifies an existing incremental algorithm, UWEP, so that it can identify emergent large itemsets. It uses incremental scheme for finding emerging frequent itemsets [28].

A formal definition of utility mining and theoretical model was proposed in [35], namely MEU, where the utility is defined as the combination of utility information in each transaction and additional resources. Since this model cannot rely on downward closure property of Apriori to restrict the number of itemsets to be examined, a heuristic is used to predict whether an itemset should be added to the candidate set. However, the prediction usually overestimates, especially at the beginning stages, where the number of candidates approaches the number of all the combinations of items. The examination of all the combinations is impractical, either in computation cost or in memory space cost, whenever the number of items is large or the utility threshold is low. Although this algorithm is not efficient or scalable, it is by far the best one to solve this specific problem.

Another algorithm named Two-Phase was proposed in [21], which is based on the definition in [35] and achieves the finding of high utility itemsets. The Two-Phase algorithm is used to prune down the number of candidates and can obtain the complete set of high utility itemsets. In the first phase, a model that applies the “transaction-weighted downward closure property” on the search space is used to expedite the identification of candidates. In the second phase, one extra database scan is performed to identify the high utility itemsets. However, this algorithm must rescan the whole database when new transactions are added from temporal databases. It incurs more cost on I/O and CPU time for finding high utility itemsets. Hence, the Two-Phase algorithm is focused on traditional databases and is not suited for mining temporal databases.

Many algorithms were proposed for mining useful information on different applications. A mining method was proposed to process computer vulnerability [38] for finding

vulnerability correlation. A heuristic reduction algorithm HeuriRed and A complete heuristic reduction algorithm HeuriComRed [29] based on discernibility matrix were proposed to process imprecise and incomplete data for attributes reduction mining.

One algorithm named RSAA (Relative support Apriori Algorithm) [36] is a method to discover the association rules for significantly rare itemsets that appear infrequently in the database but are highly associated with specific data. The technique adopts a new criterion, relative support, which can identify the strong co-relation of the significant rare itemset items with the specific data co-occurring in relatively high proportion. However, the technique of this algorithm can not be adopted in utility mining because it violates definitions of utility mining. Hence, the RSAA algorithm is focused on traditional association rules discovery and databases, and so it is not suited for utility mining and temporal databases.

Although there existed numerous studies on frequent itemsets mining and data stream analysis as described above, there is no algorithm proposed for finding emerging frequent itemsets, temporal high utility itemsets, temporal rare utility itemsets and high utility itemsets with negative item values in temporal and large databases. This motivates our exploration on the issue of efficiently mining various frequent itemsets we describe above in temporal databases like data streams in this research. Therefore, we propose four methods that can identify frequent pattern efficiently and effectively. To our best knowledge, this is the first work on mining t emerging frequent itemsets, temporal high utility itemsets, temporal rare utility itemsets and high utility itemsets with negative item values from temporal and large databases.

1.3 Organization of Thesis

The remainder of this thesis is organized as follows. In Chapter 2, we describe *EFI-Mine* algorithm for mining temporal emerging frequent itemsets from temporal databases efficiently

and effectively. Efficient *THUI-Mine* algorithm for mining temporal high utility itemsets from temporal databases is introduced in Chapter 3. In Chapter 4, we describe two novel algorithms, namely *TP-RUI (Two-Phase Rare Utility Itemsets) -Mine* and *TRUI (Temporal Rare Utility Itemsets) -Mine*, for mining temporal rare utility itemsets from temporal databases. relevance feedback methods are surveyed and a novel feedback mechanism is proposed. In Chapter 5, we address a novel method, namely *HUINIV (High Utility Itemsets with Negative Item Values) -Mine*, for efficiently and effectively mining high utility itemsets from large databases with consideration of negative item values. Last, the conclusions are given in Chapter 6.



Chapter 2

Mining Temporal Emerging Itemsets from Temporal Databases

2.1 Problem Definition

The mining of association rules for finding the relationship between data items in large databases is a well studied technique in data mining field with representative methods like *Apriori* [1][2][7]. The problem of mining association rules can be decomposed into two steps. The first step involves finding all frequent itemsets (or say large itemsets) in databases. Once the frequent itemsets are found, generating association rules is straightforward and can be accomplished in linear time.

An important research issue extended from the association rules mining is the discovery of temporal association patterns in temporal databases due to the wide applications on various domains. Temporal data mining can be defined as the activity of looking for interesting correlations or patterns in large sets of temporal data accumulated for other purposes [6]. For a database with a specified transaction window size, we may use the algorithm like *Apriori* to obtain frequent itemsets from the database. For time-variant temporal databases, there is a strong demand to develop an efficient and effective method to mine various temporal patterns [4]. However, most methods designed for the traditional databases cannot be directly applied for mining temporal patterns in temporal databases because of the high complexity.

Without loss of generality, consider a typical market-basket application as illustrated in [30] has been considered. The transaction flow in such an application is shown in Figure 2-1 where items *a* to *g* stand for items purchased by customers.

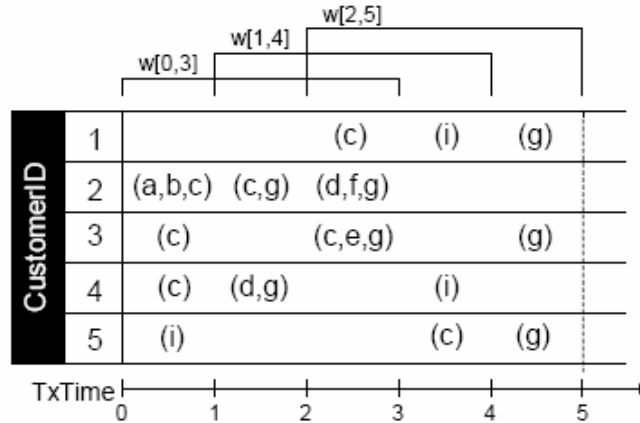


Figure 2-1. An example of online transaction flows.

In Figure 2-1, for example, the third customer bought item c during time $t=[0,1)$, items c, e and g during $t=[2, 3)$, and item g during $t=[4, 5)$. It can be seen that in such a data stream environment it is intrinsically difficult to conduct the frequent pattern identification due to the limited time and space constraints. Furthermore, it wastes too much times finding frequent itemsets in different window times. Therefore, we develop a new scheme to find potential emerging frequent itemsets before next window times.

Dong and Li [14] define an emerging pattern as an itemset the support of which increases significantly between two databases. We view emerging frequent itemsets as a special case of the emerging patterns described by Dong and Li. An *Emerging Frequent Itemset (EFI)* can be considered as an itemset that is infrequent (i.e., small) in the current database and gets increased for its support so that it will eventually become frequent (i.e., large) in the new database temporally added with new data transactions. For example, in the market basket domain, we may assume an interval as the time between wholesale purchases. Recognizing the set of items that will emerge or become frequent in the next time period with windows size may allow the storekeeper to order these emerging items much earlier than usual. Thus, the storekeeper will know what kinds of items will be popular in the next time period and avoid losing the income that their sales could have generated. Although some

related issues like mining emerging frequent itemsets [28] and incremental frequent itemsets [9][10][11][25] have been studied, they have been focused on traditional databases and are not suited for temporal databases.

In this chapter, we explore the issue of efficiently mining emerging frequent itemsets in temporal databases like data streams [15][16][17][19]. We propose an algorithm named *EFI-Mine* that can discover emerging frequent itemsets from temporal databases efficiently and effectively. The *EFI-Mine* algorithm is based on the concept of Apriori algorithm [2] for mining frequent itemsets. The novel contribution of *EFI-Mine* is that it can effectively identify the potential emerging frequent itemsets in temporal databases so that the execution time for mining frequent itemsets can be substantially reduced. That is, *EFI-Mine* can discover the itemsets that are infrequent in current time window but will become frequent ones with high probability in subsequent time windows. In this way, the process of discovering all frequent itemsets under all time windows of temporal databases can be achieved efficiently with limited memory space. This meets the critical requirements of time and space efficiency for mining temporal databases. Through experimental evaluation, *EFI-Mine* is shown to deliver high precision in finding the emerging frequent itemsets and it also achieves high scalability in terms of execution time.

Support Framework for Mining Temporal Patterns

In this chapter, the mining of temporal patterns are explored for illustrative purposes since not only the patterns should be efficiently and effectively extracted but also variations of corresponding occurrence frequencies should be tracked. In market-basket analysis, patterns along with their frequencies are extracted from sliding window in transactions. So the data expires after a user-specified time window. As time advances, new data is included while obsolete data is discarded. With the mining task for discovering frequent temporal patterns,

only patterns with occurrence frequencies no less than a specified threshold are being tracked. We focus in this chapter on handling the different sliding windows to find emerging frequent itemsets.

An example showing the basic process in transforming transactions into numerical time series, for discovering frequent temporal patterns, is provided as follows.

Example 1: Consider the transaction flows shown in Figure 2-1. Given the window size $w=3$ and the minimum support value as 40%, occurrence frequencies of the inter-transaction itemset $\{c, g\}$ from time $t=1$ to $t=5$ can be obtained as shown in Table 2-1.

Table 2-1. The support values of the inter-transaction itemset $\{c, g\}$.

TxTime		Occurrence(s) of $\{c, g\}$	Support
t=1	w[0,1]	none	0
t=2	w[0,2]	CustomerID={2, 4}	$2/5=0.4$
t=3	w[0,3]	CustomerID={2, 3, 4}	$3/5=0.6$
t=4	w[1,4]	CustomerID={2, 3}	$2/5=0.4$
t=5	w[2,5]	CustomerID={1, 3, 5}	$3/5=0.6$

With the sliding window model, the frequent temporal patterns can be discovered for different time windows. The main goal of our research is to discover interesting emerging itemsets under progressive time windows.

Emerging Frequent Itemsets and Interesting Emerging Itemsets

In a database, the frequent itemsets will be changed when new datum are added. As time progress, we can see many interesting patterns with regards to the change in status of individual itemsets. An itemset that was infrequent may become frequent (large), while frequent itemsets may become infrequent (small) and an itemset may remain frequent or infrequent. We define infrequent itemsets that are moving toward being frequent as *emerging*. Conversely, frequent itemsets moving toward infrequent are *submerging*. An infrequent

(frequent) itemset that becomes large, i.e. with support above (below) minimum support value, is said to have emerged (submerged). The problems we address in this chapter are: 1) How can we identify itemsets that are emerging (submerging)? 2) Which of these itemsets have the potential to emerge (submerge) within the next time window? That is, we focus on finding emerging frequent itemsets in this chapter.

According to the emerging itemsets of incremental scheme, we develop this concept on the temporal data mining. Temporal data mining has the limitation on window size for finding emerging itemsets. Therefore, we must change the formula for finding emerging itemsets. For the remainder of this chapter, we give definitions to the formula.

Definition 2.1 db_k is the transactions in $t=k$, i.e., db_1 is the transactions in $t=1$.

Definition 2.2 $DB_{i,i+1,\dots,j}$ is the transactions in $t=i$ to j , i.e., DB_{12345} is the transactions in $t=1$ to 5. We also view DB_{12345} as the accumulation of $db_1+db_2+db_3+db_4+db_5$.

Suppose the original database is $DB_{i,i+1,\dots,j}$ with window size= N and $N=j-i+1$. Due to the limitation of window size, we should discard the old database db_i when adding a database db_{j+1} . The new database should be $DB_{i+1,i+2,\dots,j+1}$. In our scheme, we should find emerging itemsets before a new database is added. So we should focus on the database $DB_{i+1,i+2,\dots,j}$. The old database db_i is useless for finding emerging itemsets. For example, suppose original database is DB_{1234} and we set the limitation of window size as 5. If a database db_5 is added, the new database will be DB_{12345} . Due to the limitation of window size, when adding a database db_6 , we should discard the old database db_1 . Thus, the new database becomes DB_{23456} . In our scheme, we would find potential emerging frequent itemsets before a database is added. So we should focus on the database DB_{2345} finding potential emerging frequent itemsets. And the potential emerging frequent itemsets of the database DB_{2345} can be represented more accurate in the new database DB_{23456} . In practice, with the feature of data stream, we first remove db_1 from DB_{1234} and then add db_5 to form the database DB_{2345} . So we could find potential emerging frequent itemsets from the database DB_{2345} before adding a new

database db_6 to form DB_{23456} , and this conforms the limitation of window size. Figure 2-2 shows we would find potential emerging frequent itemsets from the database DB_{2345} . So the window size should be $N-1$ for finding potential emerging itemsets.

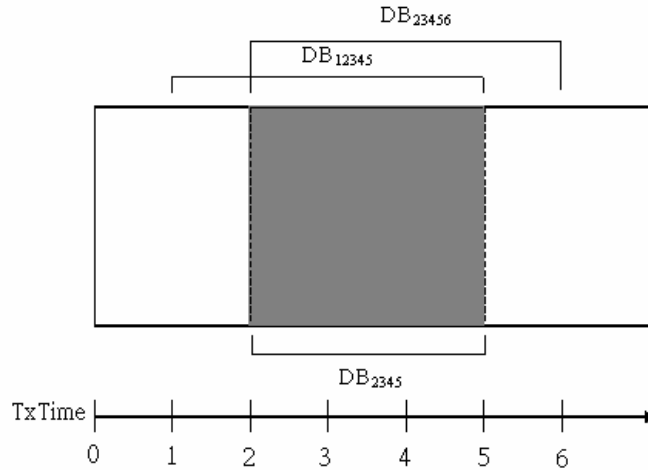


Figure 2-2. Potentially emerging frequent itemsets in DB_{2345} .

The rest of this chapter is organized as follows: Section 2.2 describes the proposed approach, *EFI-Mine*, for finding the emerging frequent itemsets. In section 2.3, we describe the experimental results for evaluating the proposed method. The conclusion of the chapter is provided in Section 2.4.

2.2 Mining Temporal Emerging Itemsets

In this Section, we give an example for mining temporal emerging itemsets from data stream. The proposed algorithm, *EFI-Mine*, is also described in details in this Section.

An example for mining emerging itemsets

Figure 2-3 shows an example of emerging itemsets modified on that proposed by Dong and Li in [14] for the special case of EFI. It shows partitions of the space of itemsets, indicating all

possible transitions for an itemset X from original database DB to the new database DB+db.

Figure 2-3 plots the support count in DB (denoted as SC_{DB}) against the support count in db (denoted as SC_{db}). Each point in the graph depicts an ordered pair (SC_{db}, SC_{DB}) where the sum of SC_{db} and SC_{DB} is an itemset's support count in DB+db at some increment interval. If the increment adds no transactions to an itemset's support count, then its support count in DB has to be equal to $\min SC_{DB} + \min SC_{db}$ in order to achieve $\min SC_{DB+db}$. This corresponds to point H in Figure 2-3. Alternatively, if an itemset's SC is equal to $|db|$ in db, then its support in DB has to be some $SC=n$, where $n>0$, and $n= \min SC_{DB} + \min SC_{db} - |db|$ for the itemset to be frequent. This is point C in Figure 2-3. Line HC partitions the space of all itemsets in DB+db into frequent and infrequent. The shaded area in Figure 2-3 represents all the frequent itemsets and it includes Line HC. Specific partitions under HC contain itemsets that are emerging in the current increment. For example, the area defined by ΔHFG represents those itemsets that were frequent itemsets in DB, infrequent itemsets in db, and now are infrequent in DB+db. These itemsets have therefore submerged. ΔGIC represents itemsets that were infrequent in DB and frequent in db. These itemsets have emerged. Therefore, we can find all itemsets in area ABCG are emerging in the current interval and all itemsets in area OAGH are submerging.

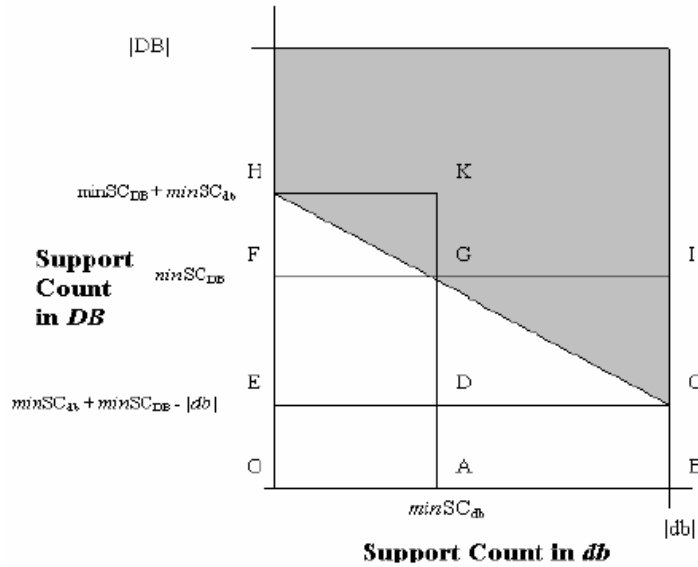


Figure 2-3. Emerging frequent itemsets.

However, there are too many emerging itemsets in area ABCG. In fact, we should focus more potential emerging itemsets. To have the potential to emerge in the next increment, the support count of the itemset in $DB+db$ needs to be greater than or equal to $2minSC_{db}+minSC_{DB} - |db|$ in the current increment. All points with this value are represented by line RS in Figure 2-4.

For example, if we have a database with $|DB|= 10000$, $|db|= 1000$ and $minsup =0.2$, then the minimum support count for the current increment is 2,200 (2,000 from DB plus 200 from db). If an itemset can add the maximum support incremental support count, a total of 1,000 from db, in the next increment, it would need a support count of at least 1400 in the current increment to be able to attain the minimum support count of 2,400 $((11000+1000)*0.2=2400)$ needed to become frequent.

The band of itemsets between line RS and line HC are all itemsets that have the potential to become frequent in the next increment, by this formula. Intersecting area ABCG and HCSR, we get itemsets in GDSC are most likely to emerge in the next increment.

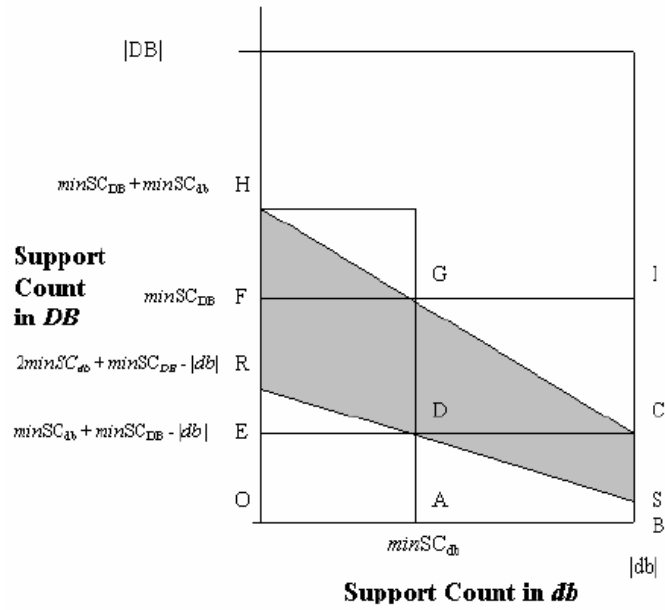
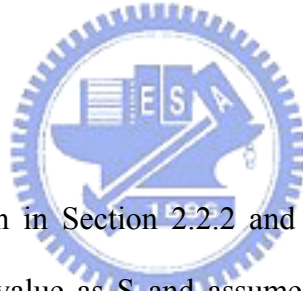


Figure 2-4. Potentially emerging frequent itemsets.

Algorithm of EFI-Mine



With window size we mention in Section 2.2.2 and the concepts of emerging itemsets in section 2.3.1, we set support value as S and assume the original database as $DB_{i,i+1,\dots,j-1}$. According to the scheme we mentioned previously, if we want to find frequent itemsets from $DB_{i+1,i+2,\dots,j+1}$, we should focus on $DB_{i+1,i+2,\dots,j}$ for finding potential emerging frequent itemsets after adding database db_j and then find potential emerging frequent itemsets of the database $DB_{i+1,i+2,\dots,j+1}$ before adding next incremental new database db_{j+1} . It means db_i would be an old database that needs not be considered. After adding new database db_{j+1} , the new database would be $DB_{i+1,i+2,\dots,j+1}$. So the window size is N when database is changed from db_{i+1} to db_{j+1} . It also indicates $N=(j+1)-(i+1)+1$. By the feature of temporal data mining, we set $|db|=|db_i|=|db_{i+1}|=\dots=|db_j|$. In Figure 2-4, various lines bear the following meaning:

$$LineHC = \min SC_{DB_{i+1,i+2,\dots,j-1}} + \min SC_{db_j}$$

$$LineFI = \min SC_{DB_{i+1,i+2,\dots,j-1}}$$

$$LineRS = 2 \min SC_{db_j} + \min SC_{DB_{i+1, i+2, \dots, j-1}} - |db_j|$$

$$LineEC = \min SC_{db_j} + \min SC_{DB_{i+1, i+2, \dots, j-1}} - |db_j|$$

$$LineAK = \min SC_{db_j}$$

According to the feature of window size in temporal mining, incremental database means adding length of original transactions and also promoting the probability of infrequent itemsets to become frequent. Because we focus on N-1 window size for finding potential emerging frequent itemsets, these formulas should be divided by N-1 based on the number of database as follows:

$$LineHC = (\min SC_{DB_{i+1, i+2, \dots, j-1}} + \min SC_{db_j}) / N - 1$$

$$LineRS = (2 \min SC_{db_j} + \min SC_{DB_{i+1, i+2, \dots, j-1}} - |db_j|) / N - 1$$

Because line FI does not add new database, it should be divided by (N-1)-1. It means line FI should be divided by N-2 as follows:

$$LineFI = \min SC_{DB_{i+1, i+2, \dots, j-1}} / N - 2$$

Line EC means that adding new database db_j and an itemset's SC is equal to $|db_j|$ in db_j , so it should be divided by (N-1) as follows:

$$LineEC = (\min SC_{db_j} + \min SC_{DB_{i+1, i+2, \dots, j-1}} - |db_j|) / N - 1$$

Because db_j belongs to one of N window size, the formula should be divided by N as follows:

$$LineAK = \min SC_{db_j} / N$$

Figure 2-5 illustrates the potentially emerging frequent itemsets in area GDSC with window size limitation. The formula for each line is as mentioned above.

According to these formulas, we can simplify these lines as follows:
 $HC = [S*(j-1-(i+1)+1)*|db| + S*|db|] / N - 1 = [S*(N-2)*|db| + S*|db|] / N - 1 = S*|db|$

$$FI = [S * (j-1 - (i+1) + 1) |db|] / N - 2 = S * |db|$$

$$RS = [2 * S * |db| + S * [(j-1) - (i+1) + 1] * |db| - |db|] / N - 1 = [2 * S * |db| + S * (N-2) * |db| - |db|] / N - 1 = [(S * N) - 1] * |db| / N - 1$$

$$EC = [S * |db| + S * [(j-1) - (i+1) + 1] * |db| - |db|] / N - 1 = [S * |db| + S * (N-2) * |db| - |db|] / N - 1 = [S * (N-1) - 1] * |db| / N - 1$$

$$AK = S * db / N$$

We can also find potentially emerging frequent itemsets in area HRSC without concerning support count in db_j . However, it will reduce the accuracy with potentially emerging frequent itemsets. Taking into consideration of db_j would get the trend of itemsets and get better accuracy with potentially emerging frequent itemsets. Therefore, itemsets in GDSC are most likely to emerge in the next increment.

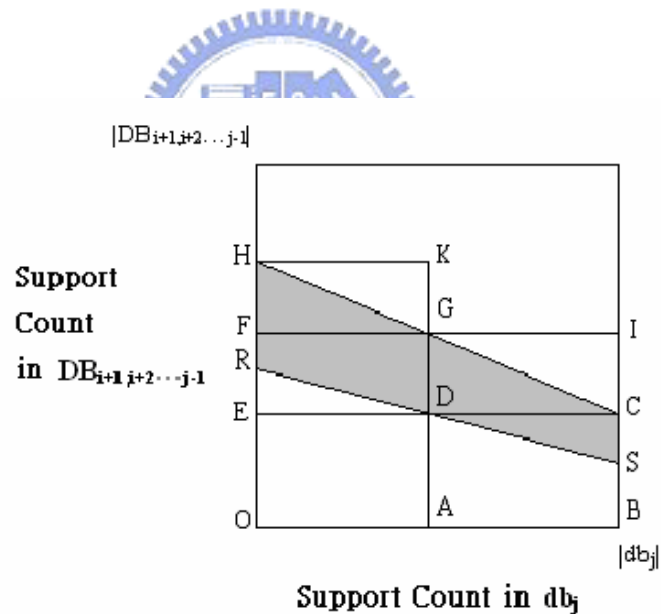


Figure 2-5. Potentially emerging frequent itemsets for temporal patterns.

