

國立交通大學

資訊科學與工程研究所

博士論文

漸進式探勘與多維度即時探勘之研究

A Study of Incremental Mining and
Multidimensional Online Mining for Knowledge
Discovery in Database



研究生: 王慶堯

指導教授: 曾憲雄 博士

中華民國九十四年十月

漸進式探勘與多維度即時探勘之研究

A Study of Incremental Mining and Multidimensional Online Mining for Knowledge Discovery in Database

研 究 生: 王慶堯

Student: Ching-Yao Wang

指導教授: 曾憲雄

Advisor: Dr. Shian-Shyong Tseng

國立交通大學

資訊工程系

博士論文



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

October 2005

Hsinchu, Taiwan, Republic of China

中華民國九十四年十月

漸進式探勘與多維度即時探勘之研究

學 生: 王慶堯

指導教授: 曾憲雄

國立交通大學 資訊工程系

摘要

近年來,利用資料探勘技術從大型資料庫和資料倉儲中發掘出隱含且有用的知識變得愈來愈重要。然而,大部分傳統的資料探勘方法都是採用批次處理的方式,當面對資料庫中資料新增時,就必須耗時地對整個資料庫重新進行處理,以更新之前所獲得的知識;此外,這些方法通常對資料一視同仁地處理,鮮少考慮資料所生成之相關背景資訊,乃至所獲取的知識無法滿足使用者特定的需求、提供線上決策支援。因此在本篇論文中,我們將發展一些新穎的漸進式演算法,分別對關聯法則、循序樣式和文件分類器進行知識更新,減少每當資料新增時,必須對整個資料庫重新進行處理的龐大成本;此外,為了讓資料探勘能夠即時地提供線上決策支援,我們進而將漸進式演算法加以延伸,並將資料所生成之相關背景資訊納入考量,利用所發展之結構化儲存體,系統化地儲存之前所獲得的知識和其相關的背景資訊,以滿足使用者特定的知識查詢和多維度即時探勘的需求;最後,我們企圖將所發展的漸進式探勘與多維度即時探勘的技術應用到半導體製程,幫助偵測並發掘出造成良率偏低的瑕疵機器。

關鍵詞: 關聯法則、循序樣式、漸進式探勘、封閉項目及、準大項目集、文件分類、文件表示、限制式探勘、瑕疵偵測、重要性測量法。

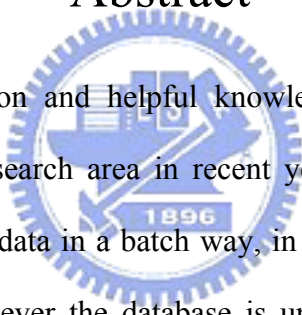
A Study of Incremental Mining and Multidimensional Online Mining for Knowledge Discovery in Database

Student: Ching-Yao Wang

Advisor: Dr. Shian-Shyong Tseng

Department of Computer Science
National Chiao Tung University

Abstract

The logo of National Chiao Tung University is a circular emblem with a gear-like border. Inside the circle, there is a stylized blue figure that appears to be a person or a symbol, and the year '1896' is written at the bottom of the inner circle.

Mining useful information and helpful knowledge from large databases has evolved into an important research area in recent years. However, most of classic mining approaches processed data in a batch way, in which they must re-process the entire updated database whenever the database is updated, and focused on finding rules or patterns in a specified part of a database, in which they can not consider problems at different aspects and provide on-line decision supports. This seems to be inefficient and insufficient for knowledge discovery process in real-world applications. As a result, in this dissertation, we will develop some novel incremental mining algorithms respectively for maintaining association rules, sequential patterns and a document classifier without re-processing the original database whenever the database is updated. For providing ad-hoc, query-driven and online mining supports, we will extend the concept of *effectively utilizing patterns previously discovered* in incremental mining to support online mining under multidimensional considerations. Specifically, we propose a structural repository to systematically store the previously

mined information from each inserted dataset, and then develop online mining approaches to acquire user-interesting rules or patterns by integrating related mining information in the repository. Consequently, we will attempt to apply incremental mining and multidimensional online mining techniques on knowledge discovery process in semiconductor manufacture for quickly identifying *root-cause machinesets*, the major killer machine(s) that causes a low-yield situation in a regular manufacturing procedure.

Keywords: association rule, sequential pattern, incremental mining, closed itemsets, pre-large itemset, text classification, document representation, constraint-based mining, defect detection, interestingness measurement.



誌謝

還記得五年前，帶著忐忑不安的心情踏入交大校園，愁著博士班入學考試及不可預知的未來。午夜夢迴、慕然回首，資格考前的挑燈苦讀、研究方向的反覆思索、論文的枕日修訂、口試前的緊繃情緒，一切仍歷歷在目、彷彿昨日。

今日，看著自己畢業海報的張貼，內心五味雜陳，與其說是喜悅，不如說是感動。這一人生經歷，一步步踏過之足跡，實實在在。其中，最感謝的是在這茫茫學習與研究大海中，指引我方向的明燈，我的恩師 曾憲雄 博士。他不僅在研究領域上培養我獨立思考、自我突破與批判能力，更透過產學界的合作，培養我領導、掌控與解決問題的能力，開闊我的人生視野。其次，感謝從碩士班以來，一直協助我在研究領域上撐舵揚帆的 洪宗貝 博士，他那鉅細靡遺的研究精神，幫助我更精準地掌握研究的方向、更清楚地表達研究的內容和成果，建立我應有的研究態度。

在博士論文口試其間，感謝 李素瑛 博士和 胡毓志 博士從一開始論文計畫書、校內口試到校外口試，給予的鼓勵和指導，啟發我能從其他觀點去檢閱自己研究上的盲點和不足；感謝校內口試時 孫春在 博士提供在研究方法與內容闡述上的建議，讓我深刻體會過程比成果更為重要；感謝校外口試時清華大學 陳良弼 博士與 蘇豐文 博士以及高雄大學 洪宗貝 博士，給予的精闢見解與建議，才能讓本篇論文更加完整。

這五年多的求學生涯，每一屆從四面八方來到實驗室的研究伙伴，感謝你們在我研究歲月中所留下的美好回憶，447 實驗室、346 會議室、籃球場、羽球場、……，這份誠摯的同袍情誼，我將永感於心。

對於家人的支持以及鼓勵我的女友 麗煌，這一份感謝，非筆墨所能形容。在我面對課業與研究壓力時、當我生活不如意時、當我快樂歡笑時，當我需要一份關愛時、 ，你們總是陪伴在我身邊，無怨無悔，為我打氣加油，給我安慰給我愛，謝謝你們。

僅將本篇論文的完成，獻給每一位給予我幫助及支持我的人。

Table of Contents

Abstract (In Chinese)	I
Abstract (In English)	II
Acknowledgement	IV
Table of Contents	V
List of Figures	VII
List of Tables	X
Chapter 1 Introduction	1
1.1 Motivation.....	1
1.2 Contribution	8
1.3 Reader’s Guide.....	10
Chapter 2 Incremental Mining Algorithms for Association Rules	
Maintenance	11
2.1 Introduction.....	11
2.2 Related Work.....	14
2.3 Preliminary Concepts.....	18
2.4 Closed Itemsets Maintenance	20
2.5 The Closed Itemsets Maintaining (CIM) Algorithm.....	24
2.6 The CIM Algorithm with Pre-large Concept: CIM-P Algorithm.....	34
2.7 Experimental Results	39
2.8 Conclusion	45
Chapter 3 Incremental Mining Algorithms for Sequential Patterns	
Maintenance	47
3.1 Introduction.....	47
3.2 Related Work.....	48
3.3 Preliminary Concepts.....	54
3.4 Closed Sequences Maintenance	56
3.5 The Closed Sequences Maintaining (CSM) Algorithm	59
3.6 The CSM Algorithm with Pre-large Concept: CSM-P Algorithm	60
3.7 Conclusion	63
Chapter 4 Incremental Mining Algorithms for Document Classifiers	
Maintenance	64
4.1 Introduction.....	64
4.2 Related Work.....	66
4.3 Domain-space Weighting Scheme for Document Classification.....	69
4.4 Classifier Construction Based on Domain-space Document Representation	70

4.5 Document Labeling by the Constructed Classifier	80
4.6 Experimental Results	81
4.7 Conclusions	88
Chapter 5 From Incremental Mining to Multidimensional Online Mining for Knowledge Discovery.....	89
5.1 Introduction.....	89
5.2 Related Work.....	91
5.3 Knowledge Warehouse	93
5.4 Online Knowledge Discovery System (OKDS)	96
Chapter 6 Multidimensional Online Mining Algorithms for Generation of Association Rules.....	99
6.1 Introduction.....	99
6.2 Related Work.....	100
6.3 Multidimensional Pattern Relation (MPR)	101
6.4 Three-Phased Online Association Rule Mining (TOARM) based on MPR	103
6.5 Negative-Border Online Mining (NOM) based on Extended MPR (EMPR)	110
6.6 LNOM: Algorithm Design and Implementation.....	118
6.7 Experimental Results	131
6.8 Conclusion	142
Chapter 7 Using Association Rule Mining Techniques on Knowledge Discovery Process in Semiconductor Manufacture	144
7.1 Introduction.....	144
7.2 Related Work.....	145
7.3 Root-cause Machineset Identification Problem.....	147
7.4 Root-cause Machine Identifier (RMI) Approach.....	148
7.5 The Concepts of Progressive RMI (PRMI) and Multidimensional RMI (MRMI).....	158
7.6 Experimental Results	159
7.7 Conclusion	163
Chapter 8 Summary and Future Work.....	165
Reference.....	168

List of Figures

Figure 1-1: The performance and storage trade-off for batch mining, incremental mining and multidimensional online mining	3
Figure 2-1: Four cases of candidate itemsets	16
Figure 2-2: Four cases of joint closed itemsets	22
Figure 2-3: The CIM algorithm	25
Figure 2-4: A closed maintenance tree (CMT)	27
Figure 2-5: The <i>CO_generation</i> subroutine	29
Figure 2-6: An example of branch-wise processing strategy in the <i>CO_generation</i> subroutine	30
Figure 2-7: An example of updating process in the <i>CO_generation</i> subroutine	30
Figure 2-8: The <i>CP_generation</i> subroutine	33
Figure 2-9: An example of <i>CP_generation</i> subroutine	34
Figure 2-10: The concept of pre-large closed itemsets	35
Figure 2-11: The CIM-P algorithm	39
Figure 2-12: Execution times for the FUP, Pre-large and CIM algorithms respectively on the five datasets	42
Figure 2-13: The amounts of pre-stored mining information for the FUP, Pre-large and CIM algorithms respectively on the five datasets	43
Figure 2-14: The influence of the size of increment on the execution time for the FUP, Pre-large and CIM algorithms	44
Figure 2-15: Execution times for the CIM and CIM-P algorithms on <i>BMS-POS</i>	45
Figure 3-1: Four cases of candidate sequences	53
Figure 3-2: Four cases of joint closed sequences	57
Figure 3-3: The CSM algorithm	60
Figure 3-4: The CSM-P algorithm	63
Figure 4-1: An example of the support vector machine approach	68
Figure 4-2: The classifier construction algorithm	72
Figure 4-3: The operation of the classifier construction algorithm	72
Figure 4-4: The training algorithm	74
Figure 4-5: The discrimination algorithm	76
Figure 4-6: The tuning algorithm	79

Figure 4-7: The document labeling algorithm	80
Figure 4-8: Micro-averaging F_1 value vs. number of tuning documents for Reuters-21578(10)	85
Figure 4-9: Micro-averaging F_1 values vs. number of tuning documents at $\varphi = 1$, $\varphi = 15$ and $\varphi = 25$ for Reuters-21578(90)	86
Figure 4-10: Micro-averaging F_1 values vs. number of tuning documents at $\varphi = 1$, $\varphi = 15$ and $\varphi = 25$ for Reuters-21578(115)	87
Figure 4-11: Computation times spent by the batch-based classifier and the incremental-based classifier for reuters-21578(10)	87
Figure 5-1: An example of the star schema of a knowledge warehouse.....	95
Figure 5-2: The OKDS architecture.....	96
Figure 6-1: The TOARM algorithm.....	109
Figure 6-2: The graph model of candidate itemsets for Tuple 4 in Table 6-4.....	121
Figure 6-3: The STCC algorithm.....	122
Figure 6-4: The directed minimum spanning tree found from Figure 6-2.....	123
Figure 6-5: The lattice to represent the candidate itemsets illustrated in Example 6-6	124
Figure 6-6: The hash table derived from the candidate itemsets illustrated in Example 6-6	126
Figure 6-7: The updated lattice after processing all matched tuples.....	129
Figure 6-8: The algorithm of the L NOM approach with a direct hashing function...	131
Figure 6-9: Execution times for the TOARM, Apriori, Partition and FUP algorithms on Groups 1, 2, 3 and 5	134
Figure 6-10: The influence of the number of negative itemsets on execution time for Groups 1 to 6.....	137
Figure 6-11: Execution times of the NOM algorithm respectively with and without a direct hashing function on Groups 1 to 4	139
Figure 6-12: Execution times spent by the NOM and L NOM algorithms on Groups 1 to 4	140
Figure 6-13: Execution times for the TOARM, Apriori, Partition and FUP algorithms on Group 7	141
Figure 6-14: Execution times spent by the NOM and L NOM algorithms on Groups 7	142
Figure 7-1: A general manufacturing process	147

Figure 7-2: The flowchart of the RMI approach.....150
Figure 7-3: Execution times for Case 2, Case 3 and Case 7 with the minimum defect coverage set from 0.3 to 0.6.....162



List of Tables

Table 2-1: A transactional database.....	26
Table 2-2: The newly inserted transactions.....	30
Table 2-3: Characteristics of the experimental datasets.....	39
Table 2-4: Mining information for the five datasets	40
Table 2-5: The distribution of frequent itemsets for the five datasets	40
Table 3-1: The sequence database.....	51
Table 3-2: All frequent sequences generated for the sequences in Table 3-1	51
Table 3-3: Two new transactions sorted according to <i>Sequence_id</i> and <i>Trans_time</i>	51
Table 3-4: The two newly merged sequences	51
Table 3-5: The candidate 1-sequences with their support counts for newly merged sequences	52
Table 4-1: An example of a feature-domain weighting table.....	70
Table 4-2: The statistic information of features in “DM” Category	75
Table 4-3: The feature weights in “DM” Category	75
Table 4-4: An example of the tuning algorithm	79
Table 4-5: Micro- and macro-averaging F_1 values shown in [21]	82
Table 4-6: Micro- and macro-averaging F_1 values at $\varphi = 1$, $\varphi = 15$ and $\varphi = 25$	83
Table 4-7: Micro- and macro-averaging F_1 values at various δ for Reuters-21578(10)	84
Table 4-8: Micro-averaging F_1 values at various δ and φ for Reuters-21578(90).....	84
Table 4-9: Macro-averaging F_1 values at various δ and φ for Reuters-21578(90)	84
Table 4-10: Micro-averaging F_1 values at various δ and φ for Reuters-21578(115)	84
Table 4-11: Macro-averaging F_1 values at various δ and φ for Reuters-21578(115).....	85
Table 4-12: Numbers of remaining categories at various φ	85
Table 5-1: Differences between the operational database and the data warehouse	92
Table 5-2: Differences between the knowledge warehouse and the data warehouse	94
Table 6-1: An MPR with minimum support = 5%	102
Table 6-2: Matched tuples in Example 6-2	105
Table 6-3: An EMPR with minimum support = 5%	112
Table 6-4: The matched tuples in Example 6-6	112
Table 6-5: Parameters considered when generating datasets.....	132

Table 6-6: The six groups of synthetic datasets	133
Table 6-7: Mining information for the six groups	133
Table 6-8: The numbers of candidate itemsets for Group 5.....	135
Table 6-9: The numbers of candidate itemsets for Group 2.....	135
Table 6-10: Mining information for the seventh group	141
Table 7-1: A manufacturing process relation for six products in a five-stage manufacturing procedure	148
Table 7-2: An example of the machine-oriented preprocessing procedure.....	151
Table 7-3: An example of the stage-oriented preprocessing procedure.....	152
Table 7-4: Defect coverage and defective product information for each 1-machineset in Table 7-3	153
Table 7-5: Defect coverage and defective product information for each candidate 1-machineset obtained	154
Table 7-6: Defect coverage and defective product information for each 2-machinesets generated.....	154
Table 7-7: Defect coverage and defective product information for each candidate 2-machineset obtained	154
Table 7-8: Defect coverage and defective product information for each candidate machinesets obtained	155
Table 7-9: Calculated continuities for each candidate machineset in Table 7-8.....	157
Table 7-10: ϕ' for each candidate machinesets in Table 7-8.....	157
Table 7-11: Relevant information for the nine real datasets	159
Table 7-12: Accuracy results of the RMI approach for the nine datasets	160
Table 7-13: Accuracy results of the RMI approach on the nine datasets for interestingness measurements confidence, ϕ and ϕ'	163

Chapter 1

Introduction

1.1 Motivation

Data mining technology has become increasingly important in the field of large databases and data warehouses. This technology helps discover non-trivial, implicit, previously unknown and potentially useful knowledge, thus being able to aid managers in making good decision [4][18][38]. Years of effort in data mining have produced a variety of efficient techniques. Depending on the type of databases processed, these mining approaches may be classified as working on transaction databases, temporal databases, relational databases, and multimedia databases. On the other hand, depending on the classes of knowledge derived, the mining approaches may be classified as finding association rules, sequential patterns, classifiers (classification models), etc.

(1) Association rules: Recently, mining association rules from transaction databases has been one of the most interesting and popular research topics in data mining. An association rule indicates a relationship among items such that the occurrence of certain items in a transaction would imply the occurrence of some other items in the same transaction. For example, an association rule for a supermarket may be *“people often buy beer and diapers together in the same transaction”*. The discovery of interesting association rules can help decision-making processes in many potential applications, such as manufacturing defect detection, catalog design, store layout, cross-marketing, etc.

(2) Sequential patterns: Mining sequential patterns attempts to find customer purchase sequences in temporal transaction databases (sequence database), and to predict whether there is a high probability that when customers buy some products, they will buy some other products in later transactions. For example, a sequential pattern for a video shop may be “*a customer buys a television in one transaction; he/she then buys a video recorder in a later transaction within a month*”. As a result, sequential patterns are also treated as inter-transaction association rules. The discovery of interesting sequential patterns can not only model customer behaviors, but also predict weather, identify symptoms in medicine, diagnose alarms in intrusion detection, etc.

(3) Classifiers (classification models): Classification is the process of mining a classifier from a set of pre-defined training data that can describe and distinguish data classes or concepts, such that the found classifier can assign a class or concept to a new un-defined data. In general, classification (mining a classifier) involves three major tasks: *data representation*, which represents data in machine-readable structures, *classifier construction*, which constructs a classifier from a set of training data, and *classifier evaluation*, which evaluates classifier accuracy with a set of testing data and in terms of various evaluation functions. Classification has been popularly applied on document classification/management, insurance risk analysis, credit approval, medical diagnosis, etc.

In our view of points on the evolution of knowledge discovery in database, the first part of this dissertation will indicate the challenges from *batch mining* evolving into *incremental mining* and propose our solutions especially for the three above-mentioned classes of knowledge; then the second part of this dissertation will indicate the importance from *incremental mining* evolving into our proposed

multidimensional online mining and propose our methodologies especially for online generation of association rules. Figure 1-1 shows the performance and storage trade-off for batch mining, incremental mining and multidimensional online mining. Finally, the third part of this dissertation will indicate the issues of knowledge discovery in semiconductor manufacture and attempt to integrate incremental mining and multidimensional online mining techniques dealing with them.

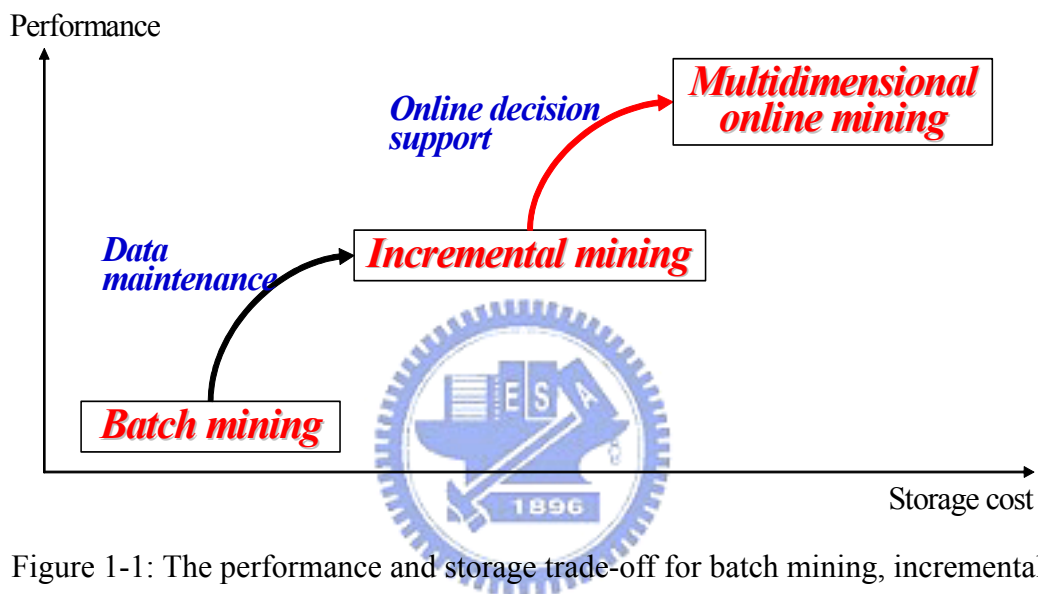


Figure 1-1: The performance and storage trade-off for batch mining, incremental mining and multidimensional online mining

Most of classic mining approaches process data in a batch way and must re-process the entire updated database whenever the database is updated, since the mined rules or patterns may become invalid or new implicitly valid rules or patterns may appear in the resulting updated database. As a result, two drawbacks may occur in maintaining database knowledge:

(a) Nearly the same computation time as that spent in mining the original database is needed. It is time-consuming and unpractical when the original database is large.

(b) Information previously mined from the original database, such as frequent itemsets and association rules, provides no help in the maintenance process.

In the first part of this dissertation, we will propose some novel incremental mining algorithms respectively for maintaining association rules, sequential patterns and a document classifier without re-processing the original database whenever the database is updated. The proposed algorithms continue using the common idea of previous incremental mining algorithms that the previously mined information should be utilized as much as possible. Furthermore, we utilize the concepts of *pre-large patterns* and *closed patterns* to improve the performance of developed algorithms for maintaining association rules and sequential patterns.

(a) Using the pre-large patterns to enlarge the amount of pre-stored mined information can reduce the cost of re-processing the original database at the expense of storage spaces, because they acts as a buffer to avoid the movements of patterns directly from valid to invalid and vice-versa when the database maintained.

(b) Using the closed patterns instead of the pre-stored mined information can reduce the comparison cost and redundant rules generated, because they can determine all the pre-stored mined patterns and their exact support without loss of any information but are orders of magnitude smaller than all pre-stored patterns.

Consequently, based on the two concepts, the CIM (*Closed Itemsets Maintenance*) and CIM-P (CIM with *Pre-large concept*) algorithms are developed to efficiently maintain association rules and the CSM (*Closed Sequences Maintenance*) and CSM-P (CSM with *Pre-large concept*) algorithms are developed to efficiently maintain sequential patterns.

As for the developed algorithm for maintaining a document classifier, in document representation, we propose a *domain-space document representation* to

represent documents in finite sets of domains. This representation is more compact and representative than classical *term-space document representation*. Based on the domain-space document representation, in classifier construction, we design a *feature-domain weighting table* to structurally retain the weights between features and all involved domains for later maintenances. Consequently, the *domain-space weighting scheme* algorithm is developed to resolve the document representation and categories adaptation problems.

Although incremental mining algorithms are rather efficient and useful for static models such as *mining all the data accumulated thus far* and *mining only a recently collected portion of data* in uncomplicated applications, they usually provide little support for user guidance and focus (e.g., limiting the computation to what interests the user) and user interaction (e.g., dynamically changing the parameters or constraints). This may neither flexibly obtain rules or patterns from their interesting portions of data, nor diversely consider problems at different aspects to provide on-line decision supports for users.

In the second part of this dissertation, we will extend the concept of *effectively utilizing previously discovered patterns* in incremental mining to support multidimensional online mining. The concept of *knowledge warehouse*, which is similar to the construction of a data warehouse for OLAP except it is not used to store data but mined patterns, and the architecture of *Online Knowledge Discovery System* (OKDS), which automatically and systematically mines patterns from data gathered in different contexts and forwards mined patterns into the knowledge warehouse, are proposed to help decision-makers diversely consider problems at different aspects and provide online mining supports.

For efficiently manipulating the mining information in the knowledge warehouse,

we focus on the knowledge class of association rules and design corresponding aggregation and generalization approaches to provide online mining supports on association rules. We first propose the *multidimensional pattern relation* (MPR) as a knowledge warehouse to structurally and systematically store context information, such as *region, time* and *branch*, and mining information, such as *the set of previously mined frequent itemsets with their supports*, for each inserted block of data (each increment of data). Based on the proposed MPR, we then develop an aggregation and generalization approach called *Three-phased Online Association Rule Mining* (TOARM) to support online generation of association rules under multidimensional considerations. By the TOARM approach, users can therefore acquire interesting and/or focused association rules or frequent itemsets by only integrating related mining information in the MPR rather than mining the underlying data. In addition, we further apply the concept of *negative border* to extend the mining information in the MPR, and develop a *Negative-Border Online Mining* (NOM) approach based on the *extended* MPR (EMPR) to improve the performance of TOARM especially for heterogeneous blocks of data.

However, from the experimental results, we can find that the NOM approach may take much computation time than the TOARM approach, especially when the numbers of itemsets kept in EMPR and candidate itemsets to be considered are large. For overcoming this problem, we thus try to use appropriate data structures to improve the performance of the NOM approach. The *lattice* data structure is utilized to organize and maintain all candidate itemsets such that the candidate itemsets with the same proper subsets can be considered at the same time. The derived *lattice-based* NOM (LNOM) approach will require only one scan of the itemsets stored in EMPR, thus saving much computation time. In addition, a hashing technique is used to further

improve the performance of the NOM approach since many itemsets stored in EMPR may be useless for calculating the counts of candidates. At last, experimental results show the effect of the improved NOM approaches.

In the third part of this dissertation, we will attempt to apply the proposed incremental mining and multidimensional online mining techniques on knowledge discovery process in semiconductor manufacture. For a semiconductor manufacturing company, one of the most essential issues is to quickly identify *root-cause machinesets*, and to meet high-yield target expectations by remedying these abnormal machines. Therefore, we first define the *root-cause machineset identification problem*, and propose the *Root-cause Machine Identifier* (RMI) approach using a batch-based association rule mining algorithm to obtain *candidate root-cause machinesets* from a shipment of *wafer in process* (WIP) data. After that, we propose the *progressive* RMI (PRMI) concept, which applies incremental mining techniques to progressively process previously mined candidate root-cause machinesets, and the *multidimensional* RMI (MRMI) concept, which designs a knowledge warehouse to structurally and systematically store the context information about a shipment and the mining information about mined candidate root-cause machinesets from each shipment for supporting decision-makers diversely considering problems at different aspects.

In this dissertation, we attempt to make data mining techniques more robust and practical for real-world applications. Experiments respectively for sparse, dense, synthetic and real datasets are made, with results showing the effectiveness and practicality of the proposed approaches.

1.2 Contribution

In the first part of this dissertation, the major contributions are as follows:

- The concepts of *pre-large patterns* and *closed patterns* have been utilized to improve the performance of developed algorithms for maintaining association rules and sequential patterns.
- Two novel incremental mining algorithms called *Closed Itemsets Maintenance* (CIM) and CIM with *Pre-large concept* (CIM-P) have been developed to efficiently maintain association rules.
- Two novel incremental mining algorithms called *Closed Sequences Maintenance* (CSM) and CSM with *Pre-large concept* (CSM-P) have been developed to efficiently maintain sequential patterns.
- The *domain-space weighting scheme* has been developed to represent documents in domain-space and incrementally construct a classifier to resolve the document representation and categories adaptation problems.

In the second part of this dissertation, the major contributions are as follows:

- The concept of *knowledge warehouse* and the architecture of *Online Knowledge Discovery System* (OKDS) have been proposed to help decision-makers diversely consider problems at different aspects and provide online mining services.
- For the knowledge class of association rules, the *multidimensional pattern relation* (MPR) has been designed as a knowledge warehouse to structurally and systematically store the context and mining information.
- The *Three-phased Online Association Rule Mining* (TOARM) approach, which is an aggregation and generalization approach corresponding to the proposed

MPR, has been developed to support online generation of association rules under multidimensional considerations.

- The concept of *negative border* has been used to extend the mining information in the MPR, and then the *Negative-Border Online Mining* (NOM) approach based on the *extended* MPR (EMPR) has been developed to improve the performance of TOARM especially for heterogeneous blocks of data.
- The *lattice-based* NOM (LNOM) approach and the hashing technique have been developed to improve the NOM approach.

In the third part of this dissertation, the major contributions are as follows:

- Identifying *root-cause machinesets*, the most likely sources of defective products, in the manufacturing processes has been defined as the *root-cause machineset identification problem* of analyzing correlations between combinations of machines.
- The *Root-cause Machine Identifier* (RMI) approach, which uses a batch-based association rule mining algorithm, has been developed to provide an efficient and effective solution for the root-cause machineset identification problem.
- The concepts of *progressive* RMI (PRMI), which applies incremental mining techniques to progressively process previously mined candidate root-cause machinesets, and *multidimensional* RMI (MRMI), which applies multidimensional online mining techniques to support online generation of candidate root-cause machinesets under multidimensional consideration, have been proposed.

1.3 Reader's Guide

The remainder of this dissertation is organized as follows. In the first part of this dissertation, we will propose some novel incremental mining algorithms respectively for maintaining association rules, sequential patterns and a document classifier. The proposed incremental mining algorithms for association rules maintenance are described in Chapter 2; the proposed incremental mining algorithms for sequential patterns maintenance are described in Chapter 3; and the proposed incremental mining algorithm for a document classifier maintenance is described in Chapter 4. In the second part of this dissertation, we will extend the concept of *effectively utilizing previously discovered patterns* in incremental mining to support multidimensional online mining. The concept of *knowledge warehouse* and the architecture of *Online Knowledge Discovery System* (OKDS) are proposed in Chapter 5. The proposed aggregation and generalization approaches, TOARM, NOM and LNOM, based on the two forms of knowledge warehouse, MPR and EMPR, are described in Chapter 6. In the third part of this dissertation, we attempt to apply the association rule mining techniques, including classical batch-based, incremental and multidimensional online mining algorithms on knowledge discovery process in semiconductor manufacture. The *Root-cause Machine Identifier* (RMI) approach and the two concepts of *progressive RMI* (PRMI) and *multidimensional RMI* (MRMI) are proposed in Chapter 7. Conclusions and future work are given in Chapter 8.

Chapter 2

Incremental Mining Algorithms for Association Rules Maintenance

2.1 Introduction

Data mining technology has become increasingly important in the field of large databases and data warehouses. This technology helps discover non-trivial, implicit, previously unknown and potentially useful knowledge, thus being able to aid managers in making good decision [4][18][38]. Among various types of databases and mined knowledge, mining association rules [3][5] from transaction databases is the most interesting and popular. In general, the process of mining association rules can roughly be decomposed into two tasks: *finding frequent itemsets* satisfying the user-specified *minimum support* threshold from a given database and *generating interesting association rules* satisfying the user-specified *minimum confidence* threshold from found frequent itemsets. Since the first task is very time-consuming when compared to the second one, the major challenges in mining association rules thus focus on how to reduce the search space and decrease the computation time in the first task. Some famous mining approaches, such as Apriori [5], DIC [16], DHP [67], Partition [78], Sampling [61] and FP-Growth [40][95], have been proposed.

In real-world applications, a database grows over time such that existing association rules may become invalid or new implicitly valid association rules may appear. Recently, some researchers [8][20][21][27][43][44][77] have developed

incremental mining algorithms to maintain association rules without reprocessing the entire updated database. The common idea of these researches lies in that, the previously mining information such as mined frequent itemsets are stored in advance; when new transactions are inserted, (a) a large portion of candidate itemsets can be decided using the pre-stored mined frequent itemsets; (b) only a small portion of candidate itemsets obtained from the new transactions without sufficient information needs to be reprocessed against the original database. Task (a) is responsible for updating previously mined frequent itemsets (known association rules), and Task (b) is responsible for finding new frequent itemsets (unknown association rules). Much computation time can thus be saved in this way.

However, for a dense database such as census data and DNA sequences or a low minimum support threshold, the computation cost of Task (a) will be getting tremendous due to a huge amount of previously mined frequent itemsets. For example, a frequent 30-itemset (a frequent itemset consisting of 30 items) implies the presence of $2^{30}-2$ additional frequent itemsets as well. The performance of classically incremental mining algorithms will degrade dramatically. On the other hand, one scan of original database is required for dealing with Task (b) by most incremental mining algorithms. When the original database is massive, this will result in excessive I/O cost. As a result, in this study, we attempt to utilize the concepts of *closed itemsets* and *pre-large itemsets* to overcome the two challenges, respectively.

In a dense database, many itemsets usually appear together, and we can consider them together. The concept of *closed itemsets* [68], which is denoted as the itemsets having no proper superset with the same support, can be treated as a lossless compression for all itemsets in the database. It can also reduce redundant rules generated [104]. Therefore, using the set of frequent closed itemsets instead of the set

of frequent itemsets from the original database as the pre-stored mining information can increase both efficiency and effectiveness of an incremental mining algorithm. The set of frequent closed itemsets can easily determine all the frequent itemsets and their exact supports, and its order of magnitude is smaller than the set of all frequent itemsets for dense databases.

In general, the number of newly inserted transactions is much smaller than the number of records in the original database. Only the candidate itemsets whose supports are slightly less than the minimum support threshold in the original database are possible to be frequent after database maintenances. The concept of *pre-large itemsets* [43] is denoted as the set of itemsets having support between a lower support threshold, which is smaller than the given minimum support threshold, and an upper support threshold, which is equal to the given minimum support threshold. Therefore, using the pre-large closed itemsets to enlarge the amount of pre-stored frequent closed itemsets can reduce the cost of reprocessing the entire database at the expense of storage spaces. This is because they act as a buffer to avoid the movement of a closed itemset directly from infrequent to frequent and vice-versa during the incremental mining process.

Although using the concept of closed itemsets can effectively reduce the number of itemsets considered, some closed itemsets for the updated database called *joint closed itemsets*, which was not closed itemsets in both the original database and the newly inserted transactions before, cannot be determined by above-mentioned Tasks (a) and (b) of a classically incremental mining algorithm such that some valid association rules may be lost. We thus propose a novel incremental mining algorithm called *Closed Itemsets Maintaining* (CIM) to extend Tasks (a) and (b) that can sufficiently and efficiently find all up-to-date association rules for the updated

database. Task (a) of the CIM algorithm is responsible for extracting the joint closed itemsets which was *absorbed (closed)* by the pre-stored frequent closed itemsets in the original database before, and updating them and all the pre-stored frequent closed itemsets against the newly inserted transactions. Task (b) of the CIM algorithm is responsible for generating the candidate itemsets for the updated database which has not been determined in Task (a), and rescanning them against the original database. Furthermore, based on the concept of pre-large itemsets, we propose the CIM-P (CIM with *Pre-large concept*) algorithm to reduce the cost of Task (b) in the CIM algorithm. Also, we design the *bucketing* strategy to improve the utility of buffer in the CIM-P algorithm. The consumption of buffer can be rigidly calculated using the maximum value of buckets.

2.2 Related Work

2.2.1 Closed itemsets mining approaches

The major challenge in mining association rules is to reduce the search space and decrease the computation time required for mining frequent itemsets. The Apriori algorithm [5], which is the most well-known, utilizes a level-wise candidate generation approach to reduce its search space such that only frequent itemsets found in the previous level are treated as seeds for generating candidate itemsets in the current level. Many later algorithms [16][53][61][67][78][95] were based on this property and attempted to further reduce candidate itemsets and I/O costs. However, this Apriori property can not work well for dense databases or a low minimum support threshold. This is because most generated candidate itemsets are also frequent itemsets such that the number of frequent itemsets will grow up explosively; the performance of an Apriori-like algorithm thus degrades dramatically.

Some researchers have then developed closed itemsets mining algorithms to reduce the number of itemsets generated. Examples include A-close [68], CLOSET [69], CLOSET+ [86] and CHARM [104]. The A-close algorithm is an Apriori-like algorithm using a breadth-first search manner to find frequent closed itemsets directly. However, breadth-first searches may encounter difficulties since there could be many candidates generated and need to scan the database many times. The CLOSET algorithm, an extension of the FP-growth algorithm, uses a depth-first search (recursive divide-and-conquer) manner and a database-projection approach to mine long patterns from the FP-tree (frequent pattern tree) structure representing all transactions of database. However, the CLOSET algorithm may suffer from a sparse database or a low minimum support threshold. An enhancement of the CLOSET algorithm, the CLOEST+ algorithm, thus combines various known search manners and closure-testing strategies to improve the performance of CLOSET. The CHARM algorithm uses a dual itemsets-tidset search tree and the *Diffset* technique to enumerate closed itemsets from a vertical-layout database.

2.2.2 Incremental mining approaches

Conventional batch-mining algorithms do not utilize previously mined patterns for later maintenance, and may require considerable computation time to reprocess the entire updated database to get all up-to-date association rules. Some researchers have developed incremental mining algorithms to maintain association rules without reprocessing the entire database whenever the database is updated. Examples include the FUP-based algorithms [20][21], an adaptive algorithm [77], an incremental mining algorithm based on the concept of *pre-large itemsets* [43], and an incremental updating technique based on the concept of *negative border* [27][85]. The common

idea of these researches lies in that, the previously mining information such as mined frequent itemsets are stored in advance; when new transactions are inserted, a large portion of candidate itemsets can be decided by using the pre-stored frequent itemsets; only a small portion of candidate itemsets obtained from the new transactions needs to be reprocessed against the original database. Much computation time can thus be saved in this way. The correctness of this idea is simply illustrated as follows.

Considering an original database and the newly inserted transactions, there are four cases of candidate itemsets shown in Figure 2-1 may arise:

- Case 1: A candidate itemset is frequent in both the original database and the newly inserted transactions;
- Case 2: A candidate itemset is frequent in the original database but infrequent in the newly inserted transactions;
- Case 3: A candidate itemset is infrequent in the original database but frequent in the newly inserted transactions;
- Case 4: A candidate itemset is infrequent in both the original database and the newly inserted transactions.

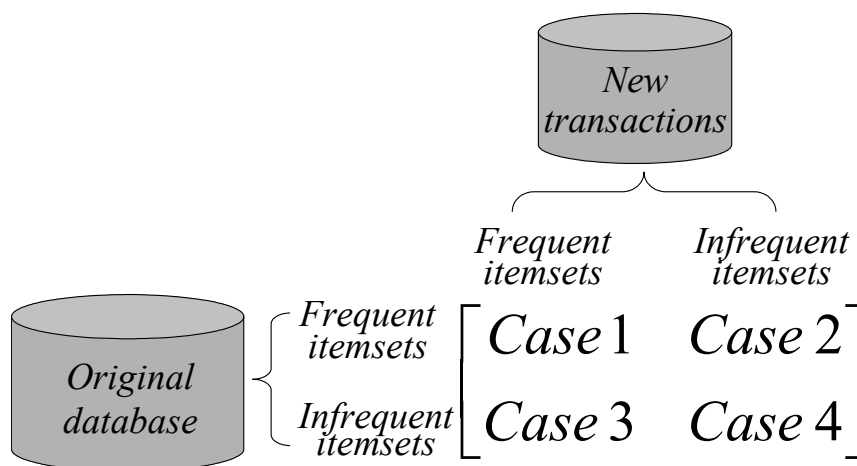


Figure 2-1: Four cases of candidate itemsets

Among the cases, since candidate itemsets in Case 1 are frequent in both the original database and the new transactions, they are still frequent after the weighted average of the supports; similarly, candidate itemsets in Case 4 are still infrequent after the new transactions are inserted. Cases 1 and 4 will not affect the final association rules; Case 2 may remove existing association rules; and Case 3 may generate new association rules.

Cheung and his co-workers proposed an incremental mining algorithm, called FUP (Fast UPdate algorithm) [20][21], to efficiently cope with these four cases by pre-storing the previously mined frequent itemsets from the original database. It handles Cases 1, 2 and 4 by updating the pre-stored frequent itemsets against the newly inserted transactions, and reprocesses only the itemsets without sufficient information in Case 3 against the original database if necessary.

The performance of the FUP algorithm will get degraded if a lot of candidate itemsets from the newly inserted transactions belong to Case 3. Thomas et al. [85] and Feldman et al. [27] thus utilized the concept of *negative border* [67] to enlarge the amount of pre-stored mining information in the FUP algorithm for improving the maintenance performance. A negative border of frequent itemsets can be easily formed by excluding the set of frequent itemsets from the set of candidate itemsets generated level by level. In other words, the negative border consists of the itemsets which are candidates but do not have enough supports. The processing time for Case 3 in the FUP algorithm can be reduced by additionally keeping the negative border of frequent itemsets. Similarly, Hong et al. [43] proposed the concept of *pre-large itemsets* to enlarge the amount of pre-stored mining information for improving the maintenance performance. The proposed algorithm does not need to reprocess the

original database until a number of new transactions have been inserted.

2.3 Preliminary Concepts

Let $I = \{i_1, i_2, \dots, i_m\}$ be a set of m items. A subset X of I consisting of k items is called a k -itemset. Let D be a transactional database consisting of a set of transactions, where each transaction T consisting of a set of items of I is associated with an identifier called TID , and $|D|$ denotes the number of transactions in D . A transaction T is said to contain X if and only if $X \subseteq T$. The support of an itemset X , $X.sup$, in D is denoted as the percentage of transactions in D which contain X ; the support count of X , $X.count$, in D is denoted as the number of transactions in D which contain X , $X.count = X.sup * |D|$. For the itemsets in D , X is called a **closed itemset** if there does not exist an itemset Y which *closes* (*absorbs*) X , where an itemset Y is said to close (absorb) X iff $X \subseteq Y$ and $X.sup = Y.sup$ ($X.count = Y.count$). CI denotes the set of all closed itemsets in D . Furthermore, if there is no superset of X existing in D , X is also called a **maximal itemset**.

An **association rule** is an implication of the form $X \Rightarrow Y$, where X and Y are subset of I , and $X \cap Y = \phi$. The support of a rule $X \Rightarrow Y$, $(X \cup Y).sup$, in D is denoted as the percentage of transactions in D which contain $X \cup Y$, and the confidence of $X \Rightarrow Y$ is computed by $(X \cup Y).sup / X.sup$. Given the user-specified minimum support threshold, $minsup$, and minimum confidence threshold, $minconf$, the problem of mining association rules is to find out all association rules in D that have support and confidence larger than $minsup$ and $minconf$, respectively. With respect to the $minsup$, the set of **frequent itemset**, FI , includes all the itemsets whose support is larger than $minsup$; the set of **infrequent itemset**, NI , includes all the itemsets whose support is less than $minsup$; the set of **frequent closed itemset**, FCI , includes all the closed

itemsets whose support is larger than $minsup$, $FCI = \{x|x \in CI, x.sup \geq minsup\}$; and the set of **infrequent closed itemset**, NCI , includes all the closed itemsets whose support is less than $minsup$, $NCI = \{x| x \in CI - FCI\}$. Note that FCI includes no itemset which has a superset with the same support, and thus $FCI \subseteq FI$. The problem of mining association rules can be reduced to the problem of finding FI or FCI in D .

