Mining Association Relationship in a Temporal Database

Chang-Hung Lee
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan, ROC
E-mail:{chlee@arbor.ee.ntu.edu.tw}
Abstract

Since the early work in algorithm Apriori, several efficient algorithms to mine association rules have been developed. These studies cover a broad spectrum of topics including: (1) fast algorithms based on the level-wise Apriori framework, partitioning, sampling, parallel methods, TreeProjection and FP-growth methods; (2) incremental updating; (3) mining of generalized multi-dimensional and multi-level rules; (4) mining of quantitative rules; (5) constraint-based rule mining and multiple minimum supports issues; (6) temporal association rule discovery; and (7) episode mining. While these are important results toward enabling the integration of association mining and fast searching algorithms, we note that these mining methods cannot effectively be applied to the mining of a *large incremental temporal database* which is of increasing popularity recently. Specifically, some phenomena are observed when we take the issues of *Incremental Updates, Weighted Transactions, Publication-like Items, Short Transactions, and Dynamic Thresholds* into consideration.

In view of this, we explore in this thesis an effective sliding-window filtering (abbreviated as SWF) algorithm for incremental mining of association rules. Under SWF, the cumulative information of mining previous partitions is selectively carried over toward the generation of candidate itemsets for the subsequent partitions. Algorithm SWF not only significantly reduces I/O and CPU cost by the concepts of cumulative filtering and scan reduction techniques but also effectively controls memory utilization by the technique of sliding-window partition. By utilizing proper scan reduction techniques, only one scan of the incremented dataset is needed by algorithm SWF.

Furthermore, without fully considering the time-variant characteristics of items and transactions, it is noted that some discovered rules may be expired from users’ interest. In other words, some discovered knowledge may be obsolete and of little use, especially when we perform the mining schemes on a transaction database of short life cycle products. This aspect is, however, rarely addressed in prior studies. In view of this, we broaden in this thesis the horizon of frequent pattern mining by introducing a weighted model of *transaction-weighted association rules* in a time-variant database. Specifically, we propose an efficient *Progressive Weighted Miner* (abbreviated as PWM) algorithm to perform the mining for this problem as well as conduct the corresponding performance studies.

In addition, in this thesis, we explore a new problem of mining *general temporal association rules* in publication databases. In essence, a publication database is a set of transactions where each transaction *T* is a set of items of which each item contains an individual exhibition period. The current model of association rule mining is not able to handle the publication database due to the following fundamental problems, i.e., (1) lack of consideration of the *exhibition period* of each individual item; (2) lack of an equitable support counting basis for each item. To remedy this, we propose an innovative algorithm *Progressive-Partition-Miner* (abbreviated as *PPM*) to discover general temporal association rules in a publication database. Explicitly, the execution time of *PPM* is, in orders of magnitude, smaller than those required by other competitive schemes which are directly extended from existing methods.

On the other hand, it is noted that the existing models of rule mining might not be able to discover user preferred frequent patterns efficiently due to the following two fundamental problems: (1) the
puzzles for mining association rules on a short transaction database; (2) lack of long patterns for sequential pattern mining. To remedy this, this thesis explores the mining of causality rules. The causality rule explored in this dissertation consists of a sequence of triggering events and a set of consequential events, and is designed with the capability of mining non-sequential, inter-transaction information across multiple categories. Hence, the causality rule mining provides a very general framework for rule derivation.

Moreover, with the fast increase in Web activities, mining of path traversal patterns plays an essential role in the Web mining. While existing methods are efficient for the mining of frequent path traversal patterns from the access information contained in a log file, these approaches are likely to over evaluate associations. Explicitly, most previous studies of mining path traversal patterns are based on the model of a uniform support threshold, where a single support threshold is used to determine frequent traversal patterns without taking into consideration such important factors as the length of the pattern, the positions of Web pages, and the importance of a particular pattern, etc. In reality, however, a Web page at a lower level of a Web site naturally has a lower occurrence frequency than their corresponding higher level concepts, e.g., the portal Web page. As a result, a low support threshold will lead to lots of uninteresting patterns derived whereas a high support threshold may cause some interesting patterns with lower supports to be ignored. Hence, to capture the very nature of the Web mining problems, it is desirable to have a more general model for the support threshold. This thesis broadens the horizon of frequent path traversal pattern mining by introducing a flexible model of mining Web traversal patterns with dynamic thresholds. Specifically, we explore a new data mining capability which involves mining path traversal patterns with the concept of dynamic thresholds in a time-variant Web environment.
Contents

1 Introduction

1.1 Motivation ........................................... 7
  1.1.1 Apriori-like algorithms .............................. 7
  1.1.2 Partition-based algorithms ........................... 8
  1.1.3 FUP-based algorithms ............................... 8
  1.1.4 FP-based algorithms ................................. 9

1.2 Overview of the Dissertation ................................ 9
  1.2.1 Incremental Updates .................................. 9
  1.2.2 Weighted Transactions ................................ 11
  1.2.3 Publication-like Items ............................... 13
  1.2.4 Short Transactions ..................................... 15
  1.2.5 Dynamic Thresholds .................................... 17

1.3 Organization of the Dissertation ............................... 20

2 Sliding Window Filtering: An Efficient Method for Incremental Mining on a Time-Variant Database 21

2.1 Introduction ........................................... 21

2.2 Preliminaries and Related Works ............................... 24
  2.2.1 Apriori-like algorithms .............................. 24
  2.2.2 Partition-based algorithms ........................... 25
  2.2.3 FUP-based algorithms ............................... 25

2.3 SWF: Incremental Mining with Sliding-Window Filtering ................................ 26
  2.3.1 An example of incremental mining by SWF ................. 26
  2.3.2 Algorithm of SWF ...................................... 29
  2.3.3 Correctness of SWF ................................... 32

2.4 Experimental Studies ...................................... 35
  2.4.1 Generation of synthetic workload ........................ 35
  2.4.2 Experiment one: Relative performance .................... 37
  2.4.3 Experiment two: Evaluation of I/O cost ................... 39
  2.4.4 Experiment three: Reduction of CPU and memory overhead ... 39
  2.4.5 Experiment four: Scaleup performance .................... 40

2.5 Summary ............................................ 42

3 Progressive Weighted Miner: An Efficient Method for Time-Constraint Mining on a Time-Variant Database 43

3.1 Introduction ........................................... 43
3.2 Problem Description ...................................... 48
3.3 Progressive Weighted Mining .......................................................... 50
  3.3.1 Algorithm of PWM ................................................................. 50
  3.3.2 An illustrative example of algorithm PWM ......................... 53
  3.3.3 Correctness of PWM .............................................................. 55
3.4 Performance Studies ................................................................. 57
  3.4.1 Generation of synthetic workload ........................................... 57
  3.4.2 Experiment one: Relative performance ................................... 58
  3.4.3 Experiment two: Evaluation of I/O cost and CPU overhead .......... 58
  3.4.4 Experiment three: Scaleup performance ................................... 59
3.5 Summary .......................................................... 60

4 Progressive Partition Miner: An Efficient Algorithm for Mining General Temporal Association Rules 61
  4.1 Introduction .......................................................... 61
  4.2 Problem Description .......................................................... 66
    4.2.1 Preliminaries .......................................................... 66
    4.2.2 Traversing the Search Space ........................................... 69
  4.3 Mining General Temporal Association Rules ................................. 71
    4.3.1 Algorithm of PPM .......................................................... 71
    4.3.2 An illustrative example of algorithm PPM ............................. 74
    4.3.3 Correctness of PPM .......................................................... 77
  4.4 Experimental Studies ........................................................ 80
    4.4.1 Generation of synthetic workload ..................................... 80
    4.4.2 Experiment one: Relative performance ................................ 81
    4.4.3 Experiment two: Evaluation of I/O cost and CPU overhead ........ 81
    4.4.4 Experiment three: Scaleup performance ................................ 83
  4.5 Summary .......................................................... 84

5 Causality Rules: Exploring the Relationship between Triggering and Consequential Events in a Database of Short Transactions 85
  5.1 Introduction .......................................................... 85
  5.2 Problem Description .......................................................... 89
    5.2.1 First Phase: Discovering One-Triggering Causality Rules .......... 90
    5.2.2 Second Phase: Generating Multi-Triggering Causality Rules ......... 90
    5.2.3 An Illustrative Example for the Mining of Causality Rules .......... 91
  5.3 Algorithms for Mining One-Triggering Causality Rules ..................... 92
    5.3.1 Hierarchical Matching .......................................................... 94
    5.3.2 Algorithm $HM_C$: Candidate-Sets-Based Hierarchical Matching ...... 94
    5.3.3 Algorithm $HM_D$: Data-Sets-Based Hierarchical Matching .......... 97
    5.3.4 Algorithm $HM_A$: Adaptive Hierarchical Matching .................. 97
  5.4 Experimental Studies ........................................................ 98
    5.4.1 Generation of Synthetic Data ............................................. 98
    5.4.2 Relative Performance ......................................................... 99
    5.4.3 Scale-up .......................................................... 101
    5.4.4 Causality Rules Generation ............................................... 103
  5.5 Summary .......................................................... 104
List of Figures

1.1 Incremental mining for an ongoing time-variant transaction database. ............ 10
1.2 An illustrative transaction database and the corresponding item information ....... 14
1.3 An illustrative example of a Web site ......................................................... 18
1.4 A time-variant transaction database ............................................................ 19

2.1 Incremental mining for an ongoing time-variant transaction database. ........... 22
2.2 An illustrative transaction database ............................................................... 26
2.3 Large itemsets generation for the incremental mining with SWF .................... 28
2.4 Extensive analysis of various values of N and |L| ....................................... 36
2.5 Relative performance .................................................................................... 38
2.6 I/O cost performance .................................................................................... 39
2.7 Reduction on candidate itemsets when datasets $T_{10} - I_{4} - D_{100} - d_{10}$ and $T_{10} - I_{4} - D_{200} - d_{20}$ were used .................................................. 40
2.8 Scaleup performance of SWF ........................................................................ 41
2.9 Scaleup performance with the execution time ratio between SWF and FUP ...... 42

3.1 A time-variant transaction database ............................................................... 44
3.2 An illustrative transaction database and the corresponding item information ... 46
3.3 The flowchart of $PWM$ .................................................................................. 53
3.4 Frequent itemsets generation for mining weighted association rules by $PWM$ ... 54
3.5 The weighted association rule generation from frequent itemsets ................. 56
3.6 Relative performance studies ........................................................................ 57
3.7 I/O cost and CPU overhead performance ..................................................... 59
3.8 Scaleup performance of $PWM$ and the execution time ratio between $PWM$ and $Apriori^W$ 60

4.1 An illustrative publication database where different items may have different publication dates .............................................................. 63
4.2 An illustrative transaction database and the corresponding item information ... 63
4.3 Traversing the search space for existing algorithms, e.g., Apriori-like algorithms 70
4.4 The flowchart of $PPM$ .................................................................................. 75
4.5 Frequent temporal itemsets generation for mining general temporal association rules by $PPM$ ................................................................. 76
4.6 Relative performance studies ........................................................................ 82
4.7 I/O cost and CPU overhead performance ..................................................... 83
4.8 Scaleup performance of $PPM$ and the execution time ratio between $PPM$ and Apriori$^+$ 84

5.1 An illustrative example for the mining on a short transaction database ........... 87
5.2 The flow chart of mining causality rules ......................................................... 91
An example of generating large one-triggering event sequences with $\min \_\text{supp} = 40\%$.