Figure 2-6 shows the algorithm of *EFI-Mine* and the processing procedure is outlined below. The basic processing procedure is like Apriori except the definition of for minimum support value for finding temporal emerging itemsets from data stream. With window size N , we would not only remove db_i but also add new database db_j for finding 1-emerging itemsets

on the database $DB_{i+1,i+2,\dots,j}$ and finding large 1-itemsets on the database db_j from Step 1 to Step 3. So the purpose is to find potential emerging frequent itemsets of the database $DB_{i+1,i+2,\dots,j+1}$ before adding next new database db_{j+1} . We generate k-candidates and find k-emerging itemsets by calculating support count as mentioned previously from Step 4 to Step 13. Then, we generate k-candidates and find k-large itemsets by support count we mention from Step 14 to Step 23. Finally, those itemsets meeting the constraints $S*|db| > c.count \geq [(S*N)-1]*|db|/N-1$ on $DB_{i+1,i+2,\dots,j}$ and $c.count \geq S*db/N$ db_j are obtained as the potentially emerging frequent itemsets.

```

1) Input:  $DB_{i+1,\dots,j-1}$ 
2) Remove  $db_i$  from  $DB_{i+1,\dots,j-1}$  and add new database  $db_j$  then the database becomes
 $DB_{i+1,i+2,\dots,j}$ 
3)  $E_1 =$  (emerging 1-itemsets on database  $DB_{i+1,i+2,\dots,j}$ ) and  $L_1 =$  (large 1-itemsets on database
 $db_j$ );
4) for (  $k = 2; E_{k-1} \neq \emptyset$   $k++$  ) do begin
5)  $C_k =$  candidate-gen( $C_{k-1}$ ); //New candidates
6) forall transactions  $t \in DB_{i+1,\dots,j}$  do begin
7)  $C_t =$  subset( $C_k, t$ ); // Candidates contained in transactions
8) forall candidates  $c \in C_t$  do
9)  $c.count++$ ;
10) end
11)  $E_k = \{c \in C_k \mid S*|db| > c.count \geq [(S*N)-1]*|db|/N-1 \}$ 
12) // SC between LineHC and LineRS
13) end
14) for (  $k = 2; L_{k-1} \neq \emptyset$   $k++$  ) do begin
15)  $C_k =$  candidate-gen( $C_{k-1}$ ); //New candidates
16) forall transactions  $t \in db_j$  do begin
17)  $C_t =$  subset( $C_k, t$ ); // Candidates contained in transactions
18) forall candidates  $c \in C_t$  do
19)  $c.count++$ ;
20) end
21)  $L_k = \{c \in C_k \mid c.count \geq S*db/N \}$ 
22) // SC larger than LineAK
23) end
24) Output:  $(\cup_k E_k) \cap L_k$ ;

```

Figure 2-6. Algorithm of EFI-Mine.

We may utilize the formulas mentioned before to discuss the following situations. Notice that an itemset is emerging or not depends on support count of the itemset. Given an itemset whose support counts in $DB_{i+1,i+2,\dots,j-1}$ and $DB_{i+1,i+2,\dots,j-1}+db_j$ are $SC_{DB_{i+1,i+2,\dots,j-1}}$ and

$SC_{DB_{i+1,i+2,\dots,j-1+db_j}}$, respectively, the growth rate of that itemset is $SC_{DB_{i+1,i+2,\dots,j-1+db_j}} - SC_{DB_{i+1,i+2,\dots,j-1}}$.

The growth rate of an itemset that maintains minimal support is $\min SC_{DB_{i+1,i+2,\dots,j-1+db_j}} - \min SC_{DB_{i+1,i+2,\dots,j-1}}$. An itemset meeting the

$\frac{SC_{DB_{i+1,i+2,\dots,j-1+db_j}} - SC_{DB_{i+1,i+2,\dots,j-1}}}{\min SC_{DB_{i+1,i+2,\dots,j-1+db_j}} - \min SC_{DB_{i+1,i+2,\dots,j-1}}} > 1$ is an emerging itemset. An itemset needs a support

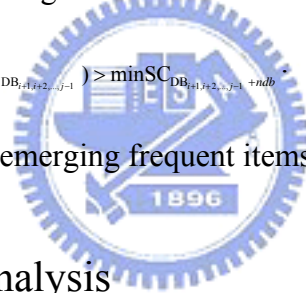
count of at least $\min SC_{DB_{i+1,i+2,\dots,j-1+db_j+db_{j+1}}} = \min SC_{DB_{i+1,i+2,\dots,j-1} + 2db}$ to emerge in adding a new database db_{j+1} with expanding one window size. A potential emerging frequent itemset is the

one that is emerging and meets the following constraint:

$SC_{DB_{i+1,i+2,\dots,j-1+db_j}} + (SC_{DB_{i+1,i+2,\dots,j-1+db_j}} - SC_{DB_{i+1,i+2,\dots,j-1}}) > \min SC_{DB_{i+1,i+2,\dots,j-1} + 2db}$. Hence, we can infer that an itemset that will

potentially emerge with expanding n window sizes is an itemset that is currently emerging

and $SC_{DB_{i+1,i+2,\dots,j-1+db_j}} + n(SC_{DB_{i+1,i+2,\dots,j-1+db_j}} - SC_{DB_{i+1,i+2,\dots,j-1}}) > \min SC_{DB_{i+1,i+2,\dots,j-1} + ndb}$. Of course, the larger n is, the less accurate with finding potential emerging frequent itemsets might be.



2.3 Experiments and Analysis

To evaluate the performance of *EFI-Mine*, we conducted experiments of using synthetic dataset generated via a randomized transaction generation algorithm in [3]. The synthetic data generation program takes the parameters as shown in Table 2-2, and the values of parameters used to generate the datasets are shown in Table 2-3. The simulation is implemented in C++ and conducted in a machine with 1.4GHz CPU and 512MB memory. The main performance metrics used are execution time and accuracy. We recorded the execution time that *EFI-Mine* spends in finding potential emerging frequent itemsets. The accuracy is to measure the number of actual emerging frequent itemset in ratio of the total potential emerging frequent itemsets that we found. Hence, the accuracy is defined as follows:

$$\text{Accuracy} = (\text{number of actual emerging frequent itemset}) / (\text{total potential emerging frequent itemset})$$

frequent itemsets)

Table 2-2. Parameters of the synthetic datasets.

N	Number of items
T	Average numbers of items per transaction
C	Number of customers
D	Number of transactions
W	Windows size
S	Support value

Table 2-3. Parameter settings of synthetic datasets.

Dataset Parameters	N	T	C	D	W
N100T5C1000	100	5	1000	100,000	10

Effects of Varying Support Threshold

The proposed approach is verified with experiments in various measurements. We vary the values of support threshold from 30% to 70% for interesting the effects on the accuracy. The other parameters were kept fixed as default values. Figure 2-7 shows the accuracy of *EFI-Mine* under different support threshold values. It is observed that the average accuracy of potential emerging frequent itemsets raises as the support value is increased. Especially, the accuracy reaches to 100% when the support value is beyond 60%. Hence, *EFI-Mine* is verified to be very effective in finding the emerging itemsets.

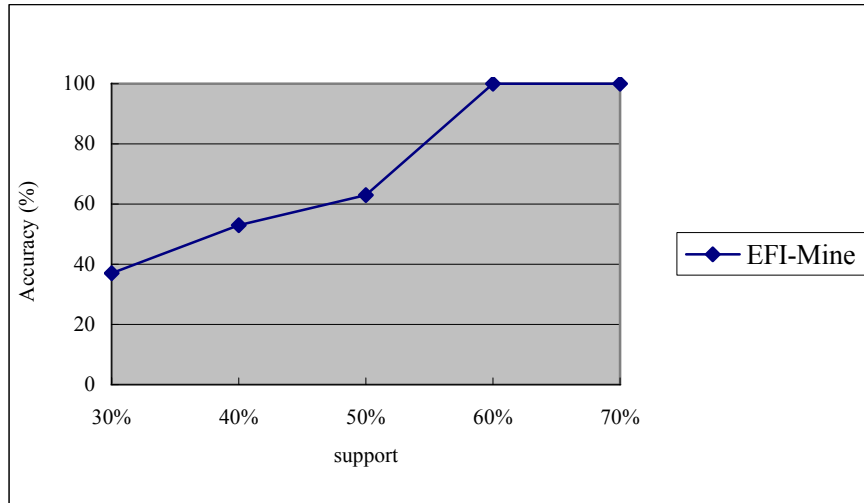


Figure 2-7. Accuracy under different support values (N100T5C1000, W=10).

Comparisons with Apriori in Execution time

The proposed algorithm is also compared to the well know Apriori algorithm. We compare the average execution time in different support values between Apriori and *EFI-Mine*. Both of these two algorithms could find frequent itemsets. However, Apriori can only find frequent itemsets, while *EFI-Mine* can find frequent itemsets that were infrequent in the past. Apriori algorithm processes $DB_{i+1,i+2,\dots,j+1}$ to find frequent itemsets, while our *EFI-Mine* algorithm needs to process fewer database $DB_{i+1,i+2,\dots,j}$ to find potentially emerging frequent itemsets. From Figure 2-8, *EFI-Mine* spends few seconds with high stability for finding potentially emerging frequent itemsets. Compared to Apriori, the improvement is about 90.6% for support values varied from 30% to 60%. Although *EFI-Mine* does not always obtain frequent itemsets with 100% accuracy, it reduces substantially the time in finding frequent itemsets. Moreover, the frequent itemsets obtained by Apriori are not suitable for applications in temporal databases since we need frequent itemsets which are infrequent in the past and frequent in the current by the time change. Hence, *EFI-Mine* meets the requirements of high efficiency and high scalability in terms of execution time for data stream mining.

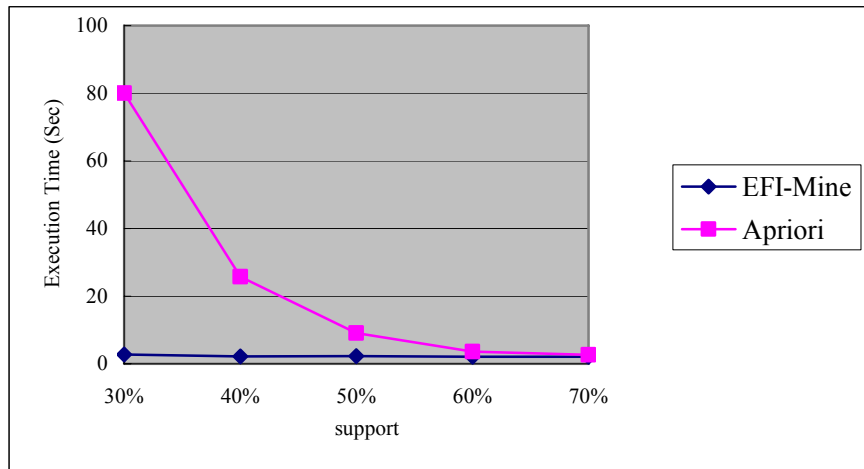


Figure 2-8. Execution time with $w=10$.

Effects of Varying Window Size

We investigate the effects of varying window size on the accuracy of mining results. As shown in Figure 2-9, we could observe that the larger window size, the higher with accuracy. In fact, the accuracy is almost 100% when window size is large than 15 in the experiments. This is because the itemsets are tended to be stable according to the past databases. This indicates that *EFI-Mine* fits for mining temporal databases with large window size.

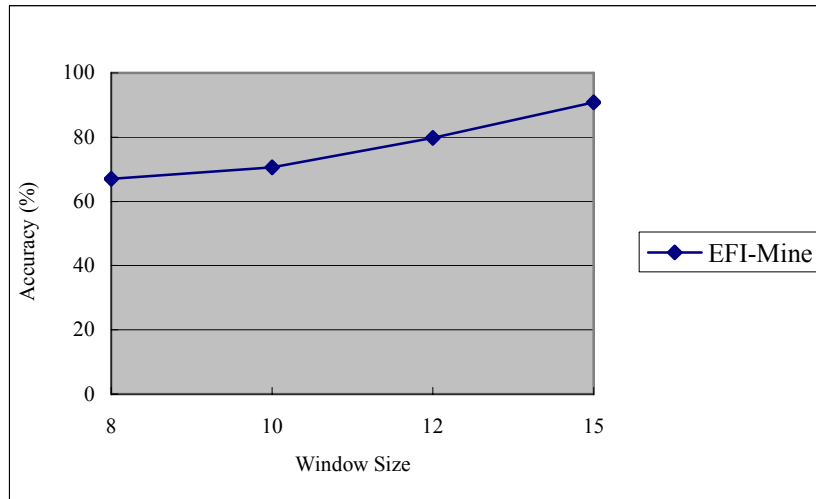


Figure 2-9. Accuracy under different window sizes.

Effects of Varying Transaction Size

We investigate the effects of varying transaction size on the accuracy of mining results i.e., the average number of items per transaction. As shown in Figure 2-10, if T is larger, the accuracy is higher than under T . This is because T can bring more information and trend from past transactions. This indicates that *EFI-Mine* fits for mining temporal databases with large transaction size.

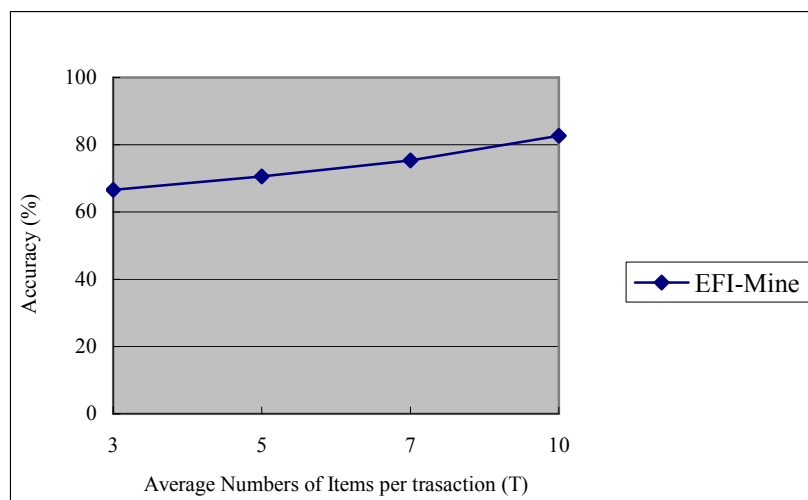


Figure 2-10. Accuracy under different numbers of items per transaction with $w=10$.

Effects of Varying Number of Items

In this last experiment, we investigate the effects of varying the numbers of items on the accuracy of mining results. The results are as shown in Figure 2-11. We observe that the accuracy decreases when the numbers of items are increased. This is because too many items will affect the stability of the patterns. On the contrary, the accuracy under smaller numbers of items could reach almost 100%. This indicates that *EFI-Mine* fits for mining temporal databases with small numbers of items.

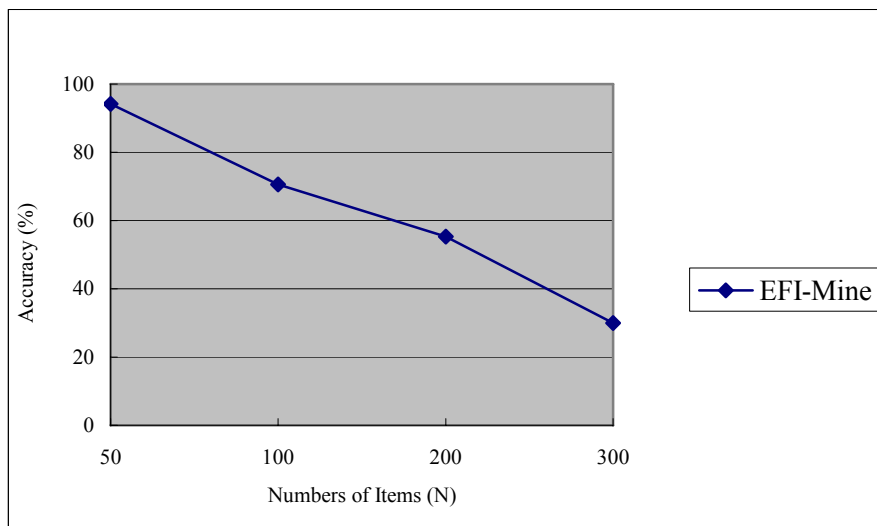


Figure 2-11. Accuracy under different numbers of items.

Chapter 3

Mining Temporal High Utility Itemsets from Temporal Databases

3.1 Problem Definition

The mining of association rules for finding the relationship between data items in large databases is a well studied technique in the data mining field with representative methods like *Apriori* [1][2]. The problem of mining association rules can be decomposed into two steps. The first step involves finding of all frequent itemsets (or say large itemsets) in databases. Once the frequent itemsets are found, generating association rules is straightforward and can be accomplished in linear time.

An important research issue extended from the mining of association rules is the discovery of temporal association patterns in temporal databases due to the wide applications on various domains. Temporal data mining can be defined as the activity of discovering interesting correlations or patterns in large sets of temporal data accumulated for other purposes [6]. For a database with a specified transaction window size, we may use an algorithm like *Apriori* to obtain frequent itemsets from the database. For time-variant temporal databases, there is a strong demand to develop an efficient and effective method to mine various temporal patterns [13]. However, most methods designed for traditional databases cannot be directly applied to the mining of temporal patterns in temporal databases because of their high complexity.

In many applications, we would like to mine temporal association patterns from the most

recent data in temporal databases. That is, in temporal data mining, one should not only include new data (i.e., data in the new hour) but also remove the old data (i.e., data in the most obsolete hour) from the mining process. Without loss of generality, consider a typical market-basket application as illustrated in Figure 3-1, where the transactional data of customer purchases are shown as time advances.

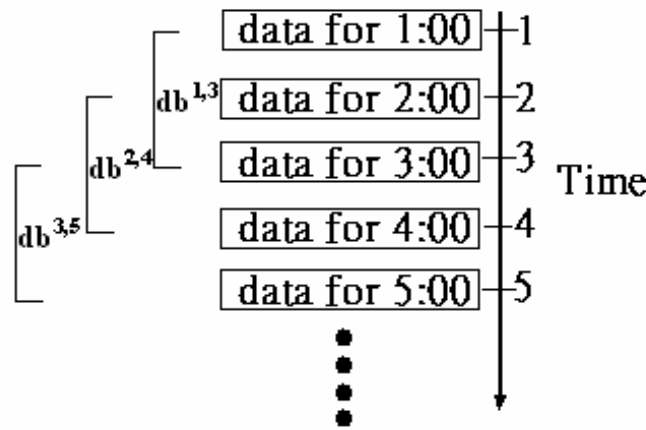


Figure 3-1. An example of online transaction flows.

In Figure 3-1, for example, data was accumulated as a function of time. Data obtained prior to some specified time interval in the past becomes useless for reference. People might be most interested in the temporal association patterns in the latest three hours (i.e., $db^{3,5}$) as shown in Figure 3-1. It can be seen that in such a temporal database environment it is intrinsically difficult to conduct the frequent pattern identification due to the constraints of limited time and memory space. Furthermore, it takes considerable time to find temporal frequent itemsets in different time windows. However, the frequency of an itemset may not be a sufficient indicator of interestingness, because it only reflects the number of transactions in the database that contain the itemset. It does not reveal the *utility* of an itemset, which can be measured in terms of cost, profit or other expressions of user preferences. On the other hand, frequent itemsets may only contribute a small portion of the overall profit, whereas

non-frequent itemsets may contribute a large portion of the profit. In reality, a retail business may be interested in identifying its most valuable customers (customers who contribute a major fraction of the profits to the company). Hence, frequency is not sufficient to answer questions such as whether an itemset is highly profitable, or whether an itemset has a strong impact. Utility mining is thus useful in a wide range of practical applications and was recently studied in [8][21][35]. This also motivates our research in developing a new scheme for finding *temporal high utility itemsets (THUI)* from temporal databases.

Table 3-1. A transaction database and its utility table.

				(a) Transaction table					(b) The utility table		
				ITEM TID	A	B	C	D	E	ITEM	PROFIT\$(per unit)
db^{1,3}	Δ^-	P₁	T₁	0	0	26	0	1	db^{2,4}	A	3
			T₂	0	6	0	1	1		B	10
			T₃	12	0	0	1	0		C	1
	D^-	P₂	T₄	0	1	0	7	0		D	6
			T₅	0	0	12	0	2		E	5
			T₆	1	4	0	0	1			
		P₃	T₇	0	10	0	0	1			
			T₈	1	0	1	3	1			
			T₉	1	1	27	0	0			
	Δ^+	P₄	T₁₀	0	6	2	0	0			
			T₁₁	0	3	0	2	0			
			T₁₂	0	2	1	0	0			

Recently, a *utility mining* model was defined in [35]. Utility is considered as a measure of how “useful” (e.g., “profitable”) an itemset is. The definition of utility $u(X)$ of an itemset X is the sum of the utilities of X in all transactions containing X . The goal of utility mining is to identify high utility itemsets which drive a large portion of the total utility. Traditional

association rules mining models assume that the utility of each item is always 1 and the sales quantity is either 0 or 1, thus it is only a special case of utility mining, where the utility or the sales quantity of each item could be any number. If $u(X)$ is greater than a utility threshold, X is a high utility itemset. Otherwise, it is a low utility itemset. Table 3-1 is an example of utility mining in a transaction database. The number in each transaction in Table 3-1(a) is the sales volume of each item, and the utility of each item is listed in Table 3-1(b). For example, $u(\{B, D\}) = (6 \times 10 + 1 \times 6) + (1 \times 10 + 7 \times 6) + (3 \times 10 + 2 \times 6) = 160$. $\{B, D\}$ is considered as a high utility itemset if the utility threshold is set at 120.

However, a high utility itemset may consist of some low utility items. Another attempt is to adopt the level-wise searching schema that exists in fast algorithms such as Apriori [3]. However, this algorithm does not apply to the *utility mining* model. For example, consider $u(D) = 84 < 120$ as shown in Table 3-1, D is a low utility item. However, its superset $\{B, D\}$ is a high utility itemset. If Apriori is used to find high utility itemsets, all combinations of the items must be generated. Moreover, the number of candidates is prohibitively large in discovering a long pattern. The cost of either computation time or memory will be intolerable, regardless of what implementation is applied. The challenge of utility mining is not only in restricting the size of the candidate set but also in simplifying the computation for calculating the utility. Another challenge of utility mining is how to find temporal high utility itemsets from temporal databases as time advances.

In this chapter, we explore the issue of efficiently mining high utility itemsets in temporal databases like data streams. We propose an algorithm named *THUI-Mine* that can discover temporal high utility itemsets from temporal databases efficiently and effectively. The underlying idea of *THUI-Mine* algorithm is to integrate the advantages of Two-Phase algorithm [21] and SWF algorithm [18] with augmentation of the incremental mining techniques for mining temporal high utility itemsets efficiently. The novel contribution of *THUI-Mine* is that it can efficiently identify the utility itemsets in temporal databases so that the execution time for mining high utility itemsets can be substantially reduced. That is,

THUI-Mine can discover the temporal high utility itemsets in current time window and also discover the temporal high utility itemsets in the next time window with limited memory space and less computation time by sliding window filter method. In this way, the process of discovering all temporal high utility itemsets under all time windows of temporal databases can be achieved effectively under less memory space and execution time. This meets the critical requirements of time and space efficiency for mining temporal databases. Through experimental evaluation, *THUI-Mine* is shown to produce fewer candidate itemsets in finding the temporal high utility itemsets, so it outperforms other methods in terms of execution efficiency. It is observed that the average improvement of *THUI-Mine* over Two-Phase algorithm reaches to about 67%. Moreover, it also achieves high scalability in dealing with large databases. To our best knowledge, this is the first work on mining temporal high utility itemsets from temporal databases.

The rest of this chapter is organized as follows: Section 3.2 describes the proposed approach, *THUI-Mine*, for finding the temporal high utility itemsets. In section 3.3, we describe the experimental results for evaluating the proposed method. The conclusion of the chapter is provided in Section 3.4.

3.2 Proposed Method

In this section, we present the *THUI-Mine* method. We describe the basic concept of *THUI-Mine* and give an example for mining temporal high utility itemsets. The procedure of the *THUI-Mine* algorithm is provided in the last paragraph the section.