Let d be an increment of new transactions which is inserted into the original database D , $|d|$ be the number of transactions in d , D^+ be the updated database which denotes $D \cup d$, and $|D^+|$ be the number of transactions in $D \cup d$. Therefore, FI_D , FI_d and CI_{D^+} denote the FI obtained from D , d and D^+ with respect to the same $minsup$, respectively, and FCI , NI , $NFCI$ or CI obtained from D , d and D^+ can have similar meanings. The problem of maintaining association rules is to find FI_{D^+} or FCI_{D^+} . Let the set of **original frequent itemsets**, O , be defined as $O = \{x|x \in FI_D\}$, and the set of **potential frequent itemsets**, P , be defined as $P = \{x|x \in FI_d - FI_D\}$. By definition, an itemset $X \in FI_{D^+}$ must belong to $O \cup P$, and thus the problem of maintaining association rules is equivalent to processing $O \cup P$. Similarly, let the set of **closed original frequent itemsets**, CO , be defined as $CO = \{x|x \in FI_D \text{ and } x \in CI_{D^+}\}$, and the set of **closed potential frequent itemsets**, CP , be defined as $CP = \{x|x \in FI_d - FI_D \text{ and } x \in CI_{D^+}\}$. The problem of maintaining association rules is also equivalent to processing $CO \cup CP$. The set of **joint closed itemsets**, JCI , which is defined as $JCI = \{x|x = y \cap z, y \in CI_D, z \in CI_d\}$, is proposed in this study and can be divided into four parts based on FCI_D , FCI_d , NCI_D and NCI_d :

- $FFJCI = \{x|x = y \cap z, y \in FCI_D, z \in FCI_d\}$;
- $FNJCI = \{x|x = y \cap z, y \in FCI_D, z \in NCI_d\}$;
- $NFJCI = \{x|x = y \cap z, y \in NCI_D, z \in FCI_d\}$;

- $NNJCI = \{x|x = y \cap z, y \in NCI_D, z \in NCI_d\}$.

2.4 Closed Itemsets Maintenance

Considering an original database D and the newly inserted transactions d , there are four cases of candidate itemsets for the updated database D^+ have been discussed in Section 2. With pre-storing previously mined frequent itemsets FI_D , a typically incremental mining process can efficiently cope with these four cases by two steps: (a) *updating O against d* and (b) *reprocessing P against D* . Following this idea, we can use two similar steps: (a) *updating CO against d* and (b) *reprocessing CP against D* to find out FCI_{D^+} dealing with the problem of maintaining association rules. However, directly obtaining $CO = \{x|x \in FI_D \text{ and } x \in CI_{D^+}\}$ and $CP = \{x|x \in FI_d - FI_D \text{ and } x \in CI_{D^+}\}$ is impractical because CI_{D^+} is unknown before processing D^+ . In the following, we attempt to utilize the pre-stored known information FCI_D from D and the information FCI_d obtained from d to approach CO and CP .

Lemma 2-1: If $x \in CI_D \cup CI_d$, then $x \in CI_{D^+}$.

Proof: We prove the lemma by contradiction. If $x \notin CI_{D^+}$, there must exist a proper superset y of x such that $y.sup_{D^+} = x.sup_{D^+}$, i.e., $y.sup_D * |D| + y.sup_d * |d| = x.sup_D * |D| + x.sup_d * |d|$. Thus $y.sup_D = x.sup_D$ and $y.sup_d = x.sup_d$, contradicting the claim that $x \in CI_D \cup CI_d$. Thus, $x \in CI_{D^+}$. ■

Let FCI_{d-D} denote $FCI_d - FCI_D$. According to Lemma 2-1, we have $FCI_D \subseteq CI_D \subseteq CI_{D^+}$ and $FCI_{d-D} \subseteq CI_d \subseteq CI_{D^+}$. FCI_D and FCI_{d-D} are both closed itemsets in D^+ . If an incremental mining algorithm can utilize FCI_D and FCI_d to obtain CO and CP , the problem of maintaining association rules in a dense database can be efficiently coped with. We first discuss the differences between FCI_D and CO and between FCI_{d-D} and CP . For example, given $D = \{ABCE, CD, BCE\}$, $d = \{ABCDE, CDE\}$ and $minsup =$

0.6, $FI_D = \{B, C, E, BC, BE, CE, BCE\}$, $FI_d = \{C, D, E, CD, CE, DE, CDE\}$, $FCI_D = \{C, BCE\}$ and $FCI_d = \{CDE\}$. By definitions, $FCI_{d-D} = \{CDE\}$, $CO = \{C, CE, BCE\}$ and $CP = \{CD, CDE\}$. As shown in this example, there exist some closed itemsets in CI_{D+} but not in CI_D or CI_d , such that FCI_D and FCI_{d-D} may be not equivalent to CO and CP . The following lemmas are used to derive the set of *joint closed itemsets* (JCI) which are closed itemsets for $D+$ but can not be determined by FCI_D and FCI_{d-D} .

Lemma 2-2: If $x \in JCI$, then $x \in CI_{D+}$.

Proof: If $x \in JCI$, x must be one of following two cases.

Case 1: If $x \in CI_D \cup CI_d$, then $x \in CI_{D+}$ according to Lemma 2-1;

Case 2: If $x \notin CI_D \cup CI_d$, there exist $y \in CI_D$ and $z \in CI_d$ such that $x \subset y$, $x \subset z$, and x is closed by both y and z . We prove this case by contradiction. If $x \notin CI_{D+}$, there must exist a proper superset x' of x such that $x'.sup_{D+} = x.sup_{D+}$, i.e., $x'.sup_D * |D| + x'.sup_d * |d| = x.sup_D * |D| + x.sup_d * |d| = y.sup_D * |D| + z.sup_d * |d|$. Thus $x' \subset y$, $x' \subset z$ (because $x'.sup_D = y.sup_D$ and $x'.sup_d = z.sup_d$) and $x' = y \cap z$, contradicting the claim that $x \in JCI$. Thus, $x \in CI_{D+}$. ■

Lemma 2-3: If $x \in CI_{D+}$, then $x \in CI_D \cup CI_d \cup JCI$.

Proof: If $x \in CI_{D+}$ and $x \notin CI_D \cup CI_d$, x must be closed in both D and d . Assume y is the itemset that closes x in D and z is the itemset that closes x in d . Then $x.sup_{D+} * |D^+| = y.sup_D * |D| + z.sup_d * |d|$. If $y \subseteq z$, x is belonging to Case 1 of Lemma 2-2, contradicting the claim that $x \notin CI_D$; if $z \subseteq y$, x is also belonging to Case 1 of Lemma 2-2, contradicting the claim that $x \notin CI_d$. Thus $y \not\subseteq z$ and $z \not\subseteq y$. According to Case 2 of Lemma 2-2, there must exist $x' = y \cap z$ and $x' \in CI_{D+}$. If $x \subset x'$, x is closed by x' (because $x'.sup_{D+} = x.sup_{D+}$), contradicting the claim that $x \in CI_{D+}$. Thus, $x = x'$ and $x \in JCI$. ■

Theorem 2-1: $CI_{D^+} = CI_D \cup CI_d \cup JCI$.

Proof: According to Lemmas 2-1 and 2-2, we have $(CI_D \cup CI_d \cup JCI) \subseteq CI_{D^+}$.

On the other hand, according to Lemma 2-3, we have $CI_{D^+} \subseteq (CI_D \cup CI_d \cup JCI)$.

Thus, $CI_{D^+} = CI_D \cup CI_d \cup JCI$. ■

Considering an original database and the newly inserted transactions, JCI can be divided into four parts based on FCI_D , FCI_d , NCI_D and NCI_d as shown in Figure 2-2:

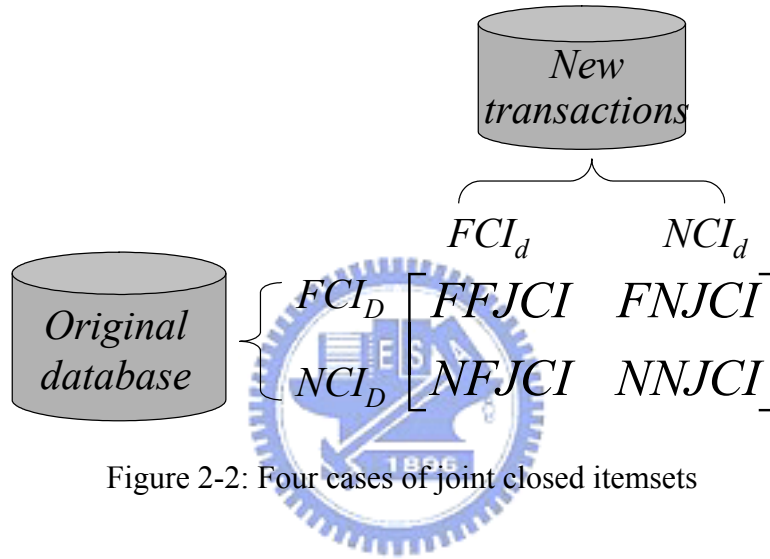


Figure 2-2: Four cases of joint closed itemsets

- The case of *FFJCI*: A closed itemset is frequent in both the original database and the newly inserted transactions;
- The case of *FNJCI*: A closed itemset is frequent in the original database but infrequent in the newly inserted transactions;
- The case of *NFJCI*: A closed itemset is infrequent in the original database but frequent in the newly inserted transactions;
- The case of *NNJCI*: A closed itemset is infrequent in both the original database and the newly inserted transactions.

Since the closed itemsets in *FFJCI* are frequent in both the original database and the new transactions, they will still be frequent in the updated database. Similarly, the

closed itemsets in $NNJCI$ will still be infrequent in the updated database. $FFJCI$ and $NNJCI$ will not affect the final association rules. $FNJCI$ may remove existing association rules, and $NFJCI$ may add new association rules.

According to Theorem 2-1, the following theorems are derived to obtain CO and CP by FCI_D , FCI_d , $FFJCI$, $FNJCI$ and $NFJCI$.

Theorem 2-2: $CO = \{x|x \in FCI_D \cup FFJCI \cup FNJCI\}$.

Proof: By definition, CO collects the closed itemsets for D^+ which is generated from FI_D . According to Theorem 2-1, $CO = \{x|x \in FI_D \text{ and } x \in CI_{D^+}\} = \{x|x \in FI_D \text{ and } x \in CI_D \cup CI_d \cup JCI\} = \{x|x \in FCI_D \cup FFJCI \cup FNJCI\}$. ■

Theorem 2-3: $CP = \{x|x \in (FCI_d - FFJCI) \cup NFJCI\}$.

Proof: By definition, CP collects the closed itemsets for D^+ which is generated from $FI_d - FI_D$. As known in Theorem 2-2, $FCI_d \cup FFJCI \cup NFJCI$ is the set of closed itemsets for D^+ which is generated from FI_d . Thus $CP = \{x|x \in FI_d - FI_D \text{ and } x \in CI_{D^+}\} = \{(FCI_d \cup FFJCI \cup NFJCI) - (FCI_D \cup FFJCI \cup FNJCI)\} = \{x|x \in FCI_d \cup FFJCI \cup NFJCI - FFJCI\} = \{x|x \in (FCI_d - FFJCI) \cup NFJCI\}$. ■

In contrast to the definitions of CO and CP , Theorems 2-2 and 2-3 provide a convenient way to obtain CO and CP . For CO , $FFJCI$ and $FNJCI$ can be obtained by processing the pre-stored mining information FCI_D against d . For CP , however, since $NFJCI$ has to be generated from NCI_D , which is usually unknown in a typically incremental mining process, the cost is too expensive to be acceptable. As a result, given a function $cover(FFJCI, FI_d)$ denoting the itemsets in FI_d which are covered by $FFJCI$, the following theorem is derived to obtain CP .

Theorem 2-4: $CP = \{x|x \in FI_d - cover(FFJCI, FI_d), x \in CI_{D^+}\}$.

Proof: By definition, the $FFJCI$ covers the itemsets which are included both in FI_d and FI_D . Thus $CP = \{x|x \in FI_d - FI_D \text{ and } x \in CI_{D^+}\} = \{x|x \in FI_d - cover(FFJCI,$

$FI_d), x \in CI_{D^+}\}$. ■

Corollary 2-1: $CP \subseteq \{FI_d - \text{cover}(FFJCI, FI_d)\}$ ■

Since $FFJCI$ has been obtained in CO generation, we only need to find FI_d and remove the itemsets in FI_d which have been determined in $FFJCI$ as candidates for CP . It seems to be a better way to generate the itemsets of FCI_{D^+} which are not included in the CO .

2.5 The Closed Itemsets Maintaining (CIM) Algorithm

We develop a novel incremental mining algorithm mainly consisting of $CO_generation$ and $CP_generation$ subroutines, called *Closed Itemsets Maintaining* (CIM), to efficiently find FCI_{D^+} . Also, an in-memory data structure called *Closed Maintenance Tree* (CMT) is proposed in the CIM algorithm to facilitate the processes of $CO_generation$ and $CP_generation$ subroutines. The CIM algorithm first updates the itemsets in the CMT against d to obtain CO by the $CO_generation$ subroutine. Then, by the $CP_generation$ subroutine, it generates candidate itemsets for the itemsets of FCI_{D^+} which have not been determined in the $CO_generation$ subroutine. Finally, by reprocessing these obtained candidate itemsets against D and checking their closure property, the CIM algorithm can find FCI_{D^+} from the CMT. Details of the CMT data structure, the $CO_generation$ and $CP_generation$ subroutines are described in Section 2.5.1 to Section 2.5.3.

The CIM algorithm($CMT, D, d, minsup$)

Parameters:

- CMT : A closed maintenance tree;
- D : An original database;
- d : A set of newly inserted transactions;
- $minsup$: A minimum support threshold.

```

Begin
Set  $FFJCISet = \phi$ , /*  $FFJCISet$  is a set used to store the
itemsets of  $FFJCI$ . */
Set  $Cand = \phi$ , /*  $Cand$  is a set used to store candidate
itemsets for  $FCI_{D^+}$ . */
 $CO\_generation$  subroutine( $CMT, d, minsup, FFJCISet, Cand$ );
Set  $F1_{dD^+} = \phi$ , /*  $F1_{dD^+}$  is a set used to store the frequent
1-itemsets in both  $d$  and  $D^+$ . */
Set  $mincount_{D^+} = minsup * (|D| + |d|)$ ;
Obtain_frequent_items( $CMT, mincount_{D^+}, F1_{dD^+}$ );
/* Obtain  $F1_{dD^+}$  from  $CMT$ . */
 $CP\_generation$  subroutine( $CMT, d, minsup, FFJCISet, Cand, F1_{dD^+}, CMT.root$ );
Reprocess_Cand( $CMT, Cand, D$ ); /* Reprocess obtained candidate  $k$ -itemsets
( $k \geq 2$ ) in  $CMT$  against  $D$ . */
Check_Closure_Cand( $CMT, Cand$ ); /* Check closure property for all candidates
itemsets in  $CMT$ . */
Remove_NCI( $CMT, mincount_{D^+}$ ); /* Remove the closed itemsets in  $CMT$ 
whose support counts are less than
 $mincount_{D^+}$ . */
Output_FCI( $CMT$ ); /* Output  $FCI_{D^+}$  for  $D^+$ . */
End.

```

Figure 2-3: The CIM algorithm

Theorem 2-5: The CIM algorithm can correctly obtain FCI_{D^+} .

Proof: As mentioned above, an incremental mining algorithm can use two steps: *updating CO against d* and *reprocessing CP against D* to find out FCI_{D^+} dealing with the problem of maintaining association rules. According to Theorem 2-2 and Corollary 2-1, since the CIM algorithm can maintain CO and candidate itemsets for CP in the CMT by the $CO_generation$ and $CP_generation$ subroutines, the CIM algorithm can correctly obtain FCI_{D^+} from the CMT. ■

2.5.1 The Closed Maintenance Tree (CMT)

A *Closed Maintenance Tree* (CMT) which is a tree structure like a *prefix tree* [1] is constructed as follows. For each itemset x , a corresponding node v_x is built in the CMT. Each node maintains its corresponding itemset with support count, denoted as $(itemset, support\ count)$. For each pair of nodes v_x and v_y corresponding to itemsets x and y , there is a directed edge from v_x to v_y if x is a *parent* of y . x is said to be a parent of y if y can be obtained by adding a new item to x , and inversely, y is said to be a child of x . Therefore, an itemset has only one parent and more than one child in the constructed CMT. Note that, the itemsets in a CMT are usually maintained in lexical order, and for saving the storage space, each node maintains only the suffix of an itemset which is regarding the itemset in its parent node. There are three types of nodes in a CMT:

- *Closed nodes*: the nodes represent the itemsets in FCI_D ;
- *Prefix-unclosed nodes*: the nodes represent the common prefixes of closed nodes;
- *Infrequent nodes*: the nodes represent infrequent 1-itemsets in D .

Among them, in particular, prefix-unclosed nodes are used to improve the searching performance of CMT, and infrequent nodes are used to reduce useless item combinations in the $CP_generation$ subroutine.

Table 2-1: A transactional database

<i>TID</i>	<i>Items</i>
100	A, C, D
200	B, C, E
300	A, B, C, E
400	B, E

Example 2-1: Given a transactional database as shown in Table 2-1, Figure 2-4 shows an example of CMT based on $minsup = 0.5$. The prefix-unclosed node $(B, 3)$

and the closed node $(CE, 2)$ stand for the closed itemset $(BCE, 2)$; $(B, 3)$ and $(E, 3)$ stand for the closed itemset $(BE, 3)$. The CMT maintains only one infrequent node $(D, 1)$. ■

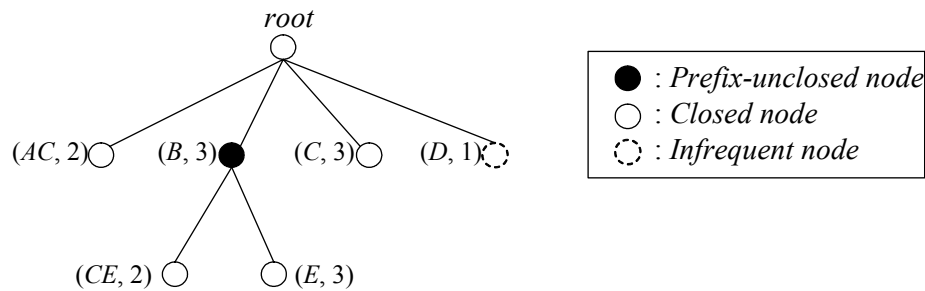


Figure 2-4: A closed maintenance tree (CMT)

2.5.2 The $CO_generation$ Subroutine of the CIM Algorithm

The $CO_generation$ subroutine is responsible for processing FCI_D against d to find $FFJCI$ and $FNJCI$, thus obtaining CO . In that, finding $FNJCI$ is the most concerned because most itemsets in NI_d are irrelative and useless. In order to reduce useless item combinations of NI_d , the $CO_generation$ subroutine adopts the *branch-wise processing strategy* to process a given CMT against d as follows. The $CO_generation$ subroutine operates from the most left branch to the most right branch in the CMT. If a branch consists of only one item x maintained in an infrequent node v_x , the $CO_generation$ subroutine updates x 's support count against d , and keeps x in a set used to store candidate itemsets for FCI_{D+} if x 's support count is not less than $minsup * |D^+|$. Detailed usage of this candidate set will be described in Section 5.3. Otherwise, for each of the other branches, which consists of closed nodes, the $CO_generation$ subroutine uses the items belonging to the branch, i.e., the items of the maximal itemset in the branch, as seeds to mine the closed itemsets in d by a

closed itemsets mining approach (such as the CHARM algorithm). Moreover, a checking mechanism is used to reduce duplicate item combinations which have been considered by a processed branch. Since the *CO_generation* subroutine considers only the items in a branch at a time, useless item combinations belonging to NI_d can be effectively reduced. The performance of *CO_generation* subroutine is greatly improved. After all branches have been processed, the *CO_generation* subroutine then updates found itemsets against CMT to obtain *CO*. Assume y is an itemset in the CMT, z is one of the found itemsets in d , and $x = y \cap z$. The *CO_generation* subroutine can find *FFJCI* and *FNJCI* by updating x with support count calculated by y 's support count + z 's support count. The updated CMT thus contains the entire *CO*.

CO_generation subroutine(CMT, d, minsup, FFJCISet, Cand)

Parameters:

CMT: The closed maintenance tree;
d: The newly inserted transactions;
minsup: The minimum support threshold;
FFJCISet: The set used to store the itemsets of *FFJCI*;
Cand: The set used to store candidate itemsets for FCI_{D^+} .

Begin

```

Set  $T = \emptyset$ ;                                     /*  $T$  is a set used to store the mining results
                                                    by the branch-wise processing strategy. */
for each item  $a_i$  only appears  $d$ , do           /* Insert each new item  $a_i$  in CMT. */
    insert  $a_i$  with  $a_i.count = 0$  into CMT;
for each branch  $b_i \in CMT$ , do
    if  $b_i$  consists of only one infrequent item  $x$ , then
        update  $x.count$  against  $d$ ;                /*  $x.count$  denotes  $x$ 's support count. */
        if  $x.count \geq minsup * |D^+|$ , then
            insert  $x$  with  $x.count$  into Cand;
    else if  $b_i \neq null$  and  $b_i$  is not contained by a processed branch  $b_j$ , then
        Closed_itemset_mining( $b_i, d, T$ ); /* Execute a closed itemsets mining
                                                    algorithm and store mining results into  $T$ . */
 $y = CMT.get\_first\_CI()$ ;                          /* Fetch the first closed itemset by lexical

```

```

order in CMT. */
z = T.get_first_CI();          /* Fetch the first closed itemset by lexical
                                order in T. */

while y ≠ null and z ≠ null, do
  if y = z, then
    y.count = y.count + z.count;
    if z.count ≥ minsup*|d|, then
      insert y with y.count into FFJCISet;
      y = CMT.get_next_CI(y);      /* Fetch the next closed itemset by lexical
                                    order in CMT. */

      z = T.get_next_CI(z);        /* Fetch the next closed itemset by lexical
                                    order in T. */

  else if y ∩ z = y, then
    y.count = y.count + z.count;
    if z.count ≥ minsup*|d|, then
      insert y with y.count into FFJCISet;
      y = CMT.get_next_CI(y);

  else if y ∩ z = z then
    if z.count ≥ minsup*|d|, then
      insert z with (y.count + z.count) into FFJCISet;
      z.count = y.count + z.count;
      insert z with z.count into CMT;
      z = T.get_next_CI(z);

  else if y ∩ z = x and x ≠ null then /* x ⊂ y and x ⊂ z. */
    if CMT.exist(x) = false, then
      x.count = y.count + z.count;
      insert x with x.count into CMT;
      if z.count ≥ minsup*|d|, then
        insert x with x.count into FFJCISet;
        y = CMT.get_next_CI(y);
    else if (y.count + z.count) > x.count, then
      x.count = y.count + z.count;
      if z.count ≥ minsup*|d|, then
        insert x with x.count into FFJCISet;
        y = CMT.get_next_CI(y);

End.

```

Figure 2-5: The *CO_generation* subroutine

Theorem 2-6: The algorithm of *CO_generation* subroutine can correctly obtain *CO*.

Proof: For a branch of the given CMT, by using the items of the branch as seeds to process *d*, the *CO_generation* subroutine can find the closed itemsets in *d* which are subsets of one of the frequent closed itemsets in the branch. After all branches have been processed, it is easily seen that these found closed itemsets in *d* can be used to obtain the entire $FFJCI \cup FNJCI$ by updating them against the frequent closed itemsets in the CMT. The updated CMT thus contains the entire $FCI_D \cup FFJCI \cup FNJCI$. ■

Table 2-2: The newly inserted transactions

TID	Items
500	B, C, D
600	C, D

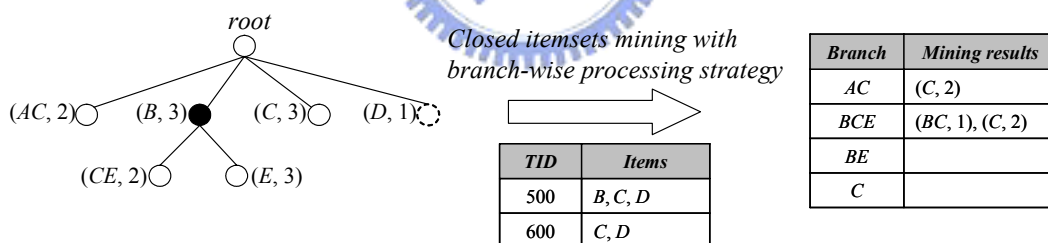


Figure 2-6: An example of branch-wise processing strategy in the *CO_generation* subroutine

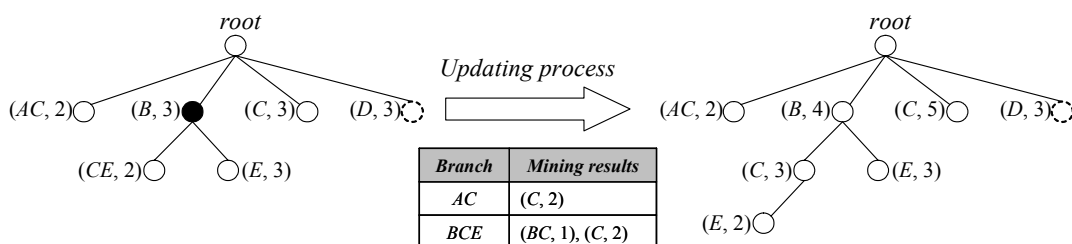


Figure 2-7: An example of updating process in the *CO_generation* subroutine

Example 2-2: When new transactions shown in Table 2-2 have been inserted into Table 2-1, the *CO_generation* subroutine first considers the most left branch of $\{AC\}$ in Figure 2-4 and uses $\{A\}$ and $\{C\}$ as seeds to mine the closed itemsets in d . Then, the branches with maximal itemsets $\{BCE\}$, $\{BE\}$, $\{C\}$ and $\{D\}$ are processed in turn. Mining results are shown in Figure 2-6, where the branches with $\{BE\}$ and $\{C\}$ can be ignored because related item combinations have been processed by the branch with $\{BCE\}$. After all branches have been processed, the *CO_generation* subroutine then updates mining results against CMT. The updated CMT is shown in Figure 2-7, where the itemsets $\{B\}$, $\{C\}$ and $\{BC\}$ are belonging to *FFJCI*, and the itemset $\{D\}$ is a candidate itemset for FCI_{D^+} . ■

2.5.3 The *CP_generation* Subroutine of the CIM Algorithm

According to Corollary 1, the *CP_generation* subroutine can find FI_d and then remove the itemsets in FI_d which have been covered by *FFJCI* as candidates for *CP* (i.e. $\{FI_d - \text{cover}(FFJCI, FI_d)\}$), but this indirect way may require an excessive computation cost for a large size of FI_d and generate many candidate itemsets irrelative to FCI_{D^+} . As a result, the *CP_generation* subroutine adopts a more effective and efficient candidate generation dealing with candidate generation. Let $F1_{dD^+}$ denote the frequent 1-itemsets in both d and D^+ , and *Cand1* denote the 1-itemsets which are infrequent in D but frequent in D^+ . They can be easily obtained from the updated CMT after the *CO_generation* subroutine. The *CP_generation* subroutine attempts to combine the found itemsets of *FFJCI* and *Cand1* with ones of $F1_{dD^+}$, to directly generate k -itemsets ($k \geq 2$) as candidates for FCI_{D^+} as follows. The *CP_generation* subroutine uses a depth-first and left-to-right search manner in the

CMT to generate the other candidates. When meeting an itemset x of $FFJCI$ in the CMT, the $CP_generation$ subroutine combines x with one of $F1_{dD^+}$ to form a new itemset x' . If x' is not covered by $FFJCI$ (i.e. x' is not a subset of an itemset in $FFJCI$) and frequent in d , x' is a new candidate itemset and a corresponding node $v_{x'}$ is built in the CMT. On the other hand, when meeting an itemset y of $Cand1$ or of new candidates generated before, the $CP_generation$ subroutine does a similar combination-and-test to generate a new candidate itemset y' and build a corresponding node $v_{y'}$ in the CMT. These two $FFJCI$ -based and $Cand$ -based candidate generations continue until no new candidate itemsets are generated.

CP_generation subroutine(*CMT, d, minsup, FFJCISet, Cand, F1_{dD⁺}, x*)

Parameters:

CMT: The closed maintenance tree;
d: The newly inserted transactions;
minsup: The minimum support threshold;
FFJCISet: The set used to store the itemsets of $FFJCI$;
Cand: The set used to store candidate itemsets for FCI_{D^+} ;
F1_{dD⁺}: The set used to store frequent 1-itemsets in both d and D^+ ;
x: A variable.

Begin

if $x = CMT.root$, **then**

for each child c_i of x , **do**

CP_generation subroutine(*CMT, d, minsup, FFJCISet, Cand, F1_{dD⁺}, c_i*);

else if $x \subseteq FFJCISet$ or $x \subseteq Cand$, **then**

for each $z_i \in F1_{dD^+}$ and the lexical order of z_i is after that of the first item of x , **do**

$x' = combine(x, z_i)$; /* Attempt to generate new candidate
itemsets for FCI_{D^+} . */

if $x' \neq null$, **then**

if $cover(FFJCISet, x') \neq null$, **then continue**;

/* If x' is covered by $FFJCISet$. */

update $x'.count$ against d ;

if $x'.count \geq minsup * |d|$, **then**

insert x' with $x'.count$ into *CMT* and *Cand*;

<p>for each child c_i of x, do</p> <p style="padding-left: 40px;">$CP_generation$ subroutine($CMT, d, minsup, FFJCISet, Cand, F1_{dD+}, c_i$);</p> <p>End.</p>

Figure 2-8: The $CP_generation$ subroutine

Theorem 2-7: The algorithm of $CP_generation$ subroutine can correctly generate candidate itemsets for the itemsets of FCI_{D+} which have not been determined in the $CO_generation$ subroutine.

Proof: It is obvious that only the itemsets of FI_d which are enumerated from $F1_{dD+}$ are possible to be contained in FCI_{D+} . The number of itemsets of $\{FI_d - cover(FFJCI, FI_d)\}$ can be further reduced regarding FCI_{D+} . Since the entire $F1_{dD+}$ can be obtained by collecting the 1-itemsets covered by $FFJCI$ and the itemsets of $Cand1$, the $CP_generation$ subroutine can directly, without loss of information, generate the candidate itemsets for the itemsets of FCI_{D+} which have not been determined in the $CO_generation$ subroutine by combining $FFJCI$ with $F1_{dD+}$ and $Cand$ with $F1_{dD+}$, respectively. Among them, the $FFJCI$ -based candidate generation can avoid the item combinations which have been covered by the found itemsets of $FFJCI$. ■

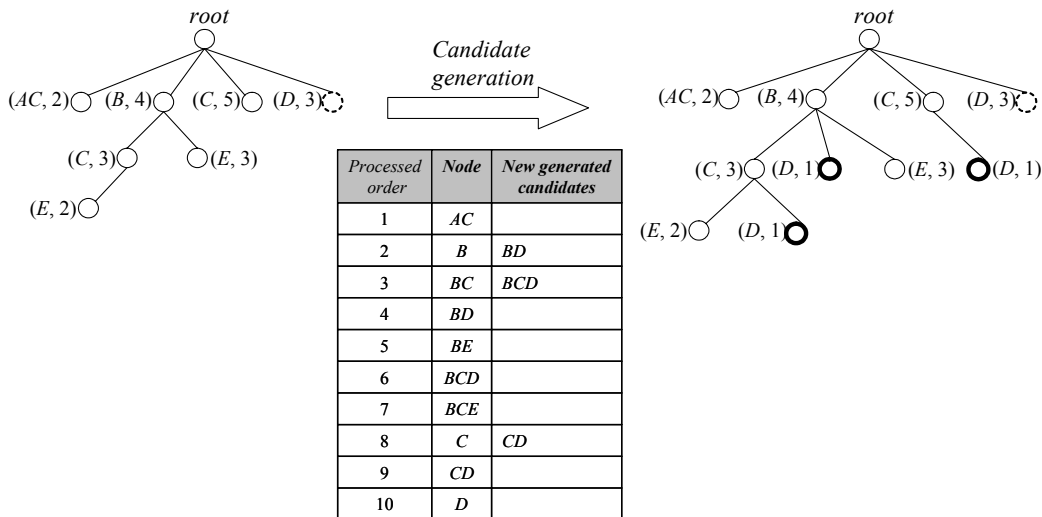


Figure 2-9: An example of *CP_generation* subroutine

Example 2-3: Continue from Example 2-2. After the *CO_generation* subroutine, $FFJCI = \{B, BC, C\}$, $Cand1 = \{D\}$ and $F1_{dD^+} = \{B, C, D\}$. As shown in Figure 2-9, the *CP_generation* subroutine mainly generates candidate itemsets as follows. It first combines $\{B\}$ of *FFJCI* with one of $F1_{dD^+}$ to form valid candidate itemsets. This will generate the candidate itemset $\{BD\}$. Then $\{BC\}$ and $\{C\}$ of *FFJCI* are processed as well to generate the candidate itemsets $\{BCD\}$ and $\{CD\}$, respectively. ■

2.6 The CIM Algorithm with Pre-large Concept: CIM-P Algorithm

Although the CIM algorithm focuses on the newly inserted transactions d and thus saves much processing time in maintaining association rules, it has to reprocess the original database D to handle the candidate itemsets generated by the *CP_generation* subroutine. This situation may occur frequently, especially when d is heterogeneous with D . For example, suppose $\{A\}$, $\{B\}$ and $\{AB\}$ are the entire *CO* and $\{C\}$, $\{D\}$ and $\{CD\}$ are the candidate itemsets. The final results can not be determined without reprocessing $\{C\}$, $\{D\}$ and $\{CD\}$ against D . If the candidate itemsets could be decided without reprocessing D at each time, the maintenance time could be further reduced.

In general, the number of records in d is much smaller than the number of records in D . Only the closed itemsets whose supports are slightly less than $minsup$ in D are possible to be frequent for D^+ after database maintenances. The concept of *pre-large closed itemsets* is denoted as the set of closed itemsets having support between a lower support threshold, which is smaller than $minsup$, and an upper

support threshold, which is equal to $minsup$. Pre-large closed itemsets are not truly frequent at present but more possible to be frequent in the future when database is updated. Therefore, using the pre-large closed itemsets to enlarge the amount of CO can reduce the cost of reprocessing D at the expense of storage spaces. They act as a buffer to avoid the movement of a closed itemset directly from infrequent to frequent and vice-versa during the incremental mining process. An infrequent closed itemset at most becomes pre-frequent (pre-large) and cannot become frequent. Based on this concept, the enhancement of CIM algorithm, CIM-P (CIM with *Pre-large* concept), does not require reprocessing D until the accumulative amount of new transactions exceeds the safety bound the buffer can afford, which depends on database size. As the database grows larger, the number of new transactions allowed also grows larger, and the CIM-P algorithm becomes increasingly efficient.

Figure 2-10 shows the concept of pre-large closed itemsets, where S_l denotes the lower support and S_u denotes the upper support. An infrequent closed itemset at most becomes pre-frequent (pre-large) and cannot become frequent after a small d is inserted into a large D .

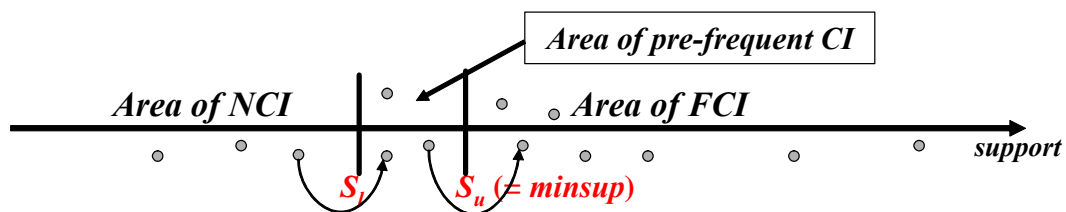


Figure 2-10: The concept of pre-large closed itemsets

Given the user-specified S_l and S_u , the safety bound of buffer can be derived by the following theorem.

Theorem 2-8: If $|d| \leq \frac{(S_u - S_l)|D|}{1 - S_u}$, then a candidate itemset will not become

frequent in D^+ after database maintenances [43]. ■

The $\frac{(S_u - S_l)|D|}{1 - S_u}$ can be used as the safety bound of buffer to determine the suitable time of reprocessing D . However, only considering whether the accumulative amount of new transactions exceeds $\frac{(S_u - S_l)|D|}{1 - S_u}$ seems too loose. For example, assume the safety bound $\frac{(S_u - S_l)|D|}{1 - S_u} = 10$ and the accumulative amount of new transactions $t = 0$ at first. When an increment d , in which all the transactions are distinct 1-itemsets and $|d| = 11$, has been inserted into D , then $t = 11$ larger than $\frac{(S_u - S_l)|D|}{1 - S_u} = 10$ and the CIM-P algorithm has to reprocess D to handle found candidate itemsets. However, these distinct closed itemsets consume only one of buffer, and the effort of reprocessing D is worthless.

Furthermore, we propose the *bucketing* strategy to improve the utility of buffer. The purpose of bucketing strategy is using some *buckets* to record the actual contributions of d for the major candidate itemsets (the itemsets with higher supports). The consumption of buffer can be rigidly calculated with the maximum value of buckets. In general, the number of candidate itemsets are much more than the number of buckets, and the bucketing strategy operates as follows. If only one bucket exists, the bucket is accumulated with the maximum support count of the candidate itemsets. Otherwise, according to the number of buckets k , k candidate itemsets with the highest support counts are selected to accumulate their corresponding bucket values:

- (a) For each selected itemset matching an itemset previously stored in the buckets, the bucketing strategy accumulates the target bucket using the support count of the selected itemset;
- (b) For the remaining selected itemsets, the bucketing strategy then finds two

having the largest and smallest support counts to accumulate the unprocessed bucket having the smallest value and all the remaining unprocessed buckets, respectively.

Example 2-4: Assume there are three buckets b_1 , b_2 and b_3 , the original database D is with $|D| = 100$, S_l is 30%, S_u is 50%, and two sets of candidate itemsets, $\{(AB, 15), (CD, 12), (CDE, 11), (BD, 10)\}$ and $\{(BCD, 11), (AB, 10), (AD, 10)\}$, are respectively obtained from two increments d_1 with $|d_1| = 20$ and d_2 with $|d_2| = 20$. By Theorem 2-8, the safety bound is $\frac{(0.5 - 0.3) * 100}{1 - 0.5} = 40$. After d_1 has been inserted into D , $b_1 = (AB, 15)$, $b_2 = (CD, 12)$ and $b_3 = (CDE, 11)$. Since the maximum value of buckets is 15 less than 40, the CIM-P algorithm does not need to reprocess D and the safety bound becomes $\frac{(0.5 - 0.3) * 120}{1 - 0.5} = 48$ for the updated database D^+ . After d_2 has been inserted into D^+ , the bucketing strategy first accumulates $b_1 = (AB, 15)$ using the support count of $(AB, 10)$ and thus $b_1 = (AB, 25)$, and then accumulates $b_2 = (CD, 12)$ and $b_3 = (CDE, 11)$ respectively using the support count of $(AD, 10)$ and $(BCD, 11)$ and thus $b_2 = (AD, 22)$ and $b_3 = (BCD, 22)$. Since the maximum value of buckets is 25 less than 48, the CIM-P algorithm still does not need to reprocess D^+ . ■

The utility of buffer would be better if we have more buckets, but the cost of storage space and accumulating buckets would be increased. This is a trade off in this strategy. In the CIM-P algorithm, according to the user-specified lower support and upper support thresholds, the large and pre-large closed itemsets with their support counts in preceding runs are stored in the CMT for later use in maintenance. When new transactions are inserted, the proposed algorithm first executes the *CO_generation* subroutine to find *FFJCI* and *FNJCI* and the *CP_generation* subroutine to generate the candidate itemsets which has not been determined in the

CO_generation subroutine. Then, the proposed algorithm utilizes the *bucketing* strategy to calculate the accumulative consumption of buffer and decide the suitable time of reprocessing *D*. If the accumulative consumption is within the safety bound of buffer, no action is needed. Otherwise, the original database has to be reprocessed to guarantee information lossless. The detail of the proposed CIM-P algorithm is shown as follows.

The CIM-P algorithm(*CMT, D, d, S_l, S_u, k*)

Parameters:

CMT: A closed maintenance tree based on *S_l*;
D: An original database;
d: A set of newly inserted transactions;
S_l: A lower support threshold;
S_u: An upper support threshold;
k: the number of buckets.

Begin

Set $SF = \frac{(S_u - S_l)|D|}{1 - S_u}$; /* *SF* is the safety bound of buffer*/

Set *FFJCISet* = ϕ ; /* *FFJCISet* is a set used to store the itemsets of *FFJCI*. */

Set *Cand* = ϕ ; /* *Cand* is a set used to store candidate itemsets for *FCI_{D+}*. */

Set_Bucket(*BucketSet*, 0, ϕ) /* Initialize the buckets in *BucketSet*, where *BucketSet* is a set used to store the most frequent *k* candidate itemsets. */

CO_generation subroutine(*CMT, d, S_u, FFJCISet, Cand*);

Set *F1_{dD+}* = ϕ ; /* *F1_{dD+}* is a set used to store frequent 1-itemsets in both *d* and *D⁺*. */

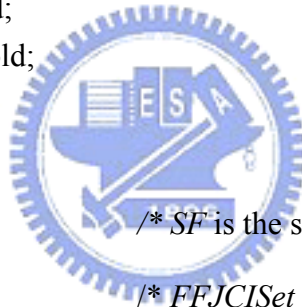
Set *Ucount_{D+}* = *S_u* * (*|D|* + *|d|*);

Set *Lcount_{D+}* = *S_l* * (*|D|* + *|d|*);

Obtain_frequent_items(*CMT, Ucount_{D+}, F1_{dD+}*); /* Obtain *F1_{dD+}* from *CMT*. */

CP_generation subroutine(*CMT, d, S_u, FFJCISet, Cand, F1_{dD+}, CMT.root*);

if Bucket_Strategy(*CMT, BucketSet, S_u*) > *SF*, **then** /* Check whether the consumption of buffer



<pre> Reconstruct(<i>CMT</i>, <i>D</i>, <i>d</i>, <i>S_i</i>); else Remove_NCI(<i>CMT</i>, <i>Lcount_{D+}</i>); Output_FCI(<i>CMT</i>); End. </pre>	<pre> is larger than the safety bound of buffer. */ /* Reconstruct <i>CMT</i> for <i>D+</i> based on <i>S_i</i> */ /* Remove the closed itemsets in <i>CMT</i> whose support counts are less than <i>mincount_{D+}</i>. */ /* Output the frequent closed itemsets for <i>D+</i>. */ </pre>
---	--

Figure 2-11: The CIM-P algorithm

2.7 Experimental Results

The experiments were conducted in C++ on a workstation with dual XEON 2.8GHz processors and 2048MB main memory, running the RedHat 9.0 operating system. For performance comparison, two classically incremental mining algorithms, FUP and Pre-large, in addition to our proposed CIM and CIM-P algorithms, were run on several synthetic and real-world dataset benchmarks which have been used in the previous performance studies [86][104][106]. The FUP and Pre-large algorithms were implemented based on the Apriori algorithm, while the CIM and CIM-P algorithms were implemented based on the CHARM algorithm. Table 2-3 shows the characteristics of the synthetic and real datasets.

Table 2-3: Characteristics of the experimental datasets

<i>Dataset</i>	<i>No. of transactions (D)</i>	<i>Avg. of transaction length (T)</i>	<i>Max. of transaction length</i>	<i>No. of Items (I)</i>
<i>T10I4D100K</i>	100,000	10	29	1000
<i>T40I10D100K</i>	100,000	40	77	1000
<i>connect</i>	67,557	43	43	130
<i>pumsb*</i>	49,046	50	63	7117
<i>BMS-POS</i>	515,597	6.5	164	1657

Two synthetic datasets, called *T10I4D100K* and *T40I10D100K*, were generated

by a generator similar to that used in [8]. The generator first generated L maximal potentially frequent itemsets, each with an average of I items. The items in the potentially frequent itemsets were randomly chosen from the total N items according to their actual sizes. The generator then generated D transactions, each with an average of T items. The items in a transaction were generated according to the L maximal potentially frequent itemsets in a probabilistic way. For example, the *T10I4D100K* dataset consists of 100,000 transactions averaging 10 items and generated according to 2000 maximal potentially frequent itemsets with an average size of 4 from a total of 1000 items.