$R^j$ sets of multi-triggering causality rules on mining the database in Figure 5.1 where $j \geq 2$.

The execution times of three hierarchical matching algorithms on $C^{10-T2.5-S4-I1.25}$ with $\min \_\text{supp} = 0.5\%$.

Relative performance of execution time.

Scale-up experiments for the number of customers.

Scale-up experiments for the number of items per customer.

Scale-up experiments for the number of items per large sequence.

Causality rules generation on the database $C^{10-T2.5-S4-I1.25}$.

6.1 An illustrative example of a Web site.

6.2 A time-variant transaction database.

6.3 The search space traversed by algorithm $UTM$.

6.4 The flowchart of Dynamic Threshold Miner.

6.5 An illustrative example with (a) structure of a Web site, (b) the example database, and (c) minimal supports of pages.

6.6 An example of $SeqGen_{C_2}$.

6.7 An example of $SeqGen_{C_k}$.

6.8 An example execution of $IDTM$.

6.9 Frequent reference sequences generation for the dynamic mining with $IDTM$.

6.10 An example using multiple hash tables.

6.11 Number of candidate and frequent reference sequence found.

6.12 The relative execution time of algorithms $DTM$ and $UTM$.

6.13 Scale-up experiments on algorithm $IDTM$.

## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Meanings of symbols used in the mining of SWF technique</td>
<td>29</td>
</tr>
<tr>
<td>2.2</td>
<td>Meanings of various parameters in the mining of SWF technique</td>
<td>35</td>
</tr>
<tr>
<td>4.1</td>
<td>Meanings of symbols used in the mining of general temporal association rules</td>
<td>69</td>
</tr>
<tr>
<td>4.2</td>
<td>Meanings of various parameters in the mining of general temporal association rules</td>
<td>81</td>
</tr>
<tr>
<td>5.1</td>
<td>Meanings of symbols used in the mining of causality rules</td>
<td>93</td>
</tr>
<tr>
<td>6.1</td>
<td>Meanings of symbols used in the Web mining with dynamic thresholds</td>
<td>119</td>
</tr>
<tr>
<td>6.2</td>
<td>Meaning of parameters for the synthetic data in the Web mining with dynamic thresholds</td>
<td>123</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation

Knowledge discovery and data mining have received much attention these last years. The approaches used for knowledge discovery are non-trivial and often domain specific, depending on the canonical mining primitives \[2, 15, 42, 47, 52, 78, 93, 97\]. Explicitly, Data Mining is a multidisciplinary field drawing work from areas including database technology, artificial intelligence, machine learning, neural networks, statistics, pattern recognition, knowledge-based systems, knowledge acquisition, information retrieval, high-performance computing, and data visualization \[3, 4, 44, 65, 76, 77, 96\]. The discovery of association relationship among a huge database has been known to be useful in selective marketing, decision analysis, and business management \[17, 22, 40\]. A popular area of applications is the market basket analysis, which studies the buying behaviors of customers by searching for sets of items that are frequently purchased either together or in sequence.

Let \[I = \{x_1, x_2, ..., x_m\}\] be a set of items. A set \(X \subseteq I\) with \(k = |X|\) is called a \(k\)-itemset or simply an itemset. Let a database \(D\) be a set of transactions, where each transaction \(T\) is a set of items such that \(T \subseteq I\). A transaction \(T\) is said to support \(X\) if and only if \(X \subseteq T\). Conventionally, an association rule is an implication of the form \(X \implies Y\), meaning that the presence of the set \(X\) implies the presence of another set \(Y\) where \(X \subseteq I, Y \subseteq I\) and \(X \cap Y = \emptyset\). The rule \(X \implies Y\) holds in the transaction set \(D\) with confidence \(c\) if \(c\%\) of transactions in \(D\) that contain \(X\) also contain \(Y\). The rule \(X \implies Y\) has support \(s\) in the transaction set \(D\) if \(s\%\) of transactions in \(D\) contain \(X \cup Y\).

For a given pair of confidence and support thresholds, the problem of mining association rules is to identify all association rules that have confidence and support greater than the corresponding minimum support threshold (denoted as \(\text{min \_ supp}\)) and minimum confidence threshold (denoted as \(\text{min \_ conf}\)). Association rule mining algorithms \[5\] work in two steps: (1) generate all frequent itemsets that satisfy \(\text{min \_ supp}\); (2) generate all association rules that satisfy \(\text{min \_ conf}\) using the frequent itemsets. This problem can be reduced to the problem of finding all frequent itemsets for the same support threshold.

Since the early work in \[5\], several efficient algorithms to mine association rules have been developed in recent years. These studies cover a broad spectrum of topics including: (1) fast algorithms based on the level-wise Apriori framework \[7, 18, 68\], partitioning \[53, 75\], and sampling \[83\]; (2) Graph-based \[51, 91\], TreeProjection \[1\] and FP-growth algorithms \[36, 37, 38, 70\]; (3) incremental updating \[11, 13, 26, 30, 49, 84, 82, 92\] and parallel algorithms \[6, 67\]; (4) mining of generalized and multi-level rules \[33, 79\]; (5) mining of quantitative rules \[80\]; (6) mining of multi-dimensional rules \[64, 88, 90\]; (7) constraint-based rule mining \[35, 46, 45, 69, 85\] and multiple minimum supports issues \[55, 87\]; (8) associations among correlated or infrequent items \[31, 41\]; and (9) temporal association rule discovery...
For better readability, some related works are reviewed below.

1.1.1 Apriori-like algorithms
Most of the previous studies, including those are [5, 19, 26, 27, 68, 79, 83], belong to Apriori-like approaches. Basically, an Apriori-like approach is based on an anti-monotone Apriori heuristic [5], i.e., if any itemset of length k is not frequent in the database, its length (k + 1) super-itemset will never be frequent. The essential idea is to iteratively generate the set of candidate itemsets of length (k + 1) from the set of frequent itemsets of length k (for k ≥ 1), and to check their corresponding occurrence frequencies in the database. As a result, if the largest frequent itemset is a j-itemset, then an Apriori-like algorithm may need to scan the database up to (j + 1) times.

1.1.2 Partition-based algorithms
The works in [53, 62, 75] are essentially based on a partition-based heuristic, i.e., if X is a frequent itemset in database D which is divided into n partitions p1, p2, ..., pn, then X must be a frequent itemset in at least one of the n partitions. The partition algorithm in [75] divides D into n partitions, and processes one partition in main memory at a time. The algorithm first scans partition pi, for i = 1 to n, to find the set of all local frequent itemsets in pi, denoted as Lpi. Then, by taking the union of Lpi for i = 1 to n, a set of candidate itemsets over D is constructed, denoted as CG. Based on the above partition-based heuristic, CG is a superset of the set of all frequent itemsets in D. Finally, the algorithm scans each partition for the second time to calculate the support of each itemset in CG and to find out which candidate itemsets are really frequent itemsets in D. Instead of constructing CG by taking the union of Lpi, for i = 1 to n, at the end of the first scan, some variations of the above partition algorithm are proposed in [53, 62]. In [62], algorithm SPINC constructs CG incrementally by adding Lpi to CG whenever Lpi is available. SPINC starts the counting of occurrences for each candidate itemset c ∈ CG as soon as c is added to CG. In [53], algorithm AS-CPA employs prior knowledge collected during the mining process to further reduce the number of candidate itemsets and to overcome the problem of data skew.

1.1.3 FUP-based algorithms
Since it is costly to find the association rules in large databases, incremental updating techniques are desirable in order to avoid redoing data mining on the whole updated database. Basically, similarly to that of Apriori, the framework of FUP, which can update the association rules in a database when new transactions are added to the database, contains a number of iterations [26, 27]. The candidate sets at each iteration are generated based on the frequent itemsets found in the previous iteration. The key steps of FUP are listed below, where Δ+ denotes the added portion of an ongoing transaction database. (1) At each iteration, the supports of the size-k frequent itemsets in L are updated against the increment Δ+ to filter out those that are no longer in the updated database. (2) While scanning the increment, a set of candidate sets, Ck, is extracted from the transactions in Δ+, together with their supports in Δ+ counted. The supports of these sets in Ck are then updated against the original database to find the “new” frequent itemsets. (3) Many sets in Ck can be pruned away by checking their supports in Δ+ before the update against the original database starts. (4) The size of the updated database is reduced at each iteration by pruning away a few items from some transactions in the updated database. The major idea is to reuse the information of the old frequent itemsets and to integrate the support information of the new frequent itemsets in order to substantially reduce the pool of candidate sets to be re-examined. An extension to FUP was reported in [27] and is referred to as FUP2. In essence,
FUP\textsubscript{2} is equivalent to FUP for the case of insertion, and is, however, a complementary algorithm of FUP for the case of deletion. Another FUP-based algorithm, call FUP\textsubscript{2H}, was also devised in [27] to utilize the hash technique for performance improvement.

1.1.4 FP-based algorithms

In essence, Frequent Pattern growth (FP-growth), which constructs a highly compact data structure (an FP-tree) to compress the original transaction database, is a method of mining frequent itemsets without candidate generation [36, 38]. Rather than employing the generate-and-test strategy of Apriori-like methods, it focuses on frequent pattern growth which avoids costly candidate generation, resulting in greater efficiency. The mining of the FP-tree proceeds as follows. Start from each frequent length-1 pattern (as an initial suffix pattern), construct its conditional pattern base (a “subdatabase” which consists of the set of prefix paths in the FP-tree co-occurring with the suffix pattern), then construct its (conditional) FP-tree, and perform mining recursively on such a tree. The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree [35, 36, 37, 38, 46, 69, 70, 85, 87]. However, in our observation, when the database is large, it is sometimes unrealistic to construct a main memory-based FP-tree. Significantly, FP-based algorithms do not have obvious extensions to deal with the problem of incremental temporal database, mentioned in this dissertation.

1.2 Overview of the Dissertation

While these are important results toward enabling the integration of association mining and fast searching algorithms, e.g., BFS and DFS which are classified in [40], we note that these mining methods cannot effectively be applied to the mining of a large incremental temporal database which is of increasing popularity recently. Specifically, some phenomena are observed when we take the issues of Incremental Updates, Weighted Transactions, Publication-like Items, Short Transactions, and Dynamic Thresholds into consideration.