Basic Concept of THUI-MINE

The goal of utility mining is to discover all the itemsets whose utility values are beyond a user specified threshold in a transaction database. In [35], the goal of utility mining is defined as

the discovery of all high utility itemsets. An itemset X is a *high utility itemset* if $u(X) \geq \epsilon$, where $X \subseteq I$ and ϵ is the minimum utility threshold; otherwise, it is a *low utility itemset*. For example, in Table 3-1, $u(A, T_8) = 1 \times 3 = 3$, $u(\{A, C\}, T_8) = u(A, T_8) + u(C, T_8) = 1 \times 3 + 1 \times 1 = 4$, and $u(\{A, C\}) = u(\{A, C\}, T_8) + u(\{A, C\}, T_9) = 4 + 30 = 34$. If $\epsilon = 120$, $\{A, C\}$ is a low utility itemset. However, if an item is a low utility item, its superset may be a high utility itemset. For example, consider $u(D) = 84 < 120$, D is a low utility item, but its superset $\{B, D\}$ is a high utility itemset since $u(\{B, D\}) = 160 > 120$. Intuitively, all combinations of items should be processed so that it never loses any high utility itemset. However, this will incur intolerable cost on computation time and memory space. A set of terms leading to the formal definition of utility mining problem can be generally defined as follows by referring to [35]:

- $I = \{i_1, i_2, \dots, i_m\}$ is a set of items.
- $D = \{T_1, T_2, \dots, T_n\}$ is a transaction database where each transaction $T_i \in D$ is a subset of I .
- $o(i_p, T_q)$, *local transaction utility value*, represents the quantity of item i_p in transaction T_q . For example, $o(A, T_3) = 12$, as shown in Table 3-1(a).
- $s(i_p)$, *external utility*, is the value associated with item i_p in the Utility Table. This value reflects the importance of an item, which is independent of transactions. For example, in Table 3-1(b), the external utility of item A, $s(A)$, is 3.
- $u(i_p, T_q)$, *utility*, the quantitative measure of utility for item i_p in transaction T_q , is defined as $o(i_p, T_q) \times s(i_p)$. For example, $u(A, T_3) = 12 \times 3$, in Table 3-1.
- $u(X, T_q)$, *utility of an itemset X in transaction Tq*, is defined as $\sum_{i_p \in X} u(i_p, T_q)$, where $X = \{i_1, i_2, \dots, i_m\}$ is a k -itemset, $X \subseteq T_q$ and $1 \leq k \leq m$.
- $u(X)$, *utility of an itemset X*, is defined as $\sum_{T_q \in D \wedge X \subseteq T_q} u(X, T_q)$.

Liu *et al.* [21] proposed the Two-Phase algorithm for pruning candidate itemsets and simplifying the calculation of utility. First, the Phase I overestimates some low utility itemsets,

but it never underestimates any itemsets. For the example in Table 3-1, the *transaction utility* of transaction T_q , denoted as $tu(T_q)$, is the sum of the utilities of all items in T_q : $tu(T_q) = \sum_{i_p \in T_q} u(i_p, T_q)$. Moreover, the *transaction-weighted utilization of an itemset* X , denoted as $twu(X)$, is the sum of the transaction utilities of all the transactions containing X : $twu(X) = \sum_{X \subseteq T_q \in D} tu(T_q)$. For example, $twu(A) = tu(T_3) + tu(T_6) + tu(T_8) + tu(T_9) = 42 + 48 + 27 + 40 = 157$ and $twu(\{D, E\}) = tu(T_2) + tu(T_8) = 71 + 27 = 98$. In fact, $u(A) = u(\{A\}, T_3) + u(\{A\}, T_6) + u(\{A\}, T_8) + u(\{A\}, T_9) = 36 + 3 + 3 + 3 = 45$ and $u(\{D, E\}) = u(\{D, E\}, T_2) + u(\{D, E\}, T_8) = 11 + 23 = 34$. Table 3-2 gives the transaction utility for each transaction in Table 3-1. Second, one extra database scan is performed to filter the overestimated itemsets in phase II. For example, by observing that $twu(A) = 157 > 120$ and $u(A) = 45 < 120$, the item $\{A\}$ is pruned. Otherwise, it is a high utility itemset. Finally, all high utility itemsets are discovered in this way.

We illustrate the detail process of Two-Phase algorithm by the following example in $db^{1,3}$ of Table 3-1. Suppose the utility threshold is set as 120 with nine transactions in $db^{1,3}$. In Phase I, the high transaction-weighted utilization 1-itemsets $\{A, B, C, D, E\}$ are generated since $twu(A) = tu(T_3) + tu(T_6) + tu(T_8) + tu(T_9) = 42 + 48 + 27 + 40 = 157 > 120$, $twu(B) = tu(T_2) + tu(T_4) + tu(T_6) + tu(T_7) + tu(T_9) = 71 + 52 + 48 + 105 + 40 = 361 > 120$, $twu(D) = tu(T_2) + tu(T_3) + tu(T_4) + tu(T_8) = 71 + 42 + 52 + 27 = 192 > 120$ and $twu(E) = tu(T_1) + tu(T_2) + tu(T_5) + tu(T_6) + tu(T_7) + tu(T_8) = 31 + 71 + 22 + 48 + 105 + 27 = 304 > 120$. Then, 10 candidate 2-itemsets $\{AB, AC, AD, AE, BC, BD, BE, CD, CE, DE\}$ are generated by the high transaction-weighted utilization 1-itemsets $\{A, B, C, D, E\}$ in the first database scan. In the same way, the high transaction-weighted utilization 2-itemset $\{BE\}$ are generated since $twu(AB) = tu(T_6) + tu(T_9) = 48 + 40 = 88 < 120$, $twu(AC) = tu(T_8) + tu(T_9) = 27 + 40 = 67 < 120$, $twu(AD) = tu(T_3) + tu(T_8) = 42 + 27 = 69 < 120$, $twu(AE) = tu(T_6) + tu(T_8) = 48 + 27 = 75 < 120$, $twu(BC) = tu(T_9) = 40 < 120$, $twu(BD) = tu(T_4) = 52 < 120$, $twu(BE) = tu(T_2) + tu(T_6) + tu(T_7) = 71 + 48 + 105 = 224 > 120$, $twu(CD) = tu(T_8) = 27 < 120$, $twu(CE) = tu(T_1) +$

$tu(T_5) + tu(T_8) = 31 + 22 + 27 = 80 < 120$ and $twu(DE) = tu(T_2) + tu(T_8) = 71 + 27 = 98 < 120$.

After processing $db^{1,3}$, the high transaction-weighted utilization itemsets in $db^{1,3}$ are obtained as $\{A, B, C, D, E, BE\}$.

In phase II, the high transaction-weighted utilization itemsets $\{A, B, C, D, E, BE\}$ is used to scan $db^{1,3}$ to find high utility itemsets. The resulting high utility itemsets are $\{B\}$ and $\{BE\}$ since $u(A) = u(\{A\}, T_3) + u(\{A\}, T_6) + u(\{A\}, T_8) + u(\{A\}, T_9) = 45 < 120$, $u(B) = u(\{B\}, T_2) + u(\{B\}, T_4) + u(\{B\}, T_6) + u(\{B\}, T_7) + u(\{B\}, T_9) = 220 > 120$, $u(C) = u(\{C\}, T_1) + u(\{C\}, T_5) + u(\{C\}, T_8) + u(\{C\}, T_9) = 66 < 120$, $u(D) = u(\{D\}, T_2) + u(\{D\}, T_3) + u(\{D\}, T_4) + u(\{D\}, T_8) = 72 < 120$, $u(E) = u(\{E\}, T_1) + u(\{E\}, T_2) + u(\{E\}, T_5) + u(\{E\}, T_6) + u(\{E\}, T_7) + u(\{E\}, T_8) = 35 < 120$ and $u(\{B, E\}) = u(\{B, E\}, T_2) + u(\{B, E\}, T_6) + u(\{B, E\}, T_7) = 215 > 120$.

Our algorithm *THUI-Mine* is based on the principle of the Two-Phase algorithm [21], and we extend it with the sliding-window-filtering (SWF) technique and focus on utilizing incremental methods to reduce the candidate itemsets and execution time. In essence, by partitioning a transaction database into several partitions from temporal databases, algorithm *THUI-Mine* employs a filtering threshold in each partition to deal with the *transaction-weighted utilization itemsets (TWUI)* generated. The cumulative information in the prior phases is selectively carried over toward the generation of TWUI in the subsequent phases by *THUI-Mine*. In the processing of a partition, a progressive transaction-weighted utilization set of itemsets is generated by *THUI-Mine*. Explicitly, a progressive transaction-weighted utilization set of itemsets is composed of the following two types of TWUI: 1) the TWUI that were carried over from the previous progressive candidate set in the previous phase and remain as TWUI after the current partition is taken into consideration; 2) the TWUI that were not in the progressive candidate set in the previous phase but are newly selected after taking only the current data partition into account. As such, after the processing of a phase, algorithm *THUI-Mine* outputs a *cumulative filter*, denoted as CF, which consists of a progressive transaction-weighted utilization set of itemsets with their occurrence counts and

the corresponding partial support required.

THUI-Mine is different from other existing methods like Lossy Counting [22], which uses bucket boundaries and maximal possible error to update or delete the itemsets with frequency. The CF computes the occurrence counts of itemsets in memory and then deletes itemsets that do not satisfy utility threshold in every partial database. With these design considerations, algorithm *THUI-Mine* is shown to have very good performance for mining temporal high utility itemsets from temporal databases.

Table 3-2. Transaction utility of the transaction database.

TID	Transaction Utility	TID	Transaction Utility
T ₁	31	T ₇	105
T ₂	71	T ₈	27
T ₃	42	T ₉	40
T ₄	52	T ₁₀	62
T ₅	22	T ₁₁	42
T ₆	48	T ₁₂	21

An Example for Mining Temporal High Utility Itemsets

The proposed *THUI-Mine* algorithm can be best understood by the illustrative transaction database in Table 3-1 and Figure 3-2 where a scenario of generating high utility itemsets from temporal databases for mining temporal high utility itemsets is given. For real life applications, this illustrative transaction database can be mapped to the customer transactions in a supermarket. We set the utility threshold as 120 with nine transactions. Without loss of generality, the temporal mining problem can be decomposed into two procedures:

1. Preprocessing procedure: This procedure deals with mining on the original transaction

database.

2. Incremental procedure: The procedure deals with the update of the high utility itemsets from temporal databases.

P ₁			P ₂			P ₃		
C ₂	start	transaction-weighted utility	C ₂	start	transaction-weighted utility	C ₂	start	transaction-weighted utility
AB	1	0	⊗ AB	2	48	⊗ AB	2	88
⊗ AD	1	42	AD	1	42	⊗ AC	3	67
AE	1	0	⊗ AE	2	48	AD	3	27
⊗ BD	1	71	⊗ BD	1	123	AE	2	75
⊗ BE	1	71	⊗ BE	1	119	⊗ BC	3	40
⊗ DE	1	71	DE	1	71	⊗ BD	1	123
						⊗ BE	1	224
						CE	3	27

db ^{1,3} - Δ = D			D + Δ* = db ^{2,4}		
C ₂	start	transaction-weighted utility	C ₂	start	transaction-weighted utility
⊗ AB	2	88	AB	2	88
⊗ AC	3	67	AC	3	67
⊗ BC	3	40	⊗ BC	3	123
BD	2	52	⊗ BD	4	42
⊗ BE	2	153	⊗ BE	2	153
			CD	4	0

Figure 3-2. Temporal high utility itemsets generated by THUI-Mine.

The preprocessing procedure is only utilized for the initial utility mining in the original database, e.g., $db^{1,n}$. For mining high utility itemsets in $db^{2,n+1}$, $db^{3,n+2}$, $db^{i,j}$, and so on, the incremental procedure is employed. Consider the database in Table 3-1. Assume that the original transaction database $db^{1,3}$ is segmented into three partitions, namely, $\{P_1, P_2, P_3\}$, in the pre processing procedure. Each partition is scanned sequentially for the generation of candidate 2-itemsets in the first scan of the database $db^{1,3}$. Since there are three partitions, the utility threshold of each partition is $120 / 3 = 40$. Such a partial utility threshold is called the filtering threshold in this chapter. After scanning the first segment of the three transactions, 1-itemsets $\{A, B, D, E\}$ are kept to generate 2-itemsets because $twu(A) = 42 > 40$, $twu(B) = 71 > 40$, $twu(C) = 31 < 40$, $twu(D) = 113 > 40$ and $twu(E) = 102 > 40$. Then, 2-itemsets $\{AB, AD, AE, BD, BE, DE\}$ are generated by 1-itemsets $\{A, B, D, E\}$ in partition P_1 as shown in Figure 3-2. In addition, each potential candidate itemset $c \in C_2$ has two attributes: (1) $c.start$, which contains the identity of the starting partition when c was added to C_2 , and (2)

transaction-weighted utility which is the sum of the transaction utilities of all the transactions containing c since c was added to C_2 . Itemsets whose transaction-weighted utility are below the filtering threshold are removed. Then, as shown in Figure 3-2, only $\{AD, BD, BE, DE\}$, marked by “ \odot ”, remain as *temporal high transaction-weighted utilization 2-itemsets (TWU2I)* whose information is then carried over to the next phase of processing. Similarly, after scanning partition P_2 , the temporal high TWU2I are recorded.

From Figure 3-2, it is noted that since there are also three transactions in P_2 , the filtering threshold of those itemsets carried out from the previous phase is $40 + 40 = 80$ and that of newly identified candidate itemsets is 40. It can be seen from Figure 3-2 that we have 4 temporal high TWU2I in C_2 after the processing of partition P_2 , and 2 of them are carried from P_1 to P_2 and 2 of them are newly identified in P_2 . Finally, partition P_3 is processed by algorithm *THUI-Mine*. The resulting temporal high TWU2I are $\{AB, AC, BC, BD, BE\}$ as shown in Figure 3-2. Note that although itemset $\{AE\}$ appears in the previous phase P_2 , it is removed from temporal high TWU2I once P_3 is taken into account since its transaction-weighted utility does not meet the filtering threshold then, i.e., $75 < 120$. However, we do have two new itemsets, i.e., AC and BC, which join the C_2 as temporal high TWU2I. Consequently, we have five temporal high TWU2I generated by *THUI-Mine*, where two of them are carried from P_1 to P_3 , one of them is carried from P_2 to P_3 , and two of them are newly identified in P_3 . Note that only 5 temporal high TWU2I are generated by *THUI-Mine*, while 10 candidate itemsets would be generated if Two-Phase algorithm were used as mentioned in section 3.3.1. After processing P_1 to P_3 , those temporal high TWUI in $db^{1,3}$ are obtained as $\{A, B, C, D, E, AB, AC, BC, BD, BE\}$.

After generating temporal high TWU2I from the first scan of database $db^{1,3}$, we use a skill to reduce the number of database scan. In fact, it will take $k-1$ database scan to generate k -candidate itemsets by using temporal high transaction-weighted utilization $(k-1)$ -itemsets directly. Instead, we use temporal high TWU2I to generate C_k ($k = 3, 4, \dots, n$), where C_n is the

candidate last itemset. It can be verified that temporal high TWU2I generated by *THUI-Mine* can be used to generate the candidate 3-itemsets. Clearly, a C_3 can be generated from temporal high TWU2I. For example, the 3-candidate itemset $\{ABC\}$ is generated from temporal high TWU2I $\{AB, AC, BC\}$ in $db^{1,3}$. However, the temporal high TWU2I generated by *THUI-Mine* is very close to the high utility itemsets. Similarly, all C_k can be stored in main memory and we can find temporal high utility itemsets together when the second scan of the database $db^{1,3}$ is performed. Thus, only two scans of the original database $db^{1,3}$ are required in the preprocessing step. In this way, the number of database scan is reduced effectively. The resulting temporal high utility itemsets are $\{B\}$ and $\{BE\}$ since $u(B) = 220 > 120$ and $u(\{B, E\}) = 215 > 120$.

One important merit of *THUI-Mine* lies in its incremental procedure. As depicted in Figure 3-2, the mining of database will be moved from $db^{1,3}$ to $db^{2,4}$. Thus, some transactions like T_1, T_2 and T_3 are deleted from the mining database and other transactions like T_{10}, T_{11} , and T_{12} , are added. To illustrate it more clearly, this incremental step can also be divided into three sub-steps: (1) generating temporal high TWU2I in $D^- = db^{1,3} - \Delta^-$, (2) generating temporal high TWU2I in $db^{2,4} = D^- + \Delta^+$ and (3) scanning the database $db^{2,4}$ only once for the generation of all temporal high utility itemsets. In the first sub-step, $db^{1,3} - \Delta^- = D^-$, we check the pruned partition P_1 and reduce the value of transaction-weighted utility and set $c.start = 2$ for those temporal TWU2I where $c.start = 1$. It can be seen that itemset $\{BD\}$ was removed. Next, in the second sub-step, we scan the incremental transactions in P_4 . The process in $D^- + \Delta^+ = db^{2,4}$ is similar to the operation of scanning partitions, e.g., P_2 , in the preprocessing step. The new itemset $\{BD\}$ joins the temporal high TWU2I after the scan of P_4 . In the third sub-step, we use temporal high TWU2I to generate C_k as mentioned above. Finally, those temporal high TWUI in $db^{2,4}$ are $\{B, C, D, E, BC, BD, BE\}$. By scanning $db_{2,4}$ only once, *THUI-Mine* obtains temporal high utility itemsets $\{B, BC, BE\}$ in $db^{2,4}$.

In contrast, Two-Phase algorithm has to scan the whole database like $db^{2,4}$ and more

candidate itemsets, i.e., {BC, BD, BE, CD, CE, DE}, will be generated whenever some transactions are deleted and other transactions are added. Then, Two-Phase algorithm needs one more database scan than *THUI-Mine* to obtain temporal high TWU2I. Finally, Two-Phase algorithm scans database again to produce temporal high utility itemsets. Hence, more database scans and candidate itemsets are incurred by Two-Phase algorithm in comparison with *THUI-Mine*.

Table 3-3. Meanings of symbols used.

$db^{i,j}$	Partitioned_database (D) from P_i to P_j
s	Utility threshold in one partition
$ P_k $	Number of transactions in partition P_k
$TUP_k(I)$	Transactions in P_k that contain itemset I with transaction utility
$UP_k(I)$	Transactions in P_k that contain itemset I with utility
$ db^{1..n}(I) $	Transactions number in $db^{1..n}$ that contain itemset I
$C^{i,j}$	The progressive candidate sets of $db^{i,j}$
$Thtw^{i,j}$	The progressive temporal high transaction-weighted utilization 2-itemsets of $db^{i,j}$
$Thu^{i,j}$	The progressive temporal high utility itemsets of $db^{i,j}$
Δ^-	The deleted portion of an ongoing database
D^-	The unchanged portion of an ongoing database
Δ^+	The added portion of an ongoing database

THUI-Mine Algorithm

For easier illustration, the meanings of various symbols used are given in Table 3-3. The preprocessing procedure and the incremental procedure of algorithm *THUI-Mine* are described as follows:

Preprocessing procedure of THUI-Mine

The preprocessing procedure of Algorithm *THUI-Mine* is shown in Figure 3-3. Initially, the database $db^{1,n}$ is partitioned into n partitions by executing the preprocessing procedure (in Step 2), and CF, the cumulative filter, is empty (in Step 3). Let $Thtw^{1,n}$ be the set of progressive temporal high TWU2I of $db^{i,j}$. Algorithm *THUI-Mine* only records $Thtw^{1,n}$ which is generated by the preprocessing procedure to be used by the incremental procedure. From Step 4 to Step 16, the algorithm processes one partition at a time for all partitions. When partition P_i is processed, each potential candidate 2-itemset is read and saved to CF. The transaction-weight utility of an itemset I and its starting partition are recorded in $I.twu$ and $I.start$, respectively. An itemset, whose $I.twu \geq s$, will be kept in CF. Next, we select $Thtw^{1,n}$ from I where $I \in CF$ and keep $I.twu$ in main memory for the subsequent incremental procedure. By employing the scan reduction technique from Step 19 to Step 26, $C_h^{1,n}$ ($h \geq 3$) are generated in main memory. After refreshing $I.count = 0$ where $I.twu = 0$ where $I \in Thtw^{1,n}$, we begin the last scan of database for the preprocessing procedure from Step 28 to Step 31. Finally, those itemsets satisfying the constraint that $I.u \geq s \times P.count$ are finally obtained as the temporal high utility itemsets.

```

1.  $n$  = Number of partitions;
2.  $|db^{1,n}| = \sum_{k=1,n} |P_k|$ ;
3.  $CF = \phi$ ;
4. begin for  $k = 1$  to  $n$  // 1st scan of  $db^{1,n}$ 
5.     begin for each 2-itemset  $I \in P_k$ 
6.         if ( $I \notin CF$ )
7.              $I.twu = TUP_k(I)$ ;
8.              $I.start = k$ ;
9.             if ( $I.twu \geq s \times P.count$ ) //  $P.count$  is number of partitions
10.                 $CF = CF \cup I$ ;
11.         if ( $I \in CF$ )
12.              $I.twu = I.twu + TUP_k(I)$ ;
13.             if ( $I.twu < s \times P.count$ )
14.                  $CF = CF - I$ ;
15.     end
16. end
17. select  $Thtw^{1,n}$  from  $I$  where  $I \in CF$ ;
18. keep  $Thtw^{1,n}$  in main memory;
19.  $h = 2$ ;
20. begin while ( $Thtw^{1,n} \neq \phi$ ) // Database scan reduction
21.     if ( $h=2$ )
22.          $C_{h+1}^{1,n} = Thtw^{1,n} * Thtw^{1,n}$ ;
23.     else
24.          $C_{h+1}^{1,n} = C_h^{1,n} * C_h^{1,n}$ 
25.      $h = h + 1$ ;
26. end
27. refresh  $I.twu = 0$  where  $I \in Thtw^{1,n}$ ;
28. begin for  $k = 1$  to  $n$  // 2nd scan of  $db^{1,n}$ 
29.     for each itemset  $I \in Thtw^{1,n} \cup C_{h+1}^{1,n}$ 
30.          $I.u = I.u + UP_k(I)$ ;
31.     end
32. for each itemset  $I \in Thtw^{1,n} \cup C_{h+1}^{1,n}$ 
33.     if ( $I.u \geq s \times P.count$ )
34.          $Thu^{1,n} = Thu^{1,n} \cup I$ ;
35. end
36. return  $Thu^{1,n}$ ;

```

Figure 3-3. Preprocessing procedure of THUI-Mine.

Incremental procedure of THUI-Mine

As shown in Table 3-3, D^- indicates the unchanged portion of an ongoing transaction database. The deleted and added portions of an ongoing transaction database are denoted by Δ^- and Δ^+ , respectively. It is worth mentioning that the sizes of Δ^+ and Δ^- , i.e., $|\Delta^+|$ and $|\Delta^-|$ respectively, are not required to be the same. The incremental procedure of *THUI-Mine* is

devised to maintain temporal high utility itemsets efficiently and effectively. This procedure is shown in Figure 3-4. As mentioned before, this incremental step can also be divided into three sub-steps: (1) generating temporal high TWU2I in $D^- = db^{1,3} - \Delta^-$, (2) generating temporal high TWU2I in $db^{2,4} = D^- + \Delta^+$ and (3) scanning the database $db^{2,4}$ only once for the generation of all temporal high utility itemsets. Initially, after some update activities, old transactions Δ^- are removed from the database $db^{m,n}$ and new transactions Δ^+ are added (in Step 6). Note that $\Delta^- \subset db^{m,n}$. Denoting the updated database as $db^{i,j}$, note that $db^{i,j} = db^{m,n} - \Delta^- + \Delta^+$. We denote the unchanged transactions by $D^- = db^{m,n} - \Delta^- = db^{i,j} - \Delta^+$. After loading $Thtw^{m,n}$ of $db^{m,n}$ into CF where $I \in Thtw^{m,n}$, we start the first sub-step, i.e., generating temporal high TWU2I in $D^- = db^{m,n} - \Delta^-$. This sub-step reverses the cumulative processing which is described in the preprocessing procedure. From Step 8 to Step 16, we prune the occurrences of an itemset I , which appeared before partition P_i , by deleting the value $I.twu$ where $I \in CF$ and $I.start < i$. Next, from Step 17 to Step 39, similarly to the cumulative processing in Section 3.3.1, the second sub-step generates temporal high TWU2I in $db^{i,j} = D^- + \Delta^+$ and employs the scan reduction technique to generate $C_{h+1}^{i,j}$. Finally, to generate temporal high utility itemsets, i.e., $Thu^{i,j}$, in the updated database, we scan $db^{i,j}$ only once in the incremental procedure to find temporal high utility itemsets. Note that $Thtw^{i,j}$ is kept in main memory for the next generation of incremental mining.