Table 2-4: Mining information for the five datasets

<i>Dataset</i>	<i>Minsup</i>	<i>No. of frequent itemsets</i>	<i>No. of frequent closed itemsets</i>	<i>length of the maximum itemset</i>
<i>T10I4D100K</i>	0.093%	29,237	25,642	12
<i>T40I10D100K</i>	1.2%	19,412	18,117	11
<i>connect</i>	94%	4,223	1,223	9
<i>pumsb*</i>	42%	12,579	1,833	12
<i>BMS-POS</i>	0.65%	2497	2473	6

Table 2-5: The distribution of frequent itemsets for the five datasets

<i>Datasets</i>	<i>Length of frequent itemsets</i>											
	1	2	3	4	5	6	7	8	9	10	11	12
<i>T10I4D100K</i> (0.093%)	806	9539	7491	5797	3407	1525	515	132	23	2	0	0
<i>T40I10D100K</i> (1.2%)	721	8336	1448	1638	1792	2192	2048	1159	66	11	1	0
<i>connect</i> (94%)	17	119	435	927	1202	952	446	113	12	0	0	0
<i>pumsb*</i> (42%)	45	268	856	1837	2729	2887	2193	1188	448	111	16	1
<i>BMS-POS</i> (0.65%)	189	739	975	508	85	1	0	0	0	0	0	0

Three real datasets, called *connect*, *pumsb** and *BMS-POS* were used to evaluate the practicality of an algorithm in the real-world applications. The *connect* dataset contains game state information; the *pumsb** dataset contains census data; and the *BMS-POS* dataset contains several years of point-of-sale data from a large electronics

retailer, where each transaction in this dataset is a customer purchase transaction consisting of all the product categories purchased at one time. The *BMS-POS* dataset was also used in the KDDCUP 2000 competition.

Table 2-4 shows the mining information for the five datasets, including the number of frequent itemsets, the number of frequent closed itemsets and the length of the maximum itemset. For example, given the $minsup = 0.093\%$ on *T10I4D100K*, the number of frequent itemsets was 29,237, the number of frequent closed itemsets was 25,642 and the length of the maximum itemset was 10. Table 2-5 shows the detailed distribution of frequent itemsets for these datasets. Among them, *connect*, *pumsb** and *T40I10D100K* can be treated as dense datasets because they still generated many long frequent itemsets even for very high $minsup$ s, whereas *T10I4D100K* and *BMS-POS* can be treated as sparse datasets because they still generated many short frequent itemsets even for very low $minsup$ s. For the dense datasets, we can find the number of frequent itemsets considered by a classically incremental mining algorithm was much larger than the number of frequent closed itemsets considered by the CIM algorithm.

First, for each dataset, we randomly selected 1,000 records as a new increment and collected the remaining records as the original database. Figures 2-12(a) to 2-12(e) shows the execution times for the FUP, Pre-large and CIM algorithms respectively on the five datasets along with various $minsup$ s in the mining requests, where the lower support threshold in the Pre-large algorithm is fixed to the initial $minsup$, e.g., for *connect*, the lower support threshold of Pre-large algorithm is fixed to 95%. Moreover, the corresponding comparisons of the amounts of pre-stored mining information considered by the three algorithms respectively on the five datasets are shown in Figures 2-13(a) to 2-13(e). We can find that the performance highly depended on the amount of pre-stored mining information.

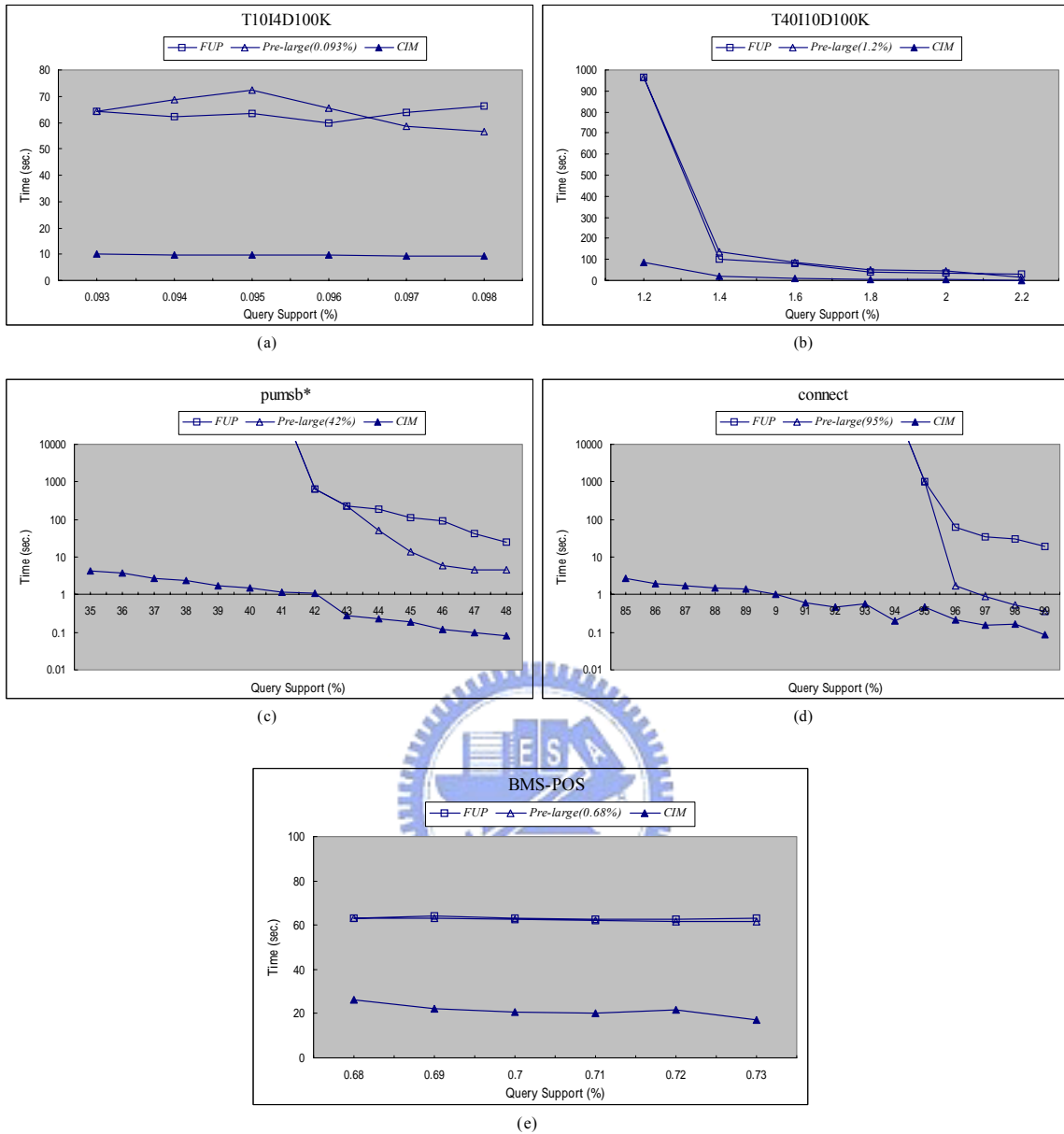


Figure 2-12: Execution times for the FUP, Pre-large and CIM algorithms respectively on the five datasets

Among the experimental results, for the dense datasets *connect*, *pumsb** and *T40I10D100K*, it can be easily seen that the CIM algorithm had several orders of magnitude better than the FUP and Pre-large algorithms for low *minsups* and it also had better performance than the two algorithms for high *minsups*. The FUP and

Pre-large algorithms performed only for very high *minsups* due to a huge amount of the previously mined frequent and pre-large itemsets, where the Pre-large algorithm had better performance than the FUP algorithm since the former, whose derived safety bound can afford the size of increment, can avoid a high cost of reprocessing original database at the expense of a low cost of processing pre-stored pre-large itemsets.



Figure 2-13: The amounts of pre-stored mining information for the FUP, Pre-large and

CIM algorithms respectively on the five datasets

On the other hand, for the sparse datasets *T10I4D100K* and *BMS-POS*, the CIM algorithm still had better performance than the FUP and Pre-large algorithms. However, since the amount of pre-stored mining information (the number of frequent closed itemsets) considered by the CIM algorithm was just slightly smaller than that (the number of frequent itemsets) considered by the FUP and Pre-large algorithms as shown in Figures 2-13(a) and 2-13(e), the CIM algorithm did not have a significant outperformance. As for the FUP and Pre-large algorithms, the former sometimes got better than the latter, because the derived safety bound can not afford the size of increment and a cost of processing pre-stored pre-large itemsets was required in addition by the latter.

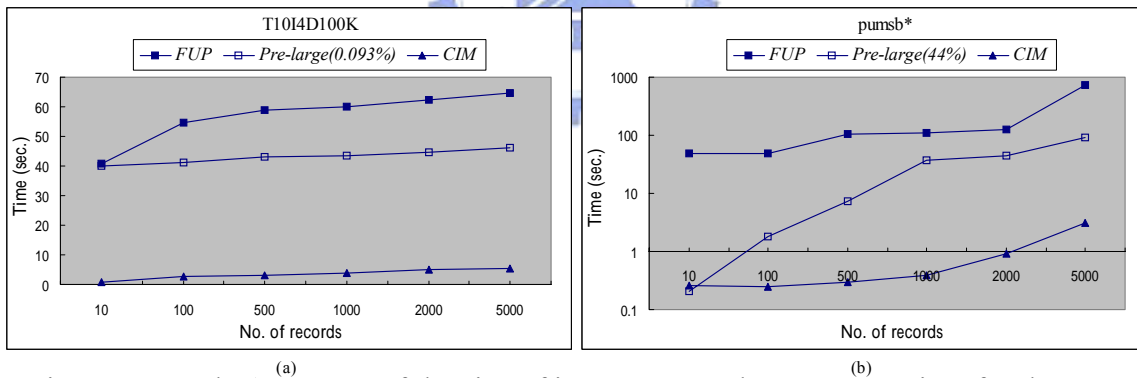


Figure 2-14: The influence of the size of increment on the execution time for the FUP, Pre-large and CIM algorithms

In general, incremental mining algorithms perform well when the size of newly inserted transactions is relatively smaller than the size of an original database because the cost of generating candidate itemsets from only new transactions is usually low and a large proportion of the candidate itemsets can be determined from previously

mining information. Figures 2-14(a) and 2-14(b) show the influence of the size of increment on the execution time for the FUP, Pre-large and CIM algorithms respectively on the datasets *T10I4D100K* and *pumst**. It is clear that the execution times required by the CIM algorithm for different sizes of increment were small, and seemed to grow slowly and linearly with the sizes of increment.

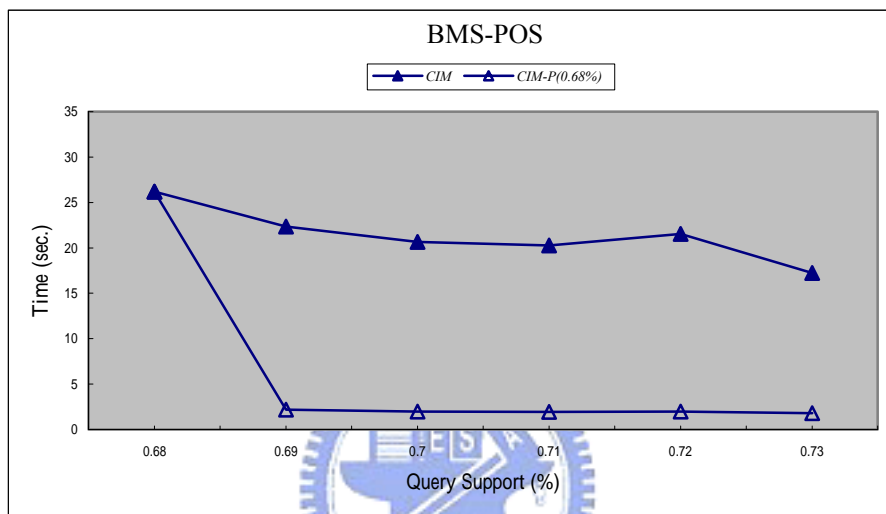


Figure 2-15: Execution times for the CIM and CIM-P algorithms on *BMS-POS*

Next, we compare the CIM-P algorithm with the CIM algorithm. Figure 2-15 shows the execution times for the CIM-P and CIM algorithms on the *BMS-POS* dataset along with various *minsup*s in the mining requests, where the lower support threshold of CIM-P algorithm is fixed to the initial *minsup* 0.68% and the number of buckets in CIM-P is set to 2. It can be seen that the execution times by the CIM-P algorithm were less than those by the CIM algorithm for the *minsup* set to a value above 0.68%.

2.8 Conclusion

In real-world applications, a database grows over time such that existing association rules may become invalid or new implicitly valid association rules may appear. Designing an incremental mining algorithm capable of updating existing association rules and discovering new association rules without reprocessing the entire updated database is a nontrivial work. Although researchers have developed some significant incremental mining algorithms to carry out this work, for dense databases or a low minimum support threshold, the performance of these approaches will degrade dramatically due to a huge amount of pre-stored mining information. On the other hand, one scan of original database to discover new association rules is required for most incremental mining algorithms. When the original database is massive, this will result in excessive I/O cost. In this study, we have thus utilized the concepts of *closed itemsets* and *pre-large itemsets* dealing with the two challenges and then designed two novel incremental mining algorithms, *Closed Itemsets Maintenance* (CIM) and CIM with *Pre-large concept* (CIM-P). Experiments respectively for sparse, dense, synthetic and real datasets are made, with results showing the effectiveness and practicality of the proposed approaches.

Chapter 3

Incremental Mining Algorithms for Sequential Patterns Maintenance

3.1 Introduction

Mining sequential patterns in sequence databases (temporal transaction databases), first proposed by Agrawal et al. in 1995 [6], is relatively useful since it can help model customer behaviors. The process of mining sequential patterns operates almost same as the process of mining association rules, except the former concerns relationships among itemsets in sequences whereas the latter concerns relationships among itemsets in transactions. Therefore, some studies extended the Apriori property [5] such that none of super-sequences of an infrequent sequence can be frequent, and proposed efficient algorithms based on the *candidates-generation-and-test* process for mining sequential patterns and other time-related frequent patterns. However, these Apriori-like sequential pattern mining algorithms, such as AprioriAll [6] and GSP [81], may suffer from the inherent drawback that a huge set of candidate sequences could be generated in a large and/or long sequence database. According to this observation, all recent studies have attempted to develop more efficient algorithms to reduce the expensive cost of candidate generation and test, such as FreeSpan [39], PrefixSpan [70], SPADE [103], SPAM [9], DISC-all [22], etc.

Studies on maintaining sequential patterns are relatively rare compared to those on maintaining association rules. Lin and Lee proposed the FASTUP algorithm [55] to

maintain sequential patterns by extending the FUP algorithm [20]; Hong *et al.* proposed an incremental mining algorithm based on the concept of *pre-large sequences* [44]. As the challenges mentioned in Chapter 2, these approaches will not work well on dense and massive database maintenances:

(a) For a dense database or a low minimum support threshold, the computation cost of updating previously mined sequential patterns will be getting tremendous due to a huge amount of previously mined frequent sequences;

(b) For a massive database, most incremental mining algorithms need one scan of original database dealing with finding new sequential patterns, and this will result in excessive I/O cost.

As a result, we attempt to utilize the concepts of *closed sequences* [99] and *pre-large sequences* [44] that are respectively extended from *closed itemsets* and *pre-large itemsets* to improve the performance of maintaining sequential patterns. Maintaining sequential patterns is much harder than maintaining association rules, since it must consider both itemsets and sequences. It is nontrivial to develop more efficient, scalable and practical mining algorithms for maintaining sequential patterns. In this chapter, we thus propose a novel incremental mining algorithm called *Closed Sequences Maintaining* (CSM) capable of sufficiently and efficiently finding all up-to-date sequential patterns for the updated database. Moreover, based on the concept of pre-large sequences, we propose the CSM-P, CSM with *Pre-large concept*, algorithm to improve the CSM algorithm.

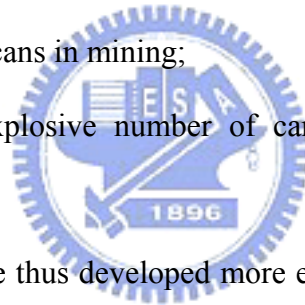
3.2 Related Work

3.2.1 Mining Sequential Patterns and Closed Sequential Patterns

Mining sequential patterns is a significant research direction of data mining. It

attempts to find customer behavior models and to assist managers in making correct and effective decisions. Among the mining sequential pattern algorithms, the first proposed AprioriAll algorithm [6] (similar to the Apriori algorithm) utilized a level-wise candidate generation approach that only *frequent sequences* (the sequence satisfying the user-specified minimum support) found in the previous level are treated as seeds for generating candidate sequences in the current level to reduce the search space. Since this *candidates-generation-and-test* process is simple and useful, many later studies [12][36][62][66][81] were based on this algorithm for further improving and refining, and deployed it in real-world applications. However, AprioriAll-based algorithms may suffer from the following inherent costs [70]:

- A huge set of candidate sequences for a large sequence database;
- Multiple database scans in mining;
- A combinatorial explosive number of candidate sequences for a dense sequence database.



Some recent studies have thus developed more efficient algorithms dealing with the three challenges. Examples include SPADE [103], PrefixSpan [70], SPAM [9]. Since AprioriAll-based algorithms using breadth-first search manner may generate many candidate sequences not appear in the database, all the three algorithms adopt depth-first search manner (i.e., recursive divide-and-conquer) to process the sequence database (SPADE also has breadth-first search option). The SPADE algorithm uses a simple join operation to enumerate frequent sequences from a vertical-layout database. The support of a sequence can be easily calculated by joining the vertical lists of its sub-sequences. The PrefixSpan algorithm uses a database-projection approach to reduce the efforts of candidate sequence generation. The sequence database is recursively projected into a set of smaller projected databases according to the

currently found frequent sequences, and then frequent sequences are grown in each projected database by exploring only local frequent fragments. Not only the support calculation but also the candidate sequence generation are highly improved. The SPAM algorithm uses the bitmap index to represent the sequence database in vertical such that the support calculation and candidate sequence generation operates similar to the SPADE algorithm.

These algorithms have provided pretty good solutions for the first two challenges. For the third one, however, they still need to be improved for a rather dense sequence database. In [99], Yan et al. proposed the concept of *closed sequences*, which is extended from the concept of *closed itemsets*, dealing with these challenges, especially for the third one.

3.2.2 Incremental Mining for Sequential Patterns

Maintaining sequential patterns is much harder than maintaining association rules since the former must consider both itemsets and sequences. In the following, we will introduce the concepts of maintaining sequential patterns when new transactions or sequences are inserted into the original sequence database, and briefly review related incremental mining algorithms.

When new transactions are inserted into a sequence database, they can be divided into two classes [44]:

Class 1: The new transactions with the same sequence identifiers as the sequences in the database;

Class 2: The new transactions with new sequence identifiers.

The newly inserted transactions are first transformed into sequences, and those belonging to Class 1 are merged with the corresponding sequences in the database and

those belonging to Class 2 are inserted into the database as new sequences.

Example 3-1: Assume that the sequence database includes eight sequences as shown in Table 3-1 and the frequent sequences found from these sequences are shown in Table 3-2 with the minimum support set to 50%.

Table 3-1: The sequence database

<i>Sequence id</i>	<i>Sequence</i>
1	<(A)(B)>
2	<(C, D)(A)(E, F, G)>
3	<(A, H, G)>
4	<(A)(E, G)(B)>
5	<(B)(C)>
6	<(A)(B, C)
7	<(A)(B, C, D)>
8	<(E, G)>

Table 3-2: All frequent sequences found from the sequences in Table 3-1

<i>Frequent sequences</i>			
<i>1-sequence</i>	<i>Support count</i>	<i>2-sequence</i>	<i>Support count</i>
<(A)>	6	<(A)(B)>	4
<(B)>	5		
<(C)>	4		
<(G)>	4		

When two new transactions shown in Table 3-3 are inserted into the sequence database, they are first transformed into the sequences and then merged with the corresponding sequences in Table 3-1. The results are shown in Table 3-4. ■

Table 3-3: Two new transactions sorted according to *Sequence_id* and *Trans_time*

<i>Sequence_id</i>	<i>Trans_time</i>	<i>Trans_content</i>
5	1998/02/01	<i>E, G</i>
9	1998/02/05	<i>E, F, G</i>

Table 3-4: The two newly merged sequences

<i>Sequence_id</i>	<i>Sequence</i>
5	$\langle(B)(C)(E, G)\rangle$
9	$\langle(E, F, G)\rangle$

The candidate sequences for the newly merged sequences in the database are then generated and counted. Note that, for the candidate sequences which have appeared in the original sequence database, their support count are only increased against the new sequences in the database. For example, the candidate 1-sequences for the newly merged sequences in Table 3-4 are shown in Table 3-5, where the support counts of $\langle(B)\rangle$ and $\langle(C)\rangle$ are not increased at all.

Table 3-5. The candidate 1-sequences with their support counts for newly merged sequences

<i>Candidate 1-sequences</i>	<i>Support count</i>
$\langle(B)\rangle$	0
$\langle(C)\rangle$	0
$\langle(E)\rangle$	2
$\langle(F)\rangle$	1
$\langle(G)\rangle$	2

Considering the original sequence database and the newly merged sequences, there are four cases of candidate sequences shown in Figure 3-1 may arise:

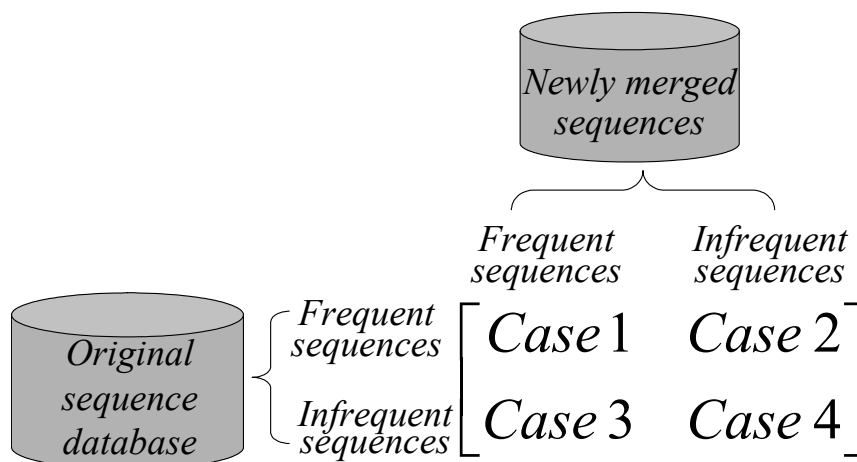


Figure 3-1: Four cases of candidate sequences

- Case 1: A candidate sequence is frequent in both the original sequence database and the newly merged sequences;
- Case 2: A candidate sequence is frequent in the original sequence database but infrequent in the newly merged sequences;
- Case 3: A candidate sequence is infrequent in the original sequence database but frequent in the newly merged sequences;
- Case 4: A candidate sequence is infrequent in both the original sequence database and the newly merged sequences.

Among the cases, since candidate sequences in Case 1 are frequent in both the original sequence database and the newly merged sequences, they are still frequent after the weighted average of the supports; similarly, candidate sequences in Case 4 are still infrequent after the new sequences are inserted. Cases 1 and 4 will not affect the final sequential patterns; Case 2 may remove existing sequential patterns; and Case 3 may generate new sequential patterns.

Lin and Lee proposed the FASTUP algorithm [55], which is an extension of the FUP algorithm proposed Cheung et al. [20], to efficiently cope with these four cases by pre-storing the previously mined frequent sequences from the original sequence database. The FASTUP can handle Cases 1, 2 and 4 by updating the pre-stored frequent sequences against the newly merged sequences, and reprocesses only the sequences without sufficient information in Case 3 against the original sequence database if necessary.

However, the performance of FASTUP algorithm will get degraded if a lot of candidate sequences from the newly merged sequences belong to Case 3. Hong et al.

[44] proposed the concept of *pre-large sequences* to enlarge the amount of pre-stored mining information in the FASTUP algorithm for further improving the maintenance performance. The concept of *pre-large sequences* is denoted as the set of sequences having support between a lower support threshold, which is smaller than the minimum support, and an upper support threshold, which is equal to the minimum support. Pre-large sequences are not truly frequent at present but more possible to be frequent in the future when database is updated. Therefore, using the pre-large sequences to enlarge the amount of pre-stored mining information can reduce the cost of reprocessing the original sequence database at the expense of storage spaces.

3.3 Preliminary Concepts

Let $I = \{i_1, i_2, \dots, i_m\}$ be a set of m items. An *itemset* is a subset of I and a k -*itemset* denotes an itemset consisting of k items. A *sequence* is an ordered list of itemsets and an l -*sequence* can be represented as $X = \langle x_1, x_2, \dots, x_l \rangle$, where x_i is an itemset and called an element of X . For a sequence, an item can occur at most once in an element, but can occur more than once in different elements. We call a sequence $Y = \langle y_1, y_2, \dots, y_q \rangle$ *contains* another sequence $X = \langle x_1, x_2, \dots, x_p \rangle$ iff there exist indexes j_1, j_2, \dots, j_p and $1 \leq j_1 \leq j_2 \leq \dots, j_p \leq q$ such that $x_1 \subseteq y_{j_1}, x_2 \subseteq y_{j_2}, \dots, x_p \subseteq y_{j_p}$; Y is also called a *supersequence* of X and inversely X is called a *subsequence* of Y . Let D be a sequence database consisting of a set of sequences, where each sequence consisting of a set of elements is associated with a *sequence identifier*, and $|D|$ denotes the number of sequences in D . The support of a sequence X , $X.sup$, in D is denoted as the percentage of sequences in D which contain X , and the support count of X , $X.count$, in D is denoted as the number of sequences in D which contain X , $X.count = X.sup * |D|$. For the sequences in D , X is called a *closed sequence* if there does not exist another

sequence Y which *closes (absorbs)* X , where a sequence Y is said to close (absorb) X iff Y contains X and $Y.sup = X.sup$ ($Y.count = X.count$). CS denotes the set of all closed sequences in D . Furthermore, if there is no supersequence of X existing in D , X is also called a **maximal sequence**.

Given the user-specified minimum support threshold, $minsup$, the problem of mining sequential patterns is to find out all sequences in D that have support larger than $minsup$. With respect to the $minsup$, the set of **frequent sequence**, FS , includes all the sequences whose support is larger than $minsup$; the set of **infrequent sequence**, NS , includes all the sequences whose support is less than $minsup$; the set of **frequent closed sequence**, FCS , includes all the closed sequences whose support is larger than $minsup$, $FCS = \{x|x \in CS, x.sup \geq minsup\}$; and the set of **infrequent closed sequence**, NCS , includes all the closed sequences whose support is less than $minsup$, $NCS = \{x|x \in CS - FCS\}$. Note that FCS includes no sequence which has a supersequence with the same support, thus $FCS \subseteq FS$. The problem of mining sequential patterns can be reduced to the problem of finding FCS in D .

Let d be a set of newly merged sequences, $|d|$ be the number of sequences in d , d' be a set of sequences in d with the same sequence identifiers as the sequences in an original sequence database D , $|d'|$ be the number of sequences in d' , D^+ be the updated database and $|D^+|$ be the number of sequences in the updated database. Therefore, FS_D , FS_d and FS_{D^+} denote the FS obtained from D , d and D^+ with respect to the same $minsup$, respectively, and NS , CS , FCS or NCS obtained from D , d and D^+ can have similar meanings. The problem of maintaining sequential patterns is to find FS_{D^+} or FCS_{D^+} . Let the set of **original frequent sequences**, OS , be defined as $OS = \{x|x \in FS_D\}$, and the set of **potential frequent sequences**, PS , be defined as $PS = \{x|x \in FS_d - FS_D\}$. By definition, a sequence $X \in FS_{D^+}$ must belong to $OS \cup PS$, and thus

the problem of maintaining sequential patterns is equivalent to processing $OS \cup PS$. Similarly, let the set of **closed original frequent sequences**, COS , be defined as $COS = \{x|x \in FS_D \text{ and } x \in CS_{D^+}\}$, and the set of **closed potential frequent sequences**, CPS , be defined as $CPS = \{x|x \in FS_d - FS_D \text{ and } x \in CS_{D^+}\}$. The problem of maintaining sequential patterns is also equivalent to processing $COS \cup CPS$. The set of **joint closed sequences**, JCS , which is defined as $JCS = \{x|x = y \wedge z, y \in CS_D, z \in CS_d\}$ is proposed in this study, where \wedge denotes the *intersection of two sequences*. We call a sequence $X = \langle x_1, x_2, \dots, x_r \rangle$ is the intersection of two sequences $Y = \langle y_1, y_2, \dots, y_p \rangle$ and $Z = \langle z_1, z_2, \dots, z_q \rangle$ iff there exist indexes j_1, j_2, \dots, j_r and $1 \leq j_1 \leq j_2 \leq \dots, j_r \leq p$ and $1 \leq j_1 \leq j_2 \leq \dots, j_r \leq q$ such that $x_1 \subseteq y_{j_1}, x_1 \subseteq y_{j_2}, \dots, x_r \subseteq y_{j_r}$ and $x_1 \subseteq z_{j_1}, x_1 \subseteq z_{j_2}, \dots, z_r \subseteq z_{j_r}$. The JCS can be divided into four parts based on FCS_D, FCS_d, NCS_D and NCS_d :

- $FFJCS = \{x|x = y \wedge z, y \in FCS_D, z \in FCS_d\}$;
- $FNJCS = \{x|x = y \wedge z, y \in FCS_D, z \in NCS_d\}$;
- $NFJCS = \{x|x = y \wedge z, y \in NCS_D, z \in FCS_d\}$;
- $NNJCS = \{x|x = y \wedge z, y \in NCS_D, z \in NCS_d\}$.

3.4 Closed Sequences Maintenance

Considering an original sequence database D and the set of newly merged sequences d , there are four cases of candidate sequences for the updated database D^+ have been discussed in Section 2. With pre-storing previously mined frequent sequences FS_D , a typically incremental mining process can efficiently cope with these four cases by two steps: (a) *updating OS against d* and (b) *reprocessing PS against D*. Following this idea, we can use two similar steps: (a) *updating COS against d* and (b)

reprocessing CPS against D to find out FCS_{D+} dealing with the problem of maintaining sequential patterns. Since directly obtaining $COS = \{x|x \in FS_D \text{ and } x \in CS_{D+}\}$ and $CPS = \{x|x \in FS_d - FS_D \text{ and } x \in CS_{D+}\}$ is impractical, we attempt to utilize the pre-stored known information FCS_D from D and the information FCS_d obtained from d to approach COS and CPS . The following lemmas and theorems can be easily derived and proven by referring to corresponding lemmas and theorems mentioned in Chapter 2, so we omit the details here.

Lemma 3-1: If $x \in CS_D \cup CS_d$, then $x \in CS_{D+}$. ■

Lemmas 3-2 and 3-3 are used to derive the set of *joint closed sequences (JCS)* which are closed sequences for $D+$ but can not be determined by FCS_D and FCS_{d-D} .

Lemma 3-2: If $x \in JCS$, then $x \in CS_{D+}$. ■

Lemma 3-3: If $x \in CS_{D+}$, then $x \in CS_D \cup CS_d \cup JCS$. ■

Theorem 3-1: $CS_{D+} = CS_D \cup CS_d \cup JCS$. ■

Considering an original sequence database and the newly merged sequences, JCS can be divided into four parts based on FCS_D , FCS_d , NCS_D and NCS_d as shown in Figure 3-2:

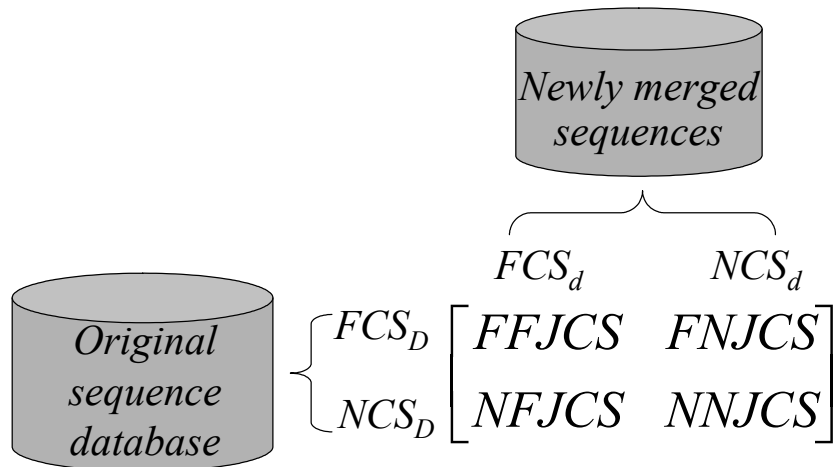


Figure 3-2: Four cases of joint closed sequences

- The case of *FFJCS*: A closed sequence is frequent in both the original sequence database and the newly merged sequences;
- The case of *FNJCS*: A closed sequence is frequent in the original sequence database but infrequent in the newly merged sequences;
- The case of *NFJCS*: A closed sequence is infrequent in the original sequence database but frequent in the newly merged sequences;
- The case of *NNJCS*: A closed sequence is infrequent in both the original sequence database and the newly merged sequences.

According to Theorem 3-1, the following theorems are derived to obtain *COS* and *CPS* by FCS_D , FCS_d , *FFJCS*, *FNJCS* and *NFJCS*.

Theorem 3-2: $COS = \{x|x \in FCS_D \cup FFJCS \cup FNJCS\}$. ■

Theorem 3-3: $CPS = \{x|x \in (FCS_d - FFJCS) \cup NFJCS\}$. ■

Theorems 3-2 and 3-3 provide a convenient way to obtain *COS* and *CPS*. For *COS*, *FFJCS* and *FNJCS* can be obtained by processing the pre-stored mining information FCS_D against d . For *CPS*, however, since *NFJCS* has to be generated from NCS_D , which is usually unknown in a typically incremental mining process, the cost is too expensive to be acceptable. As a result, given a function $cover(FFJCS, FS_d)$ denoting the sequences in FS_d which are covered by *FFJCS*, the following theorem is derived to obtain *CPS*.

Theorem 3-4: $CPS = \{x|x \in FS_d - cover(FFJCS, FS_d), x \in CS_{D+}\}$. ■

Corollary 1: $CPS \subseteq \{FS_d - cover(FFJCS, FS_d)\}$ ■

Since *FFJCS* has been obtained in *COS* generation, we only need to find FS_d and remove the sequences in FS_d which have been determined in *FFJCS* as candidates for *CPS*. It seems to be a better way to generate the sequences of FCS_{D+} which are not

included in the *COS*.

3.5 The Closed Sequences Maintaining (CSM) Algorithm

We develop a novel incremental mining algorithm mainly consisting of *COS_generation* and *CPS_generation* subroutines, called *Closed Sequences Maintaining* (CSM), to efficiently find FCS_{D+} . The proposed CSM algorithm also utilizes the CMT (*Closed Maintenance Tree*) data structure mentioned in Section 2.5.1 to facilitate the processes of *COS_generation* and *CPS_generation* subroutines. However, the CMT of the CSM algorithm is not mainly for closed itemset but mainly for closed sequence, such that the *closed nodes* and *infrequent nodes* represent the sequences in FCS_{D+} and the infrequent 1-sequences in D , respectively.

The CSM algorithm first updates the sequences in the CMT against d to obtain *COS* by the *COS_generation* subroutine. Then, by the *CPS_generation* subroutine, it generates candidate sequences for the sequences of FCS_{D+} which have not been determined in the *COS_generation* subroutine. Finally, by reprocessing these obtained candidate sequences against D and checking their closure property, the CSM algorithm can find FCS_{D+} from the CMT.

The *COS_generation* and *CPS_generation* subroutines operates similar to the *CO_generation* and *CP_generation* subroutines in the CIM algorithm mentioned in Chapter 2. The *COS_generation* subroutine is responsible for processing FCS_D against d to find *FFJCS* and *FNJCS*, thus obtaining *COS*, while *CPS_generation* subroutine is responsible for generating candidate sequences for FCS_{D+} which have not been determined in the *COS_generation* subroutine.

The CSM algorithm(CMT, D, d, d', minsup)

Parameters:

CMT: A closed maintenance tree;

D: An original sequence database;

d: A set of newly merged sequences;

d': A set of sequences in *d* with the same sequence identifiers as the sequences in *D*;

minsup: A minimum support threshold.

Begin

```

Set FFJCSSet =  $\phi$ ,                               /* FFJCSSet is a set used to store the
                                                    sequences of FFJCS. */

Set Cand =  $\phi$ ,                                     /* Cand is a set used to store candidate
                                                    sequences for FCSD+. */

COS_generation subroutine(CMT, d, d', minsup, FFJCSSet, Cand);

Set F1dD+ =  $\phi$ ,                                  /* F1dD+ is a set used to store the frequent
                                                    1-sequences in both d and D+. */

Set mincountD+ = minsup * (|D| + |d| - |d'|);

Obtain_frequent_items(CMT, mincountD+, F1dD+);
                                                    /* Obtain F1dD+ from CMT. */

CPS_generation subroutine(CMT, d, d', minsup, FFJCSSet, Cand, F1dD+,
                                                                    CMT.root);

Reprocess_Cand(CMT, Cand, D);                    /* Reprocess obtained candidate k-sequences
                                                    (k ≥ 2) in CMT against D. */

Check_Closure_Cand(CMT, Cand);                  /* Check closure property for all candidates
                                                    sequences in CMT. */

Remove_NCS(CMT, mincountD+);                    /* Remove the closed sequences in CMT
                                                    whose support counts are less than
                                                    mincountD+. */

Output_FCS(CMT);                                  /* Output FCSD+ for D+. */

```

End.

Figure 3-3: The CSM algorithm

3.6 The CSM Algorithm with Pre-large Concept: CSM-P Algorithm

Although the CSM algorithm focuses on the newly merged sequences *d* and thus saves much processing time in maintaining sequential patterns, it has to reprocess the original sequence database *D* to handle the candidate itemsets generated by the *CPS_generation* subroutine. This situation may occur frequently, especially when *d* is

heterogeneous with D . In general, the number of records in d is much smaller than the number of records in D . Only the closed sequences whose supports are slightly less than $minsup$ in D are possible to be frequent for D^+ after database maintenances. We can apply the concept of *pre-large sequences* [44] to improve the proposed CSM algorithm. Based on this concept, the enhancement of CSM algorithm, CSM-P (CSM with *Pre-large concept*), does not require reprocessing D until the accumulative amount of newly merged sequences exceeds the safety bound the buffer can afford, which depends on database size. As the database grows larger, the number of newly merged sequences allowed also grows larger, and the CSM-P algorithm becomes increasingly efficient.

Given the user-specified S_l and S_u , $|d^*|$ denotes the number of the sequences in d with the same sequence identifiers as the sequences in D . The safety bound of buffer can be derived by the following theorem.

Theorem 3-5: If $|d| \leq \frac{(S_u - S_l)|D|}{1 - S_u} - \frac{|d^*|S_u}{1 - S_u}$, then a sequence in CPS will not

become frequent in D^+ after database maintenances [44] ■

The $\frac{(S_u - S_l)|D|}{1 - S_u} - \frac{|d^*|S_u}{1 - S_u}$ can be used as the safety bound of buffer to determine the suitable time of reprocessing D .

Furthermore, we can also utilize the *bucketing* strategy mentioned in Chapter 2 to improve the utility of buffer. The purpose of bucketing strategy is using some *buckets* to record the actual contributions of d for the major candidate sequences (the sequences with higher supports). The consumption of buffer can be rigidly calculated with the maximum value of buckets.

In the CSM-P algorithm, according to the user-specified lower support and upper support thresholds, the frequent and pre-large closed sequences with their support

counts in preceding runs are stored in the CMT for later use in maintenance. When newly merged sequences are inserted, the proposed algorithm first executes the *COS_generation* subroutine to find *FFJCS* and *FNJCS* and the *CPS_generation* subroutine to generate the candidate frequent closed sequences for D^+ which has not been determined in the *COS_generation* subroutine. Then, the proposed algorithm utilizes the *bucketing* strategy to calculate the accumulative consumption of buffer and decide the suitable time of reprocessing D . If the accumulative consumption is within the safety bound of buffer, no action is needed. Otherwise, the original sequence database D has to be reprocessed to guarantee information lossless. The detail of the proposed CSM-P algorithm is shown as follows.

The CSM-P algorithm(*CMT, D, d, d', S_l, S_u, k*)

Parameters:

- CMT*: A closed maintenance tree based on S_l ;
D: An original database;
d: A set of newly inserted sequences;
d': A set of sequences in *d* with the same sequence identifiers as the sequences in *D*;
 S_l : A lower support threshold;
 S_u : An upper support threshold;
k: the number of buckets.

Begin

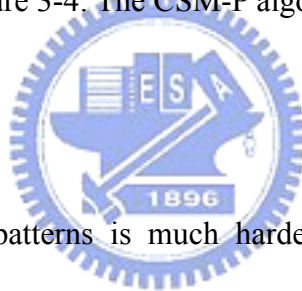
Set $SF = \left(\frac{(S_u - S_l)|D|}{1 - S_u} - \frac{|d'|S_u}{1 - S_u} \right)$; /* *SF* is the safety bound of buffer*/
Set *FFJCSSet* = ϕ ; /* *FFJCSSet* is a set used to store the sequences of *FFJCS*. */
Set *Cand* = ϕ ; /* *Cand* is a set used to store candidate sequences for FCS_{D^+} . */
Set_Bucket(*BucketSet*, 0, ϕ) /* Initialize the buckets in *BucketSet*, where *BucketSet* is a set used to store the most frequent *k* candidate itemsets. */
COS_generation subroutine(*CMT, d, d', S_u, FFJCSSet, Cand1*);
Set $F1_{dD^+} = \phi$; /* $F1_{dD^+}$ is a set used to store frequent

```

1-sequences in both  $d$  and  $D^+$ . */
Set  $Ucount_{D^+} = S_u * (|D| + |d| - |d'|)$ ;
Set  $Lcount_{D^+} = S_l * (|D| + |d| - |d'|)$ ;
Obtain_frequent_items( $CMT, Ucount_{D^+}, F1_{dD^+}$ );
/* Obtain  $F1_{dD^+}$  from  $CMT$ . */
CPS_generation subroutine( $CMT, d, d', S_u, FFJCSSet, Cand1, F1_{dD^+}, CMT.root$ );
if Bucket_Strategy( $CMT, BucketSet, S_u$ ) >  $SF$ , then
/* Check whether the consumption of buffer
is larger than the safety bound of buffer. */
    Reconstruct( $CMT, D, d, d', S_l$ ); /* Reconstruct  $CMT$  based on  $S_l$  */
else,
    Remove_NCS( $CMT, Lcount_{D^+}$ ); /* Remove the sequences in  $CMT$  whose
support counts are less than  $mincount_{D^+}$ . */
Output_FCS( $CMT$ ); /* Output the frequent closed sequences for
 $D^+$ . */
End.

```

Figure 3-4: The CSM-P algorithm



3.7 Conclusion

Maintaining sequential patterns is much harder than maintaining association rules, since it must consider both itemsets and sequences. It is nontrivial and useful to develop efficient mining algorithms for maintaining sequential patterns. As a result, in this study, we attempt to utilize the concepts of *closed sequences* and *pre-large sequences* to improve the performance of maintaining sequential patterns. The closed sequences can losslessly determine all the pre-stored mined sequences and their exact support, but is orders of magnitude small. The pre-large sequences act as a buffer to avoid the movements of sequence directly from valid to invalid and vice-versa during the incremental mining process. Based on the two concepts, two novel incremental mining algorithms, CSM and CSM-P, are thus developed to efficiently maintain sequential patterns, especially for a dense sequence database.