1.2.1 Incremental Updates

It is noted that recent important applications have called for the need of incremental mining. This is due to the increasing use of the record-based databases whose data are being continuously added. Examples of such applications include Web log records, stock market data, grocery sales data, transactions in electronic commerce, and daily weather/traffic records, to name a few. In many applications, we would like to mine the transaction database for a fixed amount of most recent data (say, data in the last 12 months). That is, in the incremental mining, one has to not only include new data (i.e., data in the new month) into, but also remove the old data (i.e., data in the most obsolete month) from the mining process.

Consider the example transaction database in Figure 1.1. Note that $db^{i,j}$ is the part of the transaction database formed by a continuous region from partition $P_i$ to partition $P_j$. Suppose we have conducted the mining for the transaction database $db^{i,j}$. As time advances, we are given the new data of January of 2001, and are interested in conducting an incremental mining against the new data. Instead of taking all the past data into consideration, our interest is limited to mining the data in the last 12 months. As a result, the mining of the transaction database $db^{i+1,j+1}$ is called for. Note that since the underlying transaction database has been changed as time advances, some algorithms, such as Apriori, may have to resort to the regeneration of candidate itemsets for the determination of new frequent itemsets, which
is, however, very costly even if the incremental data subset is small. On the other hand, while FP-tree-based methods [36, 37, 38, 70] are shown to be efficient for small databases, it is expected that their deficiency of memory overhead due to the need of keeping a portion of database in memory, as indicated in [40], could become more severe in the presence of a large database upon which an incremental mining process is usually performed.

At the time of writing, there were not many results to explicitly address the problem of incremental mining except noted below. In [26], the FUP algorithm updates the association rules in a database when new transactions are added to the database. Algorithm FUP is based on the framework of Apriori and is designed to discover the new frequent itemsets iteratively. The idea is to store the counts of all the frequent itemsets found in a previous mining operation. Using these stored counts and examining the newly added transactions, the overall count of these candidate itemsets are then obtained by scanning the original database. An extension to the work in [26] was reported in [27] where the authors propose an algorithm FUP\(_2\) for updating the existing association rules when transactions are added to and deleted from the database. In essence, FUP\(_2\) is equivalent to FUP for the case of insertion, and is, however, a complementary algorithm of FUP for the case of deletion. It is shown in [27] that FUP\(_2\) outperforms Apriori algorithm which, without any provision for incremental mining, has to re-run the association rule mining algorithm on the whole updated database. Another FUP-based algorithm, call FUP\(_2\)\(\mathcal{H}\), was also devised in [27] to utilize the hash technique for performance improvement. Furthermore, the concept of negative borders in [82] and that of UWEP, i.e., update with early pruning, in [11] are utilized to enhance the efficiency of FUP-based algorithms.

However, as will be shown by our experimental results in Chapter 2, the above mentioned FUP-based algorithms tend to suffer from two inherent problems, namely (1) the occurrence of a potentially huge set of candidate itemsets, and (2) the need of multiple scans of database. First, consider the problem of a potentially huge set of candidate itemsets. Note that the FUP-based algorithms deal
with the combination of two sets of candidate itemsets which are independently generated, i.e., from the original data set and the incremental data subset. Since the set of candidate itemsets includes all the possible permutations of the elements, FUP-based algorithms may suffer from a very large set of candidate itemsets, especially from candidate 2-itemsets. As conformed by our experimental results, this problem becomes even more severe for FUP-based algorithms when the incremented portion of the incremental mining is large. More importantly, in many applications, one may encounter new itemsets in the incremented dataset. While adding some new products in the transaction database, FUP-based algorithms will need to resort to multiple scans of database. Specifically, in the presence of a new frequent itemset $L_k$ generated in the data subset, $k$ scans of the database are needed by FUP-based algorithms in the worst case. That is, the case of $k = 8$ means that the database has to be scanned 8 times, which is very costly, especially in terms of I/O cost. As will become clear later, the problem of a large set of candidate itemsets will hinder an effective use of the scan reduction technique [68] by an FUP-based algorithm.

To remedy these problems, we shall devise in Chapter 2 an algorithm based on sliding-window filtering (abbreviated as SWF) for incremental mining of association rules. In essence, by partitioning a transaction database into several partitions, algorithm SWF employs a filtering threshold in each partition to deal with the candidate itemset generation. For ease of exposition, the processing of a partition is termed a phase of processing. Under SWF, the cumulative information in the prior phases is selectively carried over toward the generation of candidate itemsets in the subsequent phases. After the processing of a phase, algorithm SWF outputs a cumulative filter, denoted by $CF$, which consists of a progressive candidate set of itemsets, their occurrence counts and the corresponding partial support required. As will be seen, the cumulative filter produced in each processing phase constitutes the key component to realize the incremental mining. It will be seen that algorithm SWF proposed has several important advantages. First, with employing the prior knowledge in the previous phase, SWF is able to reduce the amount of candidate itemsets efficiently which in turn reduces the CPU and memory overhead. The second advantage of SWF is that owing to the small number of candidate sets generated, the scan reduction technique [68] can be applied efficiently. As a result, only one scan of the ongoing time-variant database is required. As will be validated by our experimental results, this very advantage of SWF enables SWF to significantly outperform FUP-based algorithms. The third advantage of SWF over FUP-based algorithms is the capability of SWF to avoid the data skew in nature. As mentioned in [53, 75], such instances as severe whether conditions may cause the sales of some items to increase rapidly within a short period of time. Data skew may cause FUP-based algorithms to generate many false candidate itemsets. In contrast, the performance of SWF will be less affected by the data skew since SWF employs the cumulative information for pruning false candidate itemsets in the early stage.

### 1.2.2 Weighted Transactions

In addition, note that the data mining process may produce thousands of rules, many of which are uninteresting to users. Hence, several applications have called for the use of constraint-based rule mining [10, 35, 45, 46, 69, 85, 89]. Specifically, in constraint-based mining, mining is performed under the guidance of various of constraints provided by the user. The constraints addressed in the prior works include the following: (1) Knowledge type-constraints [71, 85]; (2) Data constraints [10]; (3) Dimension/level constraints [35]; (4) Interestingness constraints [45, 46]; and (5) Rule constraints [69, 85, 89]. Such constraints may be expressed as meta-rules (rule templates), as the maximum or minimum number of predicates that can occur in the rule antecedent or consequent, or as relationships among attributes, attribute values, and/or aggregates. Recently, many constraint-based mining works have focused on the use of rule constraints. This form of constraint-based mining allows users of specifying
the rules to be mined according to their need, thereby leading to much more useful mining results. According to our observation, these kinds of rule-constraint problems are based on the concept of embedding a variety of item-constraints in the mining process.

On the other hand, a time-variant database, as shown in Figure 1.1, consists of values or events varying with time. Time-variant databases are popular in many applications, such as daily fluctuations of a stock market, traces of a dynamic production process, scientific experiments, medical treatments, weather records, to name a few. In our opinion, the existing model of the constraint-based association rule mining is not able to efficiently handle the time-variant database due to two fundamental problems, i.e., (1) lack of consideration of the exhibition period of each individual transaction; (2) lack of an intelligent support counting basis for each item. Note that the traditional mining process treats transactions in different time periods indifferently and handles them along the same procedure. However, since different transactions have different exhibition periods in a time-variant database, only considering the occurrence count of each item might not lead to interesting mining results.

For instance, when we try to analyze the past transaction data of computer and communication stores, such as sales of mobile phones and computers, we might discover little valuable information on customer buying behavior by the conventional mining methodologies. That is because that lots of mining results could be obsolete. For example, we may find such a rule as people who buying Pentium-II computers will also like to have 4GB hard disks and 32M DRAMs. Even though both of them have sufficient support and confidence from the mining processing of transaction database, they are not the selling items in this store anymore.

While most previous works are important results toward enabling the integration of association mining and fast searching algorithms, we note that these mining methods cannot effectively be applied to the problem of time-constraint mining on a time-variant database which is of increasing importance. Note that one straightforward approach to addressing the above issues is to employ the item-constraints [35, 45, 46, 69, 85, 89] and/or multiple supports strategies [55, 87], i.e., new coming items have higher weights for their item occurrences. However, as noted in [55, 87] these approaches will encounter another problem, i.e., there is no proper confidence threshold in such cases for the corresponding rule generation.

To remedy this, in Chapter 3, we broaden the horizon of frequent pattern mining by introducing a weighted model of transaction-weighted association rules (abbreviated as weighted association rules) in a time-variant database. Specifically, we propose an efficient Progressive Weighted Miner (abbreviately as PWM) algorithm to perform the mining for this problem as well as conduct the corresponding performance studies. In algorithm PWM, the importance of each transaction period is first reflected by a proper weight assigned by the user. Then, PWM partitions the time-variant database in light of weighted periods of transactions and performs weighted mining. Explicitly, algorithm PWM explores the mining of weighted association rules, denoted by \((X \Rightarrow Y)^W\), which is produced by two newly defined concepts of weighted – support and weighted – confidence in light of the corresponding weights in individual transactions. Basically, an association rule \(X \Rightarrow Y\) is termed to be a frequent weighted association rule \((X \Rightarrow Y)^W\) if and only if its weighted support is larger than minimum support required, i.e., \(\text{supp}^W(X \cup Y) > \text{min_supp}\), and the weighted confidence \(\text{conf}^W(X \Rightarrow Y)\) is larger than minimum confidence needed, i.e., \(\text{conf}^W(X \Rightarrow Y) > \text{min_conf}\). Instead of using the traditional support threshold \(\text{min}_S^T = \lfloor|D| \times \text{min_supp}\rfloor\) as a minimum support threshold for each item, a weighted minimum support, denoted by \(\text{min}_S^W = \lfloor\Sigma P_i \times W(P_i)\rfloor \times \text{min_supp}\), is employed for the mining of weighted association rules, where \(|P_i|\) and \(W(P_i)\) represent the amount of partial transactions and their corresponding weight values by a weighting function \(W(\cdot)\) in the weighted period \(P_i\) of the database \(D\). Let \(N_{P_i}(X)\) be the number of transactions in partition \(P_i\) that contain itemset \(X\). The support value of an itemset \(X\) can then be formulated as \(S^W(X) = \Sigma N_{P_i}(X) \times W(P_i)\). As a result, the weighted support
ratio of an itemset $X$ is $supp_W(X) = \frac{S_W(X)}{\sum_{Y \in D} S_W(Y)}$.

Explicitly, PWM first partitions the transaction database in light of weighted periods of transactions and then progressively accumulates the occurrence count of each candidate 2-itemset based on the intrinsic partitioning characteristics. With this design, algorithm PWM is able to efficiently produce weighted association rules for applications where different time periods are assigned with different weights. Algorithm PWM is also designed to employ a filtering threshold in each partition to early prune out those cumulatively infrequent 2-itemsets. The feature that the number of candidate 2-itemsets generated by PWM is very close to the actual number of frequent 2-itemsets allows us of employing the scan reduction technique by generating $C_k$s from $C_2$ directly to effectively reduce the number of database scans. Experimental results show that PWM produces a significantly smaller amount of candidate 2-itemsets than $Apriori^W$, i.e., an extended version of $Apriori$ algorithm. In fact, the number of the candidate itemsets $C_k$s generated by PWM approaches to its theoretical minimum, i.e., the number of actual frequent $k$-itemsets, as the value of the minimal support increases. Specifically, the execution time of PWM is, in orders of magnitude, smaller than those required by $Apriori^W$. Sensitivity analysis on various parameters of the database is also conducted to provide many insights into algorithm PWM.