1. Original database = $db^{m,n}$;
2. New database = db^{ij} ;
3. Database removed $\Delta^- = \sum_{k=m,i+1} P_k$;
4. Database added $\Delta^+ = \sum_{k=n+1,j} P_k$;
5. $D^- = \sum_{k=i,n} P_k$;
6. $db^{ij} = db^{m,n} - \Delta^- + \Delta^+$;
7. loading $Thtw^{m,n}$ of $db^{m,n}$ into CF where $I \in Thtw^{m,n}$;
8. begin for $k = m$ to $i - 1$ // one scan of Δ^-
9. begin for each 2-itemset $I \in P_k$
10. if ($I \in CF$ and $I.start \leq k$)
11. $I.twu = I.twu - TUP_k(D)$;
12. $I.start = k + 1$;
13. if ($I.twu < s \times P.count$)
14. $CF = CF - I$;
15. end
16. end
17. begin for $k = n + 1$ to j // one scan of Δ^+
18. begin for each 2-itemset $I \in P_k$
19. if ($I \notin CF$)
20. $I.twu = TUP_k(D)$;
21. $I.start = k$;
22. if ($I.twu \geq s \times P.count$)
23. $CF = CF \cup I$;
24. if ($I \in CF$)
25. $I.twu = I.twu + TUP_k(D)$;
26. if ($I.twu < s \times P.count$)
27. $CF = CF - I$;
28. end
29. end
30. select $Thtw^{ij}$ from I where $I \in CF$;
31. keep $Thtw^{ij}$ in main memory;
32. $h = 2$
33. begin while ($Thtw^{ij} \neq \phi$) //Database scan reduction
34. if ($h=2$)
35. $C_{h+1}^{i,j} = Thtw^{ij} * Thtw^{ij}$;
36. else
37. $C_{h+1}^{i,j} = C_h^{i,j} * C_h^{i,j}$
38. $h = h + 1$;
39. end
40. refresh $I.twu = 0$ where $I \in Thtw^{ij}$;
41. begin for $k = i$ to j //2nd scan of db^{ij}
42. for each itemset $I \in Thtw^{ij} \cup C_{h+1}^{i,j}$
43. $I.u = I.u + UP_k(D)$;
44. end
45. for each itemset $I \in Thtw^{ij} \cup C_{h+1}^{i,j}$
46. if ($I.u \geq s \times P.count$)
47. $Thu^{ij} = Thu^{ij} \cup I$;
48. end
49. return Thu^{ij} ;

Figure 3-4. Incremental procedure of THUI-Mine.

3.3 Experiments and Analysis

To evaluate the performance of *THUI-Mine*, we conducted experiments using synthetic

datasets generated via a randomized dataset generator provided by IBM Quest project [3]. However, the IBM Quest data generator only generates the quantity of 0 or 1 for each item in a transaction. In order to fit databases into the scenario of utility mining, we randomly generate the quantity of each item in each transaction, ranging from 1 to 5, as is similar to the model used in [21]. Utility tables are also synthetically created by assigning a utility value to each item randomly, ranging from 1 to 1000. Because it is observed from real world databases that most items are in the low profit range, we generate the utility values using a log normal distribution, as is similar to the model used in [21]. Figure 3-5 shows the utility value distribution of 1000 items.

The simulation is implemented in C++ and conducted in a machine with 2.4GHz CPU and 1G memory. For comparison with *THUI-Mine* algorithm, the two-Phase algorithm is extended with sliding window scenario. The extended Two-Phase algorithm scans the database according to the set time window and then performs the computation within the time window. This process is repeated over sliding time window for the database. The main performance metric used is execution time. We recorded the execution time of *THUI-Mine* in finding temporal high utility itemsets. The comparison on the number of generated itemsets for *THUI-Mine*, Two-Phase and MEU is presented to shows the performance comparison of *THUI-Mine* and Two-Phase. The results of scale-up experiments are presented to shows the performance comparison of *THUI-Mine* and Two-Phase on another dense dataset.

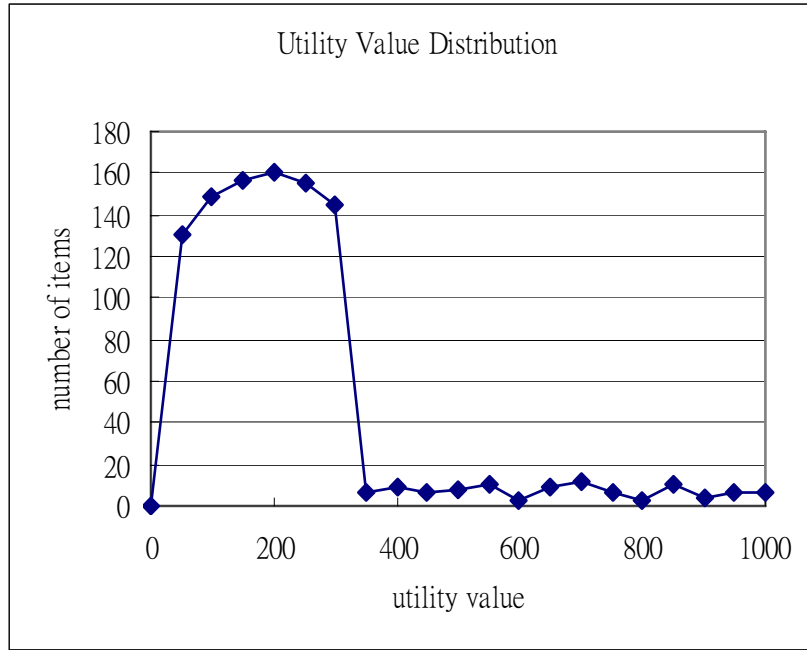
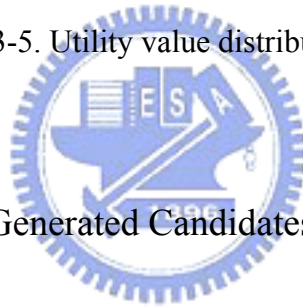


Figure 3-5. Utility value distribution in utility table.



Evaluation on Number of Generated Candidates

In this experiment, we compare the average number of candidates generated in the first database scan on the sliding windows and incremental transaction number d10K with different support values for *THUI-Mine*, Two-Phase [21] and MEU [35]. Without loss of generality, we set $|d| = |\Delta^+| = |\Delta^-|$ for simplicity. Thus, by denoting the original database as $db^{1,n}$ and the new mining database as $db^{i,j}$, we have $|db^{i,j}| = |db^{1,n} - \Delta^- + \Delta^+| = |D|$, where $\Delta^- = db^{1,i-1}$ and $\Delta^+ = db^{n+1,j}$. Table 3-4 and Table 3-5 show the average number of candidates generated by *THUI-Mine*, Two-Phase and MEU on two datasets, respectively. The number of items is set at 1000, and the minimum utility threshold varies from 0.2% to 1%. The experimental results show that the number of candidate itemsets generated by *THUI-Mine* at the first database scan decreases dramatically as the threshold goes up. Especially, when the utility threshold is set as 1%, the number of candidate itemsets is 0 in database

T10.I6.D100K.d10K where T denotes the average size of the transactions and I the average number of frequent itemsets. The default size of the sliding window is set as 30K. In fact, we also varied the size of sliding window and the experimental results show that *THUI-Mine* outperforms Two-Phase algorithm under different sliding windows sizes. Due to space limitation, we only show the representative results with the sliding window size set as 30K. However, the number of candidates generated by Two-Phase is still very large and that for MEU is always 499,500 because it needs to process all combinations of 1000 items. *THUI-Mine* generates far fewer candidates when compared to Two-Phase and MEU.

We obtain similar experimental results for different datasets. For example, only 118 candidate itemsets are generated by *THUI-Mine*, but 183,921 and 499,500 candidate itemsets are generated by Two-Phase and MEU, respectively, when the utility threshold is set as 1% in dataset T20.I6.D100K.d10K. In the case of dataset T20.I6.D100K.d10K, more candidates are generated, because the transaction is longer than that in T10.I6.D100K.d10K. In overall, our algorithm *THUI-Mine* always generates far fewer candidates compared to Two-Phase and MEU for various kinds of databases. Hence, *THUI-Mine* is verified to be very effective in pruning candidate itemsets to find temporal high utility itemsets.

Table 3-4. The number of candidate itemsets generated on database T10.I6.D100K.d10K.

Databases Threshold	T10.I6.D100K.d10K		
	<i>THUI-Mine</i>	Two-Phase	MEU
0.2%	3433	361675	499500
0.3%	666	303810	499500
0.4%	161	258840	499500
0.6%	7	182710	499500
0.8%	1	129286	499500
1%	0	91378	499500

Table 3-5. The number of candidate itemsets generated on database T20.I6.D100K.d10K.

Databases Threshold	T20.I6.D100K.d10K		
	<i>THUI-Mine</i>	Two-Phase	MEU
0.2%	27357	401856	499500
0.3%	11659	371953	499500
0.4%	5389	337431	499500
0.6%	1364	278631	499500
0.8%	371	229503	499500
1%	118	183921	499500

Evaluation of Execution Efficiency

In this experiment, we compare only the relative performance of Two-phase and *THUI-Mine* since MEU spends much higher execution time and becomes incomparable. Figure 3-6 and Figure 3-7 show the execution times for the two algorithms on datasets T20.I6.D100K.d10K and T10.I6.D100K.d10K, respectively, as the minimum utility threshold is decreased from 1% to 0.2%. It is observed that when the minimum utility threshold is high, there are only a limited number of high utility itemsets produced. However, as the minimum utility threshold decreases, the performance difference becomes prominent in that *THUI-Mine* significantly outperforms Two-Phase. As shown in Figure 3-6 and Figure 3-7, *THUI-Mine* leads to prominent performance improvement under different sizes of transaction. Explicitly, *THUI-Mine* is significantly faster than Two-Phase and the margin grows as the minimum utility threshold decreases. For example, *THUI-mine* is 10 times faster than Two-Phase when threshold is 0.2 for T20.I6.D100K.d10K. In overall, *THUI-Mine* spends much less time than Two-Phase with higher stability in finding temporal high utility itemsets. This is because the Two-Phase algorithm produces more candidate itemsets and needs more database scans to

find high utility itemsets than *THUI-Mine*. To measure the improvement on execution time for *THUI-Mine* compared to Two-Phase algorithm, we define the *Improvement Ratio* as follows:

$$\text{Improvement Ratio} = \frac{(\text{execution time of Two - Phase}) - (\text{execution time of THUI - Mine})}{\text{execution time of Two - Phase}}$$

From the data illustrated in Figure 3-6, we see that the Improvement Ratio is about 85.6% with the threshold set as 0.2%. In Figure 3-7, the average improvement is about 67% with minimum utility threshold varied from 0.2% to 1%. Obviously, *THUI-Mine* reduces substantially the time in finding high utility itemsets. Moreover, the high utility itemsets obtained by Two-Phase are not suitable for applications in temporal databases since Two-Phase needs more database scans and increased execution time in finding high utility itemsets. Hence, *THUI-Mine* meets the requirements of high efficiency in terms of execution time for temporal data mining.

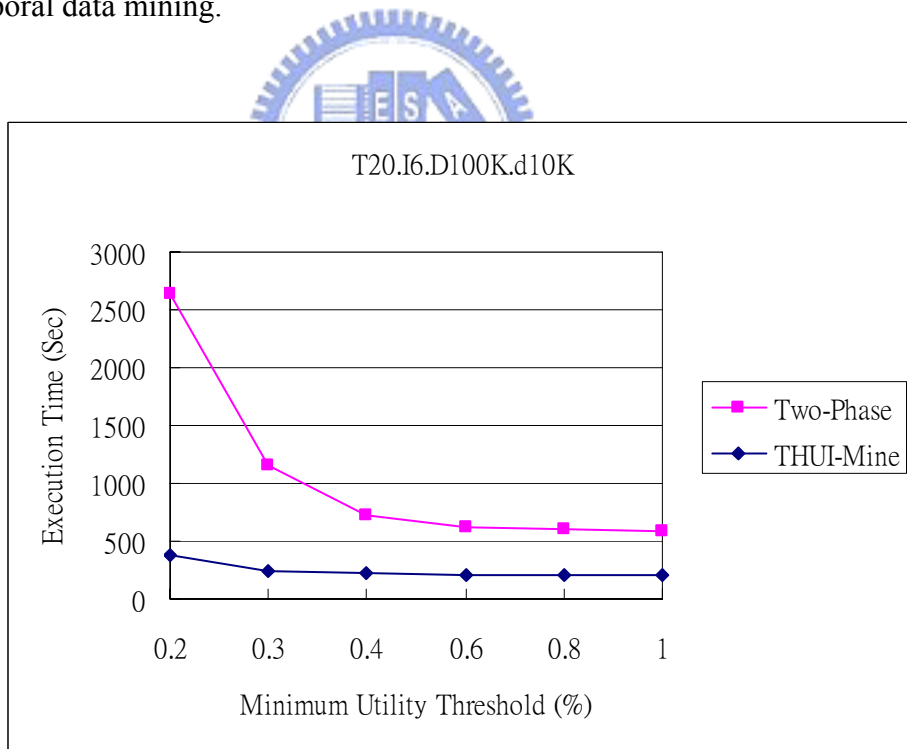


Figure 3-6. Execution time for Two-Phase and THUI on T20.I6.D100K.d10K.

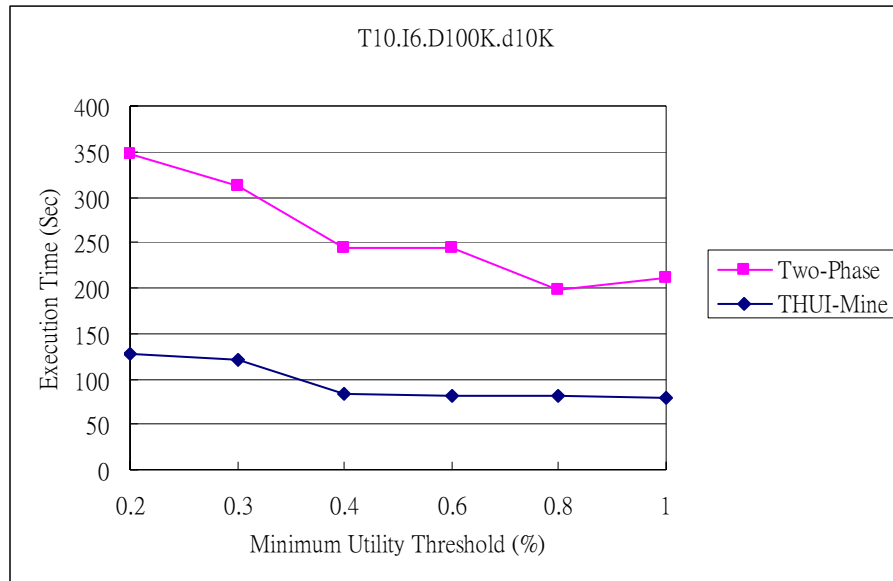


Figure 3-7. Execution time for Two-Phase and THUI on T10.I6.D100K.d10K.

Scale-up on Incremental Mining

In this experiment, we investigate the effects of varying incremental transaction size on the execution time of mining results. To further understand the impact of $|d|$ on the relative performance of *THUI-Mine* and Two-Phase, we conduct scale-up experiments which are similar to those described in [18] with minimum support thresholds being set as 0.2% and 0.4 %, respectively. Figure 3-8 shows the experimental results where the value in y-axis corresponds to the ratio of the execution time of *THUI-Mine* to that of Two-Phase under different values of $|d|$. It can be seen that the execution-time ratio remains stable with the growth of the incremental transaction number $|d|$ since the size of $|d|$ has little influence on the performance of *THUI-Mine*. Moreover, the execution time ratio of the scale-up experiments with minimum support thresholds varied from 0.6% to 1% remains constant at approximately 0.4%. This implies that the advantage of *THUI-Mine* over Two-Phase is stable and less execution time is taken as the amount of incremental portion increases. This result also indicates that *THUI-Mine* is useful for mining temporal databases with large transaction size.

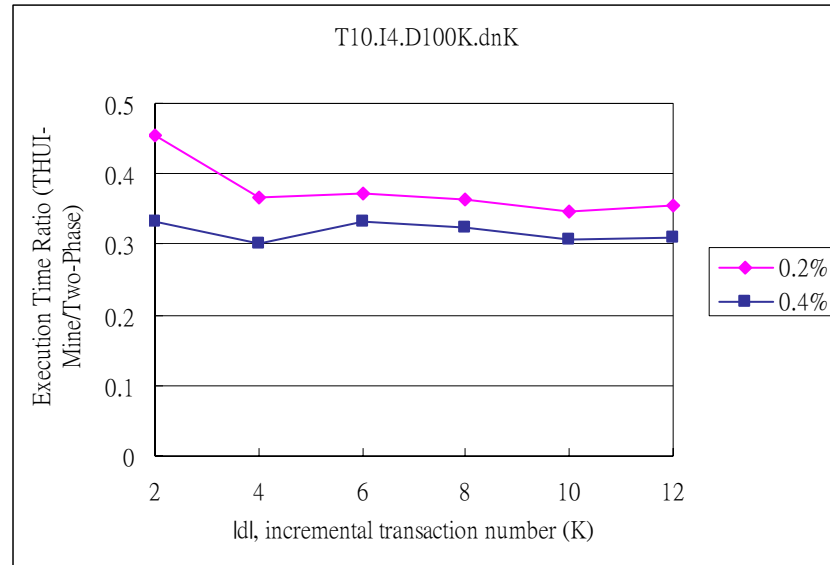


Figure 3-8. Scale-up performance results for THUI vs. Two-Phase.

Evaluation on dense data

Typically, the synthetic data sets are very sparse. For testing various kinds of databases, we evaluate another dense dataset, the gazelle data set as used in [37]. The gazelle data set comes from click-stream data from a dot-com company named Gazelle.com, a legware and legcare retailer. This data set was used in the KDD-Cup 2000 competition and publicly available from www.ecn.purdue.edu/KDDCUP. In order to fit databases into the scenario of utility mining, we also randomly generate the quantity of each item in each transaction, ranging from 1 to 5. The utility tables are also synthetically created by assigning a utility value to each item randomly, ranging from 1 to 1000.

Figure 3-9 shows the execution time for the two algorithms as the minimum utility threshold is varied from 0.1% to 0.02%. It is observed that *THUI-Mine* still spends less time than Two-Phase with higher stability for finding temporal high utility itemsets even under the dense data. This is because the Two-Phase algorithm produces more candidate itemsets and needs more database scans to find high utility itemsets than *THUI-Mine*. Hence, this result

also indicates that *THUI-Mine* is effective for mining temporal high utility itemsets under both of sparse and dense datasets.

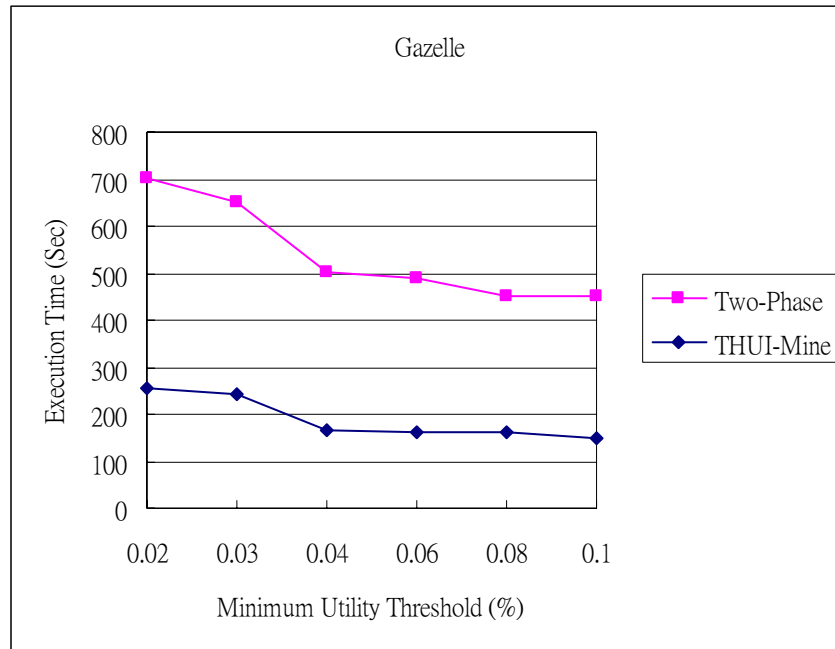


Figure 3-9. Execution time for Two-Phase and THUI on gazelle dataset.

Chapter 4

Mining Temporal Rare Utility Itemsets in Large Databases Using Relative Utility Thresholds

4.1 Problem Definition

The mining of association rules for discerning the relationship between data items in large databases is a well studied technique in the data mining field with representative methods like *Apriori* [1][2]. The problem with mining association rules can be distilled into two steps. The first step involves finding all frequent itemsets (or say large itemsets) in databases. Once the frequent itemsets are found, generating association rules is straightforward and can be accomplished in linear time.

An important research issue that extends from the mining of association rules is the discovery of temporal association patterns in temporal databases due to the wide variety of applications on various domains. Temporal data mining can be defined as the activity of looking for interesting correlations or patterns in large sets of temporal data accumulated for other purposes [6]. For a database with a specified transaction window size, we may use an algorithm like *Apriori* to obtain frequent itemsets from the database. For time-variant temporal databases, there is a strong demand to develop an efficient and effective method to mine various temporal patterns [11][19]. However, most methods designed for traditional databases cannot be directly applied to the mining of temporal patterns in temporal databases because of their high complexity.

In many applications, we would like to mine temporal association patterns in temporal databases for a specified amount of the most recent data. That is, in the temporal data mining,

one has to not only include new data (i.e., data created in the new hour) into, but also remove the old data (i.e., data created in the most obsolete hour) from the mining process. It can be seen that it is intrinsically difficult to conduct frequent pattern identification due to the constraints of limited time and space. Furthermore, it takes considerable time to find temporal frequent itemsets in different time windows. However, the frequency of an itemset may not be a sufficient indicator of interest, because it only reflects the number of transactions in the database that contain the itemset. It does not reveal the utility of an itemset, which can be measured in terms of cost, profit, or other expressions of user preference.

On the other hand, frequent itemsets may only contribute a small portion of the overall profit, whereas non-frequent itemsets may contribute a large portion of the profit. In reality, a retail business may be interested in identifying its most valuable customers (customers who contribute a major fraction of the profits to the company). Hence, frequency is not sufficient to answer questions such as whether an itemset is highly profitable, or whether an itemset has a strong impact. Utility mining is thus useful in a wide range of practical applications and has been the subject of several recent studies [8][21][32][35].

In the existing mechanisms for mining high utility itemsets, the utility threshold is unique throughout the whole database, and they assume that each data included in the database occurs over a similar frequency. In reality, however, the data composing the database may occur either relatively frequently or not, according to the characteristics of the database. In addition, the rarely occurring data in the database may be significant enough to be of good use. Nevertheless, the existing high utility itemsets discovery techniques discover the high utility itemsets using the same utility threshold over the whole range of data, so the discovered high utility itemsets with regard to rare data may be redundant, and as a result unnecessary rules may be generated.

Table 4-2. A transaction database and its utility table.

(a) Transaction table

(b) The utility table


			ITEM						
			TID	A	B	C	D	E	
db ^{1,3}	Δ^-	P ₁	T ₁	0	0	26	0	1	db ^{2,4}
			T ₂	0	6	0	1	1	
			T ₃	12	0	0	1	0	
	D ⁻	P ₂	T ₄	0	1	0	7	0	
			T ₅	0	0	12	0	2	
			T ₆	1	4	0	0	1	
		P ₃	T ₇	0	10	0	0	1	
			T ₈	1	0	1	3	1	
			T ₉	1	1	27	0	0	
	Δ^+	P ₄	T ₁₀	0	6	2	0	0	
			T ₁₁	0	3	0	2	0	
			T ₁₂	0	2	1	0	0	

ITEM	PROFIT\$(per unit)
A	3
B	10
C	1
D	6
E	5

Recently, a *utility mining* model was defined in [35]. Utility is a measure of how “useful” (i. e. “profitable”) an itemset is. The definition of utility of an itemset X , $u(X)$, is the sum of the utilities of X in all the transactions containing X . The goal of utility mining is to identify high utility itemsets which drive a large portion of the total utility. Traditional association rules mining models assume that the utility of each item is always 1 and the sales quantity is either 0 or 1; thus it is only a special case of utility mining where the utility or the sales quantity of each item could be any number. If $u(X)$ is greater than a utility threshold, X is a high utility itemset. Otherwise, it is a low utility itemset. Table 4-1 is an example of utility mining in a transaction database. The number in each transaction in Table 4-1(a) is the sales volume of each item, and the utility of each item is listed in Table 4-1(b). For example, $u(\{B, D\}) = (6 \times 10 + 1 \times 6) + (1 \times 10 + 7 \times 6) + (3 \times 10 + 2 \times 6) = 160$. $\{B, D\}$ is a high utility itemset if the

utility threshold is set at 130.