Chapter 4

Incremental Mining Algorithms for Document Classifiers Maintenance

4.1 Introduction

As digital documents evolve and become increasingly available, *automatic document classification* (a.k.a. *document categorization*) of managing and discovering useful information in documents is becoming more and more important for users. Automatic document classification refers to the activity of automatically constructing a classifier to assign category labels suggested by pre-defined training documents to undefined documents. In general, automatic document classification involves three major tasks [79]: *document representation*, which represents documents in machine-readable structures, *classifier construction*, which constructs a classifier from pre-defined training documents, and *classifier evaluation*, which evaluates classifier accuracy in terms of various evaluation functions.

Previous studies of document representation have often represented documents in finite sets of terms such as keywords and phrases, so-called *term-space document representation*. A document can be represented as $\langle w_1, w_2, w_3, \dots, w_t \rangle$, where w_i represents the weight between the i -th keyword and the document. However, this simple representation may result in highly correlated, redundant and less representative dimensions, such that the efficiency and effectiveness are decreased [31][34].

As for classifier construction, most of previously proposed batch approaches such as *C4.5* [73], *SVM* [46][47] and *Naïve Bayesian* [57] have to reconstruct the classifier when new documents or new categories are added. Therefore, considerable computation time is required to get the updated classifier. In real world, data may evolve over time, so a batch-based classifier construction approach is obviously impractical [59].

In this study, we propose a *domain-space weighting scheme* to resolve the above problems in document representation and classifier construction. The proposed scheme utilizes a more compact and meaningful document representation called *domain-space document representation* to represent documents in finite sets of domains. Based on the domain-space document representation, it utilizes three phases, *Training Phase*, *Discrimination Phase* and *Tuning Phase*, to construct a classifier and adapt the classifier along with evolving data.

In the *Training Phase*, the proposed scheme incrementally extracts and weights features from each individual category in the training documents and integrates the resulting weights into the *feature-domain weighting table*, which retain the weights between features and all involved categories. In the *Discrimination Phase*, it reduces the weights of features in the feature-domain weighting table that have lower discriminating powers. The weight between a document and each category is easily calculated by summarizing related feature weights in the feature-domain weighting table, and the classifier is thus constructed according to this table. Finally, in the *Tuning Phase*, the scheme utilizes feedback information from tuning documents to reduce the number of false positives for the constructed classifier.

We tested the constructed classifier on the standard benchmark Reuters-21578 text collection [58] based on the “ModApte” split version in terms of micro- and

macro-averaging F_1 evaluation functions. Our experiments consisted of four aspects: (1) the classification accuracy of our classifier compared to those shown in [23]; (2) the influence of the training document threshold φ and the discrimination threshold δ on classification accuracy; (3) the influence of the number of tuning documents on classification accuracy; and (4) the time performance of our classifier compared to a batch-based mining approach. The experimental results show that the classification accuracy of our classifier got better with an appropriate discrimination threshold and sufficient training documents, and the classifier was strengthened by the Tuning Phase.

4.2 Related Work

Previous studies of three major tasks (document representation, classifier construction and classifier evaluation) in automatic document classification are briefly reviewed below.



4.2.1 Document Representation

Document representation refers to representing documents in machine-readable structures such that classifiers can be constructed efficiently and effectively. The most common approach is the *vector space model* (VSM), which represents documents as sets of features. The VSM usually considers two factors: (1) how to extract representative features from documents, and (2) how to determine weights for document features. *Term-space document representation* utilizing finite sets of keywords or phrases occurring in documents as representative features and determining feature weights using the standard *tfidf* weighting function is the most popular form of VSM. A document can be therefore represented as $\langle w_1, w_2, w_3, \dots \rangle$,

w_i , where w_i represents the weight between the i -th keyword and the document. However, this simple representation may result in highly correlated, redundant and less representative dimensions in a document vector, such that the efficiency and effectiveness of a classifier are decreased [31][34].

The technique of dimension reduction has been used to resolve this problem in recent decades. Among the approaches, (1) *feature selection* which selects terms from the old ones contributing the classification most by evaluation functions such as chi-square, information gain and mutual information [25][96][102], and (2) *feature extraction* which regenerates more representative terms from the old ones [10][24][29][49][94] are the two well-known categories.

4.2.2 Classifier Construction

- *Rocchio* approach

Given a set of training documents, the *Rocchio* approach [56][76] attempts to learn a set of features used to represent each individual category from positive training documents (members of the category) and negative training documents (not members of the category). Then an undefined document x is assigned to the category w when the inner product result of w and x is more than a user-specified threshold.

- *Support Vector Machine (SVM)* approach

Given a set of training documents, the *support vector machine* approach (SVM) [46][47] finds the best decision hyper-plane separating two categories within the maximum margin of each category. Figure 4-1 shows an example of a 2-dimensional case. The decision hyper-plane, determined by only a few training documents, called the support vectors, finds the maximum distance between different categories. Then an undefined document is assigned to the closest category.

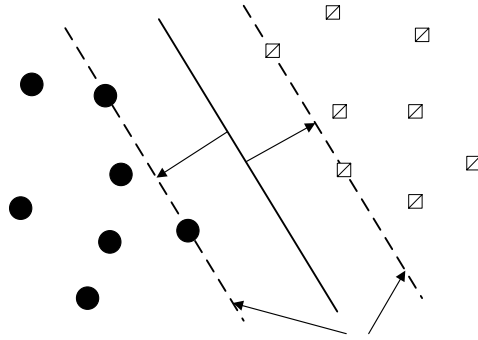


Figure 4-1: An example of the support vector machine approach

- *K-nearest neighbor (k-NN)* approach

The *K-nearest neighbor (k-NN)* [31][100] is an instance-based or lazy learning approach that treats each training document as a case and stores it in a case base. This is rather different from most classifier construction approaches, which need to construct models in advance. When classifying an undefined document d , the k -NN first finds k nearest neighbors of d from the retained cases in the case base and calculates the similarity scores between this document and categories of its k neighbors. Then d is assigned to the most similar category according to the similarity scores.

4.2.3 Classifier Evaluation

Evaluating classifier classification accuracy, the ability to make correct classification decisions, is an important task. *Precision* (π) and *Recall* (ρ) used in the field of information retrieval are well-known evaluation functions. However, considering only the precision or the recall of a classifier may sometimes be insufficient and misleading. The evaluation function F_β , which considers them simultaneously, has recently been proposed. F_β is defined as follows:

$$F_{\beta} = \frac{(\beta^2 + 1) * \pi * \rho}{\beta^2 \pi + \rho}, \quad (4-1)$$

where, β , which ranges from 0 to ∞ , denotes the importance of precision (π) and the importance of recall (ρ). When $\beta = 0$, F_{β} is identical to π . By contrast, when $\beta = \infty$, F_{β} is identical to ρ . $\beta = 1$, which gives equal importance to π and ρ for F_{β} , is used most frequently.

These evaluation functions are usually combined with *macro-averaging* or *micro-averaging* to evaluate the average classification accuracy across multiple categories [100][101]. Micro-averaging performance scores give equal weight to each document classification decision, i.e., a per-document average, while macro-averaging performance scores give equal weight to each category without considering its frequency, i.e., a per-category average.

4.3 Domain-space Weighting Scheme for Document Classification

The proposed domain-space weighting scheme utilizes a document representation called *domain-space document representation* to represent documents in finite sets of domains. In this representation, each category involved in the training documents is treated as a meaningful domain. To simplify our discussion, we assume the training documents involve c categories in the rest of this study. A document can be therefore represented as $\langle w_1, w_2, w_3, \dots, w_c \rangle$, where w_i represents the weight between this document and the i -th category. Since the number of dimensions in domain-space is much less than that in term-space and many irrelevant and redundant dimensions can be effectively eliminated, the domain-space document representation is more compact and representative. The larger the weight assigned to a document vector entry, the more relevant the entry is. Thus, the entry with the maximum weight

is chosen as the category label for an undefined document.

In order to determine the document vector, a *feature-domain weighting table* is proposed to retain the weights between features and all involved categories. Since a document is made up of a set of keywords and a keyword can be treated as a representative feature, a document vector can be calculated by summarizing all related feature vectors in the feature-domain weighting table. A document classifier can be thus constructed according to this table.

Example 4-1: Assume Table 4-1 is a feature-domain weighting table containing three categories and eight keywords. The document vector $\langle w_1, w_2, w_3 \rangle$ for an undefined document d with two keywords, ‘*Mining*’ and ‘*Clustering*’, can be simply calculated using: $\langle (0.2992+0.3282)/2, (0+0)/2, (0.7008+ 0.6718)/2 \rangle = \langle 0.3137, 0, 0.6863 \rangle$. Thus, d can be simply assigned to the “*DM*” Category. ■

Table 4-1: An example of a feature-domain weighting table

Domain Feature	<i>AI</i>	<i>DB</i>	<i>DM</i>
<i>Database</i>	0.0521	0.2387	0.2344
<i>Primary</i>	0	1	0
<i>Relation</i>	0.138	0.9852	0
<i>View</i>	0	1	0
<i>Data</i>	0.0605	0.1587	0.1592
<i>Mining</i>	0.2992	0	0.7008
<i>Clustering</i>	0.3282	0	0.6718
<i>Rule</i>	0	0	1

4.4 Classifier Construction Based on Domain-space Document Representation

Classifier construction in the domain-space weighting scheme is carried out in three phases: *Training Phase*, *Discrimination Phase* and *Tuning Phase*, to construct a

classifier. In the Training Phase, the scheme incrementally extracts and weights features from each category involved in the training documents, and then integrates the results into a feature-domain weighting table. After that, in the Discrimination Phase, it reduces the weights for the features in the feature-domain weighting table which have lower discriminating powers. A document classifier is thus constructed. In the Tuning Phase, the scheme utilizes feedback information from the tuning documents (the other pre-defined documents) to reduce the number of false positives yielded by the constructed classifier.

The proposed classifier construction algorithm is shown in Figure 4-2. It contains three subroutines corresponding to the Training, Discrimination and Tuning Phases. Let T_{in} be a given feature-domain weighting table. When a new category of documents D is added, the classifier construction algorithm first uses the training algorithm to extract and weight the features from D (Step 1) and then integrate the results into T_{in} (Step 2). The integration contains inserting the domain D and the feature only from D into T_{in} and then updating all feature weights in T_{in} . Assume T_{up} is denoted as the updated feature-domain weighting table. Next, it uses the discrimination algorithm to reduce the weights of features whose discriminating powers are less than the user-specified threshold δ . A classifier C_{up} is therefore constructed according to T_{up} (Step 3). The tuning algorithm can be used to strengthen the constructed classifier C_{up} via the set of tuning documents D' (Step 4), where ζ is the user-specified tuning parameter.

Classifier Construction Algorithm:

Input:

T_{in} : A given feature-domain weighting table.

D : A newly added category of documents.

D' : A set of tuning documents.

δ : A discrimination threshold.

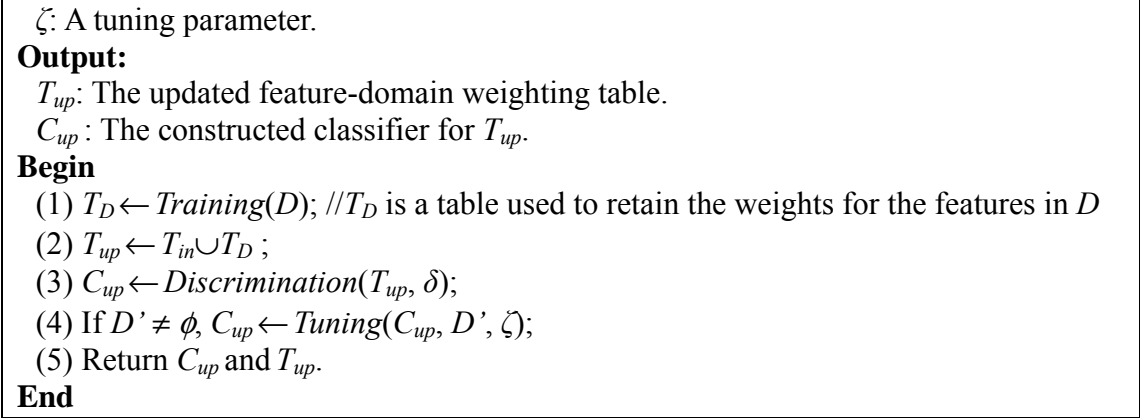


Figure 4-2: The classifier construction algorithm

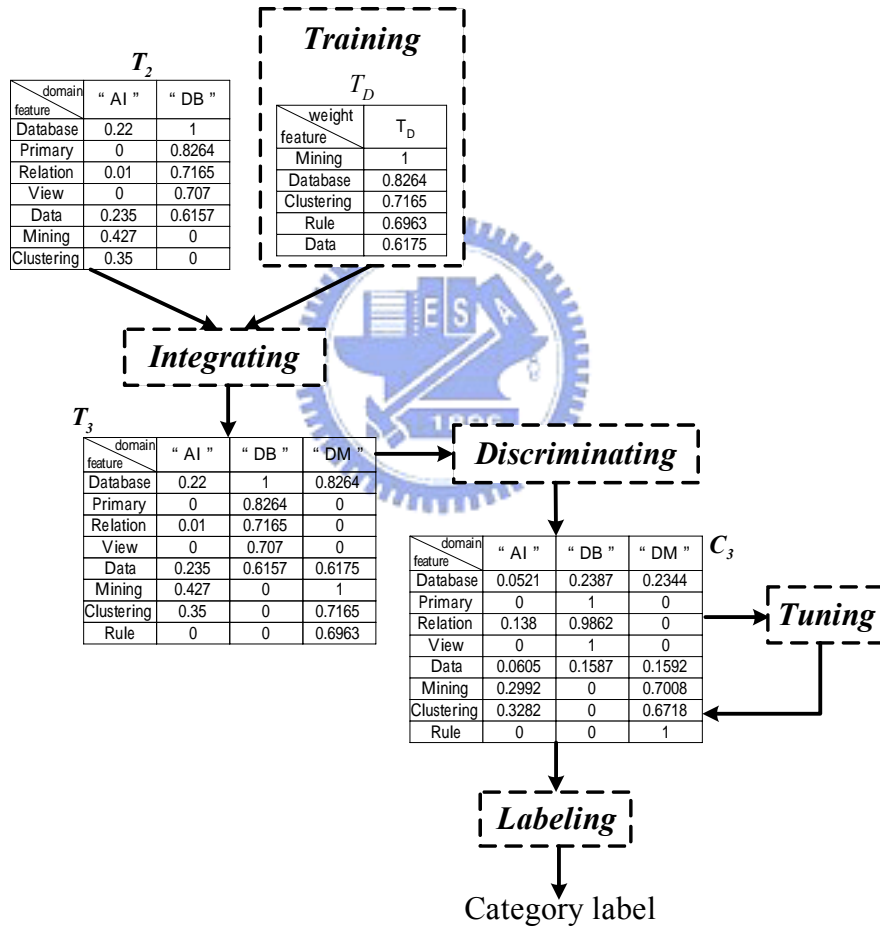


Figure 4-3: The operation of the classifier construction algorithm

Example 4-2: Figure 4-3 illustrates the operation of the classifier construction algorithm when a new category called "DM" is added. Assume T_2 is the

feature-domain weighting table which has been constructed with the “OS” Category and the “DB” Category. When the “DM” Category is added, the training algorithm extracts and weights features from the “DM” Category and stores the results in table T_D . After integrating T_D into T_2 , the feature-domain weighting table is updated to T_3 . The discrimination algorithm then reduces the weights of features in T_3 which have lower discriminating powers. The classifier C_3 is thus constructed. The tuning algorithm can use other given tuning documents to strengthen the classifier. The classifier C_3 can be used to classify an undefined document. ■

4.4.1 Training Phase

The purpose of the Training Phase is to extract representative features from documents in a given category. In this study, the features are keywords that occur more than once in at least one document in the given category, and they are extracted by a pre-processing procedure that removes stop words, punctuation and digits, converts all letters into lowercase, and stems using Porter’s stemmer. A feature is more representative for a category if it appears in more documents and has higher frequency in each document. The following formula is designed to calculate the weight w_k of the feature f_k for a given category:

$$w_k = T_k * \frac{\sum_j tf_{jk}}{\sum_k \sum_j tf_{jk}}, \text{ where } T_k = -\sum_j tf_{jk} * \log\left(\frac{tf_{jk}}{\sum_j tf_{jk}}\right), \quad (4-2)$$

where tf_{jk} denotes the frequency of f_k in document d_j .

The proposed training algorithm is shown in Figure 4-4. When a new category of documents D is added, the training algorithm extracts features from D (Step 1), and then calculates their feature weights by considering the frequency and coverage of each feature against the documents in D using Formula 4-2 (Step 2). After obtaining

and calculating feature weights, the training algorithm normalizes them in the range [0, 1] (Step 3.1), and adds them to table T_D (Step 3.2), which is used to retain the feature weights for D . Consequently, the training algorithm returns the weighting table T_D (Step 4).

Training Algorithm:

Input:
 D : A newly added category of documents.

Output:
 T_D : A table used to retain the feature weights for D .

Begin

(1) $F \leftarrow \{f_k \mid f_k \text{ is a feature in } D\}$;

(2) For each $f_k \in F$, do

(2.1) For each $d_j \in D$, count the frequency tf_{jk} of f_k in d_j ;

(2.2) Calculate the weight w_k of f_k using:

$$w_k = T_k * \frac{\sum_j tf_{jk}}{\sum_k \sum_j tf_{jk}}, \text{ where } T_k = -\sum_j tf_{jk} * \log\left(\frac{tf_{jk}}{\sum_j tf_{jk}}\right);$$

(3) For each $f_k \in F$, do

(3.1) $w_k = \frac{w_k}{\max\{w_1, w_2, \dots, w_k\}}$;

(3.2) $T_D \leftarrow T_D \cup w_k$;

(4) Return T_D .

End

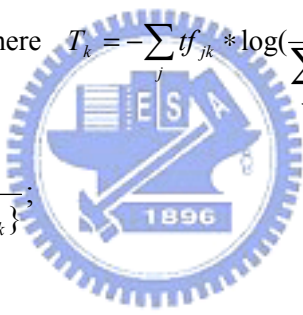


Figure 4-4: The training algorithm

Example 4-3: Assume the features, ‘Mining’, ‘Database’, ‘Clustering’, ‘Rule’ and ‘Data’, have been extracted from the three documents d_1, d_2, d_3 in the given “DM” Category. Table 4-2 shows the statistical information for these features. These five feature weights for the “DM” Category according to Formula 4-2 are shown in Table 4-3. Among them, the feature weight of ‘Mining’ is calculated as follows:

Since $\sum_j tf_{jk} = 165$, $\sum_k \sum_j tf_{jk} = 725$ and $T_k = -\sum_j tf_{jk} * \log\left(\frac{tf_{jk}}{\sum_j tf_{jk}}\right) = 78.725$, we have

$$w_k = T_k * \frac{\sum_j f_{jk}}{\sum_k \sum_j f_{jk}} = 78.725 * \frac{165}{725} = 17.917.$$

After being normalized, the feature weight of ‘Mining’ is set to 1. ■

Table 4-2: The statistic information of features in “DM” Category

Information Feature	d_1	d_2	d_3	$\sum_j f_{jk}$	T_k	w_k
Mining	55	55	55	165	78.725	17.917
Database	50	50	50	150	71.568	14.807
Clustering	40	50	50	140	66.479	12.837
Rule	60	40	40	140	64.604	12.475
Data	40	40	50	130	61.700	11.063

Table 4-3: The feature weights in “DM” Category

Feature	T_D
Mining	1
Database	0.8264
Clustering	0.7165
Rule	0.6963
Data	0.6175

4.4.2 Discrimination Phase

The purpose of the Discrimination Phase is to reduce the weights for features having lower discriminating powers. The discriminating power of a feature can be evaluated by calculating the *gini index value* [14][84] of its feature vector in the feature-domain weighting table. Assume a feature vector f_k in the feature-domain weighting table is represented as $\langle w_1, w_2, \dots, w_c \rangle$ and $w_T = \sum_{j=1}^c w_j$, where w_j denotes the weight between the feature f_k and the j -th category. The gini index value g_k of the feature f_k can be calculated using the following formula:

$$g_k = \sum_{j=1}^c \left(\frac{w_j}{w_T} \right)^2. \quad (4-3)$$

The lowest gini index value appears when $w_1 = w_2 = \dots = w_c = 1/c$, whereas the highest gini index value appears when only one $w_j = 1$ and the rest are 0. This idea is conceptually similar to the *idf* term in the *tfidf* function. A feature has higher discriminating power if it is included in fewer categories.

Discrimination Algorithm:

Input:

T : The feature-domain weighting table.
 δ : A discrimination threshold.

Output:

C : The classifier.

Begin

(1) For each feature f_k with feature vector $f_{v_k} = \langle w_1, w_2, \dots, w_c \rangle$ in T , do

(1.1) $f_{v_k} = \frac{f_{v_k}}{\sum_{j=1}^c w_j}$; //One-normalization

(1.2) Calculate the gini index value g_k of f_k using:

$$g_k = \sum_{j=1}^c w_j^2;$$

(1.3) If $g_k < \delta$, $f_{v_k} = f_{v_k} * g_k$;

(2) $C \leftarrow T$;

(3) Return C .

End

Figure 4-5: The discrimination algorithm

The proposed discrimination algorithm is shown in Figure 4-5. According to Formula 4-3, the discrimination algorithm first normalizes each feature vector in the feature-domain weighting table T such that $\|f_{v_k}\|_1 = 1$ (Step 1.1), and then calculates its corresponding gini index value (Step 1.2). If the feature's gini index value is less than the user-specified discrimination threshold δ , i.e., the feature's discriminating power does not satisfy the minimum requirement, the discrimination algorithm reduces the feature weights in T by multiplying the feature vector with its gini index value (Step

1.3). A classifier C can be therefore constructed (Step 2), since the weight between a document and each category can be easily calculated by summarizing its related feature vectors in T . Consequently, the training algorithm returns the classifier C (Step 3).

Example 4-4: Assume the discrimination threshold δ is set to 0.5. As in Figure 4-3, the discrimination algorithm will adjust the feature-domain weighting table T_3 to produce the classifier C_3 . For example, the feature vector of ‘Data’, $\langle 0.235, 0.6157, 0.6175 \rangle$, in T_3 is adjusted as follows. One-normalization of ‘Data’ is $\langle 0.235/1.4682, 0.6157/1.4682, 0.6175/1.4682 \rangle = \langle 0.16, 0.4194, 0.4206 \rangle$, and the gini index value of ‘Data’ is $0.16^2 + 0.4194^2 + 0.4206^2 = 0.3784$; since $0.3784 < 0.5$, the original feature vector is reduced to $\langle 0.16, 0.4194, 0.4206 \rangle * 0.3784 = \langle 0.0605, 0.1587, 0.1592 \rangle$. ■

4.4.3 Tuning Phase

The purpose of the Tuning Phase is to utilize feedback information from tuning documents (other pre-defined documents) to reduce the number of false positives yielded by the constructed classifier. Conceptually, it operates like the Perceptron learning algorithm [64] in neural network. Given a tuning document, the Tuning Phase first compares its pre-defined category label with the category label suggested by the constructed classifier. If they are consistent, it means that the classifier can correctly decide on this tuning document using the corresponding feature vectors in the feature-domain weighting table; the weight between each corresponding feature and the category suggested by the classifier is then emphasized, such that the classifier has *strong* weights. Otherwise, it means that the classifier may make incorrect decisions using the feature-domain weighting table. The weight between each corresponding feature and the category suggested by the classifier should be

reduced and the weight between each corresponding feature and the pre-defined category of the tuning document should be emphasized, such that the classifier has *appropriate* weights.

The proposed tuning algorithm, shown in Figure 4-6, first extracts features from each given tuning document (Step 1.1), and then obtains the category label suggested by the constructed classifier C (Step 1.2). The document labeling algorithm, described in next section, is used to carry out the suggestion procedure. If the category label suggested by C is consistent with the pre-defined category label of a tuning document, the tuning algorithm emphasizes the weight between each corresponding feature and the suggested category by ζ percent of the feature weight in the tuning document (Step 1.4), where ζ is the user-specified tuning parameter. Otherwise, the tuning algorithm reduces the weight between each corresponding feature and the suggested category by ζ percent of the feature weight in the tuning document and emphasizes the weight between each corresponding feature and the pre-defined category of a tuning document by ζ percent of the feature weight in the tuning document (Step 1.5). Consequently, the tuning algorithm returns the updated classifier C (Step 2).

Tuning Algorithm:

Input:

- C : The classifier.
- D' : A set of tuning documents.
- ζ : A tuning parameter.

Output:

- C : The updated classifier.

Begin

- (1) For each $d \in D'$, do
 - (1.1) $F \leftarrow \{f_k \mid f_k \text{ is a feature in } d\}$;
 - (1.2) $l \leftarrow \text{Document labeling}(d, C)$;
 - (1.3) $l_d =$ the pre-defined category label of d ;
 - (1.4) If $l = l_d$, do
 - (1.4.1) For each $f_k \in F$, do $w_{kl} = w_{kl} + d_l * \zeta$;
 // w_{kl} is the weight between f_k and l in C

```

//  $d_l$  is the  $l$ -th entry of  $d$ 's document vector
(1.5) If  $l \neq l_d$ , do
    (1.5.1) For each  $f_k \in F$ , do  $w_{kl} = w_{kl} - d_l * \zeta$  and  $w_{kld} = w_{kld} + d_l * \zeta$ 
(2) Return the updated classifier  $C$ .
End

```

Figure 4-6: The tuning algorithm

Example 4-5: Assume the tuning parameter ζ is set to 0.01 and a given tuning document d with two keywords, ‘Data’ and ‘Database’, belongs to “DM” Category. According to the constructed classifier C_3 in Figure 4-3, the document vector of d is thus $\langle (0.0605+0.0521)/2, (0.1587+0.2387)/2, (0.1592+0.2344)/2 \rangle = \langle 0.0563, 0.1987, 0.1968 \rangle$, and the classifier then assigns the category label “DB” to d . Obviously, the constructed classifier C_3 made an incorrect decision using the feature-domain weighting table. Thus, the tuning algorithm reduces the weight between feature ‘Data’ and “DB” Category to $0.1587-0.1987*0.01=0.1567$ and emphasizes the weight between feature ‘Data’ and “DM” Category to $0.1592+0.1968*0.01=0.1612$. On the other hand, the weight between feature ‘Database’ and “DB” Category is reduced to $0.2387-0.1987*0.01=0.2367$ and the weight between feature ‘Database’ and “DM” Category is emphasized to $0.2344+0.1968*0.01=0.2364$. The updated C_3 is shown in Table 4-4. ■

Table 4-4: An example of the tuning algorithm

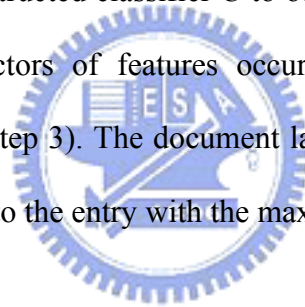
Feature \ Domain	AI	DB	DM
Database	0.0521	0.2367(1)	0.2364(3)
Primary	0	1	0
Relation	0.138	0.9852	0
View	0	1	0
Data	0.0605	0.1567(2)	0.1612(4)
Mining	0.2992	0	0.7008
Clustering	0.3282	0	0.6718
Rule	0	0	1

- (1) $0.2387 - 0.1987 * 0.01 = 0.2367$
- (2) $0.1587 - 0.1987 * 0.01 = 0.1567$
- (3) $0.2344 + 0.1968 * 0.01 = 0.2364$
- (4) $0.1592 + 0.1968 * 0.01 = 0.1612$

4.5 Document Labeling by the Constructed Classifier

According to the constructed classifier, a document vector is easily calculated by summarizing related feature vectors in the feature-domain weighting table. The larger the weight assigned to a document vector entry is the more relevant the entry is. Thus, the classifier can assign a category label to an undefined document on the basis of its entry weights.

Given an undefined document d , the document labeling algorithm, shown in Figure 4-7, first uses the constructed classifier C to obtain the document vector V_d by summarizing the feature vectors of features occurred in d from feature-domain weighting table (Step 2 and Step 3). The document labeling algorithm then assigns a category label to d according to the entry with the maximum weight in V_d (Step 4).



Document Labeling Algorithm:

Input:
 d : An undefined document.
 C : The classifier constructed by the classifier construction algorithm.

Output:
 l : The category label for d .

Begin

- (1) $V_d \leftarrow 0$; // V_d is the document vector of d and $|V_d|$ equals the number of categories
- (2) For each feature f_k in d , do
 - (2.1) Extract the feature vector f_{v_k} from C ;
 - (2.2) $V_d = V_d + f_{v_k}$;
- (3) $V_d = \frac{V_d}{\text{count}(f_k)}$; // $\text{count}(f_k)$ is the number of features in d
- (4) Return the category label l of the maximum weight in V_d .

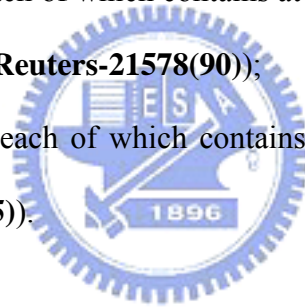
End

Figure 4-7: The document labeling algorithm

4.6 Experimental Results

Our experiments were conducted in Java on a personal computer with a Pentium 1.7GHz processor and 512MB of main memory running *Windows 2000*, and using the Reuters-21578 benchmark text collection standard (*REUTERS-21578*, Distribution 1.0) experimental dataset [58] based on the “ModApte” split version. This dataset consists of 118 categories in 12,902 documents, of which 9,603 are for training and 3,299 are for testing. The following groups of categories were used to evaluate classification accuracy:

- (1) the 10 categories with the largest number of training documents (**Reuters-21578(10)**);
- (2) the 90 categories, each of which contains at least one training document and one test document (**Reuters-21578(90)**);
- (3) the 115 categories, each of which contains at least one training document (**Reuters-21578(115)**).



We tested our classifier on four aspects of micro- and macro-averaging F_1 evaluation functions (shown in Formula 4-1):

- (1) the classification accuracy of our classifier construction algorithm compared to the algorithms shown in [23];
- (2) the influence of the training document threshold φ and the discrimination threshold δ on classification accuracy;
- (3) the influence of the number of tuning documents on classification accuracy;
- (4) the time performance of our classifier construction algorithm compared to a batch-based mining algorithm.

In [23], Debole and Sebastiani utilized six supervised term weighting functions, *chi-square*, *information gain*, and *gain ratio*, globally and locally, e.g., $\chi^2(g)$, $IG(g)$, $GR(g)$, $\chi^2(l)$, $IG(l)$, and $GR(l)$, in the *Rocchio*, *k-NN*, and *SVM* classifier construction algorithms to compare their average classification accuracy on the Reuters-21578(10), Reuters-21578(90), and Reuters-21578(115) datasets. The comparison results are shown in Table 4-5.

Table 4-5: Micro- and macro-averaging F_1 values shown in [23]

		$\chi^2(g)$	$IG(g)$	$GR(g)$	$\chi^2(l)$	$IG(l)$	$GR(l)$
<i>Micro F_1</i>	Reuters-21578(10)	0.852	0.843	0.857	0.810	0.816	0.816
	Reuters-21578(90)	0.795	0.750	0.803	0.758	0.767	0.767
	Reuters-21578(115)	0.793	0.747	0.800	0.756	0.765	0.765
<i>Macro F_1</i>	Reuters-21578(10)	0.725	0.707	0.739	0.674	0.684	0.684
	Reuters-21578(90)	0.542	0.377	0.589	0.527	0.559	0.559
	Reuters-21578(115)	0.596	0.458	0.629	0.581	0.608	0.608

We set the discrimination threshold δ in our classifier construction algorithm to 0.5 for the Reuters-21578(10) dataset, and to 0.04 for the Reuters-21578(90) and Reuters-21578(115) datasets; the number of tuning documents was set to 0. Table 4-6 shows the classification accuracy of our classifier at various training document thresholds φ . The φ was to determine the availability of categories in the training documents for our training algorithm. Thus, if the number of training documents in a category was less than the specified φ , the category was omitted from the training algorithm. For example, only 39 categories in Reuters-21578(90) satisfying $\varphi = 25$ were used in the training algorithm.

Tables 4-5 and 4-6 show the classification accuracy of our classifier construction algorithm was always better than those in [23] on Reuters-21578(10), whereas the results on Reuters-21578(90) and Reuters-21578(115) were worse when φ was less than 15. We may therefore conclude that the classification accuracy of the classifier

constructed by the domain-space weighting scheme will be getting better with sufficient training documents.

Table 4-6: Micro- and macro-averaging F_1 values at $\varphi = 1$, $\varphi = 15$ and $\varphi = 25$

		$\varphi=1$	$\varphi=15$	$\varphi=25$
<i>Micro F_1</i>	Reuters-21578(10)	0.903	0.903	0.903
	Reuters-21578(90)	0.751	0.784	0.815
	Reuters-21578(115)	0.737	0.784	0.815
<i>Macro F_1</i>	Reuters-21578(10)	0.824	0.824	0.824
	Reuters-21578(90)	0.490	0.569	0.660
	Reuters-21578(115)	0.616	0.569	0.660

Details of training document threshold φ and discrimination threshold δ affected classification accuracy on Reuters-21578(10), Reuters-21578(90), and Reuters-21578(115) are shown in Tables 4-7 to 4-11. Since each category in Reuters-21578(10) contains more than 50 training documents, the influence of φ is ignored in Table 4-7. As mentioned before, the scale of δ is determined according to the number of categories. Thus, the scale range of δ in Table 4-7 is $[1/10, 1]$, and the scale ranges of δ in Tables 4-8, 4-9 and in Tables 4-10, 4-11 are $[1/90, 1]$ and $[1/115, 1]$, respectively.

In Tables 4-7 to 4-11, we can see that the influence of δ is not evident even on Reuters-21578(10), perhaps because the one-normalization of the discrimination algorithm has achieved the purpose of discrimination such that setting δ has less influence on the classification accuracy. By contrast, setting φ had a decisive influence on classification accuracy: the larger the number of training document included, the better classification accuracy will be. Table 4-12 shows the number of remaining categories at various φ on Reuters-21578(10), Reuters-21578(90), and Reuters-21578(115). When φ was 15 or greater, the training algorithm considered the same numbers of categories on Reuters-21578(90) and Reuters-21578(115).

Table 4-7: Micro-and macro-averaging F_1 values at various δ for Reuters-21578(10)

δ	<i>Micro F_1</i>	<i>Macro F_1</i>
0.9	0.902511370	0.814721475
0.8	0.901324896	0.813716994
0.7	0.903302353	0.820149529
0.6	0.903302353	0.819969831
0.5	0.902906862	0.823657403
0.4	0.898951948	0.815825122
0.3	0.901324896	0.817534791
0.2	0.895788017	0.804622957
0.1	0.898160965	0.806951786

Table 4-8: Micro-averaging F_1 values at various δ and φ for Reuters-21578(90)

$\delta \backslash \varphi$	1	5	15	25	35	45
0.1	0.74739	0.75360	0.78372	0.81300	0.82566	0.84547
0.08	0.74827	0.75389	0.78403	0.81269	0.82631	0.84447
0.06	0.75033	0.75478	0.78464	0.81458	0.82695	0.84681
0.04	0.75063	0.75300	0.78433	0.81521	0.82824	0.84681
0.02	0.74974	0.75271	0.78555	0.81553	0.8289	0.84681
0.01	0.74974	0.75330	0.78555	0.81584	0.8289	0.84681

Table 4-9: Macro-averaging F_1 values at various δ and φ for Reuters-21578(90)

$\delta \backslash \varphi$	1	5	15	25	35	45
0.1	0.46830	0.52281	0.56963	0.66335	0.67258	0.71811
0.08	0.48881	0.54619	0.57344	0.65812	0.67529	0.71390
0.06	0.48748	0.53001	0.57152	0.66360	0.67395	0.71542
0.04	0.48997	0.52214	0.56868	0.65998	0.67747	0.71738
0.02	0.48467	0.51960	0.57205	0.66281	0.67783	0.71738
0.01	0.48922	0.52176	0.57205	0.66312	0.67783	0.71738

Table 4-10: Micro-averaging F_1 values at various δ and φ for Reuters-21578(115)

$\delta \backslash \varphi$	1	5	15	25	35	45
0.1	0.73593	0.74885	0.78372	0.81300	0.82566	0.71811
0.08	0.73505	0.74915	0.78403	0.81269	0.82631	0.71390
0.06	0.73711	0.7506	0.78464	0.81458	0.82695	0.71542
0.04	0.73681	0.74944	0.78433	0.81521	0.82824	0.71738
0.02	0.73652	0.74855	0.78555	0.81553	0.8289	0.71738
0.01	0.73711	0.74915	0.78555	0.81553	0.8289	0.71738

Table 4-11: Macro-averaging F_1 values at various δ and φ for Reuters-21578(115)

$\delta \backslash \varphi$	1	5	15	25	35	45
0.1	0.62378	0.53231	0.56963	0.66335	0.67258	0.71811
0.08	0.60384	0.55127	0.57344	0.65812	0.67529	0.71390
0.06	0.60474	0.53598	0.57152	0.66360	0.67395	0.71542
0.04	0.61597	0.53130	0.56868	0.65998	0.67747	0.71738
0.02	0.61526	0.53057	0.55990	0.66312	0.67783	0.71738
0.01	0.61526	0.52903	0.57205	0.66281	0.67783	0.71738

Table 4-12: Numbers of remaining categories at various φ

	$\varphi=1$	$\varphi=5$	$\varphi=15$	$\varphi=25$	$\varphi=35$	$\varphi=45$
Reuters-21578(10)	10	10	10	10	10	10
Reuters-21578(90)	90	69	51	39	34	27
Reuters-21578(115)	115	70	51	39	34	27

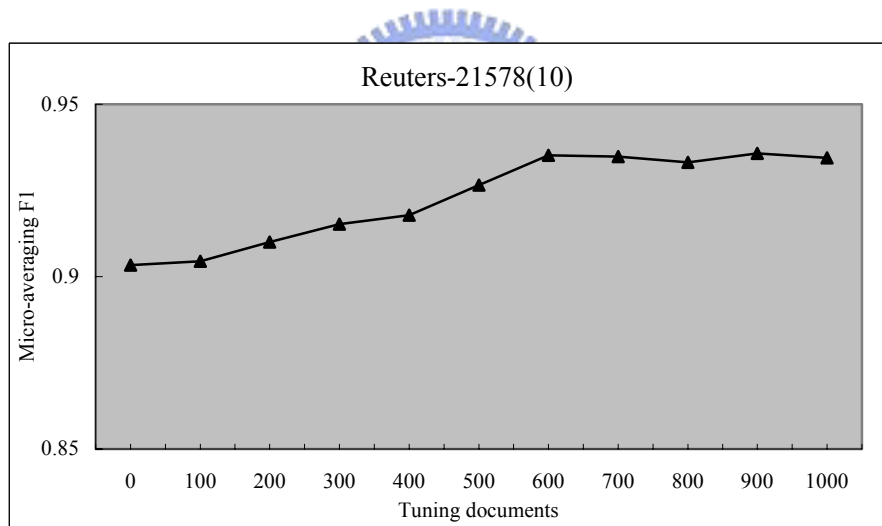


Figure 4-8: Micro-averaging F_1 value vs. number of tuning documents for Reuters-21578(10)

The influence of tuning document number on classification accuracy for Reuters-21578(10), Reuters-21578(90), and Reuters-21578(115) is shown in Figures 4-8, 4-9 and 4-10, respectively. Since the tuning documents in our experiments were selected from the test documents, the original test document dataset was divided into

tuning and test sets. Experimental results showed that setting the tuning parameter ζ to 0.000005 yielded a stably increasing trend. Too low the ζ value may lead to a tuning adjustment so tiny that the tuning effect is insignificant, and too large the ζ value may lead to an unstable and oscillatory tuning adjustment with unpredictable tuning effects. Figures 4-8 to 4-10 show that the classification accuracy of the constructed classifier improved as the number of tuning documents was increased and tended toward convergence when the number exceeded 700.

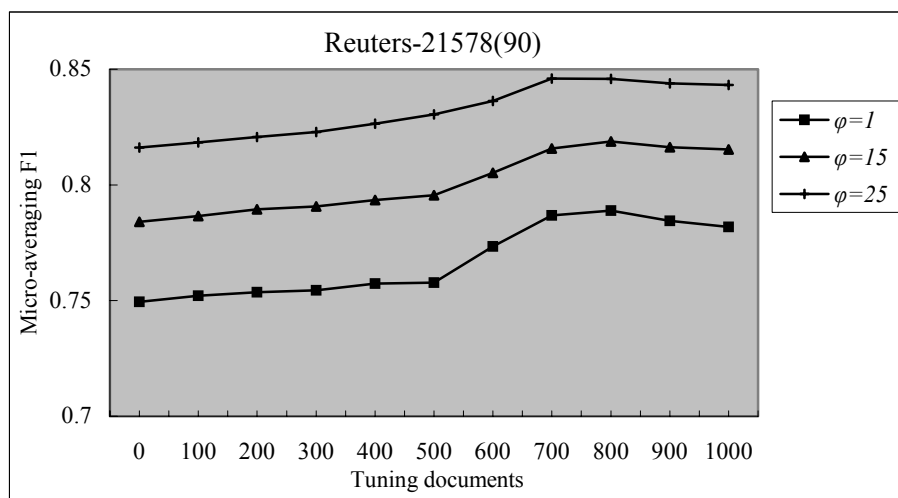


Figure 4-9: Micro-averaging F_1 values vs. number of tuning documents at $\phi=1$, $\phi=15$ and $\phi=25$ for Reuters-21578(90)

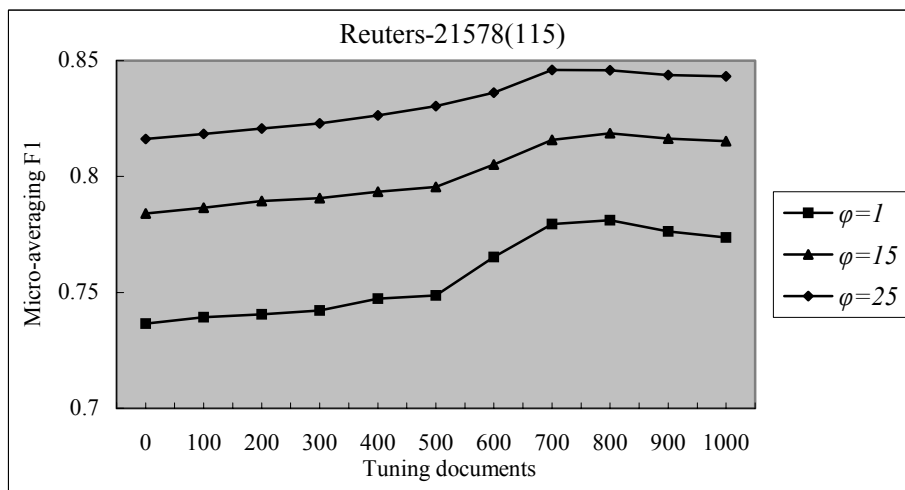


Figure 4-10: Micro-averaging F_1 values vs. number of tuning documents at $\varphi = 1$, $\varphi = 15$ and $\varphi = 25$ for Reuters-21578(115)

We evaluated the efficiency of our classifier construction algorithm in comparison with a batch-based classifier construction approach, excluding the tuning algorithm. The computation time of our classifier construction algorithm contains three major portions when a new category is added in the i -th run: (1) time to extract and weight features from a given category, denoted as t_{i1} ; (2) time to integrate the training results into the feature-domain weighting table, denoted as t_{i2} ; and (3) time to reduce the weights of features in the feature-domain weighting table having lower discriminating powers, denoted as t_{i3} . Since $t_{i1} > t_{i2} \gg t_{i3}$, total computation time can be simplified to $O(t_{i1} + t_{i2})$ in the i -th run. However, when our classifier construction algorithm mimicked a batch-based approach, and needed to re-process all previous categories to reconstruct its classifier for each run, the total computation time was $O(\sum_{j=1}^i (t_{j1} + t_{j2}))$ for the i -th run. Figure 4-11 shows the computation times spent by our classifier construction algorithm respectively in batch and in incremental for Reuters-21578(10) with increasing numbers of considered categories.