Note that the problem of mining weighted association rules will be degenerated to the traditional one of mining association rules explored in previous works if the weighting function is assigned to be $W(\cdot) = 1$, meaning that the model we consider can be viewed as a general framework of prior studies. In Chapter 3, we not only explore the new model of weighted association rules in a time-variant database, but also propose an efficient Progressive Weighted Miner methodology to perform the mining for this problem as well as conduct the corresponding performance studies.

1.2.3 Publication-like Items

Further, we note that these traditional mining methods cannot effectively be applied to the mining of a publication-like database which is of increasing popularity recently. In essence, a publication database is a set of transactions where each transaction $T$ is a set of items of which each item contains an individual exhibition period. The current model of association rule mining is not able to handle the publication database due to the following fundamental problems, i.e., (1) lack of consideration of the exhibition period of each individual item; (2) lack of an equitable support counting basis for each item. Note that the traditional mining process takes the same task-relevant tuples, i.e., the size of transaction set $D$, as a counting basis. Recall that the task of support specification is to specify the minimum transaction support for each itemset. However, since different items have different exhibition periods in a publication database, only considering the occurrence count of each item might not lead to a fair measurement.

For example, in a bookstore transaction database as shown in Figure 1.2, the minimum transaction support and confidence are assumed to be $min\_supp = 30\%$ and $min\_conf = 75\%$, respectively. A set of time series database indicates the transaction records from January 2001 to March 2001. The publication date of each transaction item is also given. Based on the traditional mining techniques, the absolute support threshold is denoted as $S^A = [12 \times 0.3] = 4$ where 12 is the size of transaction set $D$. It can be seen that only $\{B, C, D, E, BC\}$ can be termed as frequent itemsets since the amounts of their occurrences in this transaction database are respectively larger than the absolute value of support threshold. Thus, only rule $C \implies B$ is termed as a frequent association rule with support $s = 41.67\%$ and confidence $c = 83.33\%$. However, some phenomena are observed when we take the "item information" in Figure 1.2 into consideration.

1. An early publication intrinsically possesses a higher likelihood to be determined as a frequent itemset. For example, the sales volume of an early product, such as $A, B, C$ or $D$,
is likely to be larger than that of a newly exhibited product, e.g., $E$ or $F$, since an early product has a longer exhibition period. As a result, the association rules we usually get will be those with long-term products such as “milk and bread are frequently purchased together”, which, while being correct by the definition, is of less interest to us in the association rule mining. In contrast, some more recent products, such as new books, which are really “frequent” and interesting in their exhibition periods are less likely to be identified as frequent ones if a traditional mining process is employed.

2. Some discovered rules may be expired from users’ interest. Considering the generated rule $C \Rightarrow B$, both $B$ and $C$ were published from the very early dates of this mining transaction database. This information is very likely to have been explored in the previous mining database, such as the one from January 1996 to December 1997. Such mining results could be of less interest to our on-going mining works. For example, most researchers tend to pay more attention to the latest published papers.

Note that one straightforward approach to addressing the above issues is to lower the value of the minimum support threshold required. However, this naive approach will cause another problem, i.e., those interesting rules with smaller supports may be overshadowed by lots of less important information with higher supports. As a consequence, we introduce the notion of exhibition period for each transaction item in Chapter 4 and develop an algorithm, Progressive Partition Miner (abbreviatedly as PPM), to address this problem. It is worth mentioning that the application domain of this study is not limited to the mining of a publication database. Other application domains include bookstore transaction databases, video and audio rental store records, stock market data, and transactions in electronic commerce, to name a few.
Explicitly, we explore in Chapter 4 the mining of general temporal association rules, i.e., \((X \Rightarrow Y)^{t,n}\), where \(t\) is the latest-exhibition-start time of both itemsets \(X\) and \(Y\), and \(n\) denotes the end time of the publication database. In other words, \((t, n)\) is the maximal common exhibition period of itemsets \(X\) and \(Y\). An association rule \(X \Rightarrow Y\) is termed to be a frequent general temporal association rule \((X \Rightarrow Y)^{t,n}\) if and only if its probability is larger than minimum support required, i.e., \(P(X^{t,n} \cup Y^{t,n}) > \text{min\_supp}\), and the conditional probability \(P(Y^{t,n} | X^{t,n})\) is larger than minimum confidence needed, i.e., \(P(Y^{t,n} | X^{t,n}) > \text{min\_conf}\). Instead of using the absolute support threshold \(S^A = |\{D|* \text{min\_supp}\}\) as a minimum support threshold for each item in Figure 1.2, a relative minimum support, denoted by \(S^{R}_X = |\{D_X|* \text{min\_supp}\}\) where \(|D_X|\) indicates the amount of partial transactions in the exhibition period of itemset \(X\), is given to deal with the mining of temporal association rules.

To deal with the mining of general temporal association rule \((X \Rightarrow Y)^{t,n}\), an efficient algorithm, Progressive Partition Miner, is devised. The basic idea of PPM is to first partition the publication database in light of exhibition periods of items and then progressively accumulate the occurrence count of each candidate 2-itemset based on the intrinsic partitioning characteristics. Algorithm PPM is also designed to employ a filtering threshold in each partition to early prune out those cumulatively infrequent 2-itemsets. The feature that the number of candidate 2-itemsets generated by PPM is very close to the number of frequent 2-itemsets allows us to employ the scan reduction technique by generating \(C_k\)'s from \(C_2\) directly to effectively reduce the number of database scan. Experimental results show that PPM produces a significantly smaller amount of candidate 2-itemsets than Apriori\(^+\), i.e., an extended version of Apriori algorithm. In fact, the number of the candidate itemsets \(C_k\)'s generated by PPM approaches to its theoretical minimum, i.e., the number of frequent \(k\)-itemsets, as the value of the minimal support increases. Explicitly, the execution time of PPM is, in orders of magnitude, smaller than those required by Apriori\(^+\). Sensitivity analysis on various parameters of the database is also conducted to provide many insights into algorithm PPM. The advantage of PPM over Apriori\(^+\) becomes even more prominent as the size of the database increases. This is indeed an important feature for PPM to be practically used for the mining of a time series database in the real world.

It is worth mentioning that the problem of mining general temporal association rules will be degenerated to the one of mining temporal association rules explored in prior works [9, 14, 24, 25, 81] if the exhibition period \((t, n)\) of association rule \((X \Rightarrow Y)^{t,n}\) is applied to a none-maximal exhibition period of \(X \Rightarrow Y\), such as \((j, n)\) where \(j > t\). Consider for example the database in Figure 1.2 where \((C \Rightarrow B)^{1,3}\) and \((C \Rightarrow E)^{2,3}\) are two general temporal association rules in database \(D\) while the temporal subset of \((C \Rightarrow B)^{1,3}\), e.g., \((C \Rightarrow B)^{2,3}\), can also be a temporal association rule as defined before [9, 14, 24, 25, 81], showing that the model we consider can be viewed as a general framework of prior studies. This is the very reason we use the term “general temporal association rule”.

1.2.4 Short Transactions

On the other hand, while the discovery of association relationship among the data in a huge database has been known to be useful in selective marketing, decision analysis, and business management [22, 40], it is noted that the existing models of rule mining might not be able to discover user preferred frequent patterns efficiently due to the following fundamental problems.

1. The puzzle of mining association rules on a short transaction database: Consider the knowledge discovery in a transaction database of a convenience store where customers usually visit frequently and the number of items purchased in each transaction is usually small. Such a database is then composed of short transactions. Previous mining works in association rules, however, do not fully explore the inter-transaction relationship, and are thus apt to provide the
limited knowledge in the sales patterns (as discovered from intra-transactions). Note that such purchasing scenarios for short transactions also occur in electronic commerce purchasing records, pharmacy purchasing databases, bookstore transaction records, and so on, thereby unavoidably reducing the usefulness of rule mining in these applications.

2. **Lack of long patterns for sequential pattern mining:** On the other hand, due to the imposition of a strict order, e.g., people would buy $B$ after the purchase of $A$, followed by shopping $C$, the mining of sequential patterns tends to suffer from the drawback of having very low supports in long sequential patterns [8, 37, 70]. Otherwise, rules in long sequences will rarely be discovered. For instance, the occurrence probability of a seven-item sequential pattern could be in proportion to $p^6$ where $p$ is the probability of a certain product being bought after a given one. In fact, such a strict order in the sequential pattern mining might not be justifiable in some real applications, since after the purchase of a TV, one may have less interest in the exact subsequent buying order of products, say, TV set, sofa, end table, lamp, carpet, coffee table, than in exploring what set of products in general would be inspired for subsequent purchases.

Consequently, we explore the mining of **causality rules** with the **triggering** and **consequential events** for a database of short transactions. Such a short transaction database is, in our opinion, common in many real applications. Explicitly, we shall conduct in Chapter 5 the mining of causality rules from a transaction database, where each event may belong to multiple categories and the causality rule consists of (a) a sequence of **triggering events** and (b) a set of **consequential events**. The causality rule mining capability can be applied to various applications. For example, one can improve electronic commerce applications by first identifying consumer telephone calling patterns and consumer buying patterns, and then using the information discovered to attain more effective on-line advertising and offerings to the consumers. Specifically, transaction patterns can be derived from collected data by observing the customer behavior in terms of **cause** and **effect**, or what will be referred to as triggering and consequential events. In Chapter 5, the term “causality rule(s)”, denoted by $X \rightarrow Y$, will refer to a rule of describing certain customer behavior where some triggering events, i.e., $X$, lead to a set of consequential events, i.e., $Y$. This problem can be further described by the example below.

Note that no specific order is assumed among the triggering/consequential events. In essence, the problem of mining causality rules can be mapped from an event sequence database into a problem of counting large event sets. As such, we decompose the problem of mining causality rules into two phases, i.e., the phase of discovering one-triggering causality rules and the phase of generating multi-triggering causality rules:

1. **In the phase of discovering one-triggering causality rules**, we use iterative approaches to deriving one-triggering event causality rules which contain only single triggering events. To count the occurrences in the event sequence database of each candidate $k$-event rule in $C_k$, it is necessary to scan through the sequence database to do a sub-sequence matching. This procedure is very costly in the presence of a huge number of candidate sets and a large event database, and in our opinion, cannot be dealt with by direct extensions from existing rule mining methods, including GSP [8], FP-tree [36], Free-Span [38], PrefixSpan [70], episode mining algorithms [58, 59, 60], and so on. Hence, instead of comparing each candidate rule in $C_k$ directly with the event sequence in the database, the event sequence is transformed into a simple event sequence based on the concept of **hierarchical sub-sequence matching**. With the **hierarchical matching** methodologies, the detection of an occurrence of a causality rule in an event sequence can be greatly facilitated.