However, a high utility itemset may consist of some low utility items. Another approach is to adopt the level-wise searching schema that exists in fast algorithms, such as Apriori [3]. However, this algorithm doesn't apply to the utility mining model. For example, $u(D) = 84 < 130$, D is a low utility item, but its superset $\{B, D\}$ is a high utility itemset. If Apriori is used to find high utility itemsets, all the combinations of all the items must be generated. Moreover, to discover a long pattern, the number of candidates is prohibitively large. The cost of either computation time or memory is intolerable, regardless of which method of implementation is applied. The challenge of utility mining is not only in restricting the size of the candidate set but also in simplifying the computation for calculating the utility. Another challenge of utility mining is how to find temporal significant rare utility itemsets from temporal databases as time advances.

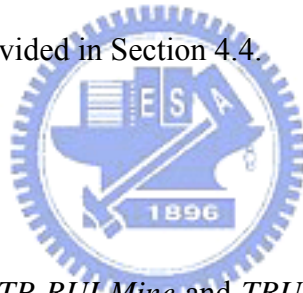


In this chapter, in addition to the preceding techniques, we not only study a technique to discover the association rules that describe the associations among data, but also suggest a high utility itemsets discovery technique that enables us to identify significant rare itemsets associated with specific data in a way that the rare data occur simultaneously with the specific data more frequently than the average co-occurrence frequency in the database. This motivates our research in developing a new scheme for identifying *temporal rare utility itemsets (TRUI)* from temporal databases. Therefore, we propose a novel method that can identify rare utility itemsets that co-occur in relatively high association with certain specific items. We adopt a relative utility threshold, a rate against the relative frequency of the data existing in a database, and explore the issue of efficiently mining rare utility itemsets in temporal databases and we will propose two algorithms named *TP-RUI-Mine* and *TRUI-Mine* that can identify the association rules that include information about the significant association among even rare data. The novel contribution of *TRUI-Mine* is particular in that it can efficiently identify the utility itemsets in temporal databases so that the execution time for

mining rare utility itemsets can be substantially reduced. In this way, the process under all time windows of temporal databases can be achieved effectively under limited memory space, less candidate itemsets and CPU I/O. This meets the critical requirements of time and space efficiency for mining temporal databases. Through experimental evaluation, *TRUI-Mine* is shown to produce fewer candidate itemsets in finding the temporal rare utility itemsets, so it outperforms the other algorithm *TP-RUI-Mine*, also proposed by us, in terms of execution efficiency. To our best knowledge, this is the first work on mining temporal rare utility itemsets from temporal databases.

The rest of this chapter is organized as follows: Section 4.2 describes the proposed approaches, *TP-RUI-Mine* and *TRUI-Mine*, for finding the temporal rare utility itemsets. In section 4.3, we describe the experimental results for evaluating the proposed methods. The conclusion of the chapter is provided in Section 4.4.

4.2 Proposed Methods



In this section, we present the *TP-RUI-Mine* and *TRUI-Mine* methods and describe the basic concept of *TP-RUI-Mine* and *TRUI-Mine*. Then we give an example of mining temporal high utility itemsets. Finally, the procedure of the *TRUI-Mine* algorithm is provided in the last paragraph of the section.

The goal of our algorithms is to discover temporal rare itemsets from temporal databases. The concept consists of utility mining and significantly rare itemsets. We describe the basic concept of utility mining and significantly rare itemset as follows.

Basic Concept of Utility Mining

The goal of utility mining is to discover all the itemsets whose utility values are beyond a user specified threshold in a transaction database. In [35] the goal of utility mining is to find all the

high utility itemsets. An itemset X is a *high utility itemset* if $u(X) \geq \epsilon$, where $X \subseteq I$ and ϵ is the minimum utility threshold, otherwise, it is a *low utility itemset*. For example, in Table 4-1, $u(A, T_8) = 1 \times 3 = 3$, $u(\{A, C\}, T_8) = u(A, T_8) + u(C, T_8) = 1 \times 3 + 1 \times 1 = 4$, and $u(\{A, C\}) = u(\{A, C\}, T_8) + u(\{A, C\}, T_9) = 4 + 30 = 34$. If $\epsilon = 130$, $\{A, C\}$ is a low utility itemset. However, if an item is a low utility item, its superset may be a high utility itemset. For example, $u(D) = 84 < 130$, D is a low utility item, but its superset $\{B, D\}$ is a high utility itemset because of $u(\{B, D\}) = 160 > 130$. Hence, all the combinations of all items should be processed so that it never loses any high utility itemset. However the cost of either computation time or memory is intolerable.

Liu *et al.* [21] proposed the Two-Phase algorithm for pruning candidate itemsets and simplifying the calculation of utility. First, Phase I overestimates some low utility itemsets, but it never underestimates any itemsets. For the example in Table 4-1, the *transaction utility of transaction T_q* , denoted as $tu(T_q)$, is the sum of the utilities of all items in T_q : $tu(T_q) = \sum_{i_p \in T_q} u(i_p, T_q)$. And the *transaction-weighted utilization of an itemset X* , denoted as $twu(X)$, is the sum of the transaction utilities of all the transactions containing X : $twu(X) = \sum_{X \subseteq T_q \in D} tu(T_q)$. For example, $twu(A) = tu(T_3) + tu(T_6) + tu(T_8) + tu(T_9) = 42 + 48 + 27 + 40 = 157$ and $twu(\{D, E\}) = tu(T_2) + tu(T_8) = 71 + 27 = 98$. In fact, $u(A) = u(\{A\}, T_3) + u(\{A\}, T_6) + u(\{A\}, T_8) + u(\{A\}, T_9) = 36 + 3 + 3 + 3 = 45$ and $u(\{D, E\}) = u(\{D, E\}, T_2) + u(\{D, E\}, T_8) = 11 + 23 = 34$. So Phase I overestimates some low utility itemsets, but it never underestimates any itemsets. Table 4-2 gives the transaction utility for each transaction in Table 4-1. Second, one extra database scan is performed to filter the overestimated itemsets in Phase II. For example, $twu(A) = 157 > 130$ but $u(A) = 45 < 130$. Then item $\{A\}$ is pruned. Otherwise, it is a high utility itemset. Finally, all of the high utility itemsets are identified in this way.

Table 4-2. Transaction utility of the database.

TID	Transaction Utility	TID	Transaction Utility
T ₁	31	T ₇	105
T ₂	71	T ₈	27
T ₃	42	T ₉	40
T ₄	52	T ₁₀	62
T ₅	22	T ₁₁	42
T ₆	48	T ₁₂	21

Basic Concept of Significant Rare Utility Itemsets

In this chapter, we use RUT (Relative Utility Threshold) which identifies the association rules containing the significantly rare itemsets that have high confidence with regard to specific data. A significantly rare itemset is one in which its frequency in the database does not satisfy the utility threshold but appears associated with the specific data in high proportion to its frequency. To identify significantly rare itemsets in the existing high utility itemsets discovery algorithms such as Two-Phase algorithm [21], we should set the utility threshold, generate high utility itemsets of which the members satisfy the utility threshold, and apply the specified confidence to all rules that can be produced by the high utility itemsets. However in some cases, these significantly rare itemsets are not discovered during the actual computation of the high utility itemsets. For example, data items a, b and c exist in the database, where each of a, b and c has support of 25%, 35% and 30% respectively in the database and the user has set the minimum utility threshold to 35%. Then a and c cannot be the members of the high utility itemset since they do not satisfy the minimum utility threshold. However, the itemset {a, b, c} may have support of 23%, and 90% of a's occurrences may come together with b and c. In the existing discovery methods, the itemset {a, b, c} is not discovered because it does not satisfy the minimum utility threshold of 35%.

To discover such significantly rare itemsets that are rarely discovered using the existing utility mining methods, our algorithms set and utilize two minimum utility thresholds. The two utility thresholds are defined as the first utility threshold and the second utility threshold. Both of the utility thresholds are defined as follows:

Definition 4.1. 1st utility threshold: Critical value of the user-specified utility threshold used in the process of high utility itemsets discovery.

Definition 4.2. 2nd utility threshold: Critical value of the user-specified utility threshold used in the process of rare utility itemsets discovery.

The first utility threshold and the second utility threshold are set so that the condition “1st utility threshold > 2nd utility threshold” is satisfied. In addition to the utility threshold, our algorithms use the relative utility threshold (RUT) that considers relative frequency between the data. RUT is one that measures the rare itemset satisfying the second utility threshold but not the first utility threshold. Using the RUT, we identify the significantly rare itemset. RUT is defined as follows:

Definition 4.3. Relative Utility Threshold (RUT): $RUT(i_1, i_2, \dots, i_k) = \max(\text{threshold}(i_1, i_2, \dots, i_k)/\text{threshold}(i_1), \text{threshold}(i_1, i_2, \dots, i_k)/\text{threshold}(i_2), \dots, \text{threshold}(i_1, i_2, \dots, i_k)/\text{threshold}(i_k))$

RUT is between 0 and 1, and is determined by selecting the largest one among the confidence values for the candidate itemset against each data item. A high value RUT implies that the user selects the items in which the percentage of the co-occurrence is high.

If we define RUT and discover the high utility itemsets using the utility threshold, we are able to discover the high utility itemsets in which the different frequencies of items are reflected. For example, it is less frequent for consumers to buy food processors or cooking pans in a supermarket than to buy bread or milk, but the former transactions are more profitable. When applying the existing methods that use only single utility threshold, we should set the utility threshold lower to discover the association with regard to food processors or cooking pans, and thus numerous unnecessary utility itemsets satisfying the low

utility threshold are produced. By using RUT, we can discover rare utility itemsets and prevent the generation of unnecessary utility itemsets.

Our algorithms *TP-RUI-Mine* and *TRUI-Mine* are based on the principle of the Two-Phase algorithm [21] and THUI-Mine [32], and we combine these with the concept of the significantly rare itemset and focus on utilizing incremental methods to improve the response time with fewer candidate itemsets and CPU I/O. In essence, by partitioning a transaction database into several partitions from temporal databases, algorithm *TRUI-Mine* employs a filtering threshold in each partition to deal with the transaction-weighted utilization itemsets generated. The cumulative information in the prior phases is selectively carried over toward the generation of transaction-weighted utilization itemsets in the subsequent phases by *TRUI-Mine*. In the processing of a partition, a progressive transaction-weighted utilization set of itemsets is generated by *TRUI-Mine*. Explicitly, a progressive transaction-weighted utilization set of itemsets is composed of the following two types of transaction-weighted utilization itemsets: (1) the transaction-weighted utilization itemsets that were carried over from the previous progressive candidate set in the previous phase and remain as transaction-weighted utilization itemsets after the current partition is taken into consideration; and (2) the transaction-weighted utilization itemsets that were not in the progressive candidate set in the previous phase but are newly selected after only taking the current data partition into account. As such, after the processing of a phase, algorithm *TRUI-Mine* outputs a cumulative filter, denoted by CF, which consists of a progressive transaction-weighted utilization set of itemsets, their occurrence counts and the corresponding partial utility threshold required. Then temporal rare utility itemsets could be generated by RUT. With these design considerations, algorithm *TRUI-Mine* is shown to have very good performance for mining temporal rare utility itemsets from temporal databases. Although another algorithm *TP-RUI-Mine* is proposed by us and based on the principle of Two-Phase algorithm [21] and uses the same concept and processes with the part of generating temporal rare utility itemsets of *TRUI-Mine*.

However, *TP-RUI-Mine* would generate too many candidate itemsets compared to *TRUI-Mine* because of the principle of Two-Phase algorithm [21]. We found that *TRUI-Mine* is a more efficient algorithm than *TP-RUI-Mine* according to both theory and experimental results. Hence, we only show the processes of *TRUI-Mine* in detail.

An Example for Mining Temporal Rare Utility Itemsets

The proposed *TRUI-Mine* algorithm can be best understood by the illustrative transaction database in Table 4-1 and Figure 4-1 where a scenario of generating high utility itemsets from temporal databases for mining temporal rare utility itemsets is given. We set the first utility threshold at 130 and second utility threshold at 90 in nine transactions. According to the characteristics of the procedure of utility mining, we should set second utility threshold to be the same as the initial threshold so as to filter utility itemsets. If we set the first utility threshold to be the initial threshold, we might lose some utility itemsets that could be rare utility itemsets. In addition, we set $RUT=0.6$ to find temporal rare utility itemsets. In fact, our algorithm *TRUI-Mine* not only could discover temporal high utility itemsets but also temporal rare utility itemsets. Without loss of generality, the temporal mining problem can be divided into two procedures:

1. Preprocessing procedure: This procedure deals with mining on the original transaction database.
2. Incremental procedure: The procedure deals with the update of the high utility itemsets and rare utility itemsets from temporal databases.

P ₁			P ₂			P ₃		
C ₂	start	transaction-weighted utility	C ₂	start	transaction-weighted utility	C ₂	start	transaction-weighted utility
AB	1	0	⊙ AB	2	48	⊙ AB	2	88
⊙ AD	1	42	AD	1	42	⊙ AC	3	67
AE	1	0	⊙ AE	2	48	AD	3	27
⊙ BD	1	71	⊙ BD	1	123	⊙ AE	2	75
⊙ BE	1	71	⊙ BE	1	119	⊙ BC	3	40
⊙ DE	1	71	⊙ DE	1	71	⊙ BD	1	123
⊙ BE			⊙ BE			⊙ BE	1	224
⊙ DE			⊙ DE			⊙ CE	3	27
⊙ DE			⊙ DE			⊙ DE	1	98

db ^{1,3} - Δ ⁻ = D ⁻			D ⁻ + Δ ⁺ = db ^{2,4}		
C ₂	start	transaction-weighted utility	C ₂	start	transaction-weighted utility
⊙ AB	2	88	AB	2	88
⊙ AC	3	67	AC	3	67
⊙ AE	2	75	⊙ AE	2	75
⊙ BC	3	40	⊙ BC	3	123
BD	2	52	⊙ BD	4	42
⊙ BE	2	153	⊙ BE	2	153
DE	2	27	CD	4	0

Figure 4-1. Temporal rare utility itemsets generated by TRUI-Mine.

The preprocessing procedure is only utilized for the initial utility mining in the original database, e.g., $db^{1,n}$. For the generation of mining high utility itemsets and rare utility itemsets in $db^{2,n+1}$, $db^{3,n+2}$, $db^{i,j}$, and so on, the incremental procedure is employed. Consider the database in Table 4-1. Assume that the original transaction database $db^{1,3}$ is segmented into three partitions, i.e., $\{P_1, P_2, P_3\}$, in the preprocessing procedure. Each partition is scanned sequentially for the generation of candidate 2-itemsets in the first scan of the database $db^{1,3}$. After scanning the first segment of 3 transactions, i.e., partition P_1 , 2-itemsets $\{AB, AD, AE, BD, BE, DE\}$ are generated as shown in Figure 4-1. In addition, each potential candidate itemset $c \in C_2$ has two attributes: (1) $c.start$ which contains the identity of the starting partition when c was added to C_2 ; and (2) transaction-weighted utility which is the sum of the transaction utilities of all the transactions containing c since c was added to C_2 . Since there are three partitions, the second utility threshold of each partition is $90 / 3 = 30$. Such a partial utility threshold is called the “filtering threshold” in this chapter. Itemsets whose transaction-weighted utility are below the filtering threshold are removed. Then, as shown in Figure 4-1, only $\{AD, BD, BE, DE\}$, marked by “⊙”, remain as temporal high transaction-weighted utilization 2-itemsets whose information is then carried over to the next

phase of processing. Similarly, after scanning partition P_2 , the temporal high transaction-weighted utilization 2-itemsets are recorded.

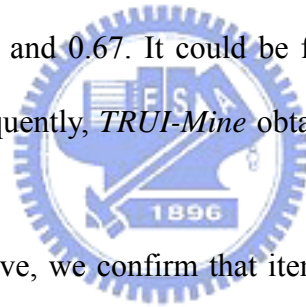
From Figure 4-1, it is noted that since there are also 3 transactions in P_2 , the filtering threshold of those itemsets carried out from the previous phase is $30 + 30 = 60$, and that of newly identified candidate itemsets is 30. It can be seen from Figure 4-1 that we have 5 temporal high transaction-weighted utilization 2-itemsets in C_2 after the processing of partition P_2 , and 3 of them are carried from P_1 to P_2 and 2 of them are newly identified in P_2 . Note that though appearing in the previous phase P_1 , itemset $\{AD\}$ is removed from temporal high transaction-weighted utilization 2-itemsets once P_2 is taken into account since its transaction-weighted utility does not meet the filtering threshold (i.e., $42 < 60$). Finally, partition P_3 is processed by algorithm *TRUI-Mine*. The resulting temporal high transaction-weighted utilization 2-itemsets are $\{AB, AC, AE, BC, BD, BE, DE\}$ as shown in Figure 4-1. After the processing of partition P_3 , we do have two new itemsets, i.e., AC and BC, which join the C_2 as temporal high transaction-weighted utilization 2-itemsets. Consequently, we have 7 temporal high transaction-weighted utilization 2-itemsets generated by *TRUI-Mine*, and 3 of them are carried from P_1 to P_3 , while 2 of them are carried from P_2 to P_3 and 2 of them are newly identified in P_3 . After processing P_1 to P_3 , those temporal high transaction-weighted utilization itemsets in $db^{1,3}$ are $\{A, B, C, D, E, AB, AC, AE, BC, BD, BE, DE\}$.

After generating temporal high transaction-weighted utilization 2-itemsets from the first scan of database $db^{1,3}$, we employ the scan reduction technique and use temporal high transaction-weighted utilization 2-itemsets to generate C_k ($k = 3, 4, \dots, n$), where C_n is the candidate last itemset. It can be verified that temporal high transaction-weighted utilization 2-itemsets generated by *TRUI-Mine* can be used to generate the candidate 3-itemsets. Clearly, a C_3 can be generated from temporal high transaction-weighted utilization 2-itemsets. For example, 3-candidate itemsets $\{ABC\}$, $\{ABE\}$ and $\{BDE\}$ are generated from temporal high

transaction-weighted utilization 2-itemsets $\{AB, AC, BC\}$, $\{AB, AE, BE,\}$ and $\{BD, BE, DE\}$ in $db^{1,3}$. Similarly, all C_k can be stored in main memory, and we can find temporal high utility itemsets together by first utility threshold and temporal rare candidate itemsets between first utility threshold and second utility threshold when the second scan of the database $db^{1,3}$ is performed. Thus, only two scans of the original database $db^{1,3}$ are required in the preprocessing step. The resulting temporal high utility itemsets are $\{B\}$ and $\{BE\}$ because $u(B) = 330 > 130$ and $u(\{B, E\}) = 215 > 130$. In addition, the temporal rare candidate itemset is $\{BD\}$ because $u(\{B, D\}) = 118$ between 90 (second utility threshold) and 130 (first utility threshold). The individual relative utility thresholds of $\{B, D\}$ are $\{B, D\}/\{B\} = 2/5 = 0.4$ and $\{B, D\}/\{D\} = 2/4 = 0.5$. So the maximum relative utility threshold of $\{B, D\}$ is 0.5. However, $RUT_{(B, D)} = 0.5 < 0.6$. Hence, there is no temporal rare utility itemset that could be found in the database $db^{1,3}$.

One important merit of *TRUI-Mine* lies in its incremental procedure. As depicted in Figure 4-1, the mining database will be moved from $db^{1,3}$ to $db^{2,4}$. Thus, some transactions, i.e., $T_1, T_2,$ and T_3 , are deleted from the mining database and other transactions, i.e., $T_{10}, T_{11},$ and T_{12} , are added. To illustrate more clearly, this incremental step can also be divided into three sub-steps: (1) generating temporal high transaction-weighted utilization 2-itemsets in $D^- = db^{1,3} - \Delta^-$, (2) generating temporal high transaction-weighted utilization 2-itemsets in $db^{2,4} = D^- + \Delta^+$ and (3) scanning the database $db^{2,4}$ only once for the generation of all temporal high utility itemsets and temporal rare utility itemsets. In the first sub-step, $db^{1,3} - \Delta^- = D^-$, we check the pruned partition P_1 , and reduce the value of transaction-weighted utility and set $c.start = 2$ for those temporal transaction-weighted utilization 2-itemsets where $c.start = 1$. It can be seen that itemsets $\{BD, DE\}$ were removed. Next, in the second sub-step, we scan the incremental transactions in P_4 . The process in $D^- + \Delta^+ = db^{2,4}$ is similar to the operation of scanning partitions, e.g., P_2 , in the preprocessing step. The new itemset $\{BD\}$ joins the temporal high transaction-weighted utilization 2-itemsets after the scan of P_4 . In the third

sub-step, we use temporal high transaction-weighted utilization 2-itemsets to generate C_k as mentioned above. Finally, those temporal high transaction-weighted utilization itemsets in $db^{2,4}$ are $\{A, B, C, D, E, AE, BC, BD, BE\}$. Note that instead of 10 2-candidate itemsets that would be generated if *TP-RUI-Mine* were used, only 4 temporal high transaction-weighted utilization 2-itemsets are generated by *TRUI-Mine*. By scanning $db_{2,4}$ only once, *TRUI-Mine* obtains temporal high utility itemsets $\{B, BE\}$ in $db^{2,4}$ because $u(B) = 270 > 130$ and $u(\{B, E\}) = 150 > 130$. In addition, the temporal rare candidate itemset are $\{BC\}$ and $\{BD\}$ because $u(\{B, C\}) = 120$ and $u(\{B, D\}) = 94$ between 90 (second utility threshold) and 130 (first utility threshold). The individual relative utility thresholds of $\{B, C\}$ are $\{B, C\}/\{B\} = 3/7 = 0.42$ and $\{B, C\}/\{C\} = 3/5 = 0.6$. The individual relative utility thresholds of $\{B, D\}$ are $\{B, D\}/\{B\} = 2/7 = 0.28$ and $\{B, D\}/\{D\} = 2/3 = 0.67$. So the maximum relative utility thresholds of $\{B, C\}$ and $\{B, D\}$ are 0.6 and 0.67. It could be found that $RUT_{(B,C)} = 0.6 \geq 0.6$ and $RUT_{(B,D)} = 0.67 > 0.6$. Consequently, *TRUI-Mine* obtains temporal rare utility itemsets $\{BC, BD\}$ in the database $db^{2,4}$.



Through the example above, we confirm that items C and D, though they are rare data items not satisfying the first utility threshold, always occur simultaneously with item B; and the algorithm *TRUI-Mine* can discover the temporal rare utility itemsets that are not included in the temporal high utility itemsets but still significant in terms of the relative utility threshold. In addition, our algorithm *TRUI-Mine* not only could discover temporal high utility itemsets but also temporal rare utility itemsets.

TRUI-Mine Algorithm

For easier illustration, the meanings of various symbols used are given in Table 4-3. The preprocessing procedure and the incremental procedure of algorithm *TRUI-Mine* are described as follows.

Preprocessing procedure of TRUI-Mine

The preprocessing procedure of Algorithm *TRUI-Mine* is shown in Figure 4-2. Initially, the database $db^{1,n}$ is partitioned into n partitions by executing the preprocessing procedure (in Step 2), and CF, the cumulative filter, is empty (in Step 3). Let $Thtw^{1,n}$ be the set of progressive temporal high transaction-weighted utilization 2-itemsets of $db^{i,j}$. Algorithm *TRUI-Mine* only records $Thtw^{1,n}$ which is generated by the preprocessing procedure to be used by the incremental procedure. From Step 4 to Step 16, the algorithm processes one partition at a time for all partitions. When partition P_i is processed, each potential candidate 2-itemset is read and saved to CF. The transaction-weight utility of an itemset I and its starting partition are recorded in $I.twu$ and $I.start$, respectively. An itemset, whose $I.twu \geq s$, will be kept in CF. Next, we select $Thtw^{1,n}$ from I where $I \in CF$ and keep $I.twu$ in main memory for the subsequent incremental procedure. By employing the scan reduction technique from Step 19 to Step 26, $C_h^{1,n}$ ($h \geq 3$) are generated in main memory. After refreshing $I.count = 0$ where $I.twu = 0$ and where $I \in Thtw^{1,n}$, we begin the last scan of the database for the preprocessing procedure from Step 28 to Step 31. Finally, those itemsets satisfying the constraint that $I.u \geq s \times P.count$ and $I.RUT \geq RUT$ are finally obtained as the temporal high utility itemsets and temporal rare utility itemsets.