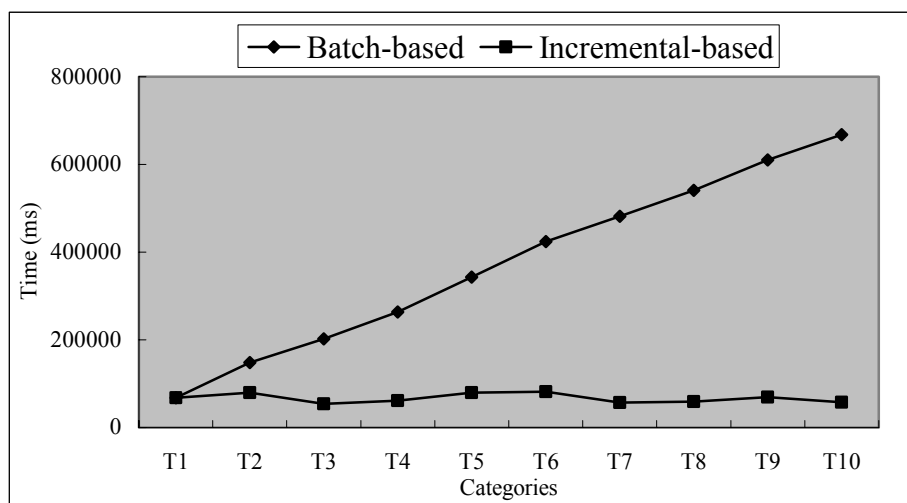


Figure 4-11: Computation times spent by the batch-based classifier and the

It is easily seen that, the computation times for the batch-based classifier increased as the number of involved categories was increased, but the computation times for the incremental-based classifier remained almost the same as the number of involved categories was increased. Since previously discovered information is all retained in the feature-domain weighting table, the classification accuracy of the incremental-based classifier is the same as that of the batch-based classifier.

4.7 Conclusions

This study proposes a *domain-space weighting scheme* to represent documents in domain-space and incrementally construct a classifier to resolve the document representation and categories adaptation problems. The scheme consists of three major phases: *Training Phase*, *Discrimination Phase* and *Tuning Phase*. The training algorithm incrementally extracts and weights features from each individual category, and then integrates the results into a *feature-domain weighting table*. The discrimination algorithm reduces feature weights with lower discriminating powers. When these algorithms finish constructing the classifier, the tuning algorithm strengthens it using feedback information from tuning documents to reduce the number of false positives. Experiments with the Reuters-21578 benchmark show that with sufficient training documents, the classifier is rather effective and efficient.

Chapter 5

From Incremental Mining to Multidimensional Online Mining for Knowledge Discovery

5.1 Introduction

Although incremental mining algorithms are rather efficient and useful for static models such as *mining all the data accumulated thus far* and *mining only a recently collected portion of data* in uncomplicated applications, they usually provide little support for user focus (e.g., limiting the computation to what interests the user) and user interaction (e.g., dynamically changing the parameters or constraints). This may produce thousands of rules that are irrelevant and uninteresting to users. On the other hand, decision-makers usually diversely consider problems at different aspects: they may need to analyze market demands, customer preferences, localities, and short-term/long-term trends; they may want to understand the change of discovered patterns or rules in different dimensions. This may neither flexibly obtain rules or patterns from their interesting portions of data, nor diversely consider problems at different aspects to provide online decision supports for users.

Some examples about that a decision-maker usually requires online mining supports of association rules are shown below.

Scenario 1: A decision-maker may have known which product combinations sold in last August were popular, and wants to know which product combinations sold in last September were also popular.

Scenario 2: A decision-maker may have known that people often buy beer and diapers together from a transaction database, and want to further know under what contexts (e.g., place, month, or branch) this pattern is significant or, oppositely, under what contexts this pattern becomes insignificant.

Scenario 3: A decision-maker may want to know how the mined patterns this year differ from those last year, such as what new patterns appear and what old patterns disappear.

Scenario 4: A marketing analyst may want to analyze the data collected from the branches in Los Angeles and San Francisco in all the first quarters in the last five years.

Scenario 5: A marketing analyst may want to know what patterns are significant in the recent month when the minimum support increases from 5% to 10%.

The examples above all require more context information to describe the problem domain. A mining algorithm that can handle relevant context information in mining requests will thus help decision-makers consider various aspects of problems in diverse ways.

Constraint-based and multidimensional mining techniques [11][15][35][37][48][51][52][65][72] which allow users to specify constraints as a guidance have thus been developed to identify and extract interesting and focused knowledge from a data warehouse or a database. Users can continually express his focus and change not only the parameters but also the constraints in the mining process. For example, Kamber *et al.* [48] proposed a famous approach that allowed users to specify the predicates that appear in antecedent and consequent parts of association rules. However, putting all data gathered in different contexts (such as different branches, different time intervals and different regions) together for centralized mining seems to be time-consuming

and infeasible for online mining support because of the size of data. Users may need to wait for a long period of time for the mining results.

Different from the techniques of constraint-based and multidimensional mining, we attempt to extend the concept of *effectively utilizing previously discovered patterns* in incremental mining to support multidimensional online mining. We first systematically mines rules or patterns from data gathered in different contexts according to the pre-defined parameter setting, and forwards the rules or patterns with the corresponding context information to a structural repository called *knowledge warehouse* for centralized post-mining and refining. Then, we can efficiently acquire user-interesting and/or user-focused association rules or patterns by integrating related mining information from the knowledge warehouse, and greatly reduce the cost of mining the underlying data at each time.

Consequently, a systematic, automatic, integrated, and on-demand architecture, called *Online Knowledge Discovery System* (OKDS), can be developed to help managers and decision-makers diversely consider problems at different aspects and provide online mining supports. The OKDS mainly consists of five major components, *knowledge client*, *knowledge warehouse*, *knowledge organizer*, *mining agent*, and *underlying storage facility*. Through the mining agents systematically and continuously mine potentially useful patterns from each underlying storage facilities, the knowledge organizer structurally stores these mined patterns into the knowledge warehouse, and thus users can utilize aggregation and generalization functions in the knowledge client for online patterns generation.

5.2 Related Work

Data warehouse is an integrated, subject-oriented, and nonvolatile data

repository containing historical and aggregated data from operational and legacy systems for supporting decision-making processes [17][45][97]. Comparing to routine works of *On-Line Transaction Processing* (OLTP) in the operational databases, the purpose of data warehouse is to help analysts *On-Line Analytical Processing* (OLAP). Therefore, to facilitate complex analyses and achieve high query throughput is the most important consideration in the data warehouse. Table 5-1 lists the major differences between the operational database and the data warehouse [17][45]. Thus, data warehouses are usually maintained separately from the organization's operational databases.

Table 5-1: Differences between the operational database and the data warehouse

<i>Aspects</i>	<i>Operational database</i>	<i>Data Warehouse</i>
<i>User</i>	<ul style="list-style-type: none"> ● Data entry clerk ● System designer ● System administrator 	<ul style="list-style-type: none"> ● Decision maker ● Knowledge worker ● Executives
<i>Function</i>	<ul style="list-style-type: none"> ● Daily operations ● OLTP 	<ul style="list-style-type: none"> ● Decision support ● OLAP
<i>DB Design</i>	<ul style="list-style-type: none"> ● Application oriented 	<ul style="list-style-type: none"> ● Subject oriented
<i>Data</i>	<ul style="list-style-type: none"> ● Current ● Up-to-date atomic ● Relational(normalized) ● Isolated 	<ul style="list-style-type: none"> ● Historical ● Summarized ● Multidimensional ● Integrated
<i>Usage</i>	<ul style="list-style-type: none"> ● Repetitive routine 	<ul style="list-style-type: none"> ● Ad hoc
<i>Access</i>	<ul style="list-style-type: none"> ● Read/write ● Simple transaction 	<ul style="list-style-type: none"> ● Read mostly ● Complex query
<i>System Requirements</i>	<ul style="list-style-type: none"> ● Transaction throughput ● Data consistency 	<ul style="list-style-type: none"> ● Query throughput ● Data accuracy

Developing a data warehouse often extracts user-interesting information from each source (operational database) in advance, then merging the relevant information, and consequently installing into a structurally centralized repository for later analysis. The data warehouse often adopts a *multidimensional data model* to prepare the data for analytical processing under multidimensional consideration. The *star schema*

consisting of a *fact table* and a set of *dimension tables* is the most used form in the multidimensional data model. The fact table contains user-interesting measure attributes, which are the objects for analysis, and key attributes (identifiable attributes) to each of the related dimension tables. The dimension table contains additional attributes to further describe each of key attributes in the fact table. The multidimensional data model provides users a clear and multidimensional view of data. Data can be easily accessed by manipulating the dimensions.

5.3 Knowledge Warehouse

For providing efficient online mining, the knowledge warehouse is initiated from the concept of *effectively utilizing previously discovered patterns* in incremental mining. As we know, for not wasting the previously mined patterns and improving rule maintenance performance, incremental mining algorithms always keep the mined patterns into the storage for later use. For providing multidimensional consideration, the knowledge warehouse is further referred to the multidimensional data model of data warehouse capable of supporting ad-hoc queries and decision making by aggregation functions and OLAP operations.

As the data under decision-support consideration does not evolve in an arbitrary way (e.g., the data in the data warehouse may be inserted or deleted in a block during an interval of a month [32]), the knowledge warehouse is thus proposed to structurally and systematically store the *context information* and *mining information* for each inserted dataset. The context information is used to represent the contexts of each individual block of data which are gathered together from a specific business viewpoint, such as region, time and branch. The mining information is used to record the available information mined from each individual block of data by a batch mining

algorithm, such as the number of data, the number of mined patterns, and the set of previously mined patterns with related information. Conceptually, the knowledge warehouse is similar to the data warehouse for OLAP. Both of them systematically preprocess the underlying data in advance, integrate related information, and store the results in a centralized structural repository for later use and analysis. However, the data warehouse is mainly used to store mined patterns at knowledge level but not data at information level. Table 5-2 lists the major differences between the knowledge warehouse and the data warehouse.

Table 5-2: Differences between the knowledge warehouse and the data warehouse

<i>Aspects</i>	<i>Data Warehouse</i>	<i>Knowledge Warehouse</i>
<i>Function</i>	● OLAP	● Online mining
<i>Data</i>	● Historical ● Summarized ● Multidimensional ● Integrated	● Mined ● Multidimensional
<i>Access</i>	● Read mostly ● Complex query	● Read only ● Mining query
<i>System Requirements</i>	● Query throughput ● Data accuracy	● Mining throughput ● Knowledge usability

The star schema can still be a concise and organized structure to model the knowledge warehouse. The context information and mining information can be represented by dimensions and measures, respectively. Example 5-1 shows a star schema of the knowledge warehouse used to provide online generation of association rules for product sales in a bicycle manufacturer. However, unlike the summarized information on measure attributes in the data warehouse, the mining information in the knowledge warehouse, such as the mined patterns, may not be directly aggregated to satisfy users' mining requests. Thus, the major challenge of the knowledge warehouse is how to efficiently aggregate, generalize and manipulate the mining

information. In the next chapter, we will design corresponding aggregation and generalization approaches to provide online mining supports on association rules.

Example 5-1: Figure 5-1 is a star schema of the knowledge warehouse for a bicycle manufacturer. It consists of three dimensions, *Time*, *Branch* and *Minsup*, and three measures, *No_Trans*, *No_Patterns* and *Pattern_Set*. Of the three dimensions, *Time* and *Branch* are nonnumeric dimension similar to that in a typical data warehouse, and *Minsup* is a numeric dimension indicating the minimum supports for the measures, *No_Patterns* and *Pattern_Set*. Of the three measures, *No_Trans* is a numeric measure that can be calculated similar to that in a typical data warehouse, and *No_Patterns* is also a numeric measure and decided by *Pattern_Set*, and *Pattern_Set* is a set measure that represents a collection of frequent itemsets with their supports under the corresponding time and branches and satisfying a minimum support in *Minsup* dimension.

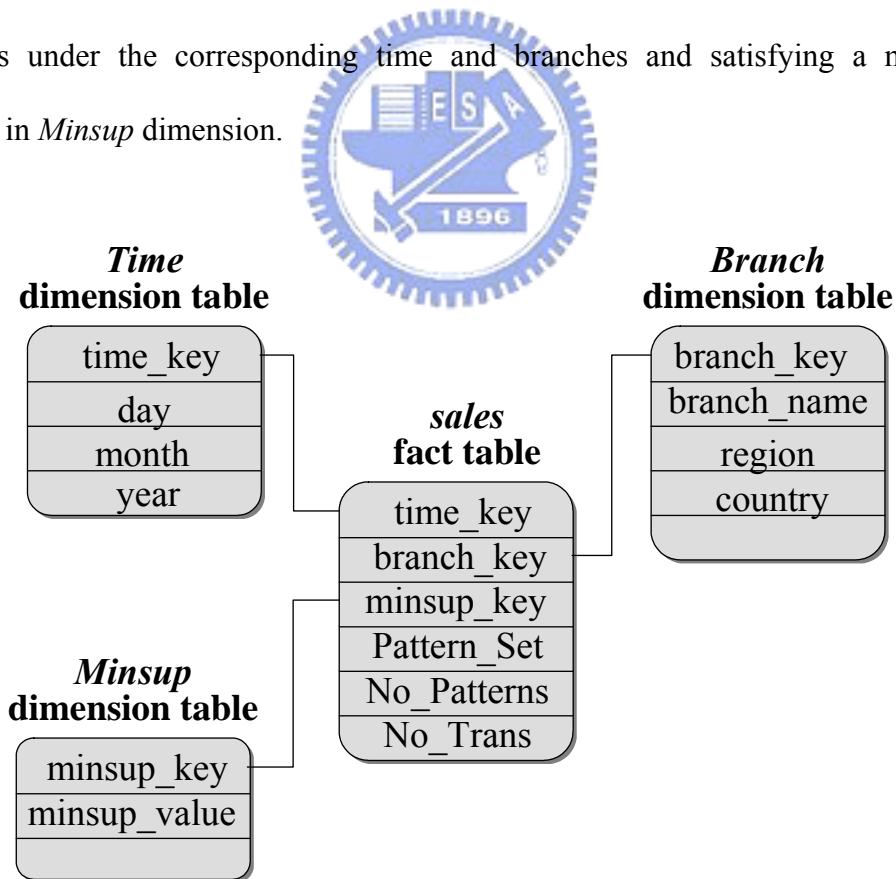


Figure 5-1: An example of the star schema of a knowledge warehouse

5.4 Online Knowledge Discovery System (OKDS)

Based on the proposed knowledge warehouse, a systematic, automatic, integrated, and on-demand architecture, called *Online Knowledge Discovery System (OKDS)*, can be developed to provide managers and decision-makers multidimensional online mining supports. The OKDS, as shown in Figure 5-2, mainly consists of five major components, *knowledge client*, *knowledge warehouse*, *knowledge organizer*, *mining agent*, and *underlying storage facility*.

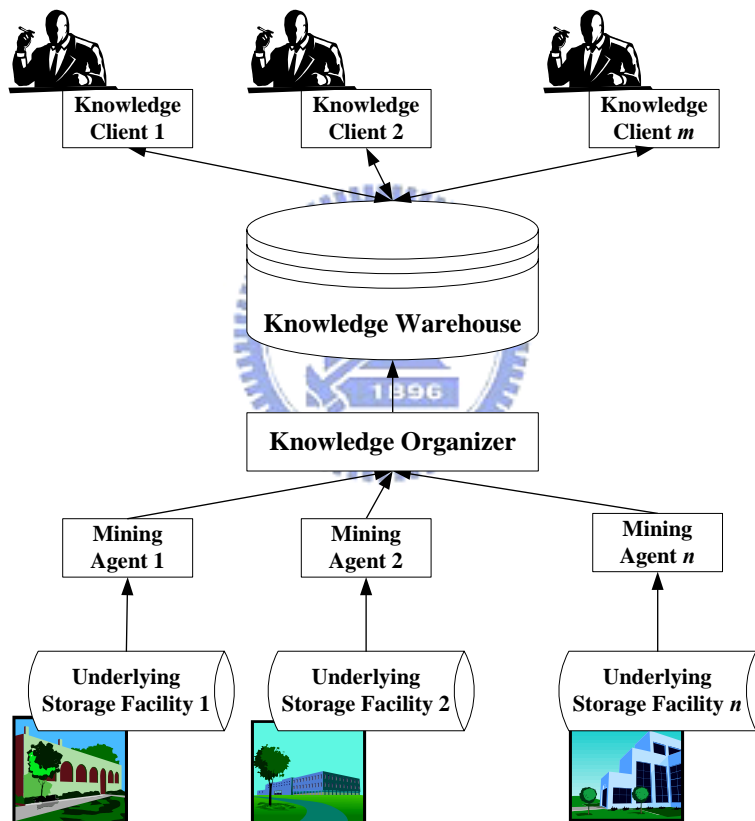


Figure 5-2: The OKDS architecture

Whenever a new block of data is inserted into a underlying storage facility, the corresponding mining agent will systematically and continuously mine potentially useful patterns from the block of data as the mining information; then the knowledge

organizer will structurally store the mining information associated with related context information in the knowledge warehouse; and thus users can utilize aggregation and generalization functions in the knowledge client for online generation of patterns. On the other hand, when an old block of data is deleted from a underlying storage facility, its corresponding context and mining information will be removed from the knowledge warehouse by the knowledge organizer.

- Underlying storage facility: A underlying storage facility is served as materials supplier in OKDS to provide underling, purpose-oriented and pre-processed data. Therefore, it can be a data warehouse, a preprocessed database or a cleaned file.
- Mining agent: *Agents* often play autonomous, adaptive and intelligent roles in a distributed system. For example, for an intelligent travel service system, a traveling agent follows the user setting or the user profile to collect interesting traveling paths and hotel coupons; a scheduling agent follows the user program, weather prediction and news to provide proper periods for traveling; and a coordinator agent is responsible for coordinate the traveling and scheduling agents capable of obtaining proper traveling packages and suitable schedules for users. Thus, a mining agent mainly follows the user setting to periodically detect the data changes in a underlying storage facility, automatically mining potential patterns from a underlying storage facility and reporting the results to the knowledge organizer. The knowledge organizer is served as the coordinator agent of mining agents.
- Knowledge organizer: The knowledge organizer is responsible for periodically maintaining the patterns in the knowledge warehouse. Its works include collecting the discovered patterns from each mining agent, merging or summarizing these ones as the mining information, and then storing them

associated with corresponding context information into the knowledge warehouse. For improving the performance of fulfilling user requests, the knowledge organizer is also responsible for constructing and maintaining the materialized views of knowledge warehouse.

- Knowledge client: A knowledge client is an interface used for receiving user's mining requests, transferring a mining request to an operating procedure for the knowledge warehouse, and reporting mining results to users. For convenient to understand and evaluate the mining results, it is also responsible for providing visualization services.



Chapter 6

Multidimensional Online Mining Algorithms for Generation of Association Rules

6.1 Introduction

Previous works on mining association rules can be classified into batch mining [5][16][40][53][61][67][78][95] and incremental mining approaches [8][20][21][27][43][77][85] according to their processing procedures. Most focus on finding association rules in specified parts of databases that satisfy the user-specified minimum support and minimum confidence [11][15][37][52][65]. Some contexts (circumstances) such as region, time, and branch are usually ignored in mining requests, and thus they usually can not flexibly obtain association rules from portions of data, diversely consider problems and provide on-line decision supports for users.

To provide ad-hoc, query-driven and online mining support for generation of association rules, we first propose a relation called the *Multidimensional Pattern Relation* (MPR) as a form of knowledge warehouse to structurally and systematically store context and mining information for later analysis [90][92]. We then develop an online mining approach called *Three-phase Online Association Rule Mining* (TOARM) based on this proposed MPR to support online generation of association rules under multidimensional considerations. The TOARM approach consists of three phases, *candidate itemset generation*, *candidate itemset reduction*, and *association rule generation*, during which final sets of patterns satisfying various mining requests

are found. The candidate itemset generation phase selects tuples that satisfy the context constraints in mining requests and generates candidate itemsets from the matching tuples. The candidate itemset reduction phase then calculates upper-bound supports for the candidate itemsets and uses two pruning strategies to reduce the number of candidates. Finally, the association rule generation phase finds final frequent itemsets and derives association rules from them.

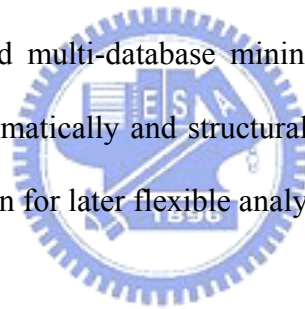
6.2 Related Work

Recently, researchers have developed online mining algorithms to obtain required sets of association rules without re-processing the entire database whenever user-specified thresholds are changed. Examples are the OLAP-style algorithm proposed by Aggarwal and Yu [1] and the Carma algorithm proposed by Hider [41]. The OLAP-style algorithm is quite similar to a typical incremental mining algorithm that utilizes previously mined patterns to save on I/O and computation. It first stores *primary itemsets* based on a low minimum support criterion in a *latticed* data structure, and then responds to users' queries with higher minimum support criteria by processing the lattice. It thus preprocesses the data just once, but can efficiently handle multiple user queries. The Carma algorithm attempts to provide intermediate results as feedback to users while databases or minimum support thresholds are being changed. Users are thus able to dynamically adjust thresholds according to intermediate results. The Carma algorithm uses two runs. During the first run, it constructs a lattice composed of all potential frequent itemsets from the transactions. Each itemset in the lattice uses a lower bound and an upper bound to record its possible support range. When a mining request is input, itemsets in the lattice whose support ranges cover or are larger than the new minimum support threshold are output

to the second run. During the second run, the Carma algorithm finds the precise support for each itemset from the first run to determine whether it is truly large.

Interestingly, many large organizations have multiple databases distributed at different branches. Traditional data mining algorithms may put all data from different databases in a common repository for centralized analysis. This kind of mining causes some problems. The collected data may be too huge to be coped with, and some useful rules or patterns regarding local databases may be lost. As a result, multi-database mining has recently been recognized as an important research topic and some studies [50][98][105] on mining association rules over multi-databases have been proposed. These approaches mine rules or patterns at different databases and then gather the mined results.

These online mining and multi-database mining approaches do not, however, maintain a repository to systematically and structurally store the mining information and related context information for later flexible analysis.



6.3 Multidimensional Pattern Relation (MPR)

In this section, we formally define the *Multidimensional Pattern Relation* (MPR) for storing context information and mining information for later analysis. First, a relation schema R , denoted by $R(A_1, A_2, \dots, A_n)$, is made up of a relation name R and a list of attributes A_1, A_2, \dots, A_n . Each attribute A_i is associated with a set of attribute values, called the domain of A_i and denoted by $dom(A_i)$. A relation r of the relation schema $R(A_1, A_2, \dots, A_n)$ is a set of tuples $\{t_1, t_2, \dots, t_m\}$. Each tuple t_i is an ordered list of n values $\langle v_{i1}, v_{i2}, \dots, v_{in} \rangle$, where each value v_{ij} is an element of $dom(A_j)$.

A *Multidimensional Pattern Relation Schema* (MPRS) is a special relation schema for storing mining information. An MPRS consists of three types of attributes:

identification (ID), *context*, and *content*. There is only one identification attribute for an MPRS. It is used to uniquely label tuples. Context attributes describe the contexts (circumstances) of an individual data block, gathered together from a specific business viewpoint. Examples of context attributes are region, time, and branch. Content attributes describe available mining information discovered from each individual data block by a batch mining algorithm. Examples of content attributes are number of transactions, number of mined patterns, and the set of previously mined frequent itemsets with supports.

The set of all patterns, with supports, previously mined from an individual data block is called a *pattern set (ps)* in this study. Assume the minimum support is s and l frequent itemsets are discovered in a data block. A pattern set can be represented as $ps = \{(x_i, s_i) \mid s_i \geq s \text{ and } 1 \leq i \leq l\}$, where x_i is a frequent itemset and s_i is its support. The pattern set is thus an essential content attribute of an inserted block of data.

An MPRS with n_1 context attributes and n_2 content attributes can be represented as $MPRS(ID, CX_1, CX_2, \dots, CX_{n_1}, CN_1, CN_2, \dots, CN_{n_2})$, where ID is an identification attribute, CX_i , $1 \leq i \leq n_1$, is a context attribute, and CN_i , $1 \leq i \leq n_2$, is a content attribute. Assume the MPR to be an instance of a given MPRS that includes the tuples $\{t_1, t_2, \dots, t_m\}$. Each tuple $t_i = (id_i, cx_{i1}, cx_{i2}, \dots, cx_{in_1}, cn_{i1}, cn_{i2}, \dots, cn_{in_2})$ in MPR indicates that for the block of data identified by the contexts cx_{i1}, cx_{i2}, \dots , and cx_{in_1} , the mined information contains cn_{i1}, cn_{i2}, \dots , and cn_{in_2} .

Table 6-1: An MPR with minimum support = 5%

<i>ID</i>	<i>Region</i>	<i>Branch</i>	<i>Time</i>	<i>No_Trans</i>	<i>No_Patterns</i>	<i>Pattern_Sets (Itemset, Support)</i>
1	CA	San Francisco	2003/10	10000	7	(A,10%),(B,11%),(C,9%), (AB,8%),(AC,7%),(BC,6 %),(ABC,6%)

2	CA	San Francisco	2003/11	15000	3	(A,5%),(B,7%),(C,5%)
3	CA	San Francisco	2003/12	12000	2	(A,5%),(C,9%)
4	CA	Los Angeles	2003/10	20000	3	(A,8%),(B,6%),(F,5%)
5	CA	Los Angeles	2003/11	25000	2	(A,5%),(C,6%)
6	CA	Los Angeles	2003/12	30000	4	(A,6%),(B,6%),(C,9%), (AB,6%)
7	NY	New York	2003/10	18000	3	(B,8%),(C,7%),(BC,6%)
8	NY	New York	2003/11	18500	2	(B,8%),(C,6%)
9	NY	New York	2003/12	19000	5	(A,5%),(B,9%),(C,8%), (D,6%),(BC,6%)

Example 6-1: Table 6-1 shows an MPR with the initial minimum support set to 5%. *ID* is the identification attribute, *Region*, *Branch* and *Time* are context attributes, and *No_Trans*, *No_Patterns* and *Pattern_Sets* are content attributes. The *Pattern_Sets* attribute records the sets of frequent itemsets mined from previous data blocks. For example, the tuple *ID* = 1 shows that seven frequent itemsets, {(A, 10%), (B, 11%), (C, 9%), (AB, 8%), (AC, 7%), (BC, 6%), and (ABC, 6%)}, were discovered from 10000 transactions and in the contexts of *Region* = CA, *Branch* = San Francisco, and *Time* = 2003/10. The other tuples have similar meanings. ■

6.4 Three-Phased Online Association Rule Mining (TOARM) based on MPR

The goal of online mining is to find association rules satisfying the constraints in mining requests. The flexibility of mining requests can be increased by using the proposed MPR. An online mining approach called *Three-phase Online Association Rule Mining* (TOARM) is proposed to carry out mining tasks with an MPR. TOARM first selects tuples from the relation that satisfy the constraints in a mining request. It then integrates the mined information in these tuples and outputs them to users. Before describing the TOARM approach, we first formally define the problem to be solved and some related terminology. Some lemmas are also derived and proven.

Assume $MPR = \{t_1, t_2, \dots, t_m\}$ is a multidimensional pattern relation based on an

initial minimum support s . Given a mining request q with the set of contexts cx_q , the new minimum support s_q ($s_q \geq s$), and the new minimum confidence $conf_q$, the TOARM approach will effectively and efficiently derive association rules satisfying s_q , $conf_q$ and cx_q . Tuples with cx_q in an MPR are called *matched tuples* (mt). Let t_i denote the i -th tuple in an MPR, $t_i.trans$ the number of transactions in t_i , $t_i.ps$ the pattern set in t_i , and $t_i.s_x$ the actual support of an itemset x in t_i . Lemma 6-1 is easily derived as follows.

Lemma 6-1: For each itemset x satisfying s_q and cx_q in a mining request q , there exists at least a matched tuple t , such that $t.s_x$ satisfies s_q .

Proof: We prove the lemma by contradiction. If $t_i.s_x < s_q$ for each matched tuple t_i , then:

$$\sum_{t_i \in mt} t_i.trans * t_i.s_x < \sum_{t_i \in mt} t_i.trans * s_q. \quad (6-1)$$

It implies that the itemset x does not satisfy s_q , contradicting the claim that x satisfies s_q . Thus, there must exist at least a matched tuple t with $t.s_x \geq s_q$. ■

According to Lemma 6-1, an itemset with support greater than or equal to s_q in at least one matched tuple is a possible candidate. The following lemma about *candidate itemsets* can thus be derived.

Lemma 6-2: Each itemset x satisfying s_q and cx_q in a mining request q must be among the candidate itemsets obtained by collecting the ones whose supports are greater than or equal to s_q in at least one matched tuple. ■

Example 6-2: For the MPR given in Table 6-1, assume that a mining request q calls for getting the patterns under the contexts cx_q of *Region = CA* and *Time = 2003/11~2003/12* and satisfying the minimum support $s_q = 5.5\%$. The matched tuples are shown in Table 6-2. According to Lemma 6-2, the set of candidate itemsets is $\{\{A\}, \{B\}, \{C\}, \{AB\}\}$, which is the union of the itemsets appearing in the pattern

sets with supports greater than 5.5%. ■

Table 6-2: Matched tuples in Example 6-2

<i>ID</i>	<i>Region</i>	<i>Branch</i>	<i>Time</i>	<i>No_Trans</i>	<i>No_Patterns</i>	<i>Pattern_Sets (Itemset, Support)</i>
2	CA	San Francisco	2003/11	15000	3	(A,5%),(B,7%),(C,5%)
3	CA	San Francisco	2003/12	12000	2	(A,5%),(C,9%)
5	CA	Los Angeles	2003/11	25000	2	(A,5%),(C,6%)
6	CA	Los Angeles	2003/12	30000	4	(A,6%),(B,6%),(C,9%), (AB,6%)

The following relation can be derived for a candidate itemset x and its proper subsets.

Lemma 6-3: If x is a candidate itemset, then $\forall x' \subset x$, x' is also a candidate itemset.

Proof: If $x' \subset x$, then $t_{i.S_{x'}} \geq t_{i.S_x}$ for each tuple t_i in an MPR. According to Lemma 6-2, if x is a candidate itemset, there must exist at least a matched tuple t with $t.S_x \geq s_q$. Thus, $t.S_{x'} \geq t.S_x \geq s_q$ for the tuple t . x' is thus a candidate itemset. ■

The *appearing count* $Count_x^{appearing}$ of a candidate itemset x is defined as the count of x calculated from the matched tuples in which x appears. Thus:

$$Count_x^{appearing} = \sum_{t_i \in mt \ \& \ x \in t_i.ps} t_i.trans * t_{i.S_x}. \quad (6-2)$$

The *upper-bound count* $Count_x^{UB}$ of a candidate itemset x is defined as the upper bound count of x calculated from the matched tuples in which x does not appear. Thus:

$$Count_x^{UB} = \sum_{t_i \in mt \ \& \ x \notin t_i.ps} (t_i.trans * s - 1). \quad (6-3)$$

Let *Match_Trans* denote the number of transactions in the matched tuples. Thus:

$$Match_Trans = \sum_{t_i \in mt} t_i.trans. \quad (6-4)$$

The upper-bound support s_x^{UB} of a candidate itemset x is thus calculated as:

$$s_x^{UB} = \frac{Count_x^{appearing} + Count_x^{UB}}{Match_Trans}. \quad (6-5)$$

Lemma 6-4: If x is a candidate itemset and s_x is its actual support, then $s_x \leq$

$$s_x^{UB}.$$

Proof:

$$\begin{aligned} s_x &= \frac{\sum_{t_i \in mt} t_i.trans * t_i.s_x}{\sum_{t_i \in mt} t_i.trans} \\ &= \frac{\sum_{t_i \in mt \& x \in t_i.ps} t_i.trans * t_i.s_x + \sum_{t_i \in mt \& x \notin t_i.ps} t_i.trans * t_i.s_x}{\sum_{t_i \in mt} t_i.trans} \\ &\leq \frac{\sum_{t_i \in mt \& x \in t_i.ps} t_i.trans * t_i.s_x + \sum_{t_i \in mt \& x \notin t_i.ps} (t_i.trans * s - 1)}{\sum_{t_i \in mt} t_i.trans} \\ &= \frac{Count_x^{appearing} + Count_x^{UB}}{Match_Trans} \\ &= s_x^{UB}. \end{aligned}$$

Thus $s_x \leq s_x^{UB}$. ■

Example 6-3: Continuing Example 2, the upper-bound supports of the four candidate itemsets $\{A\}$, $\{B\}$, $\{C\}$, and $\{AB\}$, are calculated as follows:

$$\begin{aligned} s_A^{UB} &= \frac{Count_A^{appearing} + Count_A^{UB}}{Match_Trans} \\ &= \frac{15000 * 5\% + 12000 * 5\% + 25000 * 5\% + 30000 * 6\%}{15000 + 12000 + 25000 + 30000} = 0.0537, \end{aligned}$$

$$s_B^{UB} = \frac{Count_B^{appearing} + Count_B^{UB}}{Match_Trans}$$

$$= \frac{15000 * 7\% + 30000 * 6\% + 12000 * 5\% - 1 + 25000 * 5\% - 1}{15000 + 12000 + 25000 + 30000} = 0.0573,$$

$$s_C^{UB} = \frac{Count_C^{appearing} + Count_{\bar{C}}^{UB}}{Match_Trans}$$

$$= \frac{15000 * 5\% + 12000 * 9\% + 25000 * 6\% + 30000 * 9\%}{15000 + 12000 + 25000 + 30000} = 0.0735, \text{ and}$$

$$s_{AB}^{UB} = \frac{Count_{AB}^{appearing} + Count_{AB}^{UB}}{Match_Trans}$$

$$= \frac{30000 * 6\% + 15000 * 5\% - 1 + 12000 * 5\% - 1 + 25000 * 5\% - 1}{15000 + 12000 + 25000 + 30000} = 0.0536.$$

■

Lemma 6-5: If x is a candidate itemset, then $\forall x' \subset x, s_{x'}^{UB} \geq s_x^{UB}$.

Proof: If $x' \subset x$, then $t_{i.S_{x'}} \geq t_{i.S_x}$ for each tuple t_i in an MPR. Therefore:

$$s_{x'}^{UB} = \frac{Count_{x'}^{appearing} + Count_{\bar{x'}}^{UB}}{Match_Trans}$$

$$= \frac{\sum_{t_i \in mt \ \& \ x' \in t_i.ps} t_i.trans * t_{i.S_{x'}} + \sum_{t_i \in mt \ \& \ x' \notin t_i.ps} (t_i.trans * s - 1)}{\sum_{t_i \in mt} t_i.trans}$$

$$= \frac{\sum_{t_i \in mt \ \& \ x' \in t_i.ps \ \& \ x \in t_i.ps} t_i.trans * t_{i.S_{x'}} + \sum_{t_i \in mt \ \& \ x' \in t_i.ps \ \& \ x \notin t_i.ps} t_i.trans * t_{i.S_{x'}} + \sum_{t_i \in mt \ \& \ x' \notin t_i.ps} (t_i.trans * s - 1)}{\sum_{t_i \in mt} t_i.trans}$$

$$\geq \frac{\sum_{t_i \in mt \ \& \ x' \in t_i.ps \ \& \ x \in t_i.ps} t_i.trans * t_{i.S_x} + \sum_{t_i \in mt \ \& \ x' \in t_i.ps \ \& \ x \notin t_i.ps} (t_i.trans * s - 1) + \sum_{t_i \in mt \ \& \ x' \notin t_i.ps} (t_i.trans * s - 1)}{\sum_{t_i \in mt} t_i.trans}$$

$$= \frac{\sum_{t_i \in mt \ \& \ x \in t_i.ps} t_i.trans * t_{i.S_x} + \sum_{t_i \in mt \ \& \ x \notin t_i.ps} (t_i.trans * s - 1)}{\sum_{t_i \in mt} t_i.trans}$$

$$= \frac{Count_x^{appearing} + Count_{\bar{x}}^{UB}}{Match_Trans}$$

$$= s_x^{UB}.$$

Thus, $s_{x'}^{UB} \geq s_x^{UB}$. ■

Lemma 6-6: If a candidate itemset x is contained in all matched tuples, then $s_x^{UB} = s_x$.

Proof: If x is contained in all the matched tuples, then:

$$s_x^{UB} = \frac{\text{Count}_x^{\text{appearing}} + \text{Count}_x^{UB}}{\text{Match} _ \text{Trans}} = \frac{\sum_{t_i \in mt} t_i.\text{trans} * t_i.s_x}{\sum_{t_i \in mt} t_i.\text{trans}} = s_x. \quad \blacksquare$$

Example 6-4: Continuing Examples 2 and 3, according to Lemmas 6-4 and 6-5, candidate itemsets $\{A\}$ and $\{AB\}$ will be pruned since $\{AB\}$ is a proper superset of $\{A\}$ and the upper-bound support of $\{A\}$ is less than $s_q (= 5.5\%)$. According to Lemma 6-6, the candidate itemset $\{C\}$ will be put into the set of final frequent itemsets since it appears in all matched tuples and its support is greater than 5.5%. Only the remaining candidate itemset $\{B\}$ needs further processing. ■

The TOARM approach for carrying out mining tasks with an MPR consists of three main phases, *candidate itemset generation*, *candidate itemset reduction*, and *association rule generation*. The *candidate itemset generation* phase selects tuples that satisfy the context constraints in mining requests and generates candidate itemsets from matched tuples. The *candidate itemset reduction* phase then calculates the upper-bound supports for the candidate itemsets and uses two pruning strategies to reduce the number of candidates. Finally, the *association rule generation* phase finds final frequent itemsets and derives association rules from them. The proposed three-phase online mining approach is described in Figure 6-1.

The Three-phase Online Association Rule Mining (TOARM) approach:

INPUT: An MPR based on an initial minimum support s and a mining request q with a context set cx_q , a minimum support s_q ($s_q \geq s$) and a minimum confidence $conf_q$.

OUTPUT: A set of association rules satisfying the mining request q .

Phase 1: Candidate itemset generation:

- (a) Select tuples satisfying cx_q from the MPR.
- (b) Gather the candidate itemsets appearing in the matched tuples.
- (c) Calculate $Count_x^{appearing}$ and $Count_x^{UB}$ for each candidate itemset x .

Phase 2: Candidate itemset reduction:

- (a) Calculate the upper-bound support s_x^{UB} for each candidate itemset x using:

$$s_x^{UB} = \frac{Count_x^{appearing} + Count_x^{UB}}{Match_Trans}.$$

- (b) Discard candidate itemset x and its proper supersets if $s_x^{UB} < s_q$.
- (c) Put x into the set of frequent itemsets if $s_x^{UB} = \frac{Count_x^{appearing}}{Match_Trans}$ and $s_x^{UB} \geq s_q$.

Phase 3: Association rule generation:

- (a) Check whether each remaining candidate itemset x is large by scanning the underlying blocks of data for the matched tuples in which x does not appear.
- (b) Generate association rules satisfying the minimum confidence $conf_q$ from the set of frequent itemsets.

Figure 6-1: The TOARM algorithm

The TOARM approach considers only itemsets appearing in matched tuples and satisfying minimum support as candidates. It also uses two pruning strategies to reduce the number of candidate itemsets. It therefore only needs to re-process the remaining candidate itemsets against the underlying blocks of data for matched tuples in which they do not appear. For this reason, the cost of re-processing underlying blocks of data by the TOARM approach is less than that of typical batch mining or incremental mining approaches (experimental results presented below show this).

Theorem 6-1: The TOARM approach can correctly obtain association rules in response to an on-line mining request q as long as its minimum support s_q is greater than or equal to the initial minimum support s for getting the MPR.

Proof: According to Lemma 6-2, all candidate itemsets for q are collected in Phase 1 of the TOARM approach. After that, the candidate itemsets whose

upper-bound supports are less than s_q are pruned in Phase 2 (b) of the TOARM approach according to Lemmas 6-4 and 6-5. Also, the candidate itemsets which appear in all the matched tuples can know their actual supports according to Lemma 6-6. If they satisfy s_q , they are put into the set of final frequent itemsets in Phase 2 (c) of the TOARM approach. Finally, the actual supports of the remaining candidate itemsets can be found by Phase 3 (a) of the TOARM approach from the underlying blocks of data. The final frequent itemsets can then be determined. The association rules can thus be derived by Phase 3 (b) of the TOARM approach. ■

6.5 Negative-Border Online Mining (NOM) based on Extended MPR (EMPR)

Although the proposed TOARM approach based on a well-defined MPR can flexibly obtain association rules or patterns from portions of data, diversely consider problems at different aspects and provide on-line decision supports for users, it may get loose upper-bound supports of candidate itemsets for heterogeneous blocks of data and thus cause excessive I/O and computation costs to re-process them against the underlying database. As a result, we attempt to apply the concept of *negative border* [60] to calculate tighter upper-bound supports of candidate itemsets and then reduce the number of candidate itemsets to be considered [91][93]. The MPR is first extended for keeping the additional negative-border information. Based on the *extended* MPR (EMPR), we then develop an online mining approach called *Negative-Border Online Mining* (NOM) to efficiently and effectively utilize the information of *negative itemset* in the negative border.

Definition 6-1: An *Extended Multidimensional Pattern Relation Schema* (EMPRS) with n_1 context attributes and n_2 content attributes can be represented as

$EMPRS(ID, CX_1, CX_2, \dots, CX_{n_1}, CN_1, CN_2, \dots, CN_{n_2})$, where ID is an identification attribute, CX_i , $1 \leq i \leq n_1$, is a context attribute, and CN_i , $1 \leq i \leq n_2$, is a content attribute. ■

Definition 6-2: An *Extended Multidimensional Pattern Relation* (EMPR) including tuples $\{t_1, t_2, \dots, t_m\}$ is an instance of the given $EMPRS(ID, CX_1, CX_2, \dots, CX_{n_1}, CN_1, CN_2, \dots, CN_{n_2})$. A tuple $t_i = (id_i, cx_{i1}, cx_{i2}, \dots, cx_{in_1}, cn_{i1}, cn_{i2}, \dots, cn_{in_2})$ in an EMPR indicates that for the block of data under the contexts of $cx_{i1}, cx_{i2}, \dots, cx_{in_1}$, the mining information contains $cn_{i1}, cn_{i2}, \dots, cn_{in_2}$. ■

The *frequent pattern set* and the *negative pattern set* are two essential content attributes which are defined as follows.

Definition 6-3: A *frequent pattern set* (fps) for a block of data D is the set of all previously mined frequent itemsets with their supports for D . Assume the minimum support is s and the number of frequent itemsets discovered from D is l . A frequent pattern set can be represented as $fps = \{(x_i, s_i) \mid s_i \geq s \text{ and } 1 \leq i \leq l\}$, where x_i is a frequent itemset and s_i is its support. ■

Definition 6-4: A *negative pattern set* (nps) for a block of data D is the set of all previously mined negative itemsets with their supports from NB (fps) for D . ■

Below, an example is given to illustrate the above concepts.