2. **In the phase of generating multi-triggering causality rules**, newly identified one-triggering event causality rules are used to generate the next set of candidate rules to be evaluated, by
increasing either (1) the size of the set of consequential events triggered by triggering events or (2) the number of triggering events. For example, the causality rule of “A and B triggering C” can hold true only if both rules “A triggers C” and “B triggers C” hold.

Since the corresponding causality rules can be derived in a straightforward manner in the phase of generating multi-triggering causality rule, the overall performance of mining causality rules is in fact determined by the first phase, i.e., the phase of discovering one-triggering causality rules. To minimize the corresponding computational cost in the first phase, we develop, in light of the concept of hierarchical matching, three algorithms, namely candidate-sets-based hierarchical matching (referred to as algorithm $HM_C$), data-sets-based hierarchical matching (referred to as algorithm $HM_D$) and adaptive hierarchical matching (referred to as algorithm $HM_A$), to explore the mining of causality rules. Extensive experiments are performed to assess the performance of the proposed algorithms. Sensitivity analysis on various parameters of the event database is also conducted to provide many insights into algorithms proposed. According to the experimental results, it is shown that algorithm $HM_C$ is effective in generating the higher order large event sets whereas algorithm $HM_D$ is good at dealing with the huge numbers of the lower order event sets. The adaptive matching algorithm $HM_A$ is shown to outperform algorithms $HM_C$ and $HM_D$, by adaptively employing matching techniques of algorithms $HM_C$ and $HM_D$. These experimental results conform with the complexity analysis of algorithms proposed. Scale-up experiments show that all three algorithms scale linearly with the number of customer transactions. They also have good scale-up properties with respect to the number of transactions per customer and the number of items in a transaction.

We mention in passing that association rules deal with intra-transaction information, in which the events have occurred effectively simultaneously, with no regard for cause and effect or trigger and consequence. Causality rules, in contrast, deal with inter-transaction behavior explicitly. The triggering event must occur earlier in time than all of the consequential events where no specific order is assumed among the triggering/consequential events. On the other hand, sequential events are inter-transaction events which are necessarily ordered in time [8, 37, 70], such that a second event always follows the first, and a third event follows the second, but the third would never directly follow the first. Further, the works in [58, 59, 60] consider frequent episodes discovered from a long event sequence such as the one composed of signals in a telecommunication database. With the use of a moving window, the episode mining algorithms explore the temporal relationship (parallel or sequence) of signals in individual long transactions. The inter-transaction behavior is, however, not addressed. It is worth mentioning that since the causality relationship can only be captured by the mining on non-sequential, inter-transaction information across multiple categories, the causality rule can be viewed as a more general framework than those in prior studies.

1.2.5 Dynamic Thresholds

Moreover, with the fast increase in Web activities, Web data mining has recently become an important research topic and is receiving an increasing amount of research interest from both academic and industrial environments [16, 28, 63, 72]. Among others, mining of path traversal patterns plays an essential role in the Web mining [20, 21]. Several prototypes and implemented systems are using mining techniques on path traversal patterns to explain aspects of behavior associated with the implicit time-variant nature of page viewers. In essence, log data which are collected by Web servers contain information about user access to the Web documents of the site. The size of logs increases rapidly due to two reasons: the rate that data are collected and the increase in the number of Web sites themselves. The analysis of these large volumes of log data requires the employment of data mining methods. Following
the paradigm of mining association rules [7], mined patterns are those access sequences of frequent occurrences. An example of this kind of pattern is a sequence $<D_1, ..., D_n>$ of visited pages in a Web site. If such a sequence appears frequently enough, then this sequence indicates a frequent traversal pattern. Understanding user access patterns in such a Web environment will not only help improve the Web site design but also be able to lead to better marketing decisions.

While existing methods are efficient for the mining of frequent path traversal patterns from the access information contained in a log file, these approaches are likely to over evaluate associations. Explicitly, most previous studies of path traversal pattern mining are based on the model of a uniform support threshold, where a single support threshold is used to determine frequent traversal patterns without taking into consideration such important factors as the length of the pattern, the positions of Web pages, and the importance of a particular pattern, etc. As a result, a low support threshold will lead to lots of uninteresting patterns derived while a high support threshold may cause some interesting patterns with lower supports to be ignored. Hence, different support thresholds are deemed necessary for Web pages at different levels of Web sites. More specifically, we have the following observations to justify the model of mining traversal patterns with different support thresholds considered in Chapter 6.

1. A Web page at lower level of a Web site, e.g., the Web page $H$ as shown in Figure 1.3, will naturally have a lower occurrence frequency than their corresponding higher level concepts, e.g., the portal Web page $A$. Thus, if the minimum support is set to too high, those patterns that involve Web pages at lower levels of a Web site will not be found. On the other hand, if the minimum support is set too low, a lot of uninteresting patterns will be produced. For example, consider a Web site shown in Figure 1.3 and a database that contains four sequences: $ABC$, $AB$, $AO$, $ABEGH$. Suppose the minimum support (abbreviated as $min\_sup$) is $min\_sup = 2$. Web pages $A$ and $B$ at higher levels of Web site are deemed frequent in this case, which might, however, be due more to their locations than to their contents.

2. The access design of Web pages is not the same. A single-linked Web page usually occurs less frequently than a multiple-linked Web page. An example of a multiple-linked Web page is a logo page of a company that is included in many Web documents. It is therefore undesirable to use
the same support threshold for mining both single-linked and multiple-linked patterns.

3. The importance of Web pages is not the same. A free information Web page tends to attract more attention than Web pages designed for sales transactions. For Web sites owners, however, analyzing the access behavior of sales Web pages may be more important than that of free information Web pages.

4. The natural occurrence frequencies of items often vary greatly in real world [55, 87]. For instance, people access more Web pages of popular musics than those of classical musics. It is natural to have different support constraints on different groups of data items, or to set certain constraints on items of particular interest.

A naive way to handle the non-uniform supports is to apply existing algorithms with a very small support threshold and filter the results using higher minimum supports. This method is referred to as Uniform Threshold Miner (abbreviated as UTM) in Chapter 6. As will be validated by our experimental results later, UTM suffers from the drawback that many candidates have to be generated and later discarded, which implies not only knowledge of low interest produced but also excessive execution time involved.

Consequently, this thesis broadens the horizon of frequent path traversal pattern mining by introducing a flexible model of mining Web traversal patterns with dynamic thresholds. Specifically, we explore a new data mining capability which involves mining path traversal patterns with the concept of dynamic thresholds in a time-variant Web environment. Such a time-variant database is very popular in many applications, including daily fluctuations of a stock market, traces of a dynamic production process, scientific experiments, medical treatments, Web log data, weather records, to name a few. By properly employing some effective techniques devised for joining reference sequences, the proposed algorithm DTM (standing for Dynamic Threshold Miner) not only possesses the capability of mining with dynamic thresholds, but also significantly improves the execution efficiency of mining Web traversal patterns. In addition, an innovative hybrid hash method with multiple hash tables is designed as an efficient technique for the mining with dynamic support thresholds.

Note that time advances, one has to include new data (e.g., data in Oct 2001 as shown in Figure 1.4) and \( D' = D + D^+ \) for mining. This scenario calls for the incremental mining capability. Consequently, an incremental version of DTM (referred to as incremental DTM or IDTM) is also developed. It is
noted that since the occurrence frequency, i.e., $frequency(p, t)$, of Web page $p$ might be variant as the time, i.e., $t$, advances, the support threshold, e.g., $min\_sup(p, t_i)$, of Web page $p$ at the time point $t_i$ might be different from the one, e.g., $min\_sup(p, t_j)$, at another time point $t_j$. This is the very reason we use the term “dynamic support threshold” in this thesis. Further, it is worth mentioning that the problem of mining Web path traversal patterns with dynamic support threshold will be degenerated to the traditional one of mining Web path traversal patterns explored in previous works if the minimum support threshold function $min\_sup(\cdot)$ is assigned to be a uniform support threshold $min\_sup$, meaning that the model we consider can be viewed as a general framework of prior studies. Performance of algorithm $DTM$ and the extension of existing methods is comparatively analyzed. It is shown that the option of algorithm $DTM$ is very advantageous and leads to prominent performance improvement. Also, algorithm $IDTM$ is shown to possess very good scalability as the data size increases. Sensitivity analysis on various parameters is conducted.

It is noted that there are some studies conducted on the incorporation of relatively flexible models in frequent pattern mining, including mining association rules by incorporation of multi-level (or taxonomy) concepts [33], mining of frequent patterns adaptive to user-specified support constraints [55, 87] and mining of frequent patterns in multi-dimensional circumstances [88]. Recently, the work in [73] developed some FP-tree based algorithms to deal with the flexible support constraints on mining multi-dimensional frequent patterns. However, these works were mainly conducted to deal with individual issues and are not designed to deal with incremental mining, thus not providing a general framework for the mining with dynamic thresholds in a Web environment.

1.3 Organization of the Dissertation

The rest of this dissertation is organized as follows. We examine in Chapter 2 sliding window filtering scheme to improve the efficiency of incremental mining on a time-variant database. In Chapter 3, we present a new data mining algorithm which involves an efficient method for progressive weighted mining on a time-variant database. In addition, in view of the asymmetric features, we devise progressive partition miner for mining general temporal association rules in Chapter 4. In Chapter 5, we explore the issue of mining causality rules which exploring the relationship between triggering and consequential events in a database of short transactions. Furthermore, we propose an efficient algorithm, namely dynamic threshold miner, for mining web traversal patterns with dynamic thresholds in Chapter 6. This dissertation concludes with Chapter 7.
Chapter 2

Sliding Window Filtering: An Efficient Method for Incremental Mining on a Time-Variant Database

2.1 Introduction

Due to the increasing use of computing for various applications, the importance of data mining is growing at rapid pace recently. It is noted that analysis of past transaction data can provide very valuable information on customer buying behavior, and thus improve the quality of business decisions. In essence, it is necessary to collect and analyze a sufficient amount of sales data before any meaningful conclusion can be drawn therefrom. Since the amount of these processed data tends to be huge, it is important to devise efficient algorithms to conduct mining on these data. Various data mining capabilities have been explored in the literature [5, 8, 9, 24, 22, 23, 32, 64, 66, 88, 90]. Among them, the one receiving a significant amount of research attention is on mining association rules over basket data [5, 7, 29, 31, 33, 38, 56, 54, 55, 61, 68, 74, 75, 87, 94, 95]. For example, given a database of sales transactions, it is desirable to discover all associations among items such that the presence of some items in a transaction will imply the presence of other items in the same transaction, e.g., 90% of customers that purchase milk and bread also purchase eggs at the same time.