Table 4-3. Meanings of symbols used.

$db^{i,j}$	Partitioned_database (D) from P_i to P_j
s	Second utility threshold in one partition
F	First utility threshold
RUT	Relative utility threshold
$ P_k $	Number of transactions in partition P_k
$TUP_k(I)$	Transactions in P_k that contain itemset I with transaction utility
$UP_k(I)$	Transactions in P_k that contain itemset I with utility
$ db^{1,n}(I) $	Transactions number in $db^{1,n}$ that contain itemset I
$C^{i,j}$	The progressive candidate sets of $db^{i,j}$
$Thtw^{i,j}$	The progressive temporal high transaction-weighted utilization 2-itemsets of $db^{i,j}$
$Thu^{i,j}$	The progressive temporal high utility itemsets of $db^{i,j}$
$Tru^{i,j}$	The progressive temporal rare utility itemsets of $db^{i,j}$
Δ^-	The deleted portion of an ongoing database
D^-	The unchanged portion of an ongoing database
Δ^+	The added portion of an ongoing database

```

1. n = Number of partitions;
2.  $|db^{1,n}| = \sum_{k=1,n} |P_k|$ ;
3.  $CF = \phi$ ;
4. begin for k = 1 to n // 1st scan of  $db^{1,n}$ 
5.   begin for each 2-itemset  $I \in P_k$ 
6.     if ( $I \notin CF$ )
7.       I.twu = TUPk(I);
8.       I.start = k;
9.       if (I.twu  $\geq$  s $\times$ P.count) // P.count is number of partitions
10.        CF = CF  $\cup$  I;
11.     if ( $I \in CF$ )
12.       I.twu = I.twu + TUPk(I);
13.       if (I.twu < s $\times$ P.count)
14.        CF = CF - I;
15.   end
16. end
17. select Thtw1,n from I where  $I \in CF$ ;
18. keep Thtw1,n in main memory;
19. h = 2;
20. begin while (Thtw1,n  $\neq \phi$ ) //Database scan reduction
21.   if (h=2)
22.      $C_{h+1}^{1,n} = Thtw^{1,n} * Thtw^{1,n}$ ;
23.   else
24.      $C_{h+1}^{1,n} = C_h^{1,n} * C_h^{1,n}$ 
25.   h = h + 1;
26. end
27. refresh I.twu = 0 where  $I \in Thtw^{1,n}$ ;
28. begin for k = 1 to n //2nd scan of  $db^{1,n}$ 
29.   for each itemset  $I \in Thtw^{1,n} \cup C_{h+1}^{1,n}$ 
30.     I.u = I.u + UPk(I);
31. end
32. for each itemset  $I \in Thtw^{1,n} \cup C_{h+1}^{1,n}$ 
33.   if (I.u  $\geq$  F)
34.     Thu1,n = Thu1,n  $\cup$  I;
35.   else if (F > I.u  $\geq$  s $\times$ P.count)
36.     I.RUT = max (threshold( $i_1, i_2, \dots, i_k$ )/threshold( $i_1$ ),
37.                  threshold( $i_1, i_2, \dots, i_k$ )/threshold( $i_2$ ),
38.                  ...,
39.                  threshold( $i_1, i_2, \dots, i_k$ )/threshold( $i_k$ ))
40.     if (I.RUT  $\geq$  RUT)
41.       Tru1,n = Tru1,n  $\cup$  I;
42. end
43. return Thu1,n and Tru1,n;

```

Figure 4-2. Preprocessing procedure of TRUI-Mine.

Incremental procedure of TRUI-Mine

As shown in Table 4-3, D^- indicates the unchanged portion of an ongoing transaction database.

The deleted and added portions of an ongoing transaction database are denoted by Δ^- and Δ^+ , respectively. It is worth mentioning that the sizes of Δ^+ and Δ^- , i.e., $|\Delta^+|$ and $|\Delta^-|$ respectively, are not required to be the same. The incremental procedure of *TRUI-Mine* is devised to maintain temporal high utility itemsets efficiently and effectively. This procedure is shown in Figure 4-3. As mentioned before, this incremental step can also be divided into three sub-steps: (1) generating temporal high transaction-weighted utilization 2-itemsets in $D^- = db^{1,3} - \Delta^-$; (2) generating temporal high transaction-weighted utilization 2-itemsets in $db^{2,4} = D^- + \Delta^+$; and (3) scanning the database $db^{2,4}$ only once for the generation of all temporal high utility itemsets. Initially, after some update activities, old transactions Δ^- are removed from the database $db^{m,n}$ and new transactions Δ^+ are added (in Step 6). Note that $\Delta^- \subset db^{m,n}$. Then the updated database is denoted as $db^{i,j}$. Note that $db^{i,j} = db^{m,n} - \Delta^- + \Delta^+$. We denote the unchanged transactions by $D^- = db^{m,n} - \Delta^- = db^{i,j} - \Delta^+$. After loading $Thtw^{m,n}$ of $db^{m,n}$ into CF where $I \in Thtw^{m,n}$, we start the first sub-step, i.e., generating temporal high transaction-weighted utilization 2-itemsets in $D^- = db^{m,n} - \Delta^-$. This sub-step reverses the cumulative processing which is described in the preprocessing procedure. From Step 8 to Step 16, we prune the occurrences of an itemset I , which appeared before partition P_i , by deleting the value $I.twu$ where $I \in CF$ and $I.start < i$. Next, from Step 17 to Step 39, similarly to the cumulative processing in Section 3.3.1, the second sub-step generates temporal high transaction-weighted utilization 2-itemsets in $db^{i,j} = D^- + \Delta^+$ and employs the scan reduction technique to generate $C_{h+1}^{i,j}$. Finally, to generate temporal high utility itemsets and temporal rare utility itemsets, i.e., $Thu^{i,j}$ and $Tru^{i,j}$ in the updated database, we scan $db^{i,j}$ only once in the incremental procedure to find temporal high utility itemsets and temporal rare utility itemsets. Note that $Thtw^{i,j}$ is kept in main memory for the next generation of incremental mining.

```

1. Original database =  $db^{m,n}$ ;
2. New database =  $db^{i,j}$ ;
3. Database removed  $\Delta^- = \sum_{k=m,j-1} P_k$ ;
4. Database added  $\Delta^+ = \sum_{k=n+1,j} P_k$ ;
5.  $D^- = \sum_{k=i,n} P_k$ ;
6.  $db^{i,j} = db^{m,n} - \Delta^- + \Delta^+$ ;
7. loading  $Thtw^{m,n}$  of  $db^{m,n}$  into CF where  $I \in Thtw^{m,n}$ ;
8. begin for  $k = m$  to  $i - 1$  // one scan of  $\Delta^-$ 
9.     begin for each 2-itemset  $I \in P_k$ 
10.        if ( $I \in CF$  and  $I.start \leq k$ )
11.             $I.twu = I.twu - TUP_k(I)$ ;
12.             $I.start = k + 1$ ;
13.            if ( $I.twu < s \times P.count$ )
14.                 $CF = CF - I$ ;
15.        end
16.    end
17. begin for  $k = n + 1$  to  $j$  // one scan of  $\Delta^+$ 
18.     begin for each 2-itemset  $I \in P_k$ 
19.        if ( $I \notin CF$ )
20.             $I.twu = TUP_k(I)$ ;
21.             $I.start = k$ ;
22.            if ( $I.twu \geq s \times P.count$ )
23.                 $CF = CF \cup I$ ;
24.            if ( $I \in CF$ )
25.                 $I.twu = I.twu + TUP_k(I)$ ;
26.                if ( $I.twu < s \times P.count$ )
27.                     $CF = CF - I$ ;
28.            end
29.        end
30. select  $Thtw^{i,j}$  from  $I$  where  $I \in CF$ ;
31. keep  $Thtw^{i,j}$  in main memory;
32.  $h = 2$ 
33. begin while ( $Thtw^{i,j} \neq \phi$ ) //Database scan reduction
34.     if ( $h=2$ )
35.          $C_{h+1}^{i,j} = Thtw^{i,j} * Thtw^{i,j}$ ;
36.     else
37.          $C_{h+1}^{i,j} = C_h^{i,j} * C_h^{i,j}$ 
38.      $h = h + 1$ ;
39. end
40. refresh  $I.twu = 0$  where  $I \in Thtw^{i,j}$ ;
41. begin for  $k = i$  to  $j$  //2nd scan of  $db^{i,j}$ 
42.     for each itemset  $I \in Thtw^{i,j} \cup C_{h+1}^{i,j}$ 
43.          $I.u = I.u + UP_k(I)$ ;
44.     end
45. for each itemset  $I \in Thtw^{i,j} \cup C_{h+1}^{i,j}$ 
46.     if ( $I.u \geq F$ )
47.          $Thu^{i,j} = Thu^{i,j} \cup I$ ;
48.     else if ( $F > I.u \geq s \times P.count$ )
49.          $I.RUT = \max(\text{threshold}(i_1, i_2, \dots, i_k) / \text{threshold}(i_1),$ 
50.                     $\text{threshold}(i_1, i_2, \dots, i_k) / \text{threshold}(i_2),$ 
51.                     $\dots,$ 
52.                     $\text{threshold}(i_1, i_2, \dots, i_k) / \text{threshold}(i_k))$ 
53.         if ( $I.RUT \geq RUT$ )
54.              $Tru^{i,j} = Tru^{i,j} \cup I$ ;
55.     end
56. return  $Thu^{i,j}$  and  $Tru^{i,j}$ ;

```

Figure 4-3. Incremental procedure of TRUI-Mine.

4.3 Experiments and Analysis

As described in previous sections, the proposed algorithms *TP-RUI-Mine* and *TRUI-Mine* are

based on the principle of the Two-Phase algorithm [21] and THUI-Mine [32] respectively, with consideration of the concept of significant rare itemsets. To evaluate the performance of *TP-RUI-Mine* and *TRUI-Mine*, we conducted experiments using synthetic datasets generated via a randomized dataset generator provided by IBM Quest project [3]. However, the IBM Quest data generator only generates the quantity of 0 or 1 for each item in a transaction. In order to fit databases into the scenario of utility mining, we randomly generated the quantity of each item in each transaction, ranging from 1 to 5, in a similar fashion to the model used in [21][32]. Utility tables are also synthetically created by assigning a utility value to each item randomly, ranging from 1 to 1000. Because it is observed from real world databases that most items are in the low profit range, we generated the utility values using a log normal distribution, in a similar fashion to the model used in [21][32]. Figure 4-4 shows the utility value distribution of 1000 items.

The simulation is implemented in C++ and conducted in a machine with 2.4GHz CPU and 1G memory. The main performance metric utilized is execution time. We recorded the execution time that *TP-RUI-Mine* and *TRUI-Mine* used in finding temporal high utility itemsets and temporal rare utility itemsets. The number of itemsets compared by *TP-RUI-Mine* and *TRUI-Mine* is presented to show the performance comparison of *TP-RUI-Mine* and *TRUI-Mine*. Finally, we show the results of scale-up experiments.

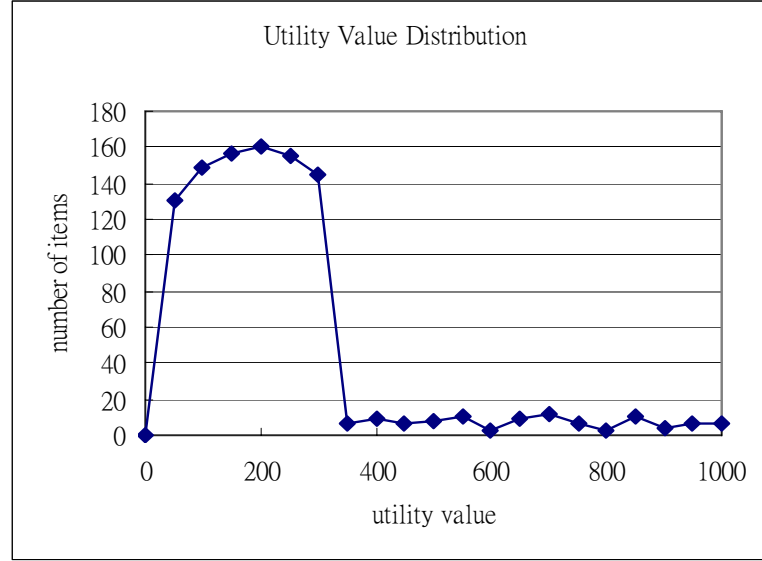


Figure 4-4. Utility value distribution in utility table.

Evaluation on Number of Generated Candidates

In this experiment, we compared the average number of candidates generated in the first database scan on the sliding windows and the incremental transaction number d10K with the difference between the first utility threshold and the second utility threshold for *TRUI-Mine* and *TP-RUI-Mine*. The relative utility threshold is set to 0.1. Without loss of generality, we set $|d| = |\Delta^+| = |\Delta^-|$ for simplicity. Thus, by denoting the original database as $db^{1,n}$ and the new mining database as $db^{i,j}$, we have $|db^{i,j}| = |db^{1,n} - \Delta^- + \Delta^+| = |D|$, where $\Delta^- = db^{1,i-1}$ and $\Delta^+ = db^{n+1,j}$. Table 4-4 shows the average number of candidates, temporal high utility and temporal rare utility generated by *TRUI-Mine*, and *TP-RUI-Mine*. The number of items is set at 1000, and the minimum first and second utility threshold varies from 0.2% to 1%. The number of candidate itemsets generated by *TRUI-Mine* at the first database scan decreases dramatically as the threshold goes up. When the second utility threshold and the first utility threshold are set to 0.8% and 1%, the number of temporal rare utility itemsets is 0 in database T10.I6.D100K.d10K where T denotes the average size of the transactions and I the average

number of frequent itemsets. We also use the same way to find the average number of candidates, temporal high utility and temporal rare utility in different database T20.I6.D100K.d10K and T10.I4.D100K.d10K. Although it still has many of temporal high utility itemsets, all of these temporal high utility itemsets are signal item. However, the number of candidates generated by *TP-RUI-Mine* is still very large. *TRUI-Mine* generates far fewer candidates when compared to *TP-RUI-Mine*.

Overall, our algorithm *TRUI-Mine* always generated far fewer candidates compared to *TP-RUI-Mine* for various kinds of databases. Hence, *TRUI-Mine* was verified to be very effective in pruning candidate itemsets to find temporal high utility itemsets and temporal rare utility itemsets.

Table 4-4. Number of candidate itemsets, temporal high utility itemsets and temporal rare utility itemsets generated on dataset T10.I6.D100K.d10K.

T10.I6.D100K.d10K (RUT = 0.1)					
Second Utility threshold	First utility threshold	<i>TRUI-Mine</i>	<i>TP-RUI-Mine</i>	Temporal high utility itemsets	Temporal rare utility itemsets
0.2%	0.4%	55967	445096	334	292
0.3%	0.5%	33101	431985	275	30
0.4%	0.6%	20088	412686	221	2
0.6%	0.8%	8060	385881	151	1
0.8%	1%	3433	361675	104	0

Evaluation of Execution Efficiency

In this experiment, we demonstrated the relative performance of *TP-RUI-Mine* and *TRUI-Mine*. Figure 4-5, Figure 4-6 and Figure 4-7 show the execution times for the two

algorithms as the various cases that the difference between first utility threshold and second utility threshold. It is observed that when the first and second utility threshold is high, there are only a limited number of high utility itemsets and rare utility itemsets produced. However, as the second utility threshold decreases, the performance difference becomes prominent in that *TRUI-Mine* significantly outperforms *TP-RUI-Mine*. As shown in Figure 4-5, Figure 4-6 and Figure 4-7, *TRUI-Mine* leads to significant performance improvement for different average sizes of transaction. Explicitly, *TRUI-Mine* is orders of magnitude faster than *TP-RUI-Mine*, and the margin grows as the second utility threshold decreases. It is observed that *TRUI-Mine* spends less time than *TP-RUI-Mine* and maintains high stability while finding temporal high utility itemsets and temporal rare utility itemsets. This is because the *TP-RUI-Mine* produces more candidate itemsets and needs more database scans to find high utility itemsets and rare utility itemsets than the *TRUI-Mine* algorithm. To measure how much execution time could be reduced substantially in using *TRUI-Mine* compared to *TP-RUI-Mine*, we define the *Improvement Ratio* as follows:

$$\text{Improvement Ratio} = \frac{(\text{execution time of TP - RUI}) - (\text{execution time of TRUI})}{\text{execution time of TP - RUI}}$$

From the data illustrated in Figure 4-5 and Figure 4-6, we see that the Improvement Ratio is about 86.8% and 83% with the second utility threshold set as 0.2%. In Figure 4-7, the average improvement is about 64% with second utility threshold varied from 0.2% to 0.6%. Obviously, *TRUI-Mine* reduces substantially the time taken in finding high utility itemsets and rare utility itemsets. Moreover, the high utility itemsets and the rare utility itemsets obtained by *TP-RUI-Mine* are not suitable for applications in temporal databases since *TP-RUI-Mine* needs more database scans and increased execution times in finding high utility itemsets and rare utility itemsets by the time change. Hence, *TRUI-Mine* meets the requirements of high efficiency in terms of execution time for data stream mining.

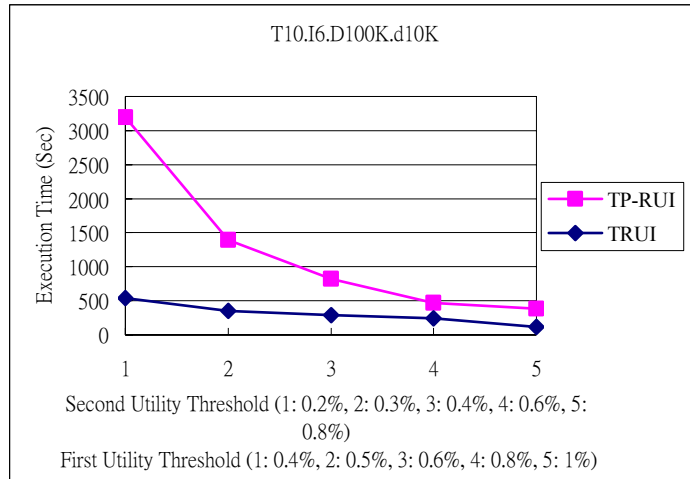


Figure 4-5. Execution time on T10.I6.D100K.d10K.

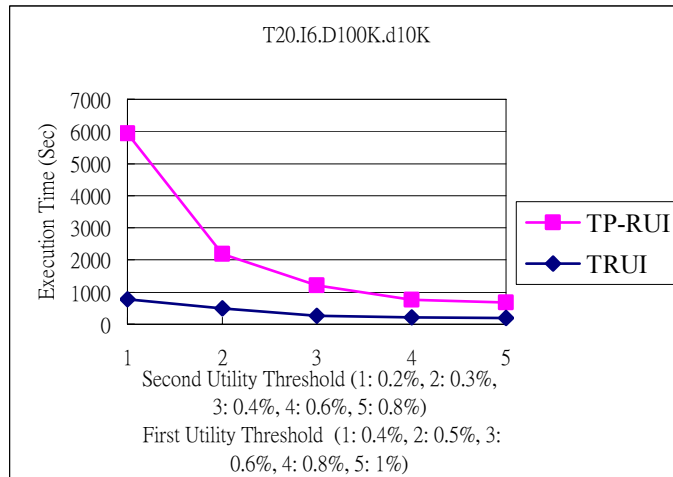


Figure 4-6. Execution time on T20.I6.D100K.d10K.

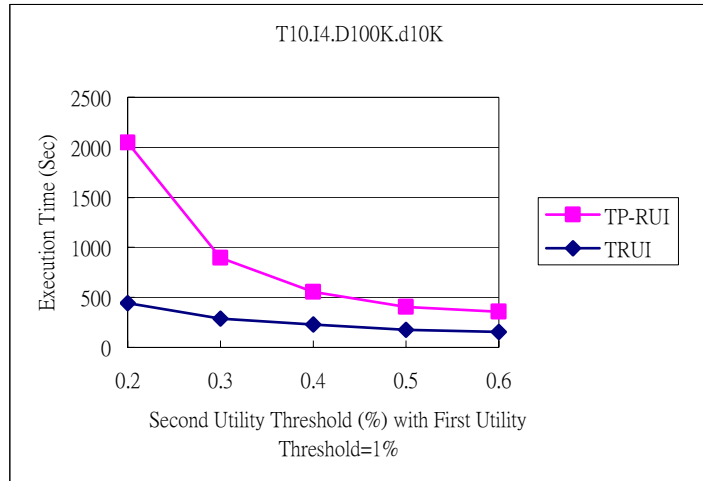


Figure 4-7. Execution time on T10.I4.D100K.d10K.

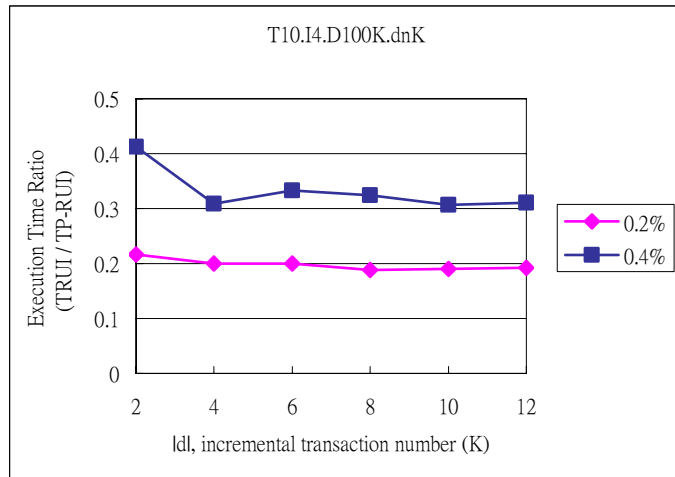


Figure 4-8. Scale-up performance results.

Scale-up on Incremental Mining

In this experiment, we investigated the effects of varying incremental transaction size on the execution time of mining results. To further understand the impact of $|d|$ on the relative performance of algorithms *TRUI-Mine* and *TP-RUI-Mine*, we conducted scale-up experiments which were similar to those described in [18] for both *TRUI-Mine* and *TP-RUI-Mine* with second utility thresholds being varied from 0.2% and 0.4 % and first utility thresholds being

varied from 0.4% and 0.6 %, respectively. Figure 4-8 shows the experimental results where the value of y-axis corresponds to the ratio of the execution time of *TRUI-Mine* to that of *TP-RUI-Mine*. Figure 4-8 shows the execution time-ratio for different values of $|d|$. It can be seen that the execution time-ratio remains stable with the growth of the incremental transaction number $|d|$ since the size of $|d|$ has little influence on the performance of *TRUI-Mine*. Moreover, the execution time ratios of the scale-up experiments with second utility thresholds varied from 0.6% to 1% remain constant at approximately 0.4%. This implies that the advantage of *TRUI-Mine* over *TP-RUI-Mine* is stable and less execution time is taken as the amount of incremental portion increases. This result also indicates that *TRUI-Mine* is more useful for mining temporal databases with large transaction size.



Chapter 5

Mining High Utility Itemsets with Negative Item Values

5.1 Problem Definition

Mining of association rules in large databases is a well studied technique in the field of data mining with typical methods like *Apriori* [1][2]. The problem surrounding association rules mining can be decomposed into two steps. The first step involves finding all frequent itemsets (or large itemsets) in a database. Once the frequent itemsets are found, generating association rules is straightforward and can be accomplished in linear time.

Most methods in finding frequent itemsets are designed for traditional databases. However, the frequency of an itemset may not be a sufficient indicator of significance, because frequency reflects only the number of transactions in the database that contain that itemset. It does not reveal the *utility* of an itemset, which can be measured in terms of cost, profit, or other expressions of user preference. On the other hand, frequent itemsets may only contribute a small portion of the overall profit, whereas non-frequent itemsets may contribute a large portion of the profit. In reality, a retail business may be most interested in identifying its most valuable customers (customers who contribute a major fraction of the profits to the company). Hence, frequency is not sufficient to answer questions such as whether an itemset is highly profitable, or whether an itemset has a strong impact.

Utility mining is thus useful in a wide range of practical applications and was recently studied in [8][21][32][35]. However, a retail business may sale item with negative value. For example, many super markets may promote certain items to attract customers. In this scenario

customers may buy specific items and then receive free goods. Free goods result in negative value for super markets. However, supermarkets may earn higher profits from other items that are cross-promoted with these free items. This practice is common. For example, if a customer bought three of item A, he would then receive one free item B as a promotion from the supermarket. Suppose the supermarket gets 5 dollars of profit from each unit of item A sold, and loses 2 dollars for each unit of item B given away. Although giving away a unit of item B results in a loss of 2 dollars for supermarkets, they could possibly earn 15 dollars from the three units of item A that are cross-promoted with item B. The supermarket thus may have a net gain of 13 dollars from this promotion. This example demonstrates why we propose the concept of mining for negative item values in utility mining. This also motivates our research in developing a new scheme for finding *high utility itemsets with negative item values (HUINIV)* from large databases.