Example 6-5: Table 6-3 shows an EMPR with the initial minimum support set to 5%. ID is an identification attribute, *Region*, *Branch* and *Time* are context attributes, and *No_Trans*, *No_Patterns*, *Frequent_Pattern_Set* and *Negative_Pattern_Set* are content attributes. The two attributes of *Frequent_Pattern_Set* and *Negative_Pattern_Set* respectively record the sets of mined frequent itemsets and negative itemsets from the corresponding data blocks. For example, the tuple with ID

= 1 shows that seven frequent itemsets $\{(A, 10\%), (B, 11\%), (C, 9\%), (AB, 8\%), (AC, 7\%), (BC, 6\%), (ABC, 6\%)\}$ and one negative itemset $(D, 2\%)$ are discovered from 10000 transactions under the contexts of $Region = CA$, $Branch = San Francisco$ and $Time = 2003/10$. The other tuples have similar meanings. ■

Table 6-3: An EMPR with minimum support = 5%

<i>ID</i>	<i>Region</i>	<i>Branch</i>	<i>Time</i>	<i>No_Trans</i>	<i>No_Patterns</i>	<i>Frequent_Pattern_Set (Itemset, Support)</i>	<i>Negative_Pattern_Set (Itemset, Support)</i>
1	CA	San Francisco	2003/10	10000	8	(A,10%),(B,11%),(C,9%),(AB,8%),(AC,7%),(BC,6%),(ABC,6%)	(D,2%)
2	CA	San Francisco	2003/11	15000	7	(A,5%),(B,7%),(C,5%)	(D,1%),(AB,2%),(AC,2%),(BC,1%)
3	CA	San Francisco	2003/12	12000	5	(A,5%),(C,9%)	(B,4%),(D,1%),(AC,4%)
4	CA	Los Angeles	2003/10	20000	8	(A,8%),(B,6%),(F,5%)	(C,2%),(D,3%),(AB,3%),(AF,4%),(BF,3%)
5	CA	Los Angeles	2003/11	25000	5	(A,5%),(C,6%)	(B,3%),(D,4%),(AC,3%)
6	CA	Los Angeles	2003/12	30000	7	(A,6%),(B,6%),(C,9%),(AB,6%)	(D,3%),(AC,4%),(BC,3%)
7	NY	New York	2003/10	18000	5	(B,8%),(C,7%),(BC,6%)	(A,2%),(D,2%)
8	NY	New York	2003/11	18500	5	(B,8%),(C,6%)	(A,4%),(D,2%),(BC,3%)
9	NY	New York	2003/12	19000	10	(A,5%),(B,9%),(C,8%),(D,6%),(BC,6%)	(AB,4%),(AC,4%),(AD,2%),(BD,4%)(CD,4%)

Example 6-6: For the EMPR in Table 6-3, assume a mining request q wants to get the patterns with the contexts cx_q of $Region = CA$ and $Time = 2003/10$ and satisfying the minimum support $s_q = 5.5\%$. The matched tuples are shown in Table 6-4. According to Lemma 6-2, the set of candidate itemsets is $\{\{A\}, \{B\}, \{C\}, \{AB\}, \{AC\}, \{BC\}, \{ABC\}\}$, which is the union of the itemsets appearing in the frequent pattern sets and with their supports larger than 5.5%. ■

Table 6-4: The matched tuples in Example 6-6

<i>ID</i>	<i>Region</i>	<i>Branch</i>	<i>Time</i>	<i>No_Trans</i>	<i>No_Patterns</i>	<i>Frequent_Pattern_Set (Itemset, Support)</i>	<i>Negative_Pattern_Set (Itemset, Support)</i>
1	CA	San Francisco	2003/10	10000	8	(A,10%),(B,11%),(C,9%),(AB,8%),(AC,7%),(BC,6%),(ABC,6%)	(D,2%)
4	CA	Los Angeles	2003/10	20000	8	(A,8%),(B,6%),(F,5%)	(C,2%),(D,3%),(AB,3%),

							(AF,4%),(BF,3%)
--	--	--	--	--	--	--	-----------------

Based on the EMPR, the appearing and upper-bound counts of a candidate itemset is re-defined as follows.

Definition 6-5: The *appearing count* $Count_x^{appearing}$ of a candidate itemset x is the sum of the counts of x appearing in the frequent pattern sets or negative pattern sets of matched tuples. Thus:

$$Count_x^{appearing} = \sum_{t_i \in mt \ \& \ x \in t_i \cdot fps \cup t_i \cdot nps} t_i \cdot trans * t_i \cdot s_x. \quad (6-6)$$

■

Definition 6-6: The *upper-bound count* $Count_x^{UB}$ of a candidate itemset x is the sum of the upper-bound counts of x not appearing in the frequent pattern sets and negative pattern sets of matched tuples. Thus:

$$Count_x^{UB} = \sum_{t_i \in mt \ \& \ x \notin t_i \cdot fps \cup t_i \cdot nps} \min(t_i \cdot trans * s - 1, t_i \cdot trans * \min_{\forall x' \subset x} (t_i \cdot s_{x'})). \quad (6-7)$$

■

Example 6-7: Continuing from Example 6-6, the upper-bound supports of the seven candidate itemsets $\{A\}$, $\{B\}$, $\{C\}$, $\{AB\}$, $\{AC\}$, $\{BC\}$ and $\{ABC\}$ are calculated as follows:

$$S_A^{UB} = \frac{Count_A^{appearing} + Count_A^{UB}}{Match_Trans} = \frac{(10000 * 10\% + 20000 * 8\%) + 0}{10000 + 20000} = 0.0867,$$

$$S_B^{UB} = \frac{Count_B^{appearing} + Count_B^{UB}}{Match_Trans} = \frac{(10000 * 11\% + 20000 * 6\%) + 0}{10000 + 20000} = 0.0767,$$

$$S_C^{UB} = \frac{Count_C^{appearing} + Count_C^{UB}}{Match_Trans} = \frac{(10000 * 9\% + 20000 * 2\%) + 0}{10000 + 20000} = 0.0433,$$

$$S_{AB}^{UB} = \frac{Count_{AB}^{appearing} + Count_{AB}^{UB}}{Match_Trans} = \frac{(10000 * 8\% + 20000 * 3\%) + 0}{10000 + 20000} = 0.0467,$$

$$s_{AC}^{UB} = \frac{Count_{AC}^{appearing} + Count_{AC}^{UB}}{Match_Trans} = \frac{(10000 * 7\%) + (20000 * 2\%)}{10000 + 20000} = 0.0367 ,$$

$$s_{BC}^{UB} = \frac{Count_{BC}^{appearing} + Count_{BC}^{UB}}{Match_Trans} = \frac{(10000 * 6\%) + (20000 * 2\%)}{10000 + 20000} = 0.0333 , \text{ and}$$

$$s_{ABC}^{UB} = \frac{Count_{ABC}^{appearing} + Count_{ABC}^{UB}}{Match_Trans} = \frac{(10000 * 6\%) + (20000 * 2\%)}{10000 + 20000} = 0.0333 .$$

■

Let s_x^{UB} denote the upper-bound support of a candidate itemset x in the EMPR and $s_x^{UB^{old}}$ denote the upper-bound support of a candidate itemset x in the MPR. The following lemma can easily be derived to show s_x^{UB} is tighter than $s_x^{UB^{old}}$.

Lemma 6-7: If x is a candidate itemset, then $s_x^{UB} \leq s_x^{UB^{old}}$.

Proof:

$$\begin{aligned} Count_x^{appearing} &= \sum_{t_i \in mt \ \& \ x \in t_i \cdot fps \cup t_i \cdot nps} t_i \cdot trans * t_i \cdot s_x \\ &= \sum_{t_i \in mt \ \& \ x \in t_i \cdot fps} t_i \cdot trans * t_i \cdot s_x + \sum_{t_i \in mt \ \& \ x \in t_i \cdot nps} t_i \cdot trans * t_i \cdot s_x \\ &\leq \sum_{t_i \in mt \ \& \ x \in t_i \cdot fps} t_i \cdot trans * t_i \cdot s_x + \sum_{t_i \in mt \ \& \ x \in t_i \cdot nps} (t_i \cdot trans * s - 1) ; \end{aligned}$$

$$\begin{aligned} Count_{\bar{x}}^{UB} &= \sum_{t_i \in mt \ \& \ x \notin t_i \cdot fps \cup t_i \cdot nps} \min(t_i \cdot trans * s - 1, t_i \cdot trans * \min_{\forall x' \subset x} (t_i \cdot s_{x'})) \\ &\leq \sum_{t_i \in mt \ \& \ x \notin t_i \cdot fps \cup t_i \cdot nps} (t_i \cdot trans * s - 1) ; \end{aligned}$$

$$\begin{aligned} s_x^{UB} &= \frac{Count_x^{appearing} + Count_{\bar{x}}^{UB}}{Match_Trans} \\ &\leq \frac{\sum_{t_i \in mt \ \& \ x \in t_i \cdot fps} t_i \cdot trans * t_i \cdot s_x + \sum_{t_i \in mt \ \& \ x \in t_i \cdot nps} (t_i \cdot trans * s - 1) + \sum_{t_i \in mt \ \& \ x \notin t_i \cdot fps \cup t_i \cdot nps} (t_i \cdot trans * s - 1)}{Match_Trans} \end{aligned}$$

$$\begin{aligned}
&= \frac{\sum_{t_i \in mt \ \& \ x \in t_i.ps} t_i.trans * t_i.s_x + \sum_{t_i \in mt \ \& \ x \notin t_i.ps} (t_i.trans * s - 1)}{Match_Trans} \\
&= s_x^{UB^{old}}.
\end{aligned}$$

Thus, $s_x^{UB} \leq s_x^{UB^{old}}$. ■

The following lemmas are important to the design of the proposed mining algorithm.

Lemma 6-8: If x is a candidate itemset, then $s_x \leq s_x^{UB}$.

Proof: For each $x' \subset x$, $t_i.s_{x'} \geq t_i.s_x$ for each tuple t_i . There are two possible cases for x' .

Case 1: If $\exists x' \in t_i.nps$, then:

$$\begin{aligned}
s_x &= \frac{\sum_{t_i \in mt} t_i.trans * t_i.s_x}{\sum_{t_i \in mt} t_i.trans} \\
&= \frac{\sum_{t_i \in mt \ \& \ x \in t_i.fps \cup t_i.nps} t_i.trans * t_i.s_x + \sum_{t_i \in mt \ \& \ x \notin t_i.fps \cup t_i.nps} t_i.trans * t_i.s_x}{\sum_{t_i \in mt} t_i.trans} \\
&\leq \frac{Count_x^{appearing} + \sum_{t_i \in mt \ \& \ x \notin t_i.fps \cup t_i.nps} t_i.trans * \min_{\forall x' \subset x} (t_i.s_{x'})}{\sum_{t_i \in mt} t_i.trans} \\
&= \frac{Count_x^{appearing} + \sum_{t_i \in mt \ \& \ x \notin t_i.fps \cup t_i.nps} \min(t_i.trans * s - 1, t_i.trans * \min_{\forall x' \subset x} (t_i.s_{x'}))}{\sum_{t_i \in mt} t_i.trans} \\
&= \frac{Count_x^{appearing} + Count_{\bar{x}}^{UB}}{Match_Trans} \\
&= s_x^{UB}.
\end{aligned}$$

Case 2: If $\forall x' \notin t_i.nps$, then:

$$\begin{aligned}
s_x &= \frac{\sum_{t_i \in mt} t_i.trans * t_i.s_x}{\sum_{t_i \in mt} t_i.trans} \\
&= \frac{\sum_{t_i \in mt \& x \in t_i.fps \cup t_i.nps} t_i.trans * t_i.s_x + \sum_{t_i \in mt \& x \notin t_i.fps \cup t_i.nps} t_i.trans * t_i.s_x}{\sum_{t_i \in mt} t_i.trans} \\
&\leq \frac{Count_x^{appearing} + \sum_{t_i \in mt \& x \notin t_i.fps \cup t_i.nps} (t_i.trans * s - 1)}{\sum_{t_i \in mt} t_i.trans} \\
&= \frac{Count_x^{appearing} + \sum_{t_i \in mt \& x \notin t_i.fps \cup t_i.nps} \min(t_i.trans * s - 1, t_i.trans * \min_{\forall x' \subset x} (t_i.s_{x'}))}{\sum_{t_i \in mt} t_i.trans} \\
&= \frac{Count_x^{appearing} + Count_x^{UB}}{Match_Trans} \\
&= s_x^{UB}.
\end{aligned}$$

Thus, $s_x \leq s_x^{UB}$. ■

Lemma 6-9: If x is a candidate itemset, then $\forall x' \subset x, s_{x'}^{UB} \geq s_x^{UB}$.

Proof: For each $x' \subset x, t_i.s_{x'} \geq t_i.s_x$ for each tuple t_i . Therefore:

$$\begin{aligned}
&Count_{x'}^{appearing} \\
&= \sum_{t_i \in mt \& x' \in t_i.fps \cup t_i.nps} t_i.trans * t_i.s_{x'} \\
&= \sum_{t_i \in mt \& x' \in t_i.fps \cup t_i.nps \& x \in t_i.fps \cup t_i.nps} t_i.trans * t_i.s_{x'} + \sum_{t_i \in mt \& x' \in t_i.fps \cup t_i.nps \& x \notin t_i.fps \cup t_i.nps} t_i.trans * t_i.s_{x'} \\
&\geq Count_x^{appearing} + \sum_{t_i \in mt \& x' \in t_i.fps \cup t_i.nps \& x \notin t_i.fps \cup t_i.nps} \min(t_i.trans * s - 1, t_i.trans * \min_{\forall x'' \subset x'} (t_i.s_{x''})); \\
&Count_{x'}^{UB} \\
&= \sum_{t_i \in mt \& x' \notin t_i.fps \cup t_i.nps} \min(t_i.trans * s - 1, t_i.trans * \min_{\forall x'' \subset x'} t_i.s_{x''})
\end{aligned}$$

$$\begin{aligned}
&\geq \sum_{t_i \in mt \ \& \ x \notin t_i \cdot fps \cup t_i \cdot nps} \min(t_i \cdot trans * s - 1, t_i \cdot trans * \min_{\forall x' \subset x} t_i \cdot s_{x'}); \\
s_{x'}^{UB} &= \frac{Count_{x'}^{appearing} + Count_{x'}^{UB}}{Match_Trans} \\
&\geq \frac{Count_x^{appearing} + Count_x^{UB}}{Match_Trans} \\
&= s_x^{UB}.
\end{aligned}$$

Thus, $s_{x'}^{UB} \geq s_x^{UB}$. ■

Lemma 6-10: If a candidate itemset x is contained in all the matched tuples, then

$$s_x^{UB} = s_x.$$

Proof: If x is contained in all the matched tuples, then:

$$s_x^{UB} = \frac{Count_x^{appearing} + Count_x^{UB}}{Match_Trans} = \frac{\sum_{t_i \in mt} t_i \cdot trans * t_i \cdot s_x}{\sum_{t_i \in mt} t_i \cdot trans} = s_x. \quad \blacksquare$$

Example 6-8: Continuing from Examples 6-6 and 6-7, according to Lemmas 6-8 and 6-9, the candidate itemsets $\{C\}$, $\{AB\}$, $\{AC\}$, $\{BC\}$ and $\{ABC\}$ will be pruned since their upper-bound supports are less than s_q ($= 5.5\%$). According to Lemma 6-10, the candidate itemsets $\{A\}$ and $\{B\}$ will be put into the set of final frequent itemsets since it appears in all the matched tuples and its support is larger than 5.5%. No remaining candidate itemsets needs to be further processed. ■

The NOM approach with an EMPR consists of three main phases, *candidate itemset generation*, *candidate itemset reduction*, and *association rule generation*, which are the same as the TOARM approach. The NOM approach can correctly obtain the association rules satisfying an on-line mining request as long as the new minimum support is larger than or equal to the initial minimum support for getting the EMPR.

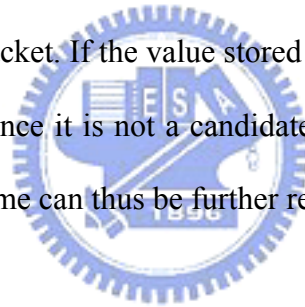
6.6 LNOM: Algorithm Design and Implementation

The NOM approach needs to calculate the appearing counts and the non-appearing upper-bound counts of the candidate itemsets derived from matched tuples. A straightforward way for finding these values is to process matched tuples one after one for each candidate itemset. Assume k is the number of matched tuples, m is the average number of itemsets in the k matched tuples, and n is the number of candidate itemsets generated from the k matched tuples. The computation cost will be $O(knm)$ when the candidate itemsets are processed one by one. The computation cost will, however, become large along with the increase of the itemsets kept in EMPR and the candidate itemsets to be considered. In fact, in the NOM approach, many candidate itemsets with the same subsets can be processed at the same time. For example, in Tuple 4 of Example 6-6, the appearing count of the candidate itemset $\{C\}$ and the upper-bound counts of the candidate itemsets $\{AC\}$, $\{BC\}$ and $\{ABC\}$ can be calculated at the same time because they have the same subset $\{C\}$. On the other hand, many itemsets kept in the matched tuples are useless for calculating the counts of candidates since they are not the subsets of candidates and can be omitted. For example, in Example 6-6, the itemsets $\{D\}$, $\{F\}$, $\{AF\}$ and $\{BF\}$ kept in the matched tuples are not the subsets of the candidate itemsets and can be omitted. We thus try to use appropriate data structures and design efficient algorithms to improve the performance of the NOM approach.

At first, the problem of calculating the appearing and upper-bound counts of candidate itemsets in a matched tuple is conceptually modeled by a graph and converted into a *directed-minimum-spanning-tree* problem. The *spanning-tree-count-calculating* (STCC) algorithm is then proposed to find the *directed minimum spanning*

tree. The *lattice* data structure [2][41] is utilized to organize and maintain all candidate itemsets such that the candidate itemsets with the same proper subsets can be considered at the same time. Consequently, by the STCC algorithm, the proposed *lattice-based* NOM (LNOM) approach requires only one scan of the itemsets for each matched tuple in Phase 1.

In addition, the hashing technique is used to filter out a part of itemsets kept in the matched tuples which are useless for calculating the counts of candidate. The NOM approach first hashes the set of candidate itemsets into a given hash table as soon as they are collected. Each bucket of the hash table consists of an integer to represent how many candidate itemsets have been hashed into this bucket. When an itemset of a matched tuple is selected, the NOM approach calculates its hash value and finds its corresponding bucket. If the value stored in the target bucket is equal to 0, the itemset must be useless since it is not a candidate itemset. It can thus be directly omitted. The computational time can thus be further reduced.



6.6.1 The Proposed Lattice-based NOM (LNOM) Approach

The problem of calculating the appearing and upper-bound counts of candidate itemsets in a matched tuple t can be conceptually modeled by a graph. Let $G = (V, E)$ be a directed graph, where V is the set of vertices representing all candidate itemsets and E is the set of directed edges representing *a-proper-subset-of* relationships between pairs of candidate itemsets. For each edge $(u, v) \in E$, a weight $w(u, v)$ specifies the possible upper-bound count of the candidate itemset v estimated from the candidate itemset u . Given a new vertex r representing the pseudo starting vertex, we make a new graph $G' = (V', E')$, where $V' = V \cup \{r\}$, $E' = E \cup \{(r, u) : u \in V\}$. For each edge (r, u) , if u appears in t , the appearing count of u is assigned as the weight

$w(r, u)$. For the case that u does not appear in t , meaning it is collected from the other matched tuple(s), then $w(r, u) = 0$ if there exists one item contained in u but not contained in t and $w(r, u) = t.trans*s-1$ otherwise, where s is the initial minimum support for deriving EMPR. The following lemmas formally show the above concepts.

Lemma 6-11: G' is an acyclic and connected graph.

Proof: It is obvious that the *a-proper-subset-of* relation on a set is transitive and anti-symmetric. G' is thus acyclic. Next, we prove G' is a connected graph by contradiction. If G' is not a connected graph, there exists a vertex u which is not reachable from the pseudo starting vertex r . This contradicts the definition of G' . Thus, G' is an acyclic and connected graph. ■

Lemma 6-12: Let k be the number of items contained in a candidate itemset x . The vertex u_x has 2^k-1 incoming edges in G' .

Proof: If x is a candidate k -itemset, it will appear in the frequent pattern set of at least a tuple. Since x is large in that tuple, all its proper subsets except ϕ are also large and appear in that tuple. There are 2^k-2 proper subsets for x except ϕ . In addition, the incoming edge (r, u_x) is used to link the two vertices r and u_x . The vertex u_x thus has 2^k-1 incoming edges in G' . ■

Lemma 6-13: For a matched tuple t in EMPR, if there exists one item contained in a candidate itemset u but not contained in t , then the upper-bound count of u is 0.

Proof: According to the concept of the negative border, all single items which are not large must be put into the negative 1-itemsets. Since all the large and negative itemsets for a block of data are stored in a corresponding tuple, if there exists one item contained in a candidate itemset but not contained in the tuple, this item does not appear in the corresponding block of data. The count of the item is thus 0 in this tuple,

causing the count of each itemset containing the item is also 0. This completes the proof. ■

Lemma 6-14: For a matched tuple t in EMPR, if a candidate itemset u does not appear in t , then the maximum possible upper-bound count of u is $t.trans*s-1$.

Proof: Since u does not appear in t , it is not a frequent itemset. The support of u in t must thus be less than the minimum support s . Therefore, the count of u in t must be less than $t.trans*s$. The maximum possible upper-bound count of u is thus $t.trans*s-1$. ■

Example 6-9: For the EMPR given in Table 6-3 and the mining request in Example 6-6, the graph model for Tuple 4 is generated as shown in Figure 6-2. ■

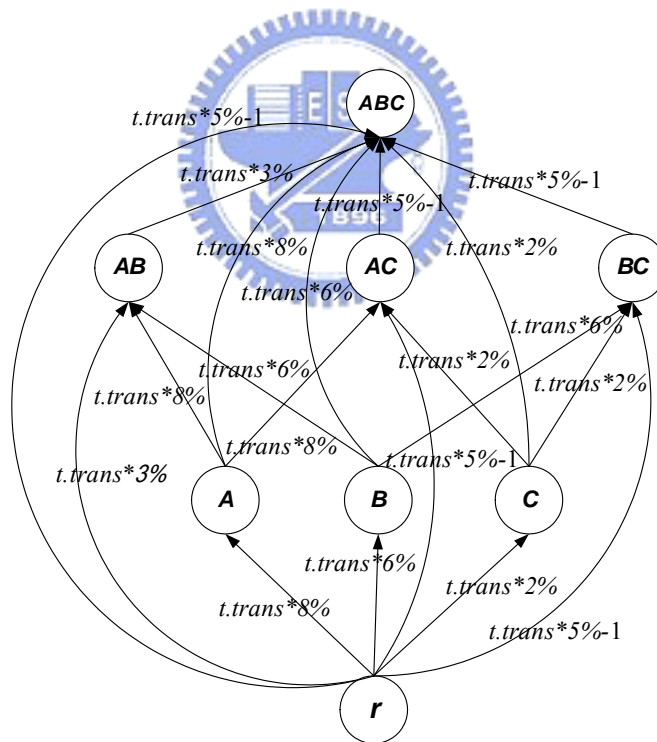
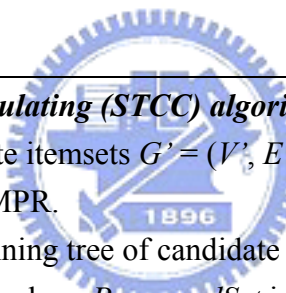


Figure 6-2: The graph model of candidate itemsets for Tuple 4 in Table 6-4

For each vertex other than r in G' , the smallest weight on all its incoming edges is its tight upper bound count. The count-calculation problem can thus be easily

thought of as the *directed-minimum-spanning-tree* problem [30], which wishes to find a rooted directed spanning tree $T = (V', S')$ from G' , such that S' is a subset of E' and $\sum_{(u,v) \in S} w(u,v)$ is a minimum. The *spanning-tree-count-calculating* (STCC) algorithm shown in Figure 6-3 is thus proposed based on the above concept for efficiently finding the counts of all candidate itemsets in a tuple. The STCC algorithm first selects an itemset appearing in t and with the smallest support. It then estimates the upper-bound count of each itemset reachable from the selected one in the graph, and thus avoids recalculating the counts of these traversed vertices in the future. This requires only one scan of the itemsets in t if they have been sorted according to their supports.



The spanning-tree-count-calculating (STCC) algorithm:

INPUT: The graph of candidate itemsets $G' = (V', E')$ derived from the EMPR, and a matched tuple t in EMPR.

OUTPUT: The minimum spanning tree of candidate itemsets $T = (V', S')$.

STEP 1: Set $ProcessedSet = \phi$, where $ProcessedSet$ is a set used to keep the vertices in G' which have been traversed.

STEP 2: Select an itemset x appearing in t and with the smallest support $t.s_x$.

STEP 3: If $x \in V'$ (i.e., x is a candidate itemset), set $Count_x^{appearing} = t.trans * t.s_x$, $ProcessedSet = ProcessedSet \cup \{x\}$, and do STEP 4; otherwise (i.e., x is not a candidate itemset), do nothing and go to STEP 5.

STEP 4: For each y reachable from x and $y \notin ProcessedSet$, set $Count_y^{UB} = \min(t.trans * s-1, t.trans * t.s_x)$ and $ProcessedSet = ProcessedSet \cup \{y\}$.

STEP 5: Repeat STEPs 2 to 4 until all the itemsets appearing in t are processed.

STEP 6: If $|ProcessedSet| \neq |V'|$ (i.e., some candidate itemsets do not appear in the underlying dataset of t), set $Count_x^{UB} = 0$ for each remaining itemset $x \in V'$.

Figure 6-3: The STCC algorithm

Example 10: Continuing Example 3, the negative itemset $\{C\}$ with 2% will be first selected by the proposed STCC algorithm to calculate the appearing count of itself and the upper-bound counts of $\{AC\}$, $\{BC\}$ and $\{ABC\}$. Then, the itemsets $\{D\}$ with 3%, $\{AB\}$ with 3%, $\{B\}$ with 6% and $\{A\}$ with 8% are selected in turn. Among them, the support information of $\{D\}$ is useless because it is not a candidate itemset. Figure 6-4 shows the directed minimum spanning tree found from Figure 6-2. ■

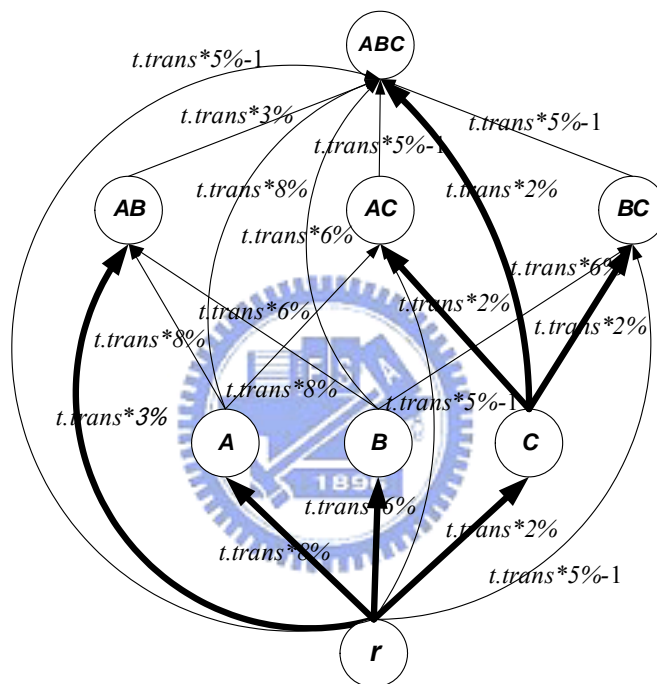


Figure 6-4: The directed minimum spanning tree found from Figure 6-2

The STCC algorithm mentioned above can be efficiently implemented by the lattice data structure [2][41], which organizes all candidate itemsets in a systematic way. The lattice is constructed as follows. For each candidate itemset x , a corresponding vertex u_x associated with a pair of values $(Count_x^{appearing}, Count_x^{UB})$ is built in the lattice. For any pair of vertices u_x and u_y corresponding to candidate itemsets x and y , there is a directed edge from u_x to u_y if x is a *parent* of y . An itemset

x is said to be a parent of an itemset y if y can be obtained by adding an item to x , and inversely, y is said to be a child of x . Therefore, a candidate itemset may have more than one parent and more than one child in the constructed lattice.

Example 6-11: Consider the candidate itemsets illustrated in Example 6-6. The lattice to represent the candidate itemsets is illustrated in Figure 6-5, where the vertex labeled “Null” denotes the greatest lower bound of the lattice. ■

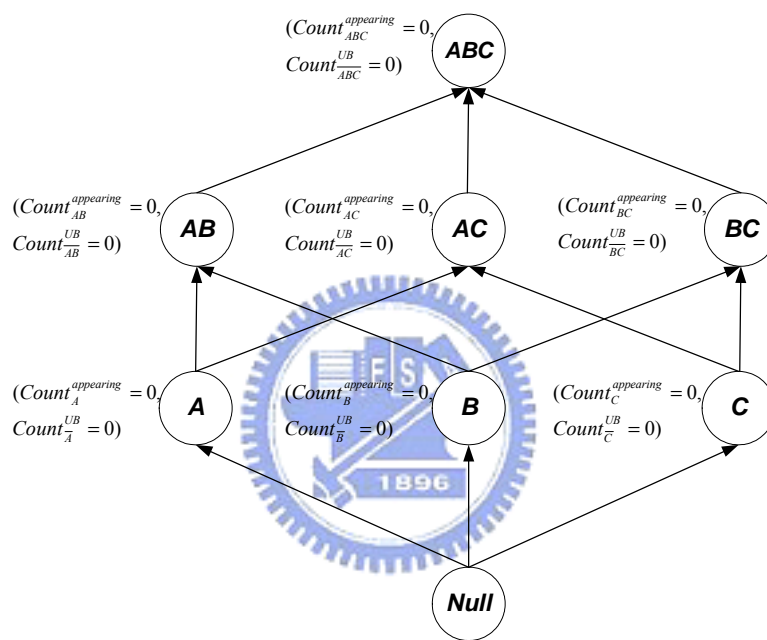


Figure 6-5: The lattice to represent the candidate itemsets illustrated in Example 6-6

The lattice structure is used to efficiently find the appearing and upper-bound counts of candidate itemsets in each tuple and to accumulate these values when the tuples are processed one by one. By the connected edges in the lattice structure, the proposed *lattice-based NOM* approach (called LNOM) can not only restrict the number of candidate itemsets to be examined, but also easily consider candidate itemsets with the same proper subsets at the same time. The detailed LNOM algorithm will be described in Section 6.6.3.

6.6.2 Using the Hashing Technique to Reduce Computation Cost Further

Many itemsets kept in matched tuples, especially negative itemsets, may be useless for calculating the counts of candidate itemsets. For example, the itemsets $\{D\}$, $\{F\}$, $\{AF\}$ and $\{BF\}$ kept in the matched tuples in Example 6-6 are not the subsets of the candidate itemsets and can be omitted. Negative itemsets are formed by excluding frequent itemsets from the candidates which are generated in a level-wise way [27][85]. In other words, a negative itemset is a candidate itemset without enough support. In general, the set of candidate itemsets generated level-wisely is usually much larger than the set of frequent itemsets found, especially in the early stage of candidate generation [5][67]. The number of negative itemsets useless for calculating the counts of candidate itemsets may thus be large. In this section, we shall utilize the hashing technique [67] to filter out a part of useless itemsets to be considered in Phase 1. Take the *direct hashing function* as an example to explain our idea. Let $x = \{a_1, a_2, \dots, a_n\}$ denote an itemset consisting of n items (from a_1 to a_n), $order(a_i)$ denote the serial number of the item a_i among the entire set of items, and $size(HT)$ denote the size of a given hash table HT . A direct hashing function for n -dimensional keys can be defined as follows:

$$h(x) = (order(a_1) * order(a_2) * \dots * order(a_n)) \bmod size(HT).$$

The hashing function is order-independent; that is, it can generate the same hash value for all permutations of items in an itemset. Each bucket of the hash table consists of only an integer to represent how many candidate itemsets have been hashed into this bucket. 0 denotes that no candidate itemsets have been hashed into this bucket. When initially obtaining the set of candidate itemsets, the NOM approach calculates their hash values, finds corresponding hash buckets, and for each candidate

add one to the value of its corresponding bucket.

Example 6-12: For the candidate itemsets $\{A\}$, $\{B\}$, $\{C\}$, $\{AB\}$, $\{AC\}$, $\{BC\}$ and $\{ABC\}$ obtained in Example 6-6, the LNom approach will hash them into a given hash table HT . Without loss of generality, assume $order(A) = 1$, $order(B) = 2$ and $order(C) = 3$. Also assume the size of the hash table is 7. The hash values of these candidate itemsets will first be calculated. Take the itemset $\{AB\}$ as an example. Its hash value is $(order(A) * order(B)) \bmod 7$, which is 2. The value in Bucket 2 is then increased by one. The other candidate itemsets are hashed in a similar way. The resulting hash table is shown in Figure 6-6. ■

HT

<i>Itemsets</i>		$\{A\}$	$\{B\}$	$\{AB\}$	$\{AC\}$	$\{C\}$	$\{BC\}$	$\{ABC\}$
<i>Bucket value</i>	0	1	2	2	0	0	2	
<i>Bucket number</i>	0	1	2	3	4	5	6	

Figure 6-6: The hash table derived from the candidate itemsets illustrated in Example

6-6

After a hash table is constructed from all the candidate itemsets, it can then be used to filter out a part of useless itemsets in a tuple. Tuples are processed one by one. When an itemset of a matched tuple is selected, the NOM approach calculates its hash value and finds its corresponding bucket. If the value stored in the target bucket is equal to 0, the itemset must be useless since it is not a candidate itemset. It can thus be directly omitted. Otherwise, the itemset may be, but not certainly, a candidate itemset. Rescanning the candidate itemsets is then necessary to determine whether it is a candidate.

Furthermore, the corresponding value in the bucket of the itemset which has been assured to be a candidate will be decreased by one. The next itemset of the same tuple is then checked according to the modified hash table, which can thus raise the probability for a useless itemset to be filtered out. After a tuple is processed, the hash table is restored to its original state, which is then used for another tuple. This is illustrated by the following example.

Example 6-13: Continuing Example 6-12, after the hash table in Figure 6-6 has been constructed, it can be used to filter out some useless itemsets in matched tuples. For example, when Tuple 4 in Example 6-6 is checked, the itemset $\{C\}$ with 2% support is first selected to process since it has the smallest support value among all the itemsets appearing in the tuple. The hash value of $\{C\}$ is calculated as 3 and the value in Bucket 3 is 2, not 0. The itemset $\{C\}$ is thus checked against the candidate itemsets and is found to be a candidate. It is then used to calculate the counts of the candidate $\{C\}$ and its superset in the lattice. In this example, the counts of the candidates $\{C\}$, $\{AC\}$, $\{BC\}$ and $\{ABC\}$ are then calculated. As a result, the value in Bucket 3 is decreased by 2 due to $\{C\}$ and $\{AC\}$. The value in Bucket 6 is decreased to 0 as well due to $\{BC\}$ and $\{ABC\}$. Bucket 6 in the modified hash table can filter out the itemsets $\{F\}$ and $\{AF\}$ in Tuple 4 since the value in Bucket 6 has been zero. After that, the hash table will be restored to the original one in Figure 6-6 for processing another matched tuple. ■

6.6.3 The L NOM Algorithm with a Direct Hashing Function

In Phase 1, by one scan of a given EMPR, the L NOM approach first collects the itemsets in the matched tuples satisfying the query support as candidates, constructs a corresponding lattice for considering candidate itemsets with the same proper subsets

at the same time, and hashes them into a given hash table for filtering out a part of useless itemsets in matched tuples. The L NOM approach then processes matched tuples one by one, selects the itemsets in the order of ascending support values for each matched tuple, and checks whether they are useful for calculating the counts of candidates according to the values of their hash buckets. If the corresponding target bucket value is 0, the itemset is omitted. Otherwise, for each itemset x , the L NOM approach will assure whether x is a candidate by checking the set of candidate itemsets. If x is a candidate, the L NOM approach will cumulate the $Count_x^{appearing}$ and each $Count_y^{UB}$ in the lattice, where y denotes an element in the proper superset of x (y is a descendant of x). This procedure is then repeated until all the matched tuples have been processed. After that, the L NOM approach can generate the candidate itemsets with appearing counts and upper-bound counts corresponding to the given mining request.

Example 6-14: Consider the mining request in Example 6-6. The L NOM approach will construct the lattice shown in Figure 6-5 and the hash table shown in Figure 6-6. It then processes the first matched tuple, and filter out $(D, 2\%)$ using the hash table. The remaining itemsets $(ABC, 6\%)$, $(BC, 6\%)$, $(AC, 7\%)$, $(AB, 8\%)$, $(C, 9\%)$, $(A, 10\%)$ and $(B, 11\%)$ are then processed in turn to update the counts of the corresponding itemsets in the lattice. After that, the L NOM approach processes the second matched tuple. Only the four itemsets $(C, 2\%)$, $(AB, 3\%)$, $(B, 6\%)$ and $(A, 8\%)$ needs to be processed after the hash-table checking. $(C, 2\%)$ is then first selected, and is used to update not only the appearing count of $\{C\}$ but also the upper-bound counts of the itemsets in its proper superset ($\{AC\}$, $\{BC\}$ and $\{ABC\}$). The updated lattice after processing all the matched tuples is shown in Figure 6-7. ■

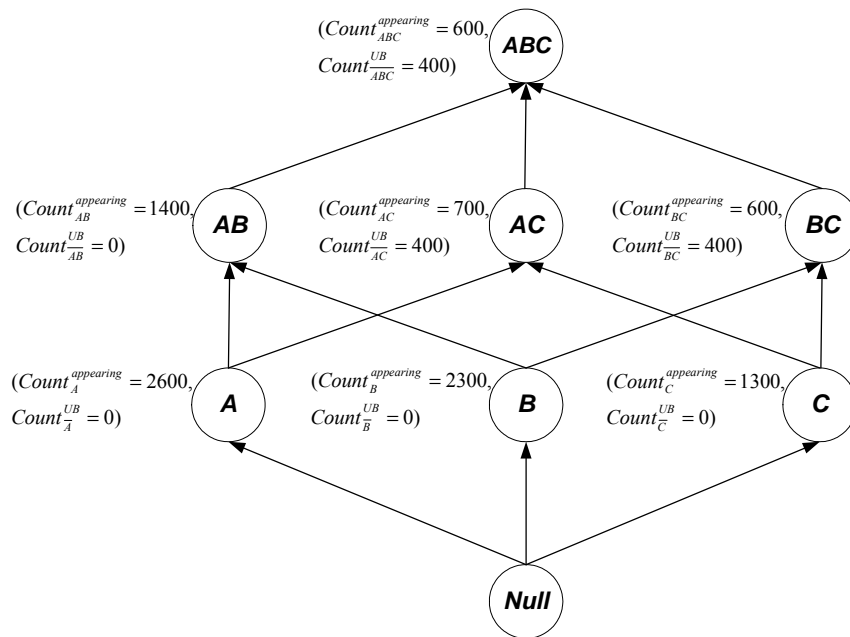


Figure 6-7: The updated lattice after processing all matched tuples

Next, Phase 2 proceeds to prune candidates in a level-wise way. Candidate 1-itemsets are then first handled. If the upper-bound support of a candidate 1-itemset is less than the query support, it and the itemsets in its proper superset are removed from the lattice. If a candidate 1-itemset appears in all the matched tuples and its upper-bound support is larger than or equal to the query support, then it is put into the set of final frequent itemsets and removed from the lattice. This procedure is repeated level-wisely until all the candidate itemsets have been processed. After Phase 2, the remaining candidate itemsets in the lattice have enough upper-bound supports but do not appear in at least one matched tuple. The L NOM approach thus re-processes them against the underlying blocks of data for the matched tuples in which they do not appear to get their actual supports. After all the frequent itemsets are found, the association rules can then be easily generated from them. The detailed algorithm of the L NOM approach with a direct hashing function is stated in Figure 6-8.

The LNOM approach with a direct hashing function:

INPUT: An EMPR based on an initial minimum support s , and a mining request q with a set of contexts cx_q , a minimum support s_q ($s_q \geq s$) and a minimum confidence $conf_q$.

OUTPUT: A set of association rules satisfying the mining request q .

Phase 1: Generation of candidate itemsets:

STEP 1: Set $C = \phi$ and $Match_Trans = 0$, where C is a lattice used to maintain the set of candidate itemsets and $Match_Trans$ is a variable used to keep the total number of transactions in the matched tuples which have been processed.

STEP 2: Initialize two equal-sized hash tables HT_1 and HT_2 with all the bucket values being zero.

STEP 3: For each tuple t in EMPR, do the following substeps:

STEP 3-1: If t satisfies cx_q , put it into the matched set and do STEP 3-2; otherwise, repeat STEP 3 to process the next tuple.

STEP 3-2: For each itemset $x \in t.fps$, if $x \notin C$ and $t.s_x \geq s_q$, set $HT_1[h(x)] =$

$HT_1[h(x)] + 1$, insert x into C with $Count_x^{appearing} = 0$ and $Count_x^{UB}$

$= 0$, and add edges to its parents and children, where $HT_1[h(x)]$ denotes the value stored in the bucket corresponding to the hash value $h(x)$ of x in HT_1 .

STEP 4: For each tuple t in the matched set, do the following substeps:

STEP 4-1: Set $ProcessedSet = \phi$, where $ProcessedSet$ is a set used to keep the itemsets in C which have been processed.

STEP 4-2: Restore the bucket values in HT_2 to those in HT_1 and set $Match_Trans = Match_Trans + t.trans$.

STEP 4-3: Select an itemset x with the smallest support $t.s_x$ from t .

STEP 4-4: If $HT_2[h(x)] \neq 0$ and $x \in C$, set $Count_x^{appearing} = Count_x^{appearing} +$

$t.trans * t.s_x$, $HT_2[h(x)] = HT_2[h(x)] - 1$, $ProcessedSet = ProcessedSet \cup \{x\}$, and do STEP 4-5; otherwise, do nothing and go to STEP 4-6.

STEP 4-5: For each itemset y in the proper superset of x in C and $y \notin$

$ProcessedSet$, set $Count_y^{UB} = Count_y^{UB} + \min(t.trans * s - 1,$

$t.trans * t.s_x)$, $HT_2[h(y)] = HT_2[h(y)] - 1$, and $ProcessedSet = ProcessedSet \cup \{y\}$.

STEP 4-6: Repeat STEPs 4-3 and 4-4 until all itemsets in t are processed.

Phase 2: Reduction of candidate itemsets:

STEP 5: Set $k = 1$, where k is used to keep the number of items in a candidate itemset currently being processed.

STEP 6: For each itemset $x \in C_k$, do the following substeps:

STEP 6-1: Calculate the upper-bound support s_x^{UB} by the formula:

$$s_x^{UB} = \frac{Count_x^{appearing} + Count_{\bar{x}}^{UB}}{Match_Trans}.$$

STEP 6-2: If $s_x^{UB} < s_q$, set $C = C - \{y \mid y \in C \text{ and } x \subseteq y\}$.

STEP 6-3: If $s_x^{UB} = \frac{Count_x^{appearing}}{Match_Trans}$ and $s_x^{UB} \geq s_q$, then set $L = L \cup \{x\}$ and $C = C - \{x\}$.

STEP 7: Set $k = k + 1$.

STEP 8: Repeat STEPs 6 and 7 until all candidate itemsets are processed.

Phase 3: Generation of association rules:

STEP 9: For each $x \in C$, re-process each underlying block of data D_i for tuple t_i in

which x does not appear to get $Count_x^{appearing}$, and then calculate the actual support of x by the following formula:

$$s_x = \frac{Count_x^{appearing} + Count_{\bar{x}}^{appearing}}{Match_Trans}.$$

STEP 10: If $s_x < s_q$, then set $C = C - \{x\}$; otherwise, set $L = L \cup \{x\}$ and $C = C - \{x\}$.