Mining association rules was first introduced in [5], where it was shown that the problem of mining association rules is composed of the following two subproblems: (1) discovering the frequent itemsets, i.e., all sets of itemsets that have transaction support above a pre-determined minimum support $s$, and (2) using the frequent itemsets to generate the association rules for the database. The overall performance of mining association rules is in fact determined by the first subproblem. After the frequent itemsets are identified, the corresponding association rules can be derived in a straightforward manner [5]. Among others, Apriori [5], DHP [68], and partition-based ones [53, 75] are proposed to solve the first subproblem efficiently. In addition, several novel mining techniques, including TreeProjection [1], FP-tree [36, 37, 38, 70], and constraint-based ones [35, 46, 69, 85, 87] also received a significant amount of research attention.

In addition, it is noted that recent important applications have called for the need of incremental mining. This is due to the increasing use of the record-based databases whose data are being continuously added. Examples of such applications include Web log records, stock market data, grocery sales data, transactions in electronic commerce, and daily weather/traffic records, to name a few. In many applications, we would like to mine the transaction database for a fixed amount of most recent data
(say, data in the last 12 months). That is, in the incremental mining, one has to not only include new data (i.e., data in the new month) into, but also remove the old data (i.e., data in the most obsolete month) from the mining process.

Consider the example transaction database in Figure 2.1. Note that $db^{i,j}$ is the part of the transaction database formed by a continuous region from partition $P_i$ to partition $P_j$. Suppose we have conducted the mining for the transaction database $db^{i,j}$. As time advances, we are given the new data of January of 2001, and are interested in conducting an incremental mining against the new data. Instead of taking all the past data into consideration, our interest is limited to mining the data in the last 12 months. As a result, the mining of the transaction database $db^{i+1,j+1}$ is called for. Note that since the underlying transaction database has been changed as time advances, some algorithms, such as Apriori, may have to resort to the regeneration of candidate itemsets for the determination of new frequent itemsets, which is, however, very costly even if the incremental data subset is small. On the other hand, while FP-tree-based methods [36, 37, 38, 70] are shown to be efficient for small databases, it is expected that their efficiency of memory overhead due to the need of keeping a portion of database in memory, as indicated in [40], could become more severe in the presence of a large database upon which an incremental mining process is usually performed.

To the best of our knowledge, there is little progress made thus far to explicitly address the problem of incremental mining except noted below. In [26], the FUP algorithm updates the association rules in a database when new transactions are added to the database. Algorithm FUP is based on the framework of Apriori and is designed to discover the new frequent itemsets iteratively. The idea is to store the counts of all the frequent itemsets found in a previous mining operation. Using these stored counts and examining the newly added transactions, the overall count of these candidate itemsets are then obtained by scanning the original database. An extension to the work in [26] was reported in [27] where the authors propose an algorithm FUP$_2$ for updating the existing association rules when transactions
are added to and deleted from the database. In essence, FUP$_2$ is equivalent to FUP for the case of insertion, and is, however, a complementary algorithm of FUP for the case of deletion. It is shown in [27] that FUP$_2$ outperforms Apriori algorithm which, without any provision for incremental mining, has to re-run the association rule mining algorithm on the whole updated database. Another FUP-based algorithm, call FUP$_2$H, was also devised in [27] to utilize the hash technique for performance improvement. Furthermore, the concept of negative borders in [82] and that of UWEP, i.e., update with early pruning, in [11] are utilized to enhance the efficiency of FUP-based algorithms.

However, as will be shown by our experimental results, the above mentioned FUP-based algorithms tend to suffer from two inherent problems, namely (1) the occurrence of a potentially huge set of candidate itemsets, and (2) the need of multiple scans of database. First, consider the problem of a potentially huge set of candidate itemsets. Note that the FUP-based algorithms deal with the combination of two sets of candidate itemsets which are independently generated, i.e., from the original data set and the incremental data subset. Since the set of candidate itemsets includes all the possible permutations of the elements, FUP-based algorithms may suffer from a very large set of candidate itemsets, especially from candidate 2-itemsets. As conformed by our experimental results, this problem becomes even more severe for FUP-based algorithms when the incremented portion of the incremental mining is large. More importantly, in many applications, one may encounter new itemsets in the incremented dataset. While adding some new products in the transaction database, FUP-based algorithms will need to resort to multiple scans of database. Specifically, in the presence of a new frequent itemset $L_k$ generated in the data subset, $k$ scans of the database are needed by FUP-based algorithms in the worst case. That is, the case of $k = 8$ means that the database has to be scanned 8 times, which is very costly, especially in terms of I/O cost. As will become clear later, the problem of a large set of candidate itemsets will hinder an effective use of the scan reduction technique [68] by an FUP-based algorithm.

To remedy these problems, we shall devise in this chapter an algorithm based on sliding-window filtering (abbreviated as SWF) for incremental mining of association rules. In essence, by partitioning a transaction database into several partitions, algorithm SWF employs a filtering threshold in each partition to deal with the candidate itemset generation. For ease of exposition, the processing of a partition is termed a phase of processing. Under SWF, the cumulative information in the prior phases is selectively carried over toward the generation of candidate itemsets in the subsequent phases. After the processing of a phase, algorithm SWF outputs a cumulative filter, denoted by $CF$, which consists of a progressive candidate set of itemsets, their occurrence counts and the corresponding partial support required. As will be seen, the cumulative filter produced in each processing phase constitutes the key component to realize the incremental mining. An illustrative example for the operations of SWF is presented in Section 2.3.1, a detailed description of algorithm SWF is given in Section 2.3.2 and the correctness of algorithm SWF is proved in Section 2.3.3. It will be seen that algorithm SWF proposed has several important advantages. First, with employing the prior knowledge in the previous phase, SWF is able to reduce the amount of candidate itemsets efficiently which in turn reduces the CPU and memory overhead. The second advantage of SWF is that owing to the small number of candidate sets generated, the scan reduction technique [68] can be applied efficiently. As a result, only one scan of the ongoing time-variant database is required. As will be validated by our experimental results, this very advantage of SWF enables SWF to significantly outperform FUP-based algorithms. The third advantage of SWF over FUP-based algorithms is the capability of SWF to avoid the data skew in nature. As mentioned in [53, 75], such instances as severe whether conditions may cause the sales of some items to increase rapidly within a short period of time. Data skew may cause FUP-based algorithms to generate many false candidate itemsets. In contrast, the performance of SWF will be less affected by the data skew since SWF employs the cumulative information for pruning false candidate itemsets in the early stage.
Extensive experiments are performed to assess the performance of SWF. As shown in the experimental results, SWF produces a significantly smaller amount of candidate 2-itemsets than FUP-based algorithms. In fact, the number of the candidate itemsets $C_{48}$ generated by SWF approaches to its theoretical minimum, i.e., the number of frequent k-itemsets, as the value of the minimal support increases. It is shown by our experiments that SWF in general significantly outperforms FUP-based algorithms. Explicitly, the execution time of SWF is, in orders of magnitude, smaller than those required by FUP-based algorithms. Sensitivity analysis on various parameters of the database is also conducted to provide many insights into algorithm SWF. The advantage of SWF over FUP-based algorithms becomes even more prominent not only as the amount of incremented dataset increases but also as the size of the database increases. This is indeed an important feature for SWF to be practically used for the mining of an ongoing transaction database.

The rest of this chapter is organized as follows. Preliminaries and related works are given in Section 2.2. Algorithm SWF is described in Section 2.3 with its correctness proved. Performance studies on various schemes are conducted in Section 2.4. This chapter concludes with Section 2.5.

2.2 Preliminaries and Related Works

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of literals, called items. Let $D$ be a set of transactions, where each transaction $T$ is a set of items such that $T \subseteq I$. Note that the quantities of items bought in a transaction are not considered, meaning that each item is a binary variable representing if an item was bought. Each transaction is associated with an identifier, called TID. Let $X$ be a set of items. A transaction $T$ is said to contain $X$ if and only if $X \subseteq T$. An association rule is an implication of the form $X \implies Y$, where $X \subseteq I$, $Y \subseteq I$ and $X \cap Y = \emptyset$. The rule $X \implies Y$ holds in the transaction set $D$ with confidence $c$ if $c\%$ of transactions in $D$ that contain $X$ also contain $Y$. The rule $X \implies Y$ has support $s$ in the transaction set $D$ if $s\%$ of transactions in $D$ contain $X \cup Y$. For a given pair of confidence and support thresholds, the problem of mining association rules is to find out all the association rules that have confidence and support greater than the corresponding thresholds. This problem can be reduced to the problem of finding all frequent itemsets for the same support threshold [5]. Before the description of algorithm SWF in Section 2.3, some related works are reviewed below.

2.2.1 Apriori-like algorithms

Most of the previous studies, including those in [5, 19, 26, 27, 68, 79, 83], belong to Apriori-like approaches. Basically, an Apriori-like approach is based on an anti-monotone Apriori heuristic [5], i.e., if any itemset of length $k$ is not frequent in the database, its length $(k+1)$ super-itemset will never be frequent. The essential idea is to iteratively generate the set of candidate itemsets of length $(k+1)$ from the set of frequent itemsets of length $k$ (for $k \geq 1$), and to check their corresponding occurrence frequencies in the database. As a result, if the largest frequent itemset is a $j$-itemset, then an Apriori-like algorithm may need to scan the database up to $(j+1)$ times.

In Apriori-like algorithms, $C_3$ is generated from $L_2 \ast L_2$. In fact, a $C_2$ can be used to generate the candidate 3-itemsets. This technique is referred to as scan reduction in [22]. Clearly, a $C'_3$ generated from $C_2 \ast C_2$, instead of from $L_2 \ast L_2$, will have a size greater than $|C_3|$ where $C_3$ is generated from $L_2 \ast L_2$. However, if $|C'_3|$ is not much larger than $|C_3|$, and both $C_2$ and $C_3$ can be stored in main memory, we can find $L_2$ and $L_3$ together when the next scan of the database is performed, thereby saving one round of database scan. It can be seen that using this concept, one can determine all $L_k$s by as few as two scans of the database (i.e., one initial scan to determine $L_1$ and a final scan to determine all other frequent itemsets), assuming that $C'_k$ for $k \geq 3$ is generated from $C'_{k-1}$ and all $C_k$ for $k > 2$
can be kept in the memory. In [23], the technique of scan-reduction was utilized and shown to result in prominent performance improvement.