Table 5-3. A transaction database and its utility table.

(a) Transaction table

TID \ ITEM	ITEM				
	A	B	C	D	E
T ₁	1	0	0	2	1
T ₂	0	1	2	6	0
T ₃	3	0	0	5	0
T ₄	1	0	0	0	1
T ₅	0	1	2	6	0
T ₆	0	1	1	0	2
T ₇	2	0	0	0	0
T ₈	3	0	0	1	0
T ₉	0	1	1	4	0
T ₁₀	1	0	0	0	1

(b) The utility table

ITEM	VALUE\$(per unit)
A	5
B	-3
C	-2
D	6
E	10

Recently, a *utility mining* model has been defined in [35]. Utility is a measure of how “useful” (i. e. “profitable”) an itemset is. The definition of the utility of an itemset X , $u(X)$, states that it is equal to the sum of the utilities of X of all the transactions containing X . The goal of utility mining is to identify high utility itemsets, which drive a large portion of the total utility. Traditional association rules of mining models assume that the utility of each item is always 1 and that the quantity of sales is either 0 or 1; thus it is only a special case of utility mining in which the utility or the quantity of sales of each item can be any number. If $u(X)$ is greater than a specified utility threshold, X is a high utility itemset; otherwise, it is a low utility itemset. Table 5-1 is an example of utility mining in a transaction database. The number associated with each transaction in Table 5-1(a) is the sales volume of each item, and the utility of each item is listed in Table 5-1(b). For example, $u(\{A, D\}) = (1 \times 5 + 2 \times 6) + (3 \times 5 + 5 \times 6) + (3 \times 5 + 1 \times 6) = 83$. $\{A, D\}$ is a high utility itemset if the utility threshold is set at 80.

However, a high utility itemset may consist of low utility items. Another possibility is to adopt the level-wise searching schema that exists in fast algorithms, such as Apriori [3]. This algorithm doesn't apply to the *utility mining* model. For example, $u(A) = 55 < 80$, A is a low utility item, but its superset $\{A, D\}$ is a high utility itemset. If Apriori is used to find high utility itemsets, all combinations of all items must be generated. Moreover, in order to discover a long pattern, the number of candidates is prohibitively large. The cost in terms of either computation time or memory is intolerable, regardless of the method utilized. The

challenge of utility mining is not only in restricting the size of the candidate set but also in simplifying the computation used to calculate its utility. Another challenge of utility mining is finding high utility itemsets with negative item values from large databases.

In this chapter, we explore the issue of efficiently mining high utility itemsets with negative item values in large databases. We propose an algorithm named *HUINIV(High Utility Itemsets with Negative Item Values)-Mine* that can discover high utility itemsets with negative item values from large databases both efficiently and effectively. The underlying idea behind the *HUINIV-Mine* algorithm is based on the principle of the Two-Phase algorithm [21] and augments with negative item value for mining high utility itemsets efficiently. The novel contribution of *HUINIV-Mine* is that it can efficiently identify the utility of itemsets in large database so that the execution time for producing high utility itemsets with negative item values can be substantially reduced. That is, *HUINIV-Mine* can discover high utility itemsets with negative item values using limited memory and comparatively less computation time by the candidate itemsets filter method. In this way, the process of discovering all high utility itemsets in which all transactions are negative can be achieved effectively with limited memory, less candidate itemsets, and CPU I/O. This meets the critical requirements of time and spatial efficiency for mining large databases. Through experimental evaluation, *HUINIV-Mine* is shown to produce fewer candidate itemsets in the process of finding high utility itemsets with negative item values, so it outperforms other methods in terms of efficiency. We found that the average improvement of *HUINIV-Mine* compared to the MEU algorithm is about 99.2%. Moreover, it also achieves high scalability in dealing with large databases. To the best of our knowledge, this is the first work to propose a negative item concept in utility mining and the first work on mining high utility itemsets with negative item values from large database.

The rest of this chapter is organized as follows: Section 5.2 describes the proposed approach, *HUINIV-Mine*, for finding the high utility itemsets with negative item values. In

section 5.3, we describe our experimental results for evaluating the proposed method. The conclusion of the chapter is provided in Section 5.4.

5.2 Proposed Method

In this section, we present the *HUINIV-Mine* method. We describe the basic concept of *HUINIV-Mine*. We give an example of mining temporal high utility itemsets and the procedure of the *HUINIV-Mine* algorithm.

Basic Concept of HUINIV-Mine

The goal of utility mining is to discover all itemsets whose utility values exceed a user specified threshold in a transaction database. In [35] the goal of utility mining is to find all high utility itemsets. An itemset X is a *high utility itemset* if $u(X) \geq \epsilon$, where $X \subseteq I$ and ϵ is the minimum utility threshold, otherwise, it is a *low utility itemset*. For example, in Table 5-1, $u(A, T_1) = 1 \times 5 = 5$, $u(\{A, E\}, T_1) = u(A, T_1) + u(E, T_1) = 1 \times 5 + 1 \times 10 = 15$, and $u(\{A, E\}) = u(\{A, E\}, T_1) + u(\{A, E\}, T_4) + u(\{A, E\}, T_{10}) = 15 + 15 + 15 = 45$. If $\epsilon = 80$, $\{A, E\}$ is a low utility itemset. However, if an item is a low utility item, its superset may be a high utility itemset. For example, $u(A) = 55 < 80$, A is a low utility item, but its superset $\{A, D\}$ is a high utility itemset because $u(\{A, D\}) = 83 > 80$. Hence, all the combinations of all items should be processed so that it never loses any high utility itemset. But the high cost of either computational time or memory is intolerable. A set of terms that leads to the formal definition of the utility mining problem can be generally defined as follows by referring to [35]:

- $I = \{i_1, i_2, \dots, i_m\}$ is a set of items.
- $D = \{T_1, T_2, \dots, T_n\}$ be a transaction database where each transaction $T_i \in D$ is a subset of I .
- $o(i_p, T_q)$, *local transaction utility value*, represents the quantity of item i_p in transaction

T_q . For example, $o(A, T_3) = 3$, in Table 5-1(a).

- $s(i_p)$, *external utility*, is the value associated with item i_p in the Utility Table. This value reflects the importance of an item, which is independent of transactions. For example, in Table 5-1(b), the external utility of item A, $s(A)$, is 5.
- $u(i_p, T_q)$, *utility*, the quantitative measure of utility for item i_p in transaction T_q , is defined as: $o(i_p, T_q) \times s(i_p)$. For example, $u(A, T_3) = 3 \times 5$, in Table 5-1.
- $u(X, T_q)$, *utility of an itemset X in transaction Tq*, is defined as $\sum_{i_p \in X} u(i_p, T_q)$, where $X = \{i_1, i_2, \dots, i_m\}$ is a k -itemset, $X \subseteq T_q$ and $1 \leq k \leq m$.
- $u(X)$, *utility of an itemset X*, is defined as $\sum_{T_q \in D \wedge X \subseteq T_q} u(X, T_q)$.

Liu *et al.* [21], proposed the Two-Phase algorithm for pruning candidate itemsets and simplifying the calculation of utility. First, Phase I overestimates some low utility itemsets, but it never underestimates any itemsets. For example in Table 5-1, the *transaction utility of transaction Tq*, denoted as $tu(T_q)$, is the sum of the utilities of all items in T_q : $tu(T_q) = \sum_{i_p \in T_q} u(i_p, T_q)$. And the *transaction-weighted utilization of an itemset X*, denoted as $twu(X)$, is the sum of the transaction utilities of all transactions containing X : $twu(X) = \sum_{X \subseteq T_q \in D} tu(T_q)$. For example, $twu(A) = tu(T_1) + tu(T_3) + tu(T_4) + tu(T_7) + tu(T_8) + tu(T_{10}) = 27 + 45 + 15 + 10 + 21 + 15 = 126$ and $twu(\{A, E\}) = tu(T_1) + tu(T_4) + tu(T_{10}) = 27 + 15 + 15 = 57$. In fact, $u(A) = u(\{A\}, T_1) + u(\{A\}, T_3) + u(\{A\}, T_4) + u(\{A\}, T_7) + u(\{A\}, T_8) + u(\{A\}, T_{10}) = 5 + 15 + 5 + 10 + 15 + 5 = 55$ and $u(\{A, E\}) = u(\{A, E\}, T_1) + u(\{A, E\}, T_4) + u(\{A, E\}, T_{10}) = 15 + 15 + 15 = 45$. So while Phase I overestimates some low utility itemsets, it never underestimates any itemsets whatsoever. Table 5-2 gives the transaction utility of each transaction in Table 5-1. One extra database scan is performed to filter the overestimated itemsets in phase II. For example, $twu(A) = 126 > 80$ but $u(A) = 55 < 80$. After that, item $\{A\}$ is pruned; otherwise, it is a high utility itemset. In the end, all of high utility itemsets have been discovered in this

way. However, we cannot apply the Two-Phase algorithm to databases whose items include negative values. Some high utility itemsets may be lost in this way. For example, $twu(\{B, D\}) = tu(T_2) + tu(T_5) + tu(T_9) = 29 + 29 + 19 = 77$. If $\epsilon = 80$, $twu(\{B, D\}) = 77 < 80$ is a low transaction-weighted utilization itemset, then $\{B, D\}$ will be deleted. In fact, $u(\{B, D\}) = u(\{B, D\}, T_2) + u(\{B, D\}, T_5) + u(\{B, D\}, T_9) = 33 + 33 + 21 = 87 > 80$. $\{B, D\}$ should be a high utility itemset. Thus, the Two-Phase algorithm is not sufficient to answer question regarding items with negative values. As one possible solution, utility mining, useful over a wide range of practical applications, was recently studied in [8][21][32][35]. This also motivates our research in developing a new scheme for finding *high utility itemsets with negative item values (HUINIV)* from large databases.

Our algorithm *HUINIV-Mine* is based on the principle of the Two-Phase algorithm [21], and focuses on utilizing transaction utility without using negative item value methods to improve the response time by concerning fewer candidate itemsets and less CPU I/O. In essence, by removing items with negative values from a transaction in a large database, algorithm *HUINIV-Mine* employs a filtering threshold within the database to deal with the transaction-weighted utilization itemsets (TWUI) generated. Table 5-3 gives the transaction utility without negative item values for each transaction in Table 5-1. In this way, *HUINIV-Mine* can overestimate some low utility itemsets, but it never underestimates any itemsets and it never loses any itemsets that may be of high utility. In processing a database, a transaction-weighted utilization set of itemsets is generated by *HUINIV-Mine*. Explicitly, a transaction-weighted utilization set of itemsets is composed of the TWUI that were generated from the previous transaction-weighted utilization candidate sets during the previous phase. After the processing, the algorithm *HUINIV-Mine* outputs a high transaction-weighted utilization set of itemsets. However, some of the high transaction-weighted utilization sets of itemsets should be pruned advance. Each item of the itemset that has negative value will never be part of a high utility itemset. At least one item's value within an itemset should be

positive, or the itemset need not scan the database. Hence, the algorithm *HUINIV-Mine* outputs real high transaction-weighted utilization candidate itemsets after filtering some itemsets. Finally, *HUINIV-Mine* computes the occurrence counts of itemsets in the memory and then deletes itemsets that do not satisfy utility threshold within the database so as to find high utility itemsets with negative item values.

Taking these design features under consideration, the algorithm *HUINIV-Mine* is shown to perform very well at mining high utility itemsets with negative item values from large databases. In Section 5.3.2, we give an example of mining high utility itemsets with negative item values from large databases. The proposed algorithm, *HUINIV-Mine*, is described in detail in Section 5.3.3.

Table 5-2. Transaction utility of the transaction database.

TID	Transaction Utility	TID	Transaction Utility
T ₁	27	T ₆	15
T ₂	29	T ₇	10
T ₃	45	T ₈	21
T ₄	15	T ₉	19
T ₅	29	T ₁₀	15

Table 5-3. Transaction utility without negative item values of the transaction database.

TID	Transaction Utility without Negative Item Values	TID	Transaction Utility without Negative Item Values
T ₁	27	T ₆	20
T ₂	36	T ₇	10
T ₃	45	T ₈	21
T ₄	15	T ₉	24
T ₅	36	T ₁₀	15

An Example of Mining High Utility Itemsets with Negative item Values

The proposed *HUINIV-Mine* algorithm can be best understood from the illustrative transaction database shown in Table 5-1 and Figure 5-1 in which a scenario for generating high utility itemsets from large databases to mine high utility itemsets with negative item values is given. This type of illustrative transaction database resembles items that are sold by supermarkets in real life. This also means that utility mining has real-life applications. We set the utility threshold at 80 with ten transactions. Without loss of generality, the mining problem can be decomposed into two procedures:

1. TWUI procedure: This procedure deals with mining the transaction database to generate TWUI.
2. Filter procedure: The procedure deals with filtering negative itemsets and generating high utility itemsets with negative item values from large databases.

C ₁	transaction-weighted utility	C ₂	transaction-weighted utility	C ₃	transaction-weighted utility
⊙ A	133	AB	0	⊙BCD	96
⊙ B	116	AC	0		
⊙ C	116	⊙ AD	93		
⊙ D	189	⊙ BC	116		
E	77	⊙ BD	96		
		⊙ CD	96		

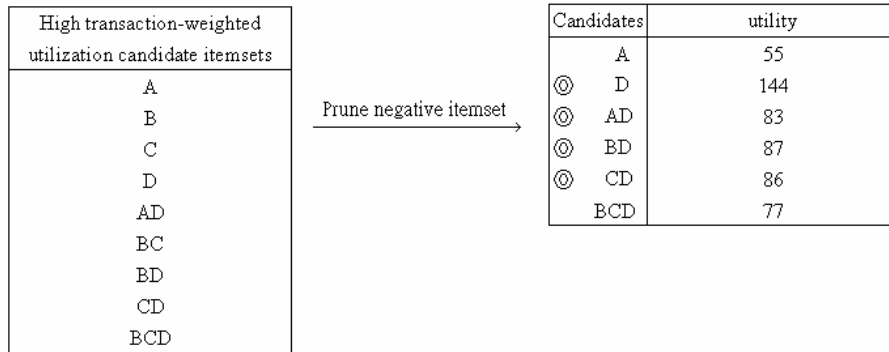


Figure 5-1. High utility itemsets generated from large databases by HUIV-Mine.



The TWUI procedure is only utilized for the initial utility mining in the database. For the mining high utility itemsets, the filter procedure is employed. Consider the database in Table 5-1. Each transaction is scanned sequentially for the generation of candidate 1-itemsets in the first scan of the database. Itemsets whose transaction-weighted utility is below the utility threshold are removed. Then, as shown in Figure 5-1, only {A, B, C, D}, marked by “⊙”, remain as high transaction-weighted utilization 1-itemsets. Although items B and C have negative values, they may constitute high utility itemset by combining with other items. These items should be preserved to combine with other items to generate the next candidate itemsets. The candidate 2-itemsets {AB, AC, AD, BC, BD, CD} are generated by high transaction-weighted utilization 1-itemsets. In the same way, only {AD, BC, BD, CD}, marked by “⊙”, remain as high transaction-weighted utilization 2-itemsets. The candidate 3-itemsets {BCD} are generated by high transaction-weighted utilization 2-itemsets; high transaction-weighted utilization 3-itemsets being those whose transaction-weighted utility is

above the utility threshold.

We could get high transaction-weighted utilization candidate itemsets $\{A, B, C, D, AD, BC, BD, CD, BCD\}$. However, some of the high transaction-weighted utilization sets of itemsets should be pruned in advance. If each item of the itemset's value is negative, it will not be a high utility itemset. For example, $\{B, C, BC\}$ should be deleted. Hence, algorithm *HUINIV-Mine* outputs only 6 real high transaction-weighted utilization candidate itemsets $\{A, D, AD, BD, CD, BCD\}$ after filtering the itemsets. Finally, all candidates can be stored in main memory, and we can find high utility itemsets with negative item values when the scan of the database is performed. The resulting high utility itemsets are $\{D\}$, $\{AD\}$, $\{BD\}$ and $\{CD\}$ because $u(D) = 144 > 80$, $u(\{A, D\}) = 83 > 80$, $u(\{B, D\}) = 87 > 80$ and $u(\{B, E\}) = 86 > 80$ as shown in Figure 5-1.

Table 5-4. Meanings of symbols used.

DB	Database
threshld	Utility threshold in database
twu	Transaction-weighted utilization itemsets without negative item values
htwu _i	High transaction-weighted utilization i-itemsets without negative item values
I.value	Each item's value
hui	High utility itemsets with negative item values

HUINIV-Mine Algorithm

For clarification, the meanings of various symbols used are given in Table 5-4. The procedure used Algorithm *HUINIV-Mine* is shown in Figure 5-2.

Initially, it input the database DB (in step 1), and it finds high transaction-weighted utilization 1-itemsets from step 2 to step 5. The transaction-weighted utility without negative

item value of itemset I is recorded in $I.twu$. An itemset, whose $I.twu \geq \text{threshold}$ (in step 3), will be kept in $htwu$ (in step 4). Next, we use high transaction-weighted utilization itemsets to generate transaction-weighted utilization candidate itemsets. Then we scan the database to find high transaction-weighted utilization itemsets from step 6 to step 13. After identifying all $htwu$, we perform a last scan of the database from Step 14 to Step 19. Since each item's value in the itemset is negative, it cannot be a high utility itemset. At least one item's value in itemset I should be positive (in step 14), or else the itemset does not need to scan the database. Finally, those itemsets satisfying the constraint that $I.htwu \geq \text{threshold}$ are finally obtained as the high utility itemsets with negative item values.

Algorithm HUINIV-Mine

```

1.  Input: DB
2.  for k = 1 to n           // Find high transaction-weighted utilization 1-itemsets
3.    if ( $I.twu \geq \text{threshold}$ )
4.       $htwu_1 = htwu_1 \cup I.twu$ ;
5.    end
6.  for i=2 ;  $htwu_{i-1} \neq \phi$ ; i++
7.     $I.twu = I.twu\_gen(htwu_{i-1})$ ; // Generate transaction-weighted utilization i-candidate itemsets
8.    if ( $I.twu \neq \phi$ )
9.      for k=1 to n
10.     if ( $I.twu \geq \text{threshold}$ )
11.        $htwu_i = htwu_i \cup I.twu$ ;
12.     end
13.   end
14.  for each itemset  $I \in htwu \neq \phi$  &&  $I.value > 0$  // Find high utility itemsets with negative item values
15.    for k = 1 to n
16.      if ( $I.htwu \geq \text{threshold}$ )
17.         $hui = hui \cup I.htwu$ ;
18.      end
19.    end
20.  return hui;
```

Figure 5-2. Procedure used by HUINIV-Mine.

5.3 Experiments and Analysis

To evaluate the performance of *HUINIV-Mine*, we conducted experiments using synthetic

datasets generated via a randomized dataset generator provided by IBM Quest [3]. However, the IBM Quest data generator only generates quantities of 0 or 1 for each item in a transaction. In order to fit databases into the scenario of utility mining, we randomly generate the quantity of each item in each transaction, ranging from 1 to 5; much like the model used in [21][32]. Utility tables are also synthetically created by assigning a utility value to each item randomly, ranging from -100 to 1000. Since it is observed from real databases that most items are in the low value range and low negative value range, we generate the utility values using a log normal distribution; similarly to the model used in [21][32]. Figure 5-3 shows the utility value distribution of 1000 items.

The simulation is implemented in C++ and conducted in a machine with 2.4GHz CPU and 1G memory. The MEU algorithm [35] is also utilized in a negative itemsets scenario for comparison with the *HUINIV-Mine* algorithm. The scenario using MEU consists of scanning the database after collecting the data to find high utility itemsets with negative item values. The main performance metric used is execution time. We recorded the time that *HUINIV-Mine* uses to find high utility itemsets with negative item values. The number of candidate itemsets compared of *HUINIV-Mine* and MEU is presented and show comparison in performance of a variety of IBM Quest data with *HUINIV-Mine*. We also give the performance of *HUINIV-Mine* with real data.

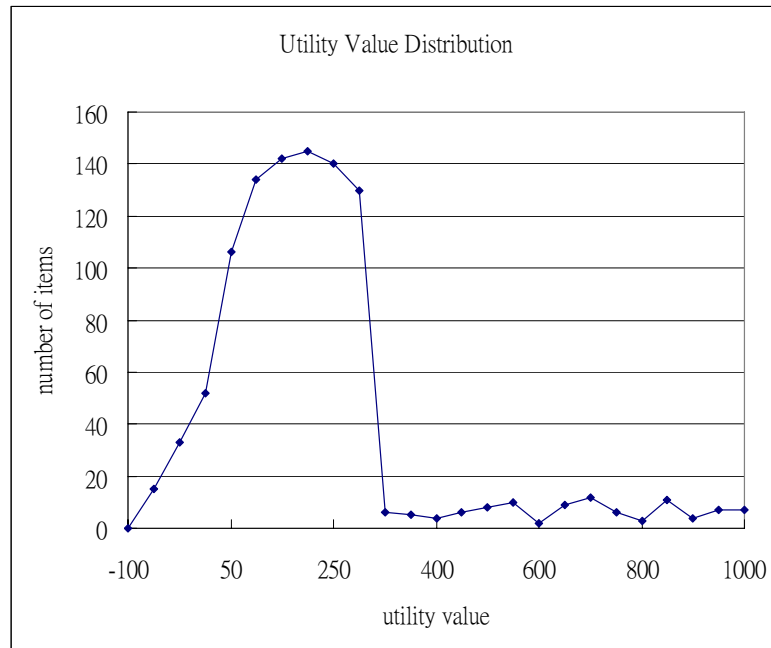


Figure 5-3. Utility value with negative distribution in utility table.

Evaluation of Number of Generated Candidates

In this experiment, we compare the average number of candidates generated in the first database scan with different support values for *HUINIV-Mine* and MEU [35]. Table 5-5, Table 5-6, Table 5-7 and Table 5-8 show the average number of candidates generated by *HUINIV-Mine* and MEU. The number of items is set at 1000, and the minimum utility threshold varies from 0.2% to 1%. The number of candidate itemsets generated by *HUINIV-Mine* during the first database scan decreases dramatically as the threshold increases. Particularly when the utility threshold is set to 1%, the number of candidate itemsets is generally 588, including all various candidate itemsets in database T10.I4.D100K where T denotes the average size of the transactions and I the average number of frequent itemsets. However, the number of candidates generated by MEU is always 499,500 because it needs to process all combinations of 1000 items to generate only 2-candidate itemsets. *HUINIV-Mine* generates far fewer candidates when compared to MEU.

We obtain similar experimental results from different datasets. For example, only 644 candidate itemsets are generated by *HUINIV-Mine*, but 499500 candidate itemsets are generated by MEU, respectively, when the utility threshold is set as 1% in dataset T10.I4.D100K. In the case of datasets T20.I4.D100K and T20.I6.D100K, more candidates are generated, because each transaction is longer than those in T10.I4.D100K and T10.I4.D100K. Overall, our algorithm *HUINIV-Mine* always generates far fewer candidates when compared to MEU for various kinds of databases. Thus, *HUINIV-Mine* is verified to be very effective in pruning candidate itemsets to find high utility itemsets with negative item values.

It is observed that *HUINIV-Mine* obtains fewer candidate itemsets than MEU with high stability with regard to finding high utility itemsets with negative item values. To measure how many candidate itemsets could be reduced substantially by using *HUINIV-Mine* compared to MEU algorithm, we define the *Improvement Ratio* as follows:


$$\text{Improvement Ratio} = \frac{(\text{candidate itemsets of MEU}) - (\text{candidate itemsets of HUIVP - Mine})}{\text{candidate itemsets of MEU}}$$

From the data illustrated in Table 5-5, we see that the Improvement Ratio is about 99.8% with the threshold set as 1%. In Table 5-8, the average improvement is about 99.2% with the minimum utility threshold varied from 0.2% to 1%. Obviously, *HUINIV-Mine* reduces substantially the candidate itemsets for finding high utility itemsets with negative item values. Moreover, the high utility itemsets obtained by MEU are not suitable for applications in large database since MEU requires more database scans, and increased execution times and candidate itemsets to find high utility itemsets with negative item values. Thus, *HUINIV-Mine* meets the requirements of being highly effective in terms of candidate itemsets for large database mining.