STEP 11: Derive the association rules satisfying $conf_q$ from the set of frequent itemsets L .

Figure 6-8: The algorithm of the LNOM approach with a direct hashing function

6.7 Experimental Results

The experiments were conducted in Java on a workstation with dual XEON 2.8GHz processors and 2048MB main memory, running the RedHat 9.0 operating system. For performance comparison, two batch-based mining algorithms, Apriori and Partition, and one incremental mining algorithm, FUP, in addition to our proposed TOARM, NOM and LNOM algorithms, were run on several synthetic and a real-world datasets.

6.7.1 Experimental Results for Synthetic Datasets

The synthetic datasets were generated by a generator similar to that used in [5]. The parameters used are listed in Table 6-5. The generator first generated L maximal potentially frequent itemsets, each with an average of I items. The items in the potentially frequent itemsets were randomly chosen from the total N items according to their actual sizes. The generator then generated D transactions, each with an average of T items. The items in a transaction were generated according to the L maximal potentially frequent itemsets in a probabilistic way. Details of the dataset generation process may be found in [5].

Table 6-5: Parameters considered when generating datasets

<i>Parameter</i>	<i>Description</i>
D	Number of transactions
N	Number of items
L	Number of maximal potentially frequent itemsets
T	Average size of items in a transaction
I	Average size of items in maximal potentially frequent itemsets

Table 6-6 listed the six groups of synthetic datasets generated and used in our experiments, where datasets in the same group had the same D , T and I values, but different L or N values. Each dataset was treated as a block of data in the database. For example, Group 1 in Table 6-6 contained ten blocks of data, from $T10I8D10KL^1$ to $T10I8D10KL^{10}$, each consisting of 10000 transactions averaging 10 items and generated according to 200 to 245 maximal potentially frequent itemsets with an average size of 8 from a total of 100 items. Let a group of heterogeneous datasets be defined as one in which the datasets have different items. Among the six groups, Groups 2, 4 and 6 may be considered heterogeneous because their varied N values

yield different items. These groups of synthetic datasets were used to show how the TOARM, NOM and L NOM algorithms dealt with heterogeneous blocks of data.

Table 6-6: The six groups of synthetic datasets

Group	Size	Datasets	D	T	I	L	N
1	10	$T10I8D10KL^1$ to $T10I8D10KL^{10}$	10000	10	8	200 to 245	100
2	10	$T10I8D10KN^1$ to $T10I8D10KN^{10}$	10000	10	8	200	100 to 145
3	10	$T20I8D100KL^1$ to $T20I8D100KL^{10}$	100000	20	8	400 to 490	200
4	10	$T20I8D100KN^1$ to $T20I8D100KN^{10}$	100000	20	8	400	200 to 290
5	5	$T10I8D500KL^1$ to $T10I8D500KL^5$	500000	10	8	400 to 560	200
6	5	$T10I8D500KN^1$ to $T10I8D500KN^5$	500000	10	8	400	200 to 360

The MPR and EMPR were first derived from each group of synthetic datasets. These are summarized in Table 6-7.

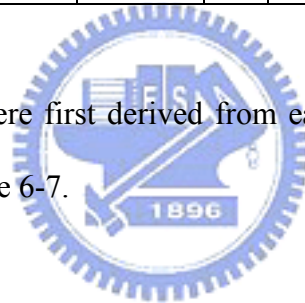


Table 6-7: Mining information for the six groups

Group	Initial minimum support	Average length of maximal frequent itemsets	Average size of frequent itemsets	Average size of negative itemsets
1	2%	11	9006	10762
2	2%	9	5093	11243
3	2%	9	12127	55625
4	2%	11	18534	49318
5	2%	5	799	11899
6	2%	8	869	14488

First, the TOARM, Apriori, Partition and FUP algorithms were run on Groups 1, 2, 3 and 5 along with various minimum supports in the mining requests, where the Partition algorithm partitioned the data sets according to group size (the number of datasets in a group) and the FUP algorithm treated each dataset in a group as a new

addition of transactions. Details of the TOARM algorithm compared with the other three algorithms are illustrated as follows.

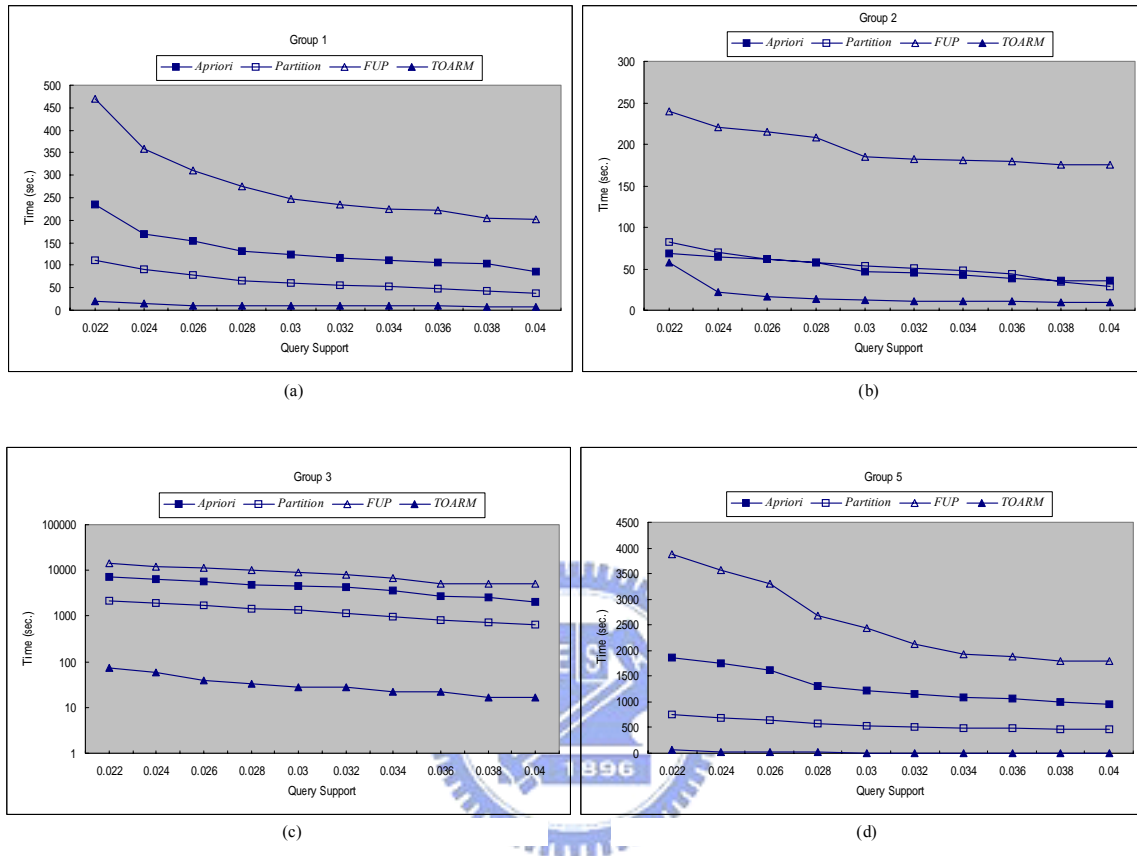


Figure 6-9: Execution times for the TOARM, Apriori, Partition and FUP algorithms on Groups 1, 2, 3 and 5

(a) Comparison with the Apriori algorithm. Figures 6-9(a), 6(c) and 6(d) show that execution times for the TOARM algorithm on Groups 1, 3 and 5 were always much less than those of the Apriori algorithm. This is because the datasets in these three groups were homogeneous, meaning they used the same set of items in each group. In this situation, the number of candidate itemsets considered by the TOARM algorithm was much closer to the number of final frequent itemsets than those considered by the Apriori algorithm. The former thus had more compact candidate

sets than the latter. For example, Table 6-8 shows the number of candidate itemsets considered by the TOARM and the Apriori algorithms for Group 5 with minimum supports ranging from 2.2% to 4% in the mining requests.

Table 6-8: The numbers of candidate itemsets for Group 5

<i>Approach \ Support</i>	0.022	0.024	0.026	0.028	0.03	0.032	0.034	0.036	0.038	0.04
<i>No. of Candidate itemsets</i>										
TOARM	959	690	550	442	372	308	269	241	220	199
Apriori	11636	10327	9165	8590	7722	7085	6603	6346	5898	5369
<i>No. of final Large itemsets</i>										
TOARM/Apriori	574	453	373	318	260	228	201	177	158	144

Table 6-9: The numbers of candidate itemsets for Group 2

<i>Approach \ Support</i>	0.022	0.024	0.026	0.028	0.03	0.032	0.034	0.036	0.038	0.04
<i>No. of Candidate itemsets</i>										
TOARM	20893	16003	11920	9016	7421	6541	5731	4984	3775	2816
Apriori	11615	10157	9158	8016	7372	6704	6070	5243	4593	4255
<i>No. of final Large itemsets</i>										
TOARM/Apriori	902	778	684	608	537	473	417	372	327	296

By contrast, the datasets in Group 2 were heterogeneous, meaning they used different sets of items. In this situation, the number of candidate itemsets considered by the TOARM algorithm was much larger than the number of final frequent itemsets considered by the Apriori algorithm since most of the candidate itemsets appeared in only one or a few tuples in the MPR. Table 6-9 shows the number of candidate itemsets considered by the TOARM and Apriori algorithms for Group 2 along with minimum supports ranging from 0.022 to 0.04 in the mining requests. Table 6-9 also shows that while the number of candidate itemsets for Group 2 considered by the TOARM algorithm was larger than that considered by the Apriori algorithm, the TOARM algorithm used two pruning strategies in Phase 2 and thus only had to re-process the remaining candidate itemsets against the underlying datasets in Phase 3.

The result was that the TOARM algorithm usually required less time than the Apriori algorithm. This is consistent with the results shown in Figure 6-9(b).

(b) Comparison with the Partition algorithm. Although the number of candidate itemsets considered by the Partition algorithm in the second pass was equal to that considered by the TOARM algorithm, the Partition algorithm must generate a set of all potentially frequent itemsets from each partition during its first pass. The TOARM algorithm can, however, use the pattern sets in the MPR to achieve this purpose. Therefore, the execution times required by the TOARM algorithm on these four groups were always less than those required by the Partition algorithm. This is also consistent with the results shown in Figure 6-9.

(c) Comparison with the FUP algorithm. The FUP algorithm can, in general, perform well when the size of newly inserted transactions is relatively smaller than the size of an original database because the cost of generating candidate itemsets from only new transactions is usually low and a large proportion of the candidate itemsets can be determined from previously mined frequent itemsets. However, the FUP algorithm treated the datasets in each of our application groups as increments and yielded even worse performance than the Apriori algorithm, especially on the heterogeneous datasets since it had to process all of them one by one. Figures 6-9(a), 6-9(c) and 6-9(d) show that the execution times for the FUP algorithm on the three homogeneous groups were about twice those of the Apriori algorithm. On the second group, which was heterogeneous, the FUP algorithm required about four times the execution time required by the Apriori algorithm.

Next, for showing the influence of the number of negative itemsets on execution time, the TOARM algorithm using no negative itemsets and the NOM using all negative itemsets were run on Groups 1 to 6. Figures 6-10(a) to 6-10(f) show the

execution times for the two algorithms on the six groups, where the query support is set to 2.4%.

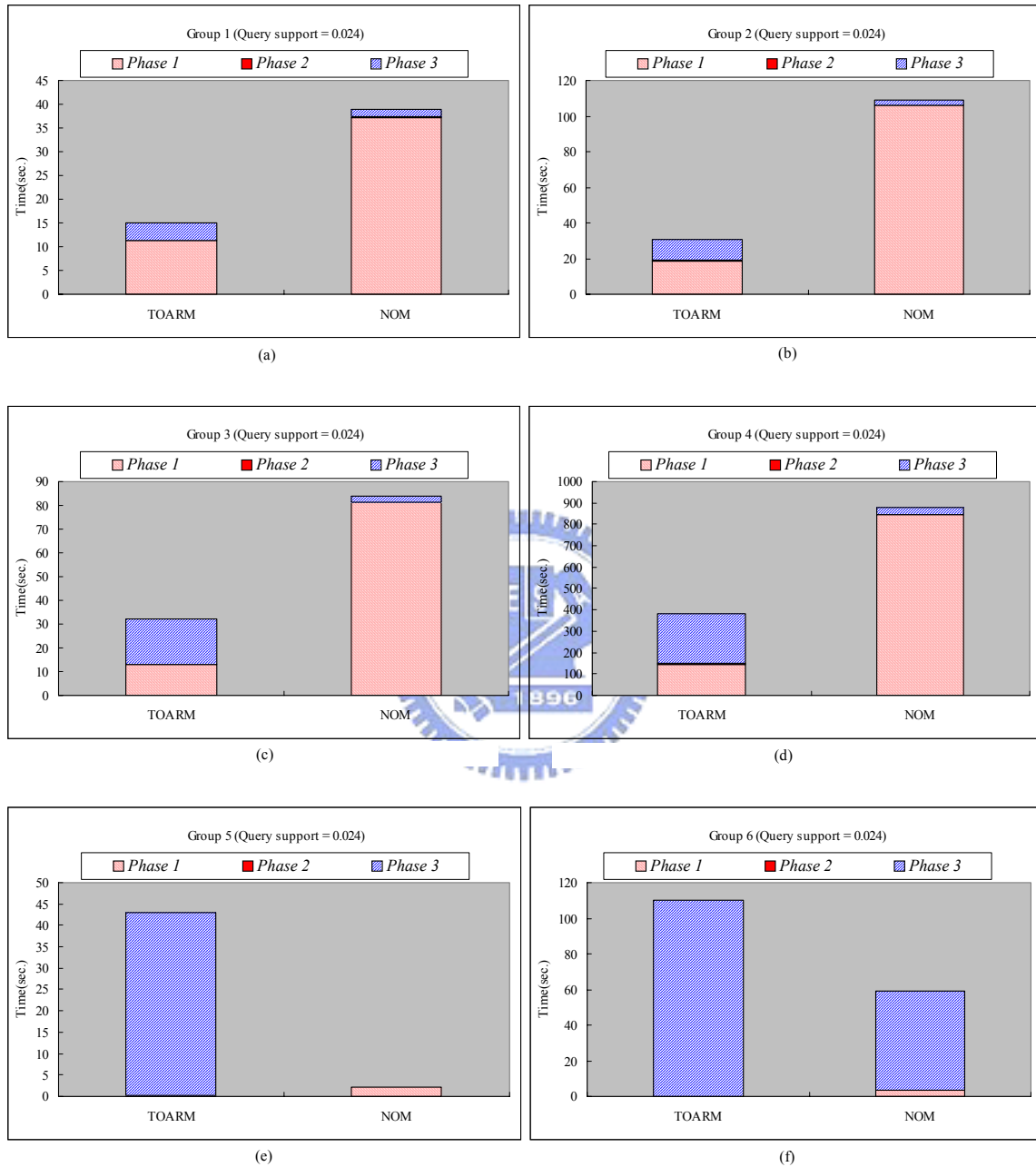


Figure 6-10: The influence of the number of negative itemsets on execution time for Groups 1 to 6

For Groups 1 and 3, most candidate itemsets appeared in nearly all tuples in

EMPR such that the negative itemsets provided little help in calculating counts of candidates. The reduced execution time in Phase 3 was thus not significant when compared to that in Phase 1. This can be easily seen from Figures 6-10(a) and 6-10(c) that the execution time by TOARM was less than that by NOM. By contrast, for Groups 2 and 4, most candidate itemsets appeared in only one or few tuples in EMPR. The effect of negative itemsets on finding tight upper-bound supports thus become apparent. However, since the computation cost in Phase 1 was much larger than that in Phase 3, the execution time by TOARM was still less than that by NOM as shown in Figures 6-10(b) and 6-10(d). Even so, it can be observed from Figures 6-10(e) and 6-10(f) that TOARM did not always outperform NOM for Groups 5 and 6, This phenomena is especially when the size of candidate itemsets is small and the size of underlying data is large. For Group 5, NOM could decide all the candidate itemsets in Phase 2 and thus no one in Phase 3 needed to be processed. For Group 6, the computation cost in Phase 3 was much higher than that in Phase 1 because the size of underlying data is large.

The performance of the NOM algorithm with a direct hashing function was then evaluated. Let $NOM(H)$ denote running the NOM algorithm with a direct hashing function. The execution times on Groups 1 to 4 are shown in Figures 6-11(a) to 6-11(d), where the query support is set to 2.4% and the size of the hash table is about 10K. It can be easily seen that the computation time in Phase 1 of the NOM algorithm can be efficiently reduced by the hashing technique.

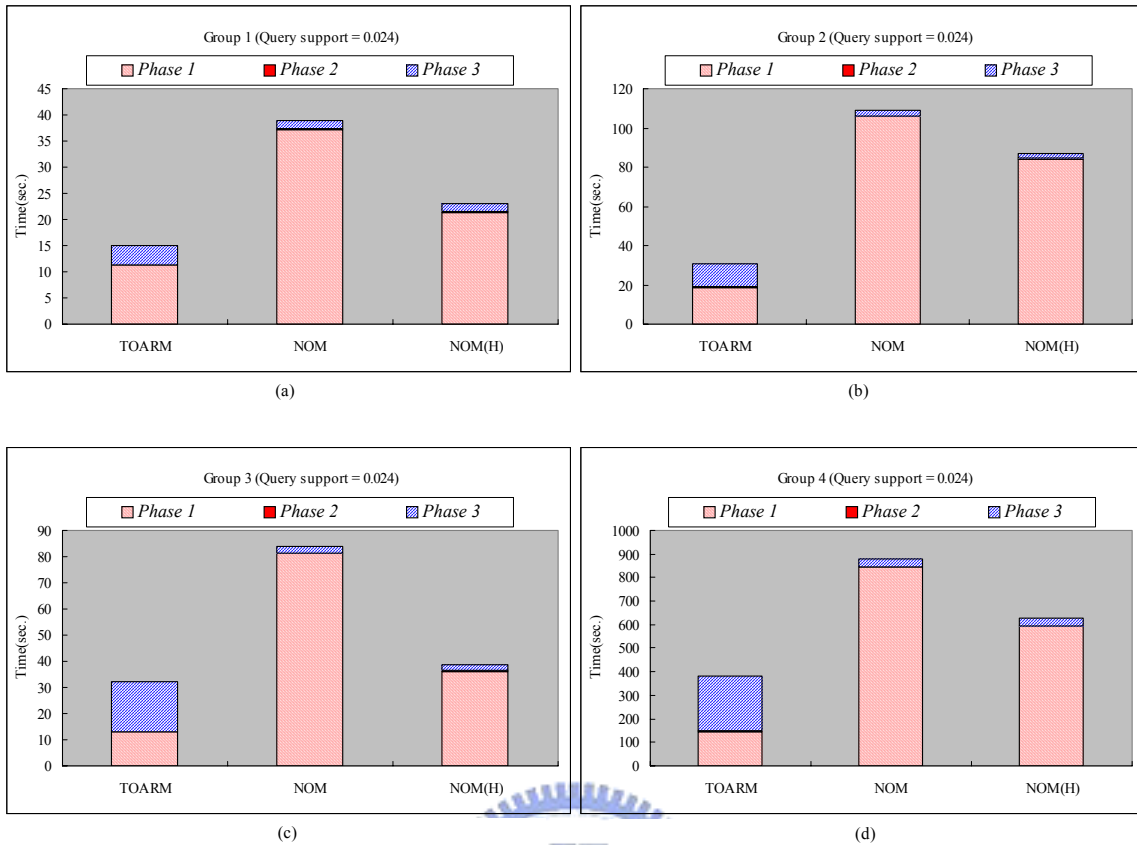


Figure 6-11: Execution times of the NOM algorithm respectively with and without a direct hashing function on Groups 1 to 4

Next, experiments were made to show the effect of using the lattice data structure on the NOM algorithm. The execution time of the NOM algorithm was compared with that of the L NOM algorithm with and without a direct hashing function. The query support is set to 2.4% and the size of the hash table is about 10K. The results for Groups 1 to 4 are shown in Figures 6-12(a) to 6-12(d), where L NOM and L NOM(H) respectively denote running L NOM algorithm with and without a direct hashing function. It is easily seen that the execution time by the L NOM algorithm was always much less than that by the NOM algorithm. The reduced computation cost in Phase 1 of the L NOM(H) algorithm was not significant because the NOM approach with the lattice data structure could effectively restrict the number

of candidate itemsets to be examined.

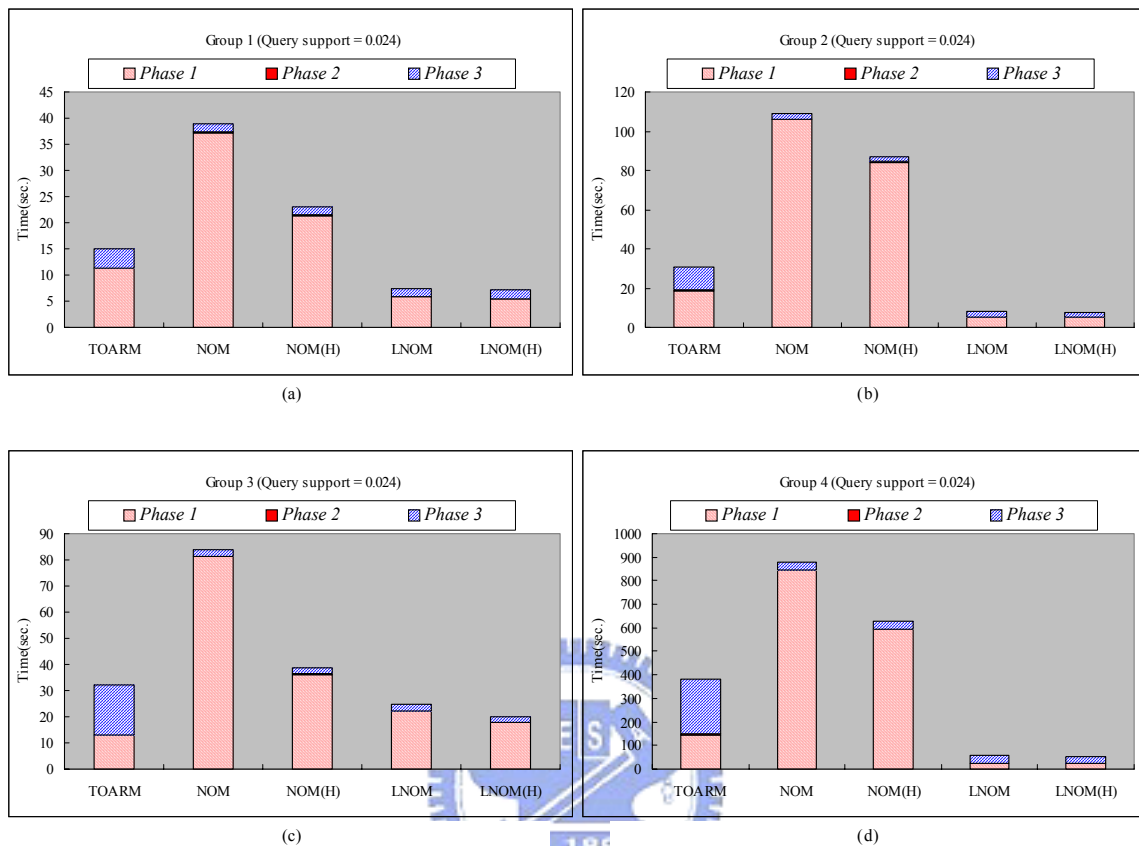


Figure 6-12: Execution times spent by the NOM and LNOM algorithms on Groups 1

to 4

6.7.2 Experimental Results for Real Datasets

In addition to the above synthetic datasets, a real one called the *BMS-POS* dataset [106] and used in the KDDCUP 2000 competition was run in our experiments. The *BMS-POS* dataset contains several years of point-of-sale data from a large electronics retailer. Each transaction in this dataset is a customer purchase transaction consisting of all the product categories purchased at one time. There are 515,597 transactions in the dataset. The number of distinct items is 1,657, the maximal transaction size is 164, and the average transaction size is 6.5. This dataset was also

used in the KDDCUP 2000 competition. In our experiments, the seventh group of data consisted of ten equal-size data subsets partitioned from the *BMS-POS* dataset, and its corresponding MPR and EMPR were shown in Table 6-10.

Table 6-10: Mining information for the seventh group

<i>Group</i>	<i>Initial minimum support</i>	<i>Average length of maximal frequent itemsets</i>	<i>Average size of frequent itemsets</i>	<i>Average size of negative itemsets</i>
7	0.1%	11	9006	10762

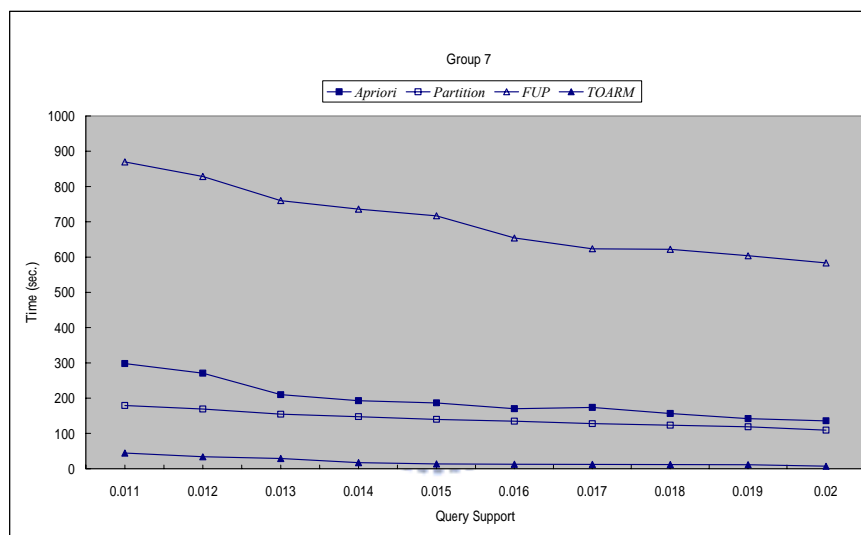


Figure 6-13: Execution times for the TOARM, Apriori, Partition and FUP algorithms on Group 7

The execution time for the TOARM, Apriori, Partition and FUP algorithms on Group 7 is shown in Figure 6-13. The TOARM algorithm had the best performance among the four algorithms. Then the execution time spent by the NOM and the LNOM algorithms for Group 7 along with query supports ranging from 0.2% to 1.1% in mining requests is shown in Figure 6-14. The experimental results were consistent with the above discussion. The LNOM algorithm had much better performance than

the NOM algorithm, especially when the number of candidate itemsets is large due to a low query support.

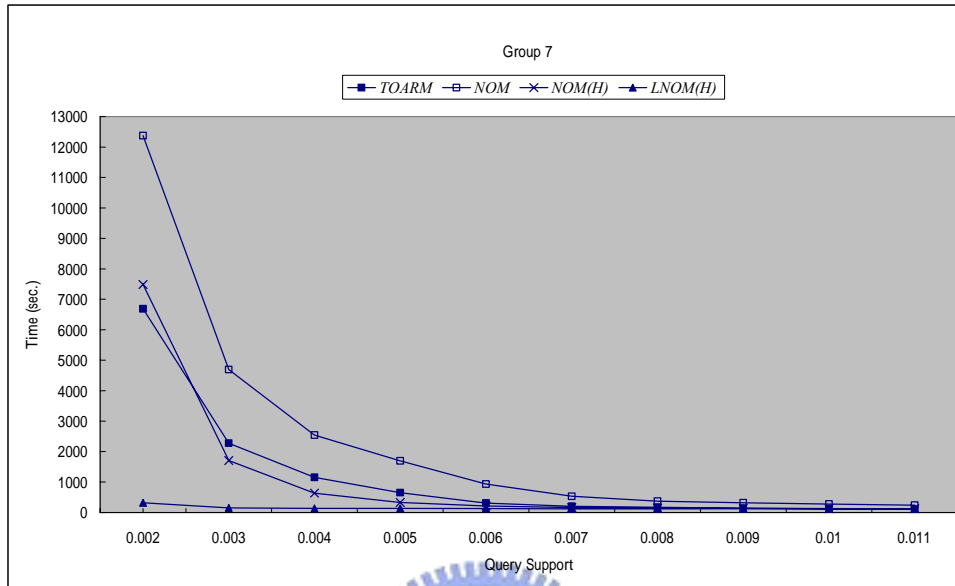


Figure 6-14: Execution times spent by the TOARM, NOM and LNom algorithms on Groups 7

6.8 Conclusion

By structurally and systematically storing context and mining information in the MPR, our proposed TOARM approach can easily and efficiently derive association rules that satisfy diverse user-concerned constraints. After that, the concept of *negative border* has been used to enlarge the mining information in the MPR to help get tight upper-bound supports of candidate itemsets and thus reduce the number of candidate itemsets to be considered. Based on the EMPR (*extended MPR*), a corresponding online mining approach called *Negative-Border Online Mining (NOM)* has been then proposed to efficiently and effectively utilize the information of *negative itemset* in the negative border. Consequently, for further improving the performance of NOM approach, the *lattice* data structure has been utilized to organize

and maintain all candidate itemsets such that the candidate itemsets with the same proper subsets can be considered at the same time. The derived *lattice-based* NOM (LNOM) approach will require only one scan of the itemsets stored in EMPR, thus saving much computation time. In addition, a hashing technique has been used to further improve the performance of the NOM approach since many itemsets stored in EMPR may be useless for calculating the counts of candidates. Experiments for both homogeneous and heterogeneous datasets are made, with results showing the effectiveness of the proposed approaches.



Chapter 7

Using Association Rule Mining Techniques on Knowledge Discovery Process in Semiconductor Manufacture


7.1 Introduction

In recent years, manufacturing processes have become more and more complex, and meeting high-yield target expectations and quickly identifying *root-cause machinesets*, the major killer machine(s) that causes a low-yield situation in a regular manufacturing procedure, also become essential issues. Although process control and statistical analysis techniques can be applied to establish a solid base for well-tuned manufacturing processes, identification of root-cause machineset is still hard and costly due to the existence of multiple coefficients among variants, nonlinear interactions, and the intermittent nature of the problem. For example, CIM/MES/EDA systems in most semiconductor manufacturing companies help users analyze collected manufacturing data in order to discover the root-cause machineset when the low-yield situation occurs; however, too many indexes and diagrams generated by the statistical methods in CIM/MES/EDA systems, such as K-W test, covariance analysis, regression analysis, etc., are usually not easy for engineers to assimilate and judge. On the other hand, lots of time is required to solve the false-alarm issue.

In the third part of this dissertation, we attempt to integrate incremental mining and multidimensional online mining techniques on knowledge discovery process in

semiconductor manufacture. We first define the *root-cause machineset identification problem* of analyzing correlations between combinations of machines and the defective products, and then propose the *Root-cause Machine Identifier* (RMI) approach [19] using a batch-based association rule mining algorithm to obtain candidate *root-cause machinesets* from a shipment of *wafer in process* (WIP) data to experts for further determination. After that, we propose the *progressive* RMI (PRMI) concept, which applies incremental mining techniques to progressively process previously mined candidate root-cause machinesets, and the *multidimensional* RMI (MRMI) concept, which applies multidimensional online mining techniques to diversely consider mined candidate root-cause machinesets from each shipment for supporting online decision services.

7.2 Related Work



As mentioned above, The process of mining association rules can be roughly divided into two tasks [5]: *finding frequent itemsets* and *generating association rules*, where the first task is used to discover statistically significant patterns while the second task is used to obtain interesting rules. Since the first task is very time-consuming compared to the second one, the major challenges in mining association rules thus focus on how to reduce the search space and decrease the computation time required for the first task.. Some famous mining algorithms, such as Apriori [5], DIC [16], DHP [67], Partition [78], Sampling [61] and FP-Growth [40][95], were proposed to achieve this purpose. Among them, the Apriori algorithm, which is the most well-known, utilizes a level-wise candidate generation approach to reduce its search space such that only the frequent itemsets found in the previous level are treated as seeds for generating the candidate itemsets in the current level. This

Apriori property can greatly reduce the number of itemsets considered in a mining process. Many later algorithms were based on this property and attempted to further reduce candidate itemsets and I/O costs. Comprehensive overviews can be found in [18][38].

Although a level-wise candidate generation algorithm can efficiently discover significant patterns, many of them may be not interesting to users. Thus, designing a useful interestingness measurement is becoming an important issue [15][18][38][82]. *Confidence*, the most typical interestingness measurement for association rule mining, measures the conditional probability of events associated with a particular rule. For example, an association rule $A \rightarrow B$ with confidence $c\%$ means that $c\%$ of all transactions containing A also contain B . However, the confidence measurement may be misleading or insufficient for many real-world applications. For example, given a minimum confidence of 60%, the association rule $milk \rightarrow cigarette$ with confidence 66% is then discovered in a supermarket. However, it is misleading since the probability of purchasing cigarette is 70%, which is even larger than 66%. In fact, milk and cigarette associate negatively since purchasing milk actually decreases the desirability of purchasing cigarettes. Thus, many researches [15][16][28][42][71][80][82] have proposed other effective interestingness measurements.

In [71], Piatetsky-Shaprio proposed a domain-independent interestingness measurement to evaluate the interestingness of discovered rule $A \rightarrow B$:

$$\phi = \frac{|A \& B| - |A||B|/N}{\sqrt{|A||B|(1 - |A|/N)(1 - |B|/N)}},$$

where, N denotes the total number of tuples in the database, $|A|$ denotes the number of tuples that contain the antecedent A , $|B|$ denotes the number of tuples that contain the consequent B , and $|A \& B|$ denotes the number of tuples that contain both A and B . The

range of this interestingness measurement is between -0.25 and 0.25 .

7.3 Root-cause Machineset Identification Problem

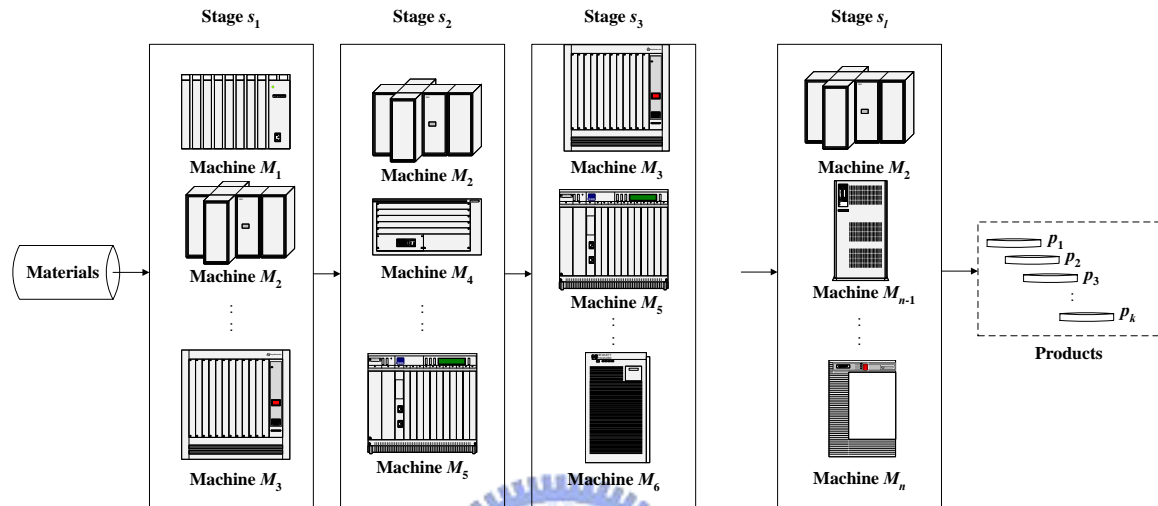


Figure 7-1: A general manufacturing process

Figure 7-1 shows a general manufacturing process requiring a multistage production procedure. Each stage may have more than one machine performing the same task. Thus, products may pass through different machines in a specific stage. Assume a shipment consists of k identical products $\{p_1, p_2, \dots, p_k\}$. Each product must pass through l stages $\langle s_1, s_2, \dots, s_l \rangle$ in sequence to be finished, and there are n manufacturing machines $\{M_1, M_2, \dots, M_n\}$ in this l -stage shipment. Note that a machine with multiple functions may appear in more than one stage in the process. The *manufacturing process relation*, $r = \{t_1, t_2, \dots, t_k\}$, based on the relation schema $R(PID, S_1, S_2, \dots, S_l, D)$, can be used to record the processing information from each stage and the test result for each product, p_i , $1 \leq i \leq k$. Among the attributes in R , PID is an identification attribute used to uniquely label the products, S_i is a context

attribute associated with a pair $\langle \text{manufacturing machine}, \text{timestamp} \rangle$ used to indicate that the *manufacturing machine* is used in the i -th stage at the *timestamp* for each product, and D is a class attribute used to state whether a product is defective or not.

Example 7-1: Table 7-1 shows a manufacturing process relation used to record five-stage ($l=5$) and seven-machine ($n=7$) processing information for a shipment consisting of six products ($k=6$). The first tuple shows that product p_1 passed through stage 1 on $\langle M_1, 1 \rangle$, stage 2 on $\langle M_5, 3 \rangle$, stage 3 on $\langle M_3, 10 \rangle$, stage 4 on $\langle M_4, 12 \rangle$, and stage 5 on $\langle M_5, 14 \rangle$, and its test result shows a defect ($D=1$). The other tuples have similar meanings. ■

Table 7-1: A manufacturing process relation for six products in a five-stage manufacturing procedure

PID	S_1	S_2	S_3	S_4	S_5	D
1	$M_1, 1$	$M_5, 3$	$M_3, 10$	$M_4, 12$	$M_5, 14$	1
2	$M_2, 5$	$M_1, 8$	$M_1, 12$	$M_2, 15$	$M_1, 17$	0
3	$M_3, 2$	$M_3, 7$	$M_5, 13$	$M_4, 17$	$M_3, 20$	0
4	$M_3, 4$	$M_1, 6$	$M_4, 14$	$M_4, 18$	$M_5, 19$	1
5	$M_4, 7$	$M_2, 11$	$M_4, 15$	$M_2, 20$	$M_5, 23$	1
6	$M_3, 9$	$M_3, 8$	$M_6, 12$	$M_4, 16$	$M_7, 20$	0

Our goal is to identify the *root-cause machineset* for a given manufacturing process relation. In recent years, many approaches have been proposed to solve similar problems. Examples are such as V. Raghavan applied decision tree to discover the root cause of yield loss in integrated circuits [74], M. Gardner and J. Bieker combined self-organizing neural networks and rule induction to identify the critical poor yield factors from normally collected wafer manufacturing data [33], F. Mieno et al. applied a regression tree analysis to failure analysis in LSI manufacturing [63].

7.4 Root-cause Machine Identifier (RMI) Approach

We attempt to apply the technique of association rule mining to solve the root-cause machineset identification problem. According to the general operation of mining association rules, there are three major scenarios need to be discussed:

(1) *Data preprocessing scenario*: Since the technique of association rule mining is usually performed on transactional data (its target of mining is not predetermined), it is important to transform the data in the manufacturing process relation into the materials and retain the *appropriate* relationships between machines and products that facilitate mining.

(2) *Mining procedure scenario*: A product may pass through hundreds of stages (machines) to be finished. The evaluation of all combinations of machines is relatively enormous and impractical. Therefore, the pruning strategy is required to remove the candidates with inadequate evidences to be the root cause such that the search space and the computation time can be reduced.

(3) *Visualization scenario*: Among the generated candidates, a suitable interestingness measurement is then needed to identify the root-cause machineset.

To overcome the above three scenarios, the *Root-cause Machine Identifier (RMI)* approach shown in Figure 7-2 consisting of three phases, *data preprocessing phase*, *candidate generation phase* and *interestingness ranking phase*, is proposed. The data preprocessing phase focuses on transforming the raw data in a given manufacturing process relation into transactional data. The candidate generation phase focuses on generating candidate machinesets from the transactional data, and the interestingness measurement phase focuses on identifying the root-cause machineset from the obtained candidate machinesets.

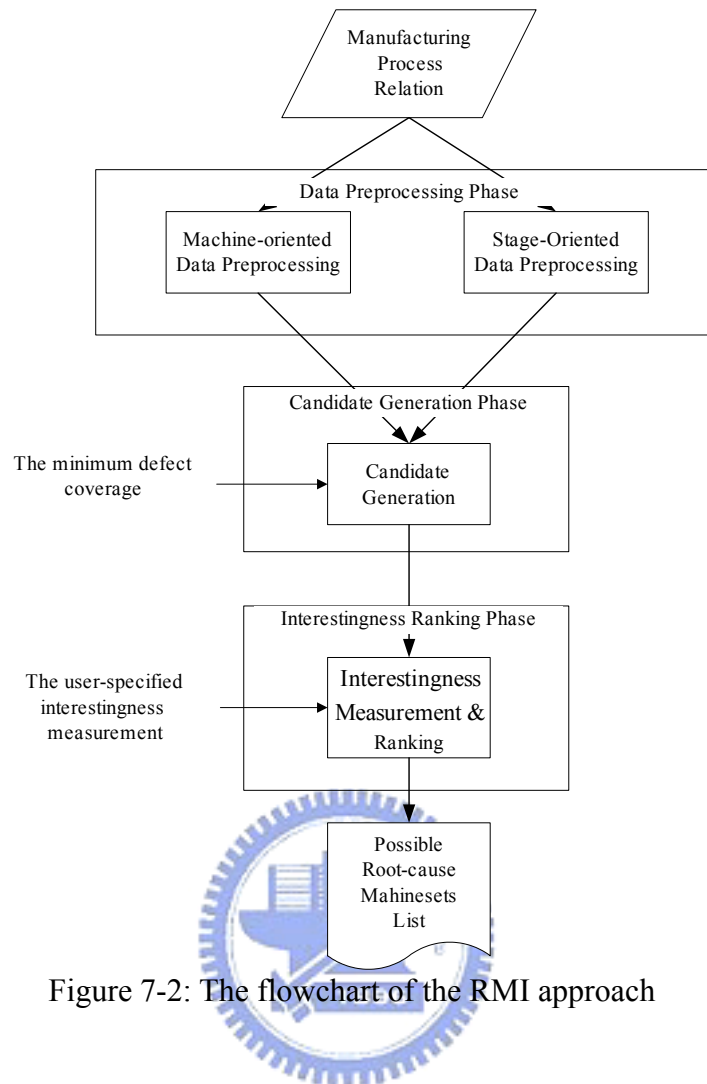


Figure 7-2: The flowchart of the RMI approach

By the user-selected preprocessing procedure in the data preprocessing phase, the RMI approach first gets materials transformed from the data in the manufacturing process relation. Then given a user-specified *minimum defect coverage*, a threshold used to remove the machinesets without enough evidences to be the root cause, the RMI approach generates all candidate machinesets by the candidate generation phase. Finally, by the interestingness ranking phase, the RMI approach ranks the candidate machinesets based upon a user-specified interestingness measurement and provides the result to experts for further determination.

7.4.1 The Data Preprocessing Phase of RMI Approach

The data preprocessing phase first selects the defective tuples from a given manufacturing process relation. Two data preprocessing procedures, *machine-oriented* and *stage-oriented* preprocessing procedures, have been proposed to handle different manufacturing defect hypotheses. The machine-oriented preprocessing procedure concentrates on the machines a product passes through, regardless of the manufacturing stage. Thus, although a machine may be used in more than one stage in a tuple because of its multi-functionality, this preprocessing procedure treats it as only a single appearance.