2.2.2 Partition-based algorithms

The works in [53, 62, 75] are essentially based on a partition-based heuristic, i.e., if $X$ is a frequent itemset in database $D$, which is divided into $n$ partitions $p_1, p_2, \ldots, p_n$, then $X$ must be a frequent itemset in at least one of the $n$ partitions. The partition algorithm in [75] divides $D$ into $n$ partitions, and processes one partition in main memory at a time. The algorithm first scans partition $p_i$, for $i = 1$ to $n$, to find the set of all local frequent itemsets in $p_i$, denoted as $L_{p_i}$. Then, by taking the union of $L_{p_i}$ for $i = 1$ to $n$, a set of candidate itemsets over $D$ is constructed, denoted as $C^G$. Based on the above partition-based heuristic, $C^G$ is a superset of the set of all frequent itemsets in $D$. Finally, the algorithm scans each partition for the second time to calculate the support of each itemset in $C^G$ and to find out which candidate itemsets are really frequent itemsets in $D$. Instead of constructing $C^G$ by taking the union of $L_{p_i}$ for $i = 1$ to $n$, some variations of the above partition algorithm are proposed in [53, 62]. In [62], algorithm SPINC constructs $C^G$ incrementally by adding $L_{p_i}$ to $C^G$ whenever $L_{p_i}$ is available. SPINC starts the counting of occurrences for each candidate itemset $c \in C^G$ as soon as $c$ is added to $C^G$. In [53], algorithm AS-CPA employs prior knowledge collected during the mining process to further reduce the number of candidate itemsets and to overcome the problem of data skew. However, these works were not devised to handle incremental updating of association rule.

2.2.3 FUP-based algorithms

Since it is costly to find the association rules in large databases, incremental updating techniques are desirable in order to avoid redoing data mining on the whole updated database. Basically, similarly to that of Apriori, the framework of FUP, which can update the association rules in a database when new transactions are added to the database, contains a number of iterations [26, 27]. The candidate sets at each iteration are generated based on the frequent itemsets found in the previous iteration. The key steps of FUP are listed below, where $\Delta^+$ denotes the added portion of an ongoing transaction database. (1) At each iteration, the supports of the size-$k$ frequent itemsets in $L$ are updated against the increment $\Delta^+$ to filter out those that are no longer in the updated database. (2) While scanning the increment, a set of candidate sets, $C_k$, is extracted from the transactions in $\Delta^+$, together with their supports in $\Delta^+$ counted. The supports of these sets in $C_k$ are then updated against the original database to find the “new” frequent itemsets. (3) Many sets in $C_k$ can be pruned away by checking their supports in $\Delta^+$ before the update against the original database starts. (4) The size of the updated database is reduced at each iteration by pruning away a few items from some transactions in the updated database. The major idea is to reuse the information of the old frequent itemsets and to integrate the support information of the new frequent itemsets in order to substantially reduce the pool of candidate sets to be re-examined. An extension to FUP was reported in [27] and is referred to as FUP$_2$. In essence, FUP$_2$ is equivalent to FUP for the case of insertion, and is, however, a complementary algorithm of FUP for the case of deletion. Another FUP-based algorithm, call FUP$_2H$, was also devised in [27] to utilize the hash technique for performance improvement. As pointed out earlier, the existing FUP-based algorithms in general suffer from two inherent problems, namely (1) the occurrence of a potentially huge set of candidate itemsets, which is particularly critical for incremental mining since the candidate sets for the original database and the incremental portion are generated separately, and (2) the need of multiple scans of database.
2.3 SWF: Incremental Mining with Sliding-Window Filtering

In essence, by partitioning a transaction database into several partitions, algorithm SWF employs a filtering threshold in each partition to deal with the candidate itemset generation. As described earlier, under SWF, the cumulative information in the prior phases is selectively carried over toward the generation of candidate itemsets in the subsequent phases. In the processing of a partition, a progressive candidate set of itemsets is generated by SWF. Explicitly, a progressive candidate set of itemsets is composed of the following two types of candidate itemsets, i.e., (1) the candidate itemsets that were carried over from the previous progressive candidate set in the previous phase and remain as candidate itemsets after the current partition is taken into consideration (Such candidate itemsets are called type $\alpha$ candidate itemsets); and (2) the candidate itemsets that were not in the progressive candidate set in the previous phase but are newly selected after only taking the current data partition into account (Such candidate itemsets are called type $\beta$ candidate itemsets). As such, after the processing of a phase, algorithm SWF outputs a cumulative filter, denoted by $CF$, which consists of a progressive candidate set of itemsets, their occurrence counts and the corresponding partial support required. With these design considerations, algorithm SWF is shown to have very good performance for incremental mining. In Section 2.3.1, an illustrative example of SWF is presented. A detailed description of algorithm SWF is given in Section 2.3.2. The correctness of SWF is proved in Section 2.3.3.

2.3.1 An example of incremental mining by SWF

Algorithm SWF proposed can be best understood by the illustrative transaction database in Figure 2.2 and Figure 2.3 where a scenario of generating frequent itemsets from a transaction database for the incremental mining is given. The minimum transaction support is assumed to be $s = 40\%$. Without loss of generality, the incremental mining problem can be decomposed into two procedures:
1. **Preprocessing procedure:** This procedure deals with mining on the original transaction database.

2. **Incremental procedure:** The procedure deals with the update of the frequent itemsets for an ongoing time-variant transaction database.

The preprocessing procedure is only utilized for the initial mining of association rules in the original database, e.g., \( \text{db}^{1,n} \). For the generation of mining association rules in \( \text{db}^{2,n+1}, \text{db}^{3,n+2}, \text{db}^{4,j} \), and so on, the incremental procedure is employed. Consider the database in Figure 2.2. Assume that the original transaction database \( \text{db}_{1}^{i} \) is segmented into three partitions, i.e., \( \{P_{1}, P_{2}, P_{3}\} \), in the preprocessing procedure. Each partition is scanned sequentially for the generation of candidate 2-itemsets in the first scan of the database \( \text{db}_{1}^{i} \). After scanning the first segment of 3 transactions, i.e., partition \( P_{1} \), 2-itemsets \{AB, AC, AE, AF, BC, BE, CE\} are generated as shown in Figure 2.3. In addition, each potential candidate itemset \( c \in C_{2} \) has two attributes: (1) \( c.\text{start} \) which contains the identity of the starting partition when \( c \) was added to \( C_{2} \), and (2) \( c.\text{count} \) which contains the number of occurrences of \( c \) since \( c \) was added to \( C_{2} \). Since there are three transactions in \( P_{1} \), the partial minimal support is \( [3 \times 0.4] = 2 \). Such a partial minimal support is called the filtering threshold in this chapter. Itemsets whose occurrence counts are below the filtering threshold are removed. Then, as shown in Figure 2.3, only \{AB, AC, BC\}, marked by “\( \circ \)”, remain as candidate itemsets (of type \( \beta \) in this phase since they are newly generated) whose information is then carried over to the next phase of processing.

Similarly, after scanning partition \( P_{2} \), the occurrence counts of potential candidate 2-itemsets are recorded (of type \( \alpha \) and type \( \beta \)). From Figure 2.3, it is noted that since there are also 3 transactions in \( P_{2} \), the filtering threshold of those itemsets carried out from the previous phase (that become type \( \alpha \) candidate itemsets in this phase) is \( [(3 + 3) \times 0.4] = 3 \) and that of newly identified candidate itemsets (i.e., type \( \beta \) candidate itemsets) is \( [3 \times 0.4] = 2 \). It can be seen from Figure 2.3 that we have 5 candidate itemsets in \( C_{2} \) after the processing of partition \( P_{2} \), and 3 of them are type \( \alpha \) and 2 of them are type \( \beta \).

Finally, partition \( P_{3} \) is processed by algorithm \( \text{SWF} \). The resulting candidate 2-itemsets are \( C_{2} = \{AB, AC, BC, BD, BE\} \) as shown in Figure 2.3. Note that though appearing in the previous phase \( P_{2} \), itemset \{AD\} is removed from \( C_{2} \) once \( P_{3} \) is taken into account since its occurrence count does not meet the filtering threshold then, i.e., \( 2 < 3 \). However, we do have one new itemset, i.e., \( BE \), which joins the \( C_{2} \) as a type \( \beta \) candidate itemset. Consequently, we have 5 candidate 2-itemsets generated by \( \text{SWF} \), and 4 of them are of type \( \alpha \) and one of them is of type \( \beta \). Note that instead of 15 candidate itemsets that would be generated if \( \text{Apriori} \) were used\(^1\), only 5 candidate 2-itemsets are generated by \( \text{SWF} \). The correctness of algorithm \( \text{SWF} \) will be formally proved later.

After generating \( C_{2} \) from the first scan of database \( \text{db}_{1}^{3} \), we employ the scan reduction technique and use \( C_{2} \) to generate \( C_{k} (k = 2, 3, ..., n) \), where \( C_{n} \) is the candidate last-itemsets. It can be verified that a \( C_{2} \) generated by \( \text{SWF} \) can be used to generate the candidate 3-itemsets and its sequential \( C_{k-1}^{'} \) can be utilized to generate \( C_{k}^{'} \). Clearly, a \( C_{3}^{'} \) generated from \( C_{2} \times C_{2} \), instead of from \( L_{2} \times L_{2} \), will have a size greater than \( |C_{3}| \) where \( C_{3} \) is generated from \( L_{2} \times L_{2} \). However, since the \( |C_{2}| \) generated by \( \text{SWF} \) is very close to the theoretical minimum, i.e., \( |L_{2}| \), the \( |C_{3}^{'}| \) is not much larger than \( |C_{3}| \). Similarly, the \( |C_{k}^{'}| \) is close to \( |C_{k}| \). All \( C_{k}^{'} \) can be stored in main memory, and we can find \( L_{k} (k = 1, 2, ..., n) \) together when the second scan of the database \( \text{db}_{1}^{3} \) is performed. Thus, only two scans of the original database \( \text{db}_{1}^{3} \) are required in the preprocessing step. In addition, instead of recording all \( L_{k} \)'s in main memory, we only have to keep \( C_{2} \) in main memory for the subsequent incremental mining of an ongoing time variant transaction database.