Table 5-5. The number of candidate itemsets and high utility itemsets generated from database T10.I4.D100K.

Databases Threshold	T10.I4.D100K		
	<i>HUINIV-Mine</i>	MEU	High Utility Itemsets
0.2%	11306	499500	285
0.3%	3928	499500	183
0.4%	1691	499500	115
0.6%	846	499500	58
0.8%	676	499500	29
1%	588	499500	14

Table 5-6. The number of candidate itemsets and high utility itemsets generated from database T10.I6.D100K.



Databases Threshold	T10.I6.D100K		
	<i>HUINIV-Mine</i>	MEU	High Utility Itemsets
0.2%	16304	499500	335
0.3%	5336	499500	197
0.4%	2469	499500	130
0.6%	1056	499500	69
0.8%	755	499500	29
1%	644	499500	19

Table 5-7. The number of candidate itemsets and high utility itemsets generated from database T20.I4.D100K.

Databases Threshold	T20.I4.D100K		
	<i>HUINIV-Mine</i>	MEU	High Utility Itemsets
0.2%	20026	499500	118
0.3%	7958	499500	55
0.4%	3754	499500	29
0.6%	1308	499500	8
0.8%	774	499500	4
1%	599	499500	2

Table 5-8. The number of candidate itemsets and high utility itemsets generated from database T20.I6.D100K.

Databases Threshold	T20.I6.D100K		
	<i>HUINIV-Mine</i>	MEU	High Utility Itemsets
0.2%	27357	499500	127
0.3%	8441	499500	56
0.4%	4095	499500	23
0.6%	1438	499500	7
0.8%	823	499500	4
1%	637	499500	3

Evaluation of Execution Time

In this experiment, we show only the performance of *HUINIV-Mine* since MEU requires much higher execution time (longer than 10 hours) to complete the second scan lacks basis for

comparison because the number of candidate itemsets generated is always 499500. Therefore, *HUINIV-Mine* meets the requirements of efficiency in terms of execution time for large database mining.

Figure 5-4 and Figure 5-5 show the execution times for *HUINIV-Mine* as the minimum utility threshold is decreased from 1% to 0.2%. It is observed that when the minimum utility threshold is high, there are only a limited number of high utility itemsets produced. However, as the minimum utility threshold decreases, the execution times increase with more high utility itemsets produced. As shown in Figure 5-4 and Figure 5-5, the margin grows as the minimum utility threshold increases for different average sizes of transaction.

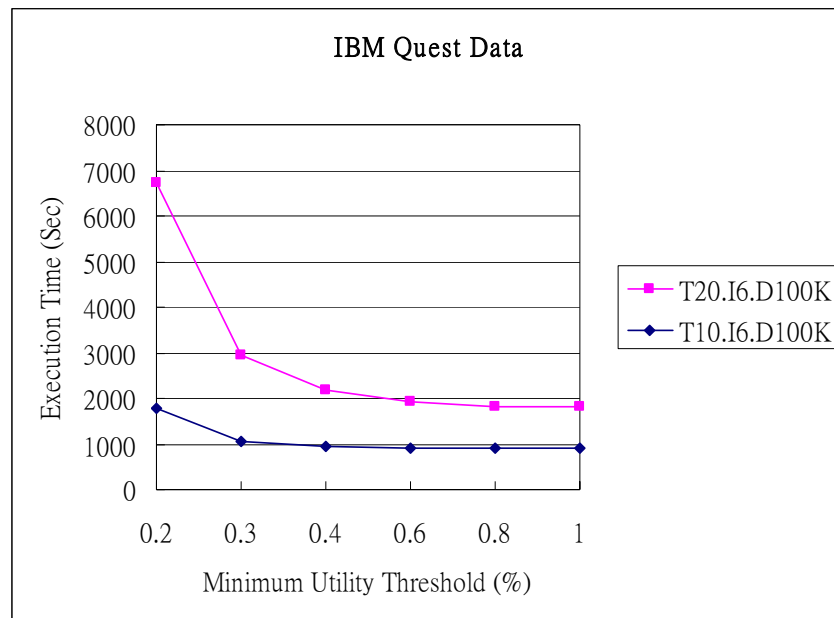


Figure 5-4. Execution time for HUINIV on T20.I6.D100K and T10.I6.D100K.

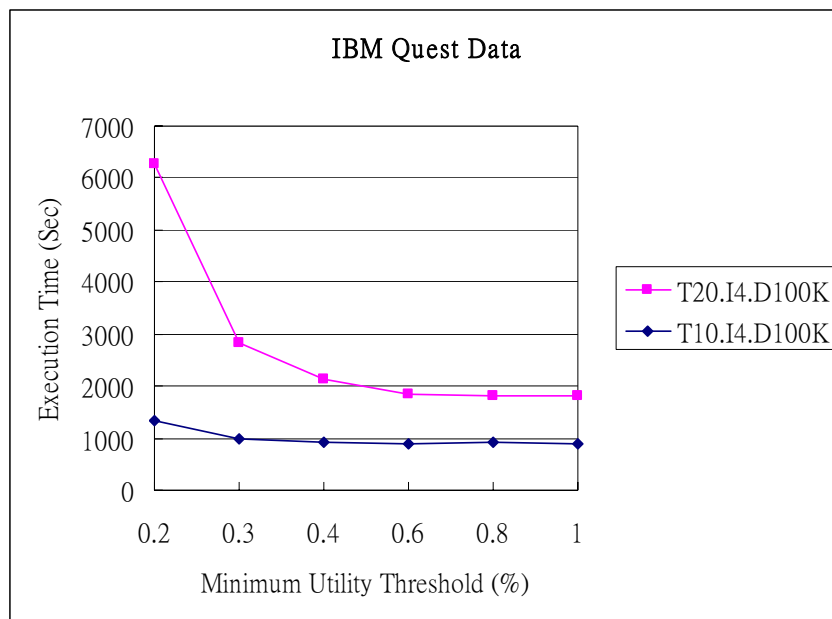


Figure 5-5. Execution time for HUIVIV on T20.I4.D100K and T10.I4.D100K.

Evaluation with Real Data



We also evaluate our algorithm *HUIVIV-Mine* with real data, BMS-POS. The BMS-POS dataset contains several years worth of point-of-sale data from a large electronics retailer. Since this retailer has so many different products, we used product categories as items. Each item thus represents a category, rather than an individual product. The transaction in this dataset is a customer's purchase transaction consisting of all product categories purchased at one time. The goal of this dataset is to find associations between product categories purchased by customers in a single visit to the retailer. Table 5-9 characterizes BMS-POS in terms of the number of transactions, the number of distinct items, the maximum transaction size, and the average transaction size.

This data set was used in the KDD-Cup 2000 competition and was recently made publicly available by Blue Martini Software (downloaded from <http://www.ecn.purdue.edu/KDDCUP>). In order to render databases suitable for utility

mining, we also randomly generate the quantity of each item in each transaction, ranging from 1 to 5. Utility tables are also synthetically created by assigning a utility value to each item randomly, ranging from 1 to 1000.

Table 5-10 shows the average number of candidates generated by *HUINIV-Mine* and MEU. The number of items is set at 1000, and the minimum utility threshold varies from 0.2% to 1%. The number of candidate itemsets generated by *HUINIV-Mine* during the first database scan decreases dramatically as the threshold increases. Particularly when the utility threshold is set to 1%, the number of candidate itemsets is generally 3789, including all various candidate itemsets within the database BMS-POS. However, the number of candidates generated by MEU is always 499,500 because it must process all combinations of 1000 items to generate only 2-candidate itemsets. It is observed that *HUINIV-Mine* still generates far fewer candidates when compared to MEU even using the real data. Hence, this result indicates that *HUINIV-Mine* is useful for mining high utility itemsets with negative item values from both artificial data and real data. Figure 5-6 shows the execution times for *HUINIV-Mine* as the minimum utility threshold is decreased from 1% to 0.2%.

Table 5-9. Database BMS-POS characteristics.

BMS-POS			
Transactions	Distinct items	Maximum Trans. Size	Average Trans. Size
515597	1657	164	6.5

Table 5-10. The number of candidate itemsets and high utility itemsets generated on database

BMS-POS.

Databases Threshold	BMS-POS		
	<i>HUINIV-Mine</i>	MEU	High Utility Itemsets
0.2%	59066	499500	151
0.3%	31485	499500	66
0.4%	19488	499500	34
0.6%	9603	499500	16
0.8%	5728	499500	7
1%	3789	499500	4

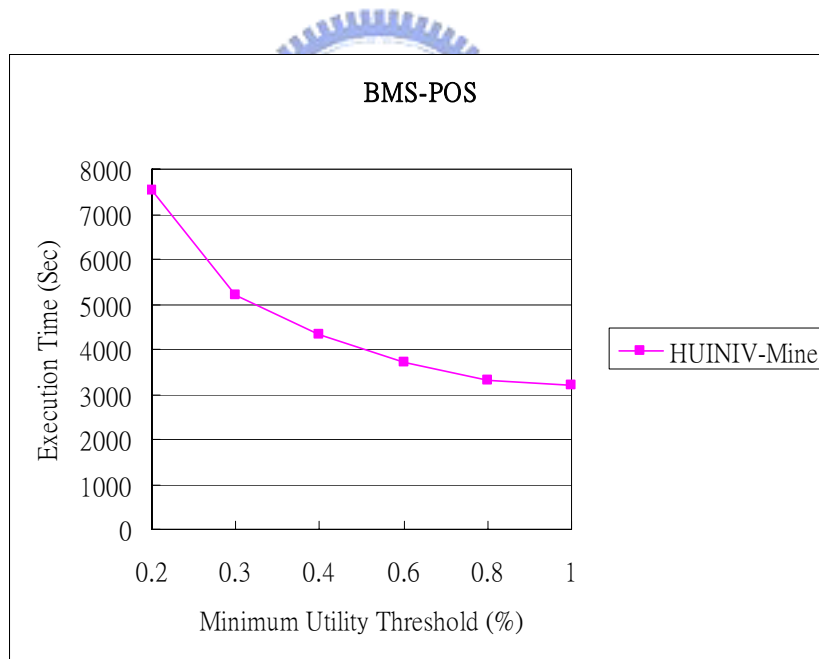


Figure 5-6. Execution time for HUINIV on BMS-POS.

Chapter 6

Conclusions and Future Work

The mining of association rules for finding the relationship between data items in large databases is a well studied technique in data mining field. For time-variant temporal databases, there is a strong demand to develop an efficient and effective method to mine various temporal patterns. However, most methods designed for the traditional databases cannot be directly applied for mining temporal patterns in temporal databases because of the high complexity. In many applications, the frequency of an itemset may not be a sufficient indicator of interestingness, because it only reflects the number of transactions in the database that contain the itemset. It does not reveal the *utility* of an itemset, which can be measured in terms of cost, profit or other expressions of user preferences. Hence, frequency is not sufficient to answer questions such as whether an itemset is highly profitable, or whether an itemset has a strong impact. Utility mining is thus useful in a wide range of practical applications. Therefore, we proposed a set of data mining methods for mining emerging frequent itemsets and high utility itemsets from temporal databases in this thesis.

First, we addressed the problem of discovering temporal emerging itemsets in temporal databases, i.e., the itemsets that are infrequent in current time window but have the high potential to become frequent in the subsequent time windows. We propose a new approach, namely *EFI-Mine*, which can discover emerging frequent itemsets from temporal databases efficiently and effectively. The novel contribution of *EFI-Mine* is that it can effectively identify the potential emerging itemsets such that the execution time can be reduced substantially in mining all frequent itemsets in temporal databases. The experimental results show that *EFI-Mine* can find the emerging frequent itemsets with high precision under different conditions like varied window size, transaction size and number of items, etc. This

also indicates that *EFI-Mine* fits for mining temporal databases with large window size transaction size and number of items. Moreover, it is highly efficient and scalable in terms of execution time. Hence, *EFI-Mine* promising for mining temporal emerging patterns in temporal databases.

Second, we addressed the problem of discovering temporal high utility itemsets in temporal databases. Under the stream database situation, the memory is often limited and it is hard to store large itemsets in memory. We propose a new algorithm, namely *THUI-Mine*, which can discover temporal high utility itemsets from temporal databases efficiently and effectively. The novel contribution of *THUI-Mine* is that it can effectively identify the temporal high utility itemsets with less candidate itemsets such that the execution time can be reduced efficiently. In this way, the process of discovering the temporal high utility itemsets in temporal databases can be achieved effectively with less memory space and execution time. This meets the critical requirements of time and space efficiency for mining temporal databases.



The experimental results show that *THUI-Mine* can discover the temporal high utility itemsets with higher performance by generating less candidate itemsets as compared to other algorithms under different experimental conditions, including both of sparse and dense datasets. Across the experiments, *THUI-mine* is faster than Two-Phase by 2 to 10 times, and the performance gain becomes more significant as the minimum utility threshold decreases. For example, *THUI-mine* is 10 times faster than Two-Phase when the threshold is 0.2 for dataset T20.I6.D100K.d10K. This performance enhancement comes mainly from the good feature of *THUI-mine* in producing far fewer candidate itemsets. Moreover, the experimental results also show that *THUI-Mine* is scalable with large databases. Therefore, it is indicated that the advantage of *THUI-Mine* over Two-Phase is stable and less execution time is taken as the amount of incremental portion of databases increases. Hence, *THUI-Mine* is promising for mining temporal high utility itemsets in temporal databases.

Third, we addressed the problem of discovering temporal rare utility itemsets in temporal databases, i.e., the itemsets that are larger than relative utility threshold in a current time window of the data stream. We propose two approaches, namely *TP-RUI-Mine* and *TRUI-Mine*, which can identify not only temporal high utility itemsets but also temporal rare utility itemsets from temporal databases both efficiently and effectively. To our best knowledge, this is the first work on mining temporal rare utility itemsets from temporal databases. The novel contribution of *TRUI-Mine* in particular is that it can effectively identify the temporal rare utility itemsets with less temporal high transaction-weighted utilization 2-itemsets such that the execution time can be reduced efficiently in mining all rare utility itemsets in temporal databases. In this way, the process of discovering all temporal rare utility itemsets under all time windows of temporal databases can be achieved effectively with limited memory space, less candidate itemsets and less CPU I/O. This meets the critical requirements of time and space efficiency for mining temporal databases.

The experimental results show that *TRUI-Mine* can find the temporal rare utility itemsets with higher performance by generating less candidate itemsets compared to *TP-RUI-Mine* under different experimental conditions. Moreover, it performs scalably in terms of execution time with large databases. Hence, *TRUI-Mine* is promising for mining temporal rare utility itemsets in temporal databases.

Fourth, we addressed the problem of discovering high utility itemsets with negative item values in large databases, i.e., the itemsets containing negative item values that are larger than threshold in large databases. We propose a new approach, namely *HUINIV-Mine*, which can identify high utility itemsets with negative item values in large databases both efficiently and effectively. The novel contribution of *HUINIV-Mine* is that it can effectively identify high utility itemsets with negative item values in less high TWUI such that the execution time can be reduced efficiently for mining all high utility itemsets with negative item values in large databases. In this way, the process of discovering all high utility itemsets containing negative

item values can be achieved effectively with limited memory space, less candidate itemsets and CPU I/O. This meets the critical requirements of time and space efficiency for mining high utility itemsets with negative item values.

The experimental results show that *HUINIV-Mine* can find high utility itemsets with negative item values with higher performance by generating fewer candidate itemsets compared to other algorithms under varied experimental conditions. It was found that *HUINIV-Mine* delivers an average improvement around 99.2% over MEU method in terms of execution performance. That is, the advantage of *HUINIV-Mine* over MEU is stable and less execution time is taken when the concept of negative item values is considered. Hence, *HUINIV-Mine* is promising for mining high utility itemsets in large databases with negative item values.

For the future work, we would extend the concepts of the proposed methods to discover other interesting patterns in temporal databases, like the frequent closed sets or other interesting patterns in temporal databases such as different ordered transactions. Moreover, we would explore to extend the concepts proposed in the thesis for discovering high utility itemsets with negative item values in temporal databases or data streams.

Bibliography

- [1] Agrawal, R., Imielinski, T., and Swami, A. Mining association rules between sets of items in large databases. *Proceedings of 1993 ACM SIGMOD International Conference on Management of Data*, pages 207--216, Washington, D. C., May 1993.
- [2] Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., and Verkamo, A. I. Fast discovery of association rules. *In book Advances in Knowledge Discovery and Data Mining*, pages 307--328. AAAI/MIT Press, 1996.
- [3] Agrawal, R., and Srikant, R. Mining Sequential Patterns. *Proceedings of the 11th International Conference on Data Engineering*, pages 3-14, March 1995.
- [4] Ayn, N.F., Tansel, A.U., and Arun, E. An efficient algorithm to update large itemsets with early pruning. *Technical Report BU-CEIS-9908 Dept of CEIS Bilkent University*, June 1999.
- [5] Ayn, N.F., Tansel, A.U., and Arun, E. An efficient algorithm to update large itemsets with early pruning. *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Diego, August 1999.
- [6] Bettini, C., Wang, X. S., and Jajodia, S. Testing complex temporal relationships involving multiple granularities and its application to data mining. *Proceedings of ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, June 3-5, 1996, Montreal, Canada, pages 68--78. ACM Press, 1996.
- [7] Brin, S., Motwani, R., Ullman, J., and Tsur, S., Dynamic Itemset Counting and Implication Rules for Market Basket Data, *Proceedings of ACM SIGMOD Conf. Management of Data*, May 1997.
- [8] Chan, R., Yang, Q., and Shen, Y. Mining high utility Itemsets. *Proc. of IEEE ICDM International Conference on Data Mining*, Florida, 2003.
- [9] Cheng, H., Yan, X., and Han, J., IncSpan: Incremental Mining of Sequential Patterns in Large Database. *International Conference on Knowledge Discovery and Data Mining (KDD'04)*, Seattle, WA, Aug. 2004.
- [10] Cheung, D., Han, J., Ng, V., and Wong, C.Y. Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique. *Proceedings of 1996 International Conference on Data Engineering*, pages 106--114, February 1996.
- [11] Cheung, D., Lee, S.D., and Kao, B., A General Incremental Technique for Updating Discovered Association Rules. *International Conference On Database Systems For Advanced Applications*, April 1997.

- [12] Chi, Y., Wang, H., Yu, P. S., and Richard, R. Muntz: Moment: Maintaining Closed Frequent Itemsets over a Stream Sliding Window. *Proceedings of the 2004 IEEE International Conference on Data Mining (ICDM'04)*.
- [13] Das, G., Lin, K. I., Mannila, H., Renganathan G., and Smyth, P. Rule Discovery from Time Series. *Proceedings of the 4th ACM SIGKDD*, pages 16—22, August 1998.
- [14] Dong G., and Li J. Efficient Mining of Emerging Patterns: Discovering Trends and Differences. *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Diego, August 1999.
- [15] Jin, C., Qian, W., Sha, C., Yu, J. X., and Zhou, A. Dynamically Maintaining Frequent Items over a Data Stream. *Proceedings of the 2003 ACM CIKM International Conference on Information and Knowledge Management*, November 2003.
- [16] Li, H.-F., Lee, S.-Y., and Shan, M.-K. An Efficient Algorithm for Mining Frequent Itemsets over the Entire History of Data Streams. *Proceedings of First International Workshop on Knowledge Discovery in Data Streams*, Pisa, Italy 2004.
- [17] Li, H.-F., Lee, S.-Y., Shan, and M.-K. Online Mining (Recently) Maximal Frequent Itemsets over Data Streams. *Proceedings of the 15th IEEE International Workshop on Research Issues on Data Engineering*, Tokyo, Japan, April 2005.
- [18] Lee, C. H., Lin, C. R., and Chen, M. S. Sliding-window filtering: An efficient algorithm for incremental mining. *International Conference on Information and Knowledge Management (CIKM01)*, pages 263—270, November 2001.
- [19] Lin, C.H., Chiu, D.Y., Wu, Y.H., and Chen, A.L.P. Mining Frequent Itemsets from Data Streams with a Time-sensitive Sliding Window. *Proceedings of 2005 Science and Industry Advance with Mathematics (SIAM) International Conference on Data Mining*, CA, April 2005.
- [20] Lin, J. L., and Dunham. M. H. Mining Association Rules: Anti-Skew Algorithms. *Proceedings of 1998 International Conference on Data Engineering*, pages 486—493, 1998.
- [21] Liu, Y., Liao, W., and Choudhary. A. A Fast High Utility Itemsets Mining Algorithm. *Proceedings of ACM KDD Utility-Based Data Mining Workshop (UBDM'05)*, August 2005.
- [22] Manku, G. S., and Motwani, R. Approximate Frequency Counts over Data Streams. *Proceedings of the 28th International Conference on Very Large Data Bases*, Hong Kong, China, 2002.
- [23] N. Pasquier., Y. Bastide., R. Taouil., and L. Lakhal. Discovering Frequent Closed Itemsets for Association Rules, *Seventh International Conference on Database Theory*, Jan. 1999.

- [24] Park, J. S., Chen, M. S., and Yu. P. S. Using a Hash-Based Method with Transaction Trimming for Mining Association Rules. *IEEE Transactions on Knowledge and Data Engineering*, 9(5):813—825, October 1997.
- [25] Parthasarathy, S., Zaki, M., Ogihara, M., and Dwarkadas, S. Incremental and interactive sequence mining. *Proceedings of the 8th International Conference on Information and Knowledge Management (CIKM'99)*, Nov 1999.
- [26] Pei. J., Han. J., and Mao. R. Closet: An Efficient Algorithm for Mining Frequent Closed Itemsets, *SIGMOD International Workshop Data Mining and Knowledge Discovery*, May 2000.
- [27] Savasere, A., Omiecinski, E., and Navathe, S. An Efficient Algorithm for Mining Association Rules in Large Databases. *Proceedings of the 21th International Conference on Very Large Data Bases*, pages 432—444, September 1995.
- [28] Susan P. Imberman., Abdullah Uz Tansel., and Eric Pacuit. An Efficient Method For Finding Emerging Frequent itemsets. *3rd International Workshop on Mining Temporal and Sequential Data*, pages 112—121 August 2004.
- [29] Tan, S., Xu, H., and Gu, J. Efficient Algorithms for Attributes Reduction Problem. *International Journal of Innovative Computing, Information and Control* vol.1, no.4, pp.767-777, December 2005.
- [30] Teng, W. G., Chen, M. S., and Yu, P. S. A Regression- Based Temporal Pattern Mining Scheme for Data Streams. *Proceedings of the 29th International Conference on Very Large Data Bases*, pages 93–104, September 2003.
- [31] Teng, W. G., Chen, M. S., and Yu, P. S. Resource-Aware Mining with Variable Granularities in Data Streams. *Proceedings of 2004 Science and Industry Advance with Mathematics (SIAM) International Conference on Data Mining (SDM) 2004*.
- [32] Tseng, V. S., Chu, C. J., and Liang, T. Efficient Mining of Temporal High Utility Itemsets from Data streams. *Proceedings of ACM KDD Workshop Utility-Based Data Mining (UBDM'06)*, Philadelphia, Pennsylvania, USA, August, 2006,.
- [33] Tseng, V. S., Chu, C. J., and Liang, T. An Efficient Method for Mining Temporal Emerging Itemsets From Data Streams. *International Computer Symposium (ICS), Workshop on Software Engineering, Databases and Knowledge Discovery*, Taipei, Taiwan, Dec. 2006.
- [34] Y. Bastide., R. Taouil., N. Pasquier., G. Stumme., and L. Lakhal. Mining Frequent Patterns with Counting Inference, *International Conference on Knowledge Discovery and Data Mining*, vol. 2, no. 2, Dec. 2000.

- [35] Yao, H., Hamilton, H. J., and Butz, C. J. A Foundational Approach to Mining Itemset Utilities from Databases. *Proceedings of 2004 Science and Industry Advance with Mathematics (SIAM) International Conference on Data Mining (SDM) 2004*.
- [36] Yim, H., Ha, D., Hwang, B., and Ryu, K. H. Mining association rules on significant rare data using relative support. *Journal of Systems and Software*. Vol. 67, no. 3, pp. 181-191. Sept, 2002.
- [37] Zaki, M. J., and Hsiao, C. J. Efficient Algorithm for Mining Closed Itemsets and Their Lattice Structure. *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 3, pages 462–478, April 2005.
- [38] Zhang, Y. Z., Yun, X. C., Fang, B. X., and Zhang, T. A Mining Method for Computer Vulnerability Correlation. *International Journal of Innovative Computing, Information and Control* vol.1, no.1, pp.43-51, March 2005.