Example 7-2: For the manufacturing process relation shown in Table 7-1, the machine-oriented preprocessing procedure transforms the defect tuples 1, 4 and 5 as shown in Table 7-2. The tuple $TID1 = \{M_1, M_2, M_4, M_5\}$ means that the product p_1 passed through four machines, M_1, M_3, M_4 and M_5 . The other tuples have similar meanings. ■

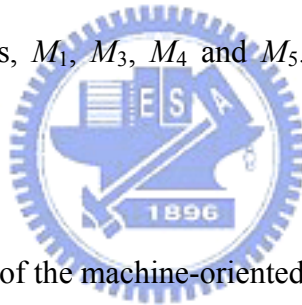


Table 7-2: An example of the machine-oriented preprocessing procedure

<i>TID</i>	<i>Items</i>
1	M_1, M_3, M_4, M_5
4	M_1, M_3, M_4, M_5
5	M_2, M_4, M_5

The machine-oriented preprocessing procedure transforms the processing information in the manufacturing process relation into intuitive transactional data and assumes a machine's functions are correlated. That is, if one function is faulty, the other may also be. By contrast, the stage-oriented preprocessing procedure assumes that a machine's functions are not correlated. If one function is faulty, the other ones may still operate normally. Therefore, this preprocessing procedure treats machines in different stages as distinct individuals.

Example 7-3: For the manufacturing process relation shown in Table 7-1, the stage-oriented preprocessing procedure transforms the defect tuples 1, 4 and 5 as shown in Table 7-3. The machine m_{11} indicating M_1 is used at stage 1 is different from the machine m_{12} indicating M_1 is used at stage 2. The tuple $TID1 = \{m_{11}, m_{52}, m_{33}, m_{44}, m_{55}\}$ means that the product p_1 passed through stage 1 on M_1 , stage 2 on M_5 , stage 3 on M_3 , stage 4 on M_4 , and stage 5 on M_5 . The other tuples have similar meanings. ■

Table 7-3: An example of the stage-oriented preprocessing procedure

<i>TID</i>	<i>Items</i>
<i>1</i>	$m_{11}, m_{52}, m_{33}, m_{44}, m_{55}$
<i>4</i>	$m_{31}, m_{12}, m_{43}, m_{44}, m_{55}$
<i>5</i>	$m_{41}, m_{22}, m_{43}, m_{24}, m_{55}$

7.4.2 The Candidate Generation Phase of RMI Approach

A level-wise processing procedure like finding frequent itemsets in association rules mining is used to generate possible sets of machines called *candidate machinesets*. The *defect coverage* of a machineset is defined as the percentage of all defective products passing through the target machineset. Therefore given the user-specified *minimum defect coverage*, in the first iteration, the proposed candidate generation phase calculates the defect coverage for each individual machine, and then retains the 1-machinesets that satisfy the minimum defect coverage as candidates. In the second iteration, the proposed phase generates machinesets consisting of two machines by joining the candidate 1-machinesets from the first iteration, and retains the 2-machinesets that satisfy the minimum defect coverage as candidates. In each subsequent iteration, candidate machinesets found in the preceding iteration are used as seeds in the current iteration, and the process continues until no new candidate machinesets can be generated.

Since this level-wise processing procedure is based on the Apriori property, each proper subset of a candidate machineset must be a candidate. In other words, if a machineset does not satisfy the user-specified minimum defect coverage, then none of its proper supersets will be. This can greatly reduce the number of candidate machinesets to be considered. Moreover, to improve the computation performance, the candidate generation phase retains defective product information for each candidate machineset in the current level so that each machineset's defect coverage information in the next level can be efficiently calculated by utilizing the retained information rather than re-processing the original database.

Example 7-4: Table 7-4 shows the defect coverage for each 1-machineset in Table 7-3. The first tuple shows that only the defective product p_1 passed through the machineset m_{11} . Thus, the defect coverage of m_{11} is $1/3 = 33\%$. ■

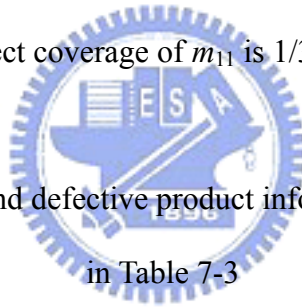


Table 7-4: Defect coverage and defective product information for each 1-machineset in Table 7-3

<i>Machineset</i>	<i>Involved Defective Products</i>	<i>Defect Coverage</i>
m_{11}	p_1	33%
m_{31}	p_4	33%
m_{41}	p_5	33%
m_{52}	p_1	33%
m_{12}	p_4	33%
m_{22}	p_5	33%
m_{33}	p_1	33%
m_{43}	p_4, p_5	66%
m_{44}	p_1, p_4	66%
m_{24}	p_5	33%
m_{55}	p_1, p_4, p_5	100%

Example 7-5: Continuing from Example 7-4 and assuming the user-specified minimum defect coverage is 50%, Table 7-5 shows candidate 1-machinesets of Table 7-4.

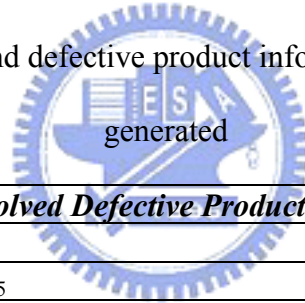
Table 7-5: Defect coverage and defective product information for each candidate

1-machineset obtained

<i>Machineset</i>	<i>Involved Defective Products</i>	<i>Defect Coverage</i>
m_{43}	p_4, p_5	66%
m_{44}	p_1, p_4	66%
m_{55}	p_1, p_4, p_5	100%

Next, 2-machinesets $\{m_{43}, m_{44}\}$, $\{m_{43}, m_{55}\}$ and $\{m_{44}, m_{55}\}$ are then generated by joining the candidate 1-machinesets in Table 7-5. The defect coverage for $\{m_{43}, m_{44}\}$ is 33% and its defective product information is $\{p_4\}$ by performing the intersection of the set of defective products of m_{43} and m_{44} . Complete results are shown in Table 7-6.

Table 7-6: Defect coverage and defective product information for each 2-machinesets



<i>Machineset</i>	<i>Involved Defective Products</i>	<i>Defect Coverage</i>
m_{43}, m_{44}	p_4	33%
m_{43}, m_{55}	p_4, p_5	66%
m_{44}, m_{55}	p_1, p_4	66%

As we can see, the machineset $\{m_{43}, m_{44}\}$ is removed since its defect coverage is less than 50%, the specified minimum defect coverage. The resulting candidate 2-machinesets are shown in Table 7-7.

Table 7-7: Defect coverage and defective product information for each candidate

2-machineset obtained

<i>Machineset</i>	<i>Involved Defective Products</i>	<i>Defect Coverage</i>
m_{43}, m_{55}	p_4, p_5	66%
m_{44}, m_{55}	p_1, p_4	66%

Next, the only 3-machineset $\{m_{43}, m_{44}, m_{55}\}$ generated by joining the candidate

2-machinesets in Table 7-7. However, since $\{m_{43}, m_{44}\}$ is not included in the set of candidate 2-machinesets, it is removed according to above-mentioned Apriori property. All candidate machinesets generated are shown in Table 7-8. ■

Table 7-8: Defect coverage and defective product information for each candidate machinesets obtained

<i>Machineset</i>	<i>Involved Defective Products</i>	<i>Defect Coverage</i>
m_{43}	p_4, p_5	66%
m_{44}	p_1, p_4	66%
m_{55}	p_1, p_4, p_5	100%
m_{43}, m_{55}	p_4, p_5	66%
m_{44}, m_{55}	p_1, p_4	66%

7.4.3 The Interestingness Ranking Phase of RMI Approach

Although a candidate machineset having high defect coverage is statistically significant, it may not have a high possibility of being the root cause. For example, the defect coverage of m_{43} is the same as that of m_{44} in Table 7-8, but intuitively, m_{43} is more probable than m_{44} since all products passing through it are defective. In this section, an interestingness ranking phase using an interestingness measurement to evaluate correlations between candidate machinesets and defective products is proposed for finding the root-cause machineset. Below, in addition to two typical interestingness measurements *confidence* and ϕ , a novel interestingness measurement called *continuity-based interestingness measurement* is proposed to extend ϕ .

Confidence, the most well-known interestingness measurement for association rule mining, calculates the conditional probability that a candidate machineset causes defective products (*machineset*→*defect*). That is, it calculates the percentage of all products passing through a candidate machineset that are defective. ϕ , a

domain-independent interestingness measurement proposed by Piattetsky-Shapiro in [71] evaluates the discovered rule $A \rightarrow B$ as follows:

$$\phi = \frac{|A \& B| - |A||B|/N}{\sqrt{|A||B|(1-|A|/N)(1-|B|/N)}}. \quad (7-1)$$

This equation indicates the degree to which “when antecedent A appears, consequent B also appears”. If A is regarded as a certain candidate machineset and B is regarded as a defective product, then the equation calculates the degree of correlation between the candidate machineset and the defect.

However, the manufacturing process characteristics, such as the observation that the root-cause machineset often produces defective products continuously, are not considered in the two above-mentioned interestingness measurements. Thus, we propose *continuity* function to measure the continuity between the defective products for a candidate machineset. High continuity may indicate a higher probability of being the root cause. We can easily extend the interestingness measurement ϕ to ϕ' , called *continuity-based interestingness measurement*, as follows:

$$\phi' = \phi * \text{continuity}. \quad (7-2)$$

The *continuity* function calculates the reciprocal of the average distance between pairs of neighboring defective products in the product sequence as follows:

$$\left\{ \begin{array}{ll} \text{Continuity} = 0 & \text{if } |X| \leq 1 \\ \text{Continuity} = \frac{1}{\sum_{i=1}^{|X|-1} d(\alpha(x_i), \alpha(x_{i+1})) / |X| - 1} & \text{if } |X| > 1 \end{array} \right. , \quad (7-3)$$

where $X = (x_1, x_2, \dots)$ denotes a defective product sequence contained in the product sequence $P = (p_1, p_2, \dots)$ which is a sequence of products passing through a candidate machineset (i.e. X is a subsequence of P), $|X|$ denotes the number of defective

products, $\alpha(x_i)$ denotes the order of the defective product x_i in P (e.g., if $\alpha(x_i) = j$, x_i is the j -th product in P), and $d(\alpha(x_i), \alpha(x_{i+1}))$ is the distance of $\alpha(x_i)$ and $\alpha(x_{i+1})$, which can easily be calculated by $\alpha(x_{i+1}) - \alpha(x_i)$.

Example 7-6: Table 7-9 shows the product sequence, defective product sequence, and calculated continuity value for each candidate machineset in Table 7-8. Among them, the continuity value of m_{44} is $\frac{1}{(d(\alpha(p_1), \alpha(p_4)))/(2-1)} = 0.5$ according to its product sequence (p_1, p_3, p_4) and defective product sequence (p_1, p_4) . ■

Table 7-9: Calculated continuities for each candidate machineset in Table 7-8

<i>Machineset</i>	<i>Product Sequence</i>	<i>Defective Product Sequence</i>	<i>Continuity</i>
m_{43}	(p_4, p_5)	(p_4, p_5)	1
m_{44}	(p_1, p_3, p_4)	(p_1, p_4)	0.5
m_{55}	(p_1, p_4, p_5)	(p_1, p_4, p_5)	1
m_{43}, m_{55}	(p_4, p_5)	(p_4, p_5)	1
m_{44}, m_{55}	(p_1, p_4)	(p_1, p_4)	1

According to the user-specified interestingness measurement, the set of candidate machinesets with their interestingness values are ranked in descending order.

Example 7-7: Continuing from Example 7-6, Table 7-10 shows the ϕ' for each candidate machineset. Since m_{55} has highest interestingness value, the machine M_5 is most likely the root-cause machineset. ■

Table 7-10: ϕ' for each candidate machinesets in Table 7-8

<i>Machineset</i>	ϕ	<i>Continuity</i>	ϕ'
m_{43}	0.67	1	0.67
m_{44}	0.167	0.5	0.0835
m_{55}	1	1	1
m_{43}, m_{55}	0.67	1	0.67
m_{44}, m_{55}	0.67	1	0.67

7.5 The Concepts of Progressive RMI (PRMI) and Multidimensional RMI (MRMI)

Although the proposed RMI approach can find candidate root-cause machinesets from a shipment to experts for further determination, it is difficult for an expert to find the actual root-cause machineset which is not apparent in the pool of candidate root-cause machinesets. For a complex manufacturing process such as the semiconductor manufacture, some root-cause machinesets are hard to be investigated and discovered in a short-term analysis due to their intermittent nature and gradually wearing. As a result, progressively monitoring previously mined candidate root-cause machinesets is a nontrivial work. In order to provide obtained evidences from processed shipments of data for later use, we can design a *progressive* RMI (PRMI) using incremental mining techniques to progressively process previously mined candidate root-cause machinesets and consider the influence of subsequent shipments on the possibility of being the root cause for each progressive candidate. Obviously, for achieving long-term analysis, the original *defect coverage* and *interestingness measurement* calculations need to be re-designed. Some efforts and works in temporal association rules mining [7][54][66][75][83], especially in [7][54], are related to PRMI and can be further referred to.

A large dedicated semiconductor company, such as TSMC (*Taiwan Semiconductor Manufacturing Corporation Ltd*), usually consists of many wafer fabs around the world and provides varied fabrication processes. Decision-makers usually may need to analyze yield situations, especially for a low-yield situation, in a shipment, fabrication, production line, wafer size or even fab location. They may also want to understand the change of yield in different dimensions. We can design a knowledge warehouse to structurally and systematically store the context information,

such as *fab location, wafer size, fabrication, product line, manufacturing time, etc.*, and the mining information, such as *the number of lots, candidate root-cause machinesets, etc.*, of each shipment for supporting decision-makers diversely considering problems at different aspects.

7.6 Experimental Results

The RMI approach was implemented in Java on a Pentium-IV 2.4G processor desktop with 512MB RAM, and nine real datasets with the known root-cause machineset provided by the *Taiwan Semiconductor Manufacturing Corporation* (TSMC) were used to evaluate its accuracy. As shown in Table 7-11, 368 and 2727 machines needed to be considered in machine-oriented and stage-oriented preprocessing procedures respectively for Case 1 having 153 products and each passing through 658 stages.

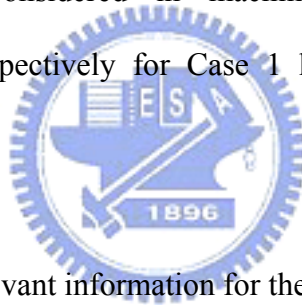


Table 7-11: Relevant information for the nine real datasets

<i>Dataset</i>	<i>Data size (Products*Stages)</i>	<i>Number of machines in machine-oriented preprocessing procedure</i>	<i>Number of machines in stage-oriented preprocessing procedure</i>
Case 1	153*658	368	2727
Case 2	145*867	497	4509
Case 3	141*837	499	4434
Case 4	116*624	416	2500
Case 5	305*733	424	3094
Case 6	53*587	411	2414
Case 7	484*709	455	3381
Case 8	106*632	419	2618
Case 9	77*1109	450	3367

With the minimum defect coverages ranging from 0.3 to 0.5 and the interestingness measurement ϕ' , the ranks of the actual root-cause machinesets among the generated candidate machinesets are shown in Table 7-12. For example, the rank

of the actual root-cause machineset for Case 1 was the 4th using machine-oriented preprocessing procedure with the minimum defect coverage = 0.3. Note that “X” means the actual root-cause machineset can not be found by the proposed RMI approach.

Table 7-12: Accuracy results of the RMI approach for the nine datasets

<i>Dataset</i>	<i>Machine-oriented preprocessing procedure</i>			<i>Stage-oriented preprocessing procedure</i>		
	<i>Min. defect coverage = 0.3</i>	<i>Min. defect coverage = 0.4</i>	<i>Min. defect coverage = 0.5</i>	<i>Min. defect coverage = 0.3</i>	<i>Min. defect coverage = 0.4</i>	<i>Min. defect coverage = 0.5</i>
	<i>Rank</i>	<i>Rank</i>	<i>Rank</i>	<i>Rank</i>	<i>Rank</i>	<i>Rank</i>
Case 1	4	4	4	22	12	6
Case 2	1	1	1	1	1	1
Case 3	1	1	1	1	1	1
Case 4	1	1	1	1	1	1
Case 5	1	1	1	1	1	1
Case 6	106	93	78	145	90	58
Case 7	6	5	5	2	1	1
Case 8	51	47	40	43	23	X
Case 9	74	50	44	10	X	X

As stated previously, the machine-oriented preprocessing procedure assumes all functions of a machine are co-affected whereas the stage-oriented preprocessing procedure assumes each function of a machine is independent. Table 7-12 shows that the RMI approach seems to have higher accuracy with the stage-oriented preprocessing procedure than with the machine-oriented preprocessing procedure in this semiconductor manufacturing experiment, if appropriate minimum defect coverages were set. By consulting with the product engineers for all above cases, the explanations of the experimental results are concluded as follows:

(a) For Cases 2, 3, 4 and 5, the actual root-cause machinesets were all ranked in the first place both with the machine-oriented and the stage-oriented preprocessing procedures. The major reasons are: (a) for Cases 2 or 3, the actual root-cause

machineset was a single-function machine. Therefore, it had the same interestingness value both with the stage-oriented and machine-oriented preprocessing procedures; (b) for Cases 4 or 5, most functions of the actual root-cause machineset had high interestingness values and were ranked in the top ten with the stage-oriented preprocessing procedure. Therefore, on the whole, the actual root-cause machineset with the machine-oriented preprocessing procedure still had a not-bad rank.

(b) For Cases 6, 7, 8, or 9, many normal products passed through the actual root-cause machineset without passing through the faulty function. Therefore the actual root-cause machineset had higher rank with the stage-oriented preprocessing procedure than with the machine-oriented preprocessing procedure, if an appropriate minimum defect coverage was set.

(c) For Case 1, the actual root-cause machineset had the same interestingness value in the machine-oriented and stage-oriented preprocessing procedures because it is a single-function machine (as in Cases 2 and 3). However, since most of the other candidate machinesets had lower interestingness values with the machine-oriented preprocessing procedure, the actual root-cause machineset with this preprocessing procedure had higher rank than with the stage-oriented preprocessing procedure. This was a special case in our experiments.

The actual root-cause machineset in most cases was ranked in the top ten with an appropriate minimum defect coverage, except in Case 6, which had only 53 products so the actual root-cause machineset was not more significant than the others. Intuitively, setting a higher minimum defect coverage will prune more machinesets from consideration during the candidate generation phase, and thus decrease the execution time. As shown in Table 7-12 and Figure 7-3, the higher minimum defect coverage is, the higher performance that RMI approach can be. However, the RMI

approach may prune the actual root-cause machinesets out once the minimum defect coverage is set too high. How to set appropriate minimum defect coverage is thus becoming a critical issue for future investigation.

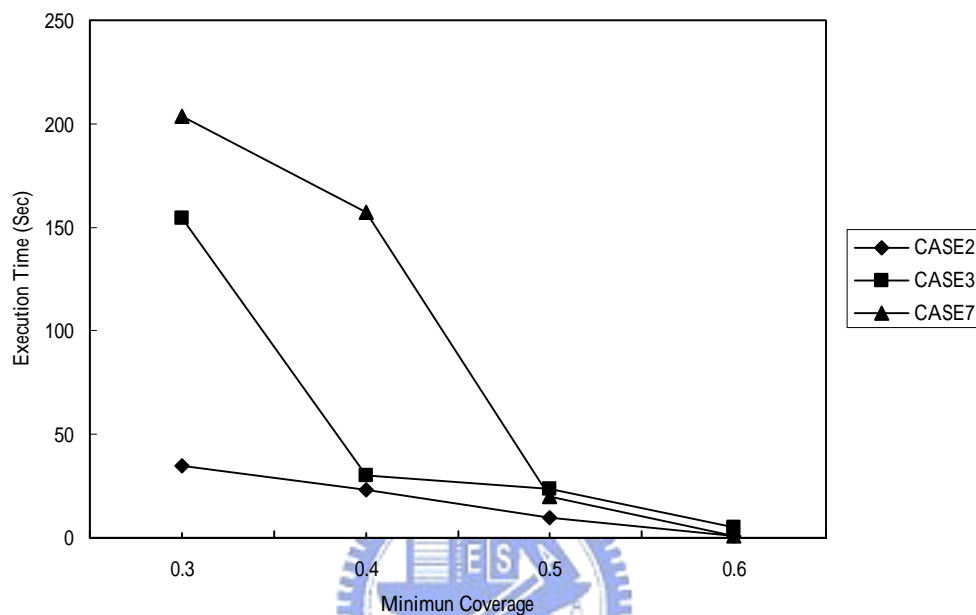


Figure 7-3: Execution times for Case 2, Case 3 and Case 7 with the minimum defect coverage set from 0.3 to 0.6

In order to demonstrate the accuracy of ϕ' compared to other known interestingness measures, Table 7-13 shows the rank of the actual root-cause machineset among all candidate machinesets generated by the RMI approach when associated with three interestingness measures, *confidence*, ϕ and ϕ' . The result shows that our proposed interestingness measurement ϕ' does not always outperform ϕ or *confidence* since the properties of all given testing case are different, and that continuity can highlight cases 1, 7 and 9 with strong continuity defect signal.

Table 7-13: Accuracy results of the RMI approach on the nine datasets for interestingness measurements confidence, ϕ and ϕ'

Dataset	Machine-oriented preprocessing procedure (Min. defect coverage = 0.3)			Stage-oriented preprocessing procedure (Min. defect coverage = 0.3)		
	Confidence	ϕ	ϕ'	Confidence	ϕ	ϕ'
	Rank	Rank	Rank	Rank	Rank	Rank
Case 1	8	4	4	41	17	22
Case 2	1	1	1	1	1	1
Case 3	1	1	1	1	1	1
Case 4	1	1	1	1	1	1
Case 5	1	1	1	3	1	1
Case 6	163	94	106	168	128	145
Case 7	9	8	6	1	4	2
Case 8	25	32	51	2	2	43
Case 9	114	57	74	46	22	10

7.7 Conclusion

Identification of the root-cause machineset in manufacturing can not only reduce manufacturing costs, but also improve manufactory performance. However, conventional methodologies for identifying root causes are restricted and dependent on experience and expertise. In this study, we have defined the *root-cause machineset identification problem* and proposed RMI approach to solve the problem efficiently and effectively. Two different data preparation procedures have proposed to transform the raw data into the desired format based on different manufacturing defect hypotheses. Also, an novel interestingness measurement considering the manufacturing continuity has proposed for the interestingness measurement phase in RMI approach. Currently, the proposed RMI approach has been considered as one of standard component in semiconductor manufacturing defect detection solution using data mining techniques of SAS[®] Taiwan Cooperation in order to help FAB users discover root causes. The experimental results show that about 80% cases can be ranked at the top ten and 20% cases are still remained unsolvable. In the future, we

will continue our research to refine interestingness measurements of RMI approach, and develop automatic/semi-automatic mechanisms to solve the low-yield situations.



Chapter 8

Summary and Future Work

Designing incremental mining algorithms that can effectively and efficiently utilize the previously mined information to reduce costs of knowledge maintenances is rather important and useful. In the first part of this dissertation, we have utilized the concepts of *pre-large patterns* and *closed patterns* to develop more efficient and practical approaches for maintaining association rules and sequential patterns especially in dense databases, and utilized the *domain-space weighting scheme* to develop a more accurate and adaptive document classifier.

For providing ad-hoc, query-driven and online mining supports, in the second part of this dissertation, the concept of *knowledge warehouse* and the architecture of *Online Knowledge Discovery System* (OKDS) have been proposed. By structurally and systematically storing context and mining information in the MPR, a form of knowledge warehouse, our proposed TOARM approach can easily and efficiently derive association rules that satisfy diverse, user-concerned constraints. In addition, the concept of *negative border* has been further applied in the MPR to form the EMPR, and based on the EMPR, the NOM and LNOM approaches have been developed to improve the performance of TOARM especially for heterogeneous blocks of data.

Consequently, in the third part of this dissertation, we attempt to apply incremental mining and multidimensional online mining techniques on knowledge discovery process in semiconductor manufacture. For a semiconductor manufacturing

company, the knowledge capable of quickly identifying *root-cause machinesets* is rather important. We have proposed the RMI approach using a batch-based association rule mining algorithm to provide an efficient and effective solution for the root-cause machineset identification problem. After that, the concepts of PRMI, which applies incremental mining techniques to progressively process previously mined candidate root-cause machinesets, and MRMI, which applies multidimensional online mining techniques to support multidimensional online generation of candidate root-cause machinesets, have been proposed to improve the accuracy and flexibility of RMI approach.

Some interesting issues may be studied in the future. In addition to record insertion, record deletion [87][89] and record modification [88] are also commonly seen in real-world applications. Processing record deletion and record modification are, however, different from processing record insertion. Design effective maintenance algorithms for association rules and sequential patterns as records are deleted or modified are thus nontrivial works. As for the proposed concept of multidimensional online mining, we can adopt other techniques to further improve the performance of the proposed methodology. For example, we can construct an iceberg cube [13][26] or use materialized views [17][97] for the proposed MPR or EMPR to provide more efficient online association rule generation and more powerful mining services. Moreover, we can also attempt to apply the multidimensional online mining concept to online decision support for other classes of knowledge, such as sequential patterns, classifications, clusters, etc. In the third part of this dissertation, although we expect the two concepts of *progressively processing previously mined patterns* and *structurally and systematically storing mined patterns* can respectively improve the accuracy of discovered knowledge and support decision-makers diversely considering

problems at different aspects, it is necessary to substantiate, test and deploy them in real-world cases in semiconductor manufacture.



Reference

1. R.C. Agarwal, C.C. Aggarwal, and V.V.V. Prasad. A tree projection algorithm for generation of frequent item sets. *Journal of Parallel and Distributed Computing*, Vol. 61, No. 3, pp. 350–371, 2001.
2. C.C. Aggarwal, P.S. Yu, A new approach to online generation of association rules, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 13, No. 4, pp. 527-540, 2001.
3. R. Agrawal, T. Imielinski, A. Swami, Mining association rules between sets of items in large database, *ACM SIGMOD Conference*, pp. 207-216, Washington DC, USA, 1993.
4. R. Agrawal, T. Imielinski, A. Swami, Database mining: a performance perspective, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 5, No. 6, pp. 914-925, 1993.
5. R. Agrawal, R. Srikant, Fast algorithm for mining association rules, *ACM VLDB Conference*, pp. 487-499, 1994.
6. R. Agrawal, R. Srikant, Mining sequential patterns, *IEEE International Conference on Data Engineering*, pp. 3-14, 1995.
7. J.M. Ale, G. Rossi, An approach to discovering temporal association rules, *ACM SAC Conference*, pp. 294-300, 2000.
8. W.G. Aref, M.G. Elfeky, A.K. Elmagarmid, Incremental, online, and merge mining of partial periodic patterns in time-series databases, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 16, No. 3, pp. 332-342, 2004.
9. J. Ayres, J.E. Gehrke, T. Yiu, J. Flannick, Sequential pattern mining using bitmaps, *The International Conference on Knowledge Discovery and Data Mining*, pp. 429-435, 2002.
10. L. Baker, A. McCallum, Distributional clustering of words for text classification, *ACM SIGIR Conference*, pp. 93-103, 1998.
11. R.J. Bayardo, R. Agrawal, D. Gunopulos, Constraint-based rule mining in large, dense databases, *IEEE International Conference on Data Engineering*, pp. 188-197, 1999.
12. C. Bettini, X.S. Wang, S. Jajodia, Mining temporal relationships with multiple granularities in time sequences, *IEEE Data Engineering Bulletin*, Vol. 21, pp. 512-521, 1999.
13. K. Beyer, R. Ramakrishnan, Bottom-up computation of sparse and iceberg cubes, *ACM SIGMOD Conference*, pp. 359-370, 1999.
14. L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, *Classification and regression trees*, Wadsworth, Belmont, CA. 1984.

15. S. Brin, R. Motwani, C Silverstein, Beyond market baskets: generalizing association rules to correlations, ACM SIGMOD Conference, pp. 265-276, Tucson, Arizona, USA, 1997.
16. S. Brin, R. Motwani, J.D. Ullman, S. Tsur, Dynamic itemset counting and implication rules for market basket data, ACM SIGMOD Conference, pp. 255-264, Tucson, Arizona, USA, 1997.
17. S. Chaudhuri, U. Dayal, An overview of data warehousing and OLAP technology, ACM SIGMOD Record, Vol. 26, No 1, pp. 65-74, 1997.
18. M.S. Chen, J. Han, P.S. Yu, Data mining: an overview from database perspective, IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 6, pp. 866-883, 1996.
19. W.C. Chen, S.S. Tseng, C.Y. Wang, A novel manufacturing defeat detection method using association rule mining techniques, An International Journal: Expert System with Application, Vol. 29, No. 4, pp. 807-815, 2005.
20. D.W. Cheung, J. Han, V.T. Ng, C.Y. Wong, Maintenance of discovered association rules in large databases: an incremental updating approach, IEEE International Conference on Data Engineering, pp. 106-114, 1996.
21. D.W. Cheung, S.D. Lee, B. Kao, A general incremental technique for maintaining discovered association rules, The International Conference on Database Systems for Advanced Applications, pp. 185-194, Melbourne, Australia, 1997.
22. D.Y. Chiu, Y.H. Wu, A.L.P. Chen, An efficient algorithm for mining frequent sequences by a new strategy without support counting, IEEE International Conference on Data Engineering, pp. 375-386, 2004.
23. F. Debole, F. Sebastiani, Supervised term weighting for automated text categorization, ACM SAC Conference, pp. 784-788, 2003.
24. S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, R. Hashman, Indexing by latent semantic indexing, Journal of the American Society for Information Science, Vol. 41, No. 6, 1990.
25. S. Dumais, J. Platt, D. Heckerman, M. Sahami, Inductive learning algorithms and representations for text categorization, ACM CIKM Conference, pp. 148-155, 1998.
26. M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, J.D. Ullman, Computing iceberg queries efficiently, ACM VLDB Conference, pp. 299-310, 1998.
27. R. Feldman, Y. Aumann, A. Amir, H. Mannila, Efficient algorithms for discovering frequent sets in incremental databases, ACM SIGMOD Workshop on DMKD, pp. 59-66, USA, 1997.
28. A.A. Freitas, On rule interestingness measures, Knowledge-Based Systems, Vol.

- 12, No. 5-6, pp. 309-315, 1999.
29. M. Fuketa, S. Lee, T. Tsuji, M. Okada, J. Aoe, A document classification method by using field association words, *An International Journal: Information Sciences*, Vol. 126, No. 1-4, pp. 57-70, 2002.
 30. H.N. Gabow, Z. Galil, T. Spencer, R.E. Tarjan, Efficient algorithms for finding minimum spanning trees in undirected and directed graphs, *Combinatorica*, Vol. 6, No. 2, pp. 109-122, 1986.
 31. L. Galavotti, F. Sebastiani, M. Simi, Experiments on the use of feature selection and negative evidence in automated text categorization. *The European Conference on Research and Advanced Technology for Digital Libraries*, 2000.
 32. V. Ganti, J. Gehrke, R. Ramakrishnan, DEMON: Mining and monitoring evolving data, *IEEE International Conference on Data Engineering*, pp. 439-448, 2000.
 33. M. Gardner, J. Bieker, Data mining solves tough semiconductor manufacturing problems, *The International Conference on Knowledge Discovery and Data Mining*, pp. 376-383, Boston, USA, 2000.
 34. H. George, J. Ron, P. Karl, Irrelevant features and the subset selection problem, *The International Conference on Machine Learning*, pp. 121-129, 1994.
 35. G. Grahne, L.V.S. Lakshmanan, X. Wang, M.H. Xie, On dual mining: from patterns to circumstances, and back, *IEEE International Conference on Data Engineering*, pp. 195-204, 2001.
 36. J. Han, G. Dong, Y. Yin, Efficient mining of partial periodic patterns in time series database, *IEEE International Conference on Data Engineering*, pp. 106-115, 1999.
 37. J. Han, L.V.S. Lakshmanan, R.T. Ng, Constraint-based, multidimensional data mining, *IEEE Computer Magazine*, pp.2-6, 1999.
 38. J. Han, M. Kamber, *Data mining: concepts and techniques*, Morgan Kaufmann, 2001.
 39. J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, M.C. Hsu, FreeSpan: Frequent pattern-projected sequential pattern mining, *The International Conference on Knowledge Discovery and Data Mining*, pp. 355-359, 2001.
 40. J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, *ACM SIGMOD Conference*, pp. 1-12, 2000.
 41. C. Hidber, Online association rule mining, *ACM SIGMOD Conference*, pp. 145-156, USA, 1999.
 42. R.J. Hilderman, H.J. Hamilton, Heuristic measures of interestingness, *The European Conference on Principles of Data Mining and Knowledge Discovery*, pp. 232-241, 1999.
 43. T.P. Hong, C.Y. Wang, Y.H. Tao, A new incremental data mining algorithm using pre-large itemsets, *An International Journal: Intelligent Data Analysis*, pp. 111-129,

- 2001.
44. T.P. Hong, C.Y. Wang, S.S. Tseng, Incremental data mining for sequential patterns using pre-large sequences, *The International Multiconference on Systemics, Cybernetics and Informatics*, Vol. 14, pp. 543-548, 2001.
 45. W.H. Immon, *Building the data warehouse*, Wiley Computer, 1996.
 46. T. Joachims, Text categorization with support vector machines: Linearizing with many relevant features, *The European Conference on Machine Learning*, vol. 1938, pp. 137-142, 1998.
 47. T. Joachims, Making large-scale SVM learning practical, *Advances in Kernel Methods-Support Vector Learning*, pp. 169-184, MIT Press, 1999.
 48. M. Kamber, J. Han, J.Y. Chiang, Metarule-guided mining of multi-dimensional association rules using data cubes, *The International Conference on Knowledge Discovery and Data Mining*, pp. 207-210, 1997.
 49. G. Karypic, E.H. Han, Concept indexing: a fast dimensionality reduction algorithm with applications to document retrieval and categorization, *ACM CIKM Conference*, pp. 12-19, 2000.
 50. H. Kona, S. Chakravarthy, Partitioned approach to association rule mining over multiple databases, *The International Conference on Data Warehousing and Knowledge Discovery*, pp. 320-330, 2004.
 51. L.V.S. Lakshmanan, C.K.S. Leung, R.T. Ng, Efficient dynamic mining of constrained frequent sets, *ACM Transaction on Database Systems*, Vol. 28, No. 4, pp. 337-389, 2003.
 52. L.V.S. Lakshmanan, R.T. Ng, J. Han, A. Pang, Optimization of constrained frequent set queries with 2-variable constraints, *ACM SIGMOD Conference*, pp. 157-168, Philadelphia, Pennsylvania, USA, 1999.
 53. B. Lan, B.C. Ooi, K.L. Tan, Efficient indexing structures for mining frequent patterns, *IEEE International Conference on Data Engineering*, pp. 453-462, 2002.
 54. C.H. Lee, M.S. Chen, C.R. Lin, Progressive partition miner: An efficient algorithm for mining general temporal association rules, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 15, No. 4, pp. 1004-1017, 2003.
 55. M.Y. Lin, S.Y. Lee, Incremental update on sequential patterns in large databases, *IEEE International Conference on Tools with Artificial Intelligence*, pp. 24-31, 1998.
 56. D.D. Lewis, R.E. Schapire, J.P. Callan, R. Papka, Training algorithms for linear text classifiers, *ACM SIGIR Conference*, pp. 13-19, 1996.
 57. D.D. Lewis, Naïve (bayes) at forty: The independence assumption in information retrieval, *The European Conference on Machine Learning*, pp. 4-15, 1998.
 58. D.D. Lewis, Reuters-21578 text categorization test collection distribution 1.0,

- <http://www.research.att.com/~lewis/reuters21578.html>, 1999.
59. R.L. Liu, and Y.L. Lu, Incremental context mining for adaptive document classification, The International Conference on Knowledge Discovery and Data Mining, pp. 599-604, 2002.
 60. H. Mannila, H. Toivonen, On an algorithm for finding all interesting sentences, The European Meeting on Cybernetics and Systems Research, pp. 973-978, 1996.
 61. H. Mannila, H. Toivonen, A.I. Verkamo, Efficient algorithm for discovering association rules, The AAAI Workshop on Knowledge Discovery in Databases, pp. 181-192, 1994.
 62. H. Mannila, H. Toivonen, A.I. Verkamo, Discovery of frequent episodes in event sequences, Data Mining and Knowledge Discovery, Vol. 1, pp. 259-289, 1997.
 63. F. Mieno, T. Santo, Y. Shibuya, K. Odagiri, H. Tsuda, R. Take, Yield improvement using data mining system, IEEE Semiconductor Manufacturing Conference, 1999.
 64. M.L. Minsky, S.A. Papert, Perceptrons: An introduction to computational geometry, MIT Press, 1969.
 65. R.T. Ng, L.V.S. Lakshmanan, J. Han, A. Pang, Exploratory mining and pruning optimizations of constrained associations Rules, ACM SIGMOD Conference, pp. 13-24, Seattle, Washington, USA, 1998.
 66. B. Ozden, S. Ramaswamy, A. Siberschatz, Cyclic association rules, IEEE International Conference on Data Engineering, pp. 412-421, 1998.
 67. J.S. Park, M.S. Chen, P.S. Yu, Using a hash-based method with transaction trimming for mining association rules, IEEE Transactions on Knowledge and Data Engineering, Vol. 9, No. 5, pp. 812-825, 1997.
 68. N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal, Discovering frequent closed itemsets for association rules. The International Conference on Database Theory, pp. 398-416, 1999.
 69. J. Pei, J. Han, R. Mao, CLOSET: An efficient algorithm for mining frequent closed itemsets, ACM SIGMOD Workshop on DMKD, pp. 11-20, May 2000.
 70. J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pino, Q. Chen, U. Dayal, M.C. Hsu, Mining sequential patterns by pattern-growth: The PrefixSpan approach, IEEE Transactions on Knowledge and Data Engineering, Vol. 16, No. 10, pp. 1-17, 2004.
 71. G. Piatetsky-Shapiro, Discovery, analysis and presentation of strong rules, G. Piatetsky-Shapiro, W.J. Frawley (Eds.), Knowledge Discovery in Databases, AAAI, pp. 229-247, 1991.
 72. H. Pinto, J. Han, J. Pei, K. Wang, Multi-dimensional sequential pattern mining, ACM CIKM Conference, pp. 81-88, 2001.
 73. J.R. Quinlan, C4.5: Programs for machine learning. Moran Kaufmann, San Mateo,

- 1993.
74. V. Raghavan, Application of decision trees for integrated circuit yield improvement, IEEE/SEMI Advanced Semiconductor Manufacturing Conference and Workshop, 2002.
 75. S. Ramaswami, S. Mahajan, A. Silberschatz, On the discovery of interesting patterns in association rules, ACM VLDB Conference, pp. 368-379, 1998.
 76. J.J. Rocchio, Relevance feedback in information retrieval, The Smart Retrieval System-Experiments in Automatic Document Processing, pp. 313-323, Prentice-Hall, 1971.
 77. N.L. Sarda, N.V. Srinivas, An adaptive algorithm for incremental mining of association rules, IEEE International Workshop on Database and Expert Systems, pp. 240-245, 1998.
 78. A. Savasere, E. Omiecinski, S. Navathe, An efficient algorithm for mining association rules in large databases, ACM VLDB Conference, pp. 432-444, 1995.
 79. F. Sebastiani, Machine learning in automated text categorization. ACM Computing Surveys, Vol. 34, No. 1, pp. 1-47, 2002.
 80. A. Silberschatz, A. Tuzhilin, What makes patterns interesting in knowledge discovery systems, IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 6, pp. 970-974, 1996.
 81. R. Srikant, R. Agrawal, Mining sequential patterns: Generalizations and performance improvements, The International Conference on Extending Database Technology, pp. 3-17, 1996.
 82. P.N. Tan, V. Kumar, Interestingness measures for association patterns: a perspective, The KDD Workshop on Postprocessing in Machine Learning and Data Mining, Boston, MA, 2000.
 83. A.U. Tansel, N.F. Ayan, Discovery of association rules in temporal databases, The AAAI Workshop on Knowledge Discovery in Databases, 1998.
 84. P.C. Taylor, B.W. Silverman, Block diagrams and splitting criteria for classification trees, Statistics and Computing, Vol. 3, pp. 147-161, 1993.
 85. S. Thomas, S. Bodagala, K. Alsabti, S. Ranka, An efficient algorithm for the incremental update of association rules in large databases, The International Conference on Knowledge Discovery and Data Mining, pp. 263-266, 1997.
 86. J. Wang, J. Han, J. Pei, Closet+: Searching for the best strategies for mining frequent closed itemsets, The International Conference on Knowledge Discovery and Data Mining, pp. 236-245, 2003.
 87. C.Y. Wang, T.P. Hong, S.S. Tseng, Maintenance of sequential patterns for record deletion, IEEE ICDM Conference, pp. 536-541, 2001.
 88. C.Y. Wang, T.P. Hong, S.S. Tseng, Maintenance of sequential patterns for record

- modification using pre-large sequences, IEEE ICDM Conference, pp. 693-696, 2002.
89. C.Y. Wang, T.P. Hong, S.S. Tseng, Maintenance of discovered sequential patterns for record deletion, *An International Journal: Intelligent Data Analysis*, Vol. 6, No. 5, pp. 399-410, 2002.
 90. C.Y. Wang, T.P. Hong, S.S. Tseng, Multidimensional on-line mining, *IEEE ICDM Foundation of Data Mining Workshop*, pp. 196-202, 2003.
 91. C.Y. Wang, S.S. Tseng, T.P. Hong, Y.S. Chu, Using extended multidimensional pattern relation for multidimensional on-line mining, *International Computer Symposium*, 2004.
 92. C.Y. Wang, S.S. Tseng, T.P. Hong, Flexible online association rule mining based on multidimensional pattern relations, to appear in *An International Journal: Information Sciences*, 2005.
 93. C.Y. Wang, S.S. Tseng, T.P. Hong, Y.S. Chu, Online generation of association rules under multidimensional consideration based on negative-border, to appear in *Journal of Information Science and Engineering*, 2005.
 94. B.B. Wang, R.I. McKay, H.A. Abbass, M. Barlow, A comparative study for domain ontology guided feature extraction, *ACM ACSC Conference*, pp. 69-78, 2003.
 95. K. Wang, L. Tang, J. Han, J. Liu, Top down FP-Growth for association rule mining, *The Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, pp. 334-340, 2002.
 96. W. Wibowo, H.E. Williams, Simple and accurate feature selection for hierarchical categorization, *ACM DocEng Conference*, pp. 111-118, 2002.
 97. J. Widom, Research problems in data warehousing, *ACM CIKM Conference*, pp. 25-30, 1995.
 98. X. Wu, S. Zhang, Synthesizing high-frequency rules from different data sources, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 15, No. 2, pp. 353-367, 2003.
 99. X. Yan, J. Han, R. Afshar, CloSpan: Mining closed sequential patterns in large database, *SIAM International Conference on Data Mining*, pp. 166-177, 2003.
 100. Y. Yang, An evaluation of statistical approaches to MEDLINE indexing, *The International Conference on American Medical Informatics Association*, pp. 358-362, 1996.
 101. Y. Yang, An evaluation of statistical approaches to text categorization, *Technical Report: CMU-CS-97-127*, Computer Science Department, Carnegie Mellon University, 1997.
 102. Y. Yang, J.O. Pedersen, A comparative study on feature selection in text

- categorization, The International Conference on Machine Learning, pp. 412-420, 1997.
103. M. Zaki, SPADE: An efficient algorithm for mining frequent sequences, Machine Learning, Vol. 40, pp. 31-60, 2001.
104. M. Zaki, C. Hsiao, CHARM: An efficient algorithm for closed itemset mining, SIAM International Conference on Data Mining, pp. 457-473, 2002.
105. S. Zhang, X. Wu, C. Zhang, Multi-database mining, IEEE Computational Intelligence Bulletin, Vol. 2, No. 1, pp. 5-13, 2003.
106. Z. Zheng, R. Kohavi, L. Mason, Real world performance of association rule algorithms, The International Conference on Knowledge Discovery and Data Mining, pp. 401-406, 2001.