The merit of \( \text{SWF} \) mainly lies in its incremental procedure. As depicted in Figure 2.3, the mining database will be moved from \( \text{db}_{1}^{3} \) to \( \text{db}_{2}^{4} \). Thus, some transactions, i.e., \( t_{1}, t_{2}, \) and \( t_{3} \), are deleted from

\(^1\)The details of the execution procedure by \( \text{Apriori} \) are omitted here. Interested readers are referred to [7].
Candidates in $db^{1,3}$:
\{A\}, \{B\}, \{C\}, \{D\}, \{E\}, \{F\}, \{AB\}, \{AC\}, \{BC\}, \{BD\}, \{BE\}, \{ABC\}

Large Itemsets in $db^{1,3}$:
\{A\}, \{B\}, \{C\}, \{D\}, \{E\}, \{F\}, \{AB\}, \{AC\}, \{BC\}, \{BE\}

$P_1$

<table>
<thead>
<tr>
<th>C_2</th>
<th>start</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>AC</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>AE</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>AF</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>BC</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>BE</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CE</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$P_2$

<table>
<thead>
<tr>
<th>C_2</th>
<th>start</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>AC</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>AD</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>BC</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>BD</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>BE</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>CF</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>DE</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

$P_3$

<table>
<thead>
<tr>
<th>C_2</th>
<th>start</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>AC</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>AD</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>AF</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>BC</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>BD</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>BE</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>CF</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>DE</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Candidates in $db^{1,3}$:
\{A\}, \{B\}, \{C\}, \{D\}, \{E\}, \{F\}, \{AB\}, \{AC\}, \{BC\}, \{BD\}, \{BE\}, \{DF\}, \{EF\}, \{BDE\}, \{DEF\}

Large Itemsets in $db^{2,4}$:
\{A\}, \{B\}, \{C\}, \{D\}, \{E\}, \{F\}, \{BD\}, \{BE\}, \{DE\}

Figure 2.3: Large itemsets generation for the incremental mining with SWF
the mining database and other transactions, i.e., \( t_{10}, t_{11}, \) and \( t_{12} \), are added. For ease of exposition, this incremental step can also be divided into three sub-steps: (1) generating \( C_2 \) in \( D^- = db^{1,3} - \Delta^- \), (2) generating \( C_2 \) in \( db^{2,4} = D^- + \Delta^+ \) and (3) scanning the database \( db^{2,4} \) only once for the generation of all frequent itemsets \( L_k \). In the first sub-step, \( db^{1,3} - \Delta^- = D^- \), we check out the pruned partition \( P_1 \), and reduce the value of \( c.\text{count} \) and set \( c.\text{start} = 2 \) for those candidate itemsets \( c \) where \( c.\text{start} = 1 \).

It can be seen that itemsets \{AB, AC, BC\} were removed. Next, in the second sub-step, we scan the incremental transactions in \( P_4 \). The process in \( D^- + \Delta^+ = db^{2,4} \) is similar to the operation of scanning partitions, e.g., \( P_3 \), in the preprocessing step. Three new itemsets, i.e., DE, DF, EF, join the \( C_2 \) after the scan of \( P_4 \) as type \( \beta \) candidate itemsets. Finally, in the third sub-step, we use \( C_2 \) to generate \( C_0^k \) as mentioned above. With scanning \( db^{2,4} \) only once, SWF obtains frequent itemsets \{A, B, C, D, E, F, BD, BE, DE\} in \( db^{2,4} \). As will be shown by experimental results later, the improvement achieved by algorithm SWF is even more prominent as the amount of the incremental portion increases and also as the size of the database \( db^{i,j} \) increases.

### 2.3.2 Algorithm of SWF

For ease exposition, the meanings of various symbols used are given in Table 2.1. The preprocessing procedure and the incremental procedure of algorithm SWF are described in Section 2.3.2.1 and Section 2.3.2.2, respectively.

#### 2.3.2.1 Preprocessing procedure of SWF

The preprocessing procedure of Algorithm SWF is outlined below. Initially, the database \( db^{1,n} \) is partitioned into \( n \) partitions by executing the preprocessing procedure (in Step 2), and CF, i.e., cumulative filter, is empty (in Step 3). Let \( C_2^{i,j} \) be the set of progressive candidate 2-itemsets generated by database \( db^{i,j} \). It is noted that instead of keeping \( L_k's \) in the main memory, algorithm SWF only records \( C_2^{1,n} \) which is generated by the preprocessing procedure to be used by the incremental procedure.

**Preprocessing procedure of Algorithm SWF**

1. \( n = \text{Number of partitions} \);
2. \( |db^{1,n}| = \sum_{k=1,n} |P_k| \);
3. \( CF = \emptyset \);
4. begin for \( k = 1 \) to \( n \)  // 1st scan of \( db^{1,n} \)
5. begin for each 2-itemset \( I \in P_k \)
6. if ( \( I \notin CF \) )
7. \( I.count = N_{pk}(I); \)
8. \( I.start = k; \)
9. \( \text{if (} I.count \geq s \cdot |P_k| \text{)} \)
10. \( CF = CF \cup I; \)
11. \( \text{if (} I \in CF \text{)} \)
12. \( I.count = I.count + N_{pk}(I); \)
13. \( \text{if (} I.count < [s \cdot \sum_{m=I.start,k} |P_m|] \text{)} \)
14. \( CF = CF - I; \)
15. end
16. end
17. select \( C_{1,n}^1 \) from \( I \) where \( I \in CF; \)
18. keep \( C_{2,n}^1 \) in main memory;
19. \( h = 2; \) //\( C_1 \) is given
20. begin while \((C_{1,n}^1 \neq \emptyset)\) //Database scan reduction
21. \( C_{h+1,n}^1 = C_{h,n}^1 \times C_{h,n}^1; \)
22. \( h = h + 1; \)
23. end
24. refresh \( I.count = 0 \) where \( I \in C_{h,n}^1; \)
25. begin for \( k = 1 \) to \( n \) //2nd scan of \( db_{1,n} \)
26. for each itemset \( I \in C_{h,n}^1 \)
27. \( I.count = I.count + N_{pk}(I); \)
28. end
29. for each itemset \( I \in C_{h,n}^1 \)
30. \( \text{if (} I.count \geq [s \cdot |db_{1,n}|] \text{)} \)
31. \( L_h = L_h \cup I; \)
32. end
33. return \( L_h \);

From Step 4 to Step 16, the algorithm processes one partition at a time for all partitions. When partition \( P_i \) is processed, each potential candidate 2-itemset is read and saved to CF. The number of occurrences of an itemset \( I \) and its starting partition are recorded in \( I.count \) and \( I.start \), respectively. An itemset, whose \( I.count \geq [s \cdot \sum_{m=I.start,k} |P_m|] \), will be kept in CF. Next, we select \( C_{2,n}^1 \) from \( I \) where \( I \in CF \) and keep \( C_{2,n}^1 \) in main memory for the subsequent incremental procedure. With employing the scan reduction technique from Step 19 to Step 23, \( C_{h,n}^1; (h \geq 3) \) are generated in main memory. After refreshing \( I.count = 0 \) where \( I \in C_{h,n}^1 \), we begin the last scan of database for the preprocessing procedure from Step 25 to Step 28. Finally, those itemsets whose \( I.count \geq [s \cdot |db_{1,n}|] \) are the frequent itemsets.

### 2.3.2.2 Incremental procedure of SWF

As shown in Table 2.1, \( D^- \) indicates the unchanged portion of an ongoing transaction database. The deleted and added portions of an ongoing transaction database are denoted by \( \Delta^- \) and \( \Delta^+ \), respectively.
It is worth mentioning that the sizes of $\Delta^+$ and $\Delta^-$, i.e., $|\Delta^+|$ and $|\Delta^-|$ respectively, are not required to be the same. The incremental procedure of SWF is devised to maintain frequent itemsets efficiently and effectively. This procedure is outlined below.

**Incremental procedure of Algorithm SWF**

1. Original database = $db^{m,n}$;
2. New database = $db^{i,j}$;
3. Database removed $\Delta^- = \sum_{k=m,i-1} P_k$;
4. Database database $\Delta^+ = \sum_{k=n+1,j} P_k$;
5. $D^- = \sum_{k=i,n} P_k$;
6. $db^{i-j} = db^{m,n} - \Delta^- + \Delta^+$;
7. loading $C_2^{m,n}$ of $db^{m,n}$ into CF where $I \in C_2^{m,n}$;
8. begin for $k = m$ to $i - 1$  
   // one scan of $\Delta^-$
9. begin for each 2-itemset $I \in P_k$
10. if ( $I \in CF$ and $I.start \leq k$ )
11.  
12.     $I.count = I.count - N_{P_k}(I)$;
13.     $I.start = k + 1$;
14.     if ( $I.count < [s * \sum_{m=i.start,n} |P_m|]$ )
15.         $CF = CF - I$;
16. end
17. end
18. begin for each 2-itemset $I \in P_k$
19.     if ( $I \notin CF$ )
20.         $I.count = N_{P_k}(I)$;
21.         $I.start = k$;
22.         if ( $I.count \geq s * |P_k|$ )
23.             $CF = CF \cup I$;
24.         if ( $I \in CF$ )
25.             $I.count = I.count + N_{P_k}(I)$;
26.             if ( $I.count < [s * \sum_{m=i.start,k} |P_m|]$ )
27.                 $CF = CF - I$;
28. end
29. end
30. select $C_2^{i,j}$ from $I$ where $I \in CF$;
31. keep $C_2^{i,j}$ in main memory
32. $h = 2$  
   //C_4 is well known.
33. begin while ($C_2^{i,j} \neq \emptyset$)  
   //Database scan reduction
34. $C_4^{i,j} = C_2^{i,j} * C_4^{i,j}$;
35. $h = h + 1$;
36. end
37. refresh $I.count = 0$ where $I \in C_4^{i,j}$;
38. begin for $k = i$ to $j$  
   //only one scan of $db^{i,j}$
39.     for each itemset $I \in C_4^{i,j}$
\[ I\.count = I\.count + N_{pk}(I) \]

end

for each itemset \( I \in C_{h}^{i,j} \)

\[ \text{if} ( I\.count \geq |s * dB^{i,j}| ) \]

\[ L_{h} = L_{h} \cup I \]

end

return \( L_{h} \);

As mentioned before, this incremental step can also be divided into three sub-steps: (1) generating \( C_{2} \) in \( D^{-} = dB^{1,3} - \Delta^{-} \), (2) generating \( C_{2} \) in \( dB^{2,4} = D^{-} + \Delta^{+} \) and (3) scanning the database \( dB^{2,4} \) only once for the generation of all frequent itemsets \( L_{k} \). Initially, after some update activities, old transactions \( \Delta^{-} \) are removed from the database \( dB^{m,n} \) and new transactions \( \Delta^{+} \) are added (in Step 6). Note that \( \Delta^{-} \subset dB^{m,n} \). Denote the updated database as \( dB^{i,j} \). Note that \( dB^{i,j} = dB^{m,n} - \Delta^{-} + \Delta^{+} \).

We denote the unchanged transactions by \( D^{-} = dB^{m,n} - \Delta^{-} = dB^{i,j} - \Delta^{+} \). After loading \( C_{2}^{m,n} \) of \( dB^{m,n} \) into \( CF \) where \( I \in C_{2}^{m,n} \), we start the first sub-step, i.e., generating \( C_{2} \) in \( D^{-} = dB^{m,n} - \Delta^{-} \). This sub-step tries to reverse the cumulative processing which is described in the preprocessing procedure. From Step 8 to Step 16, we prune the occurrences of an itemset \( I \), which appeared before partition \( P_{t} \), by deleting the value \( I\.count \) where \( I \in CF \) and \( I\.start < i \). Next, from Step 17 to Step 36, similarly to the cumulative processing in Section 2.3.2.1, the second sub-step generates new potential \( C_{2}^{i,j} \) in \( dB^{i,j} = D^{-} + \Delta^{+} \) and employs the scan reduction technique to generate \( C_{h}^{i,j} \)'s from \( C_{2}^{i,j} \). Finally, to generate new \( L_{k} \)'s in the updated database, we scan \( dB^{i,j} \) for only once in the incremental procedure to maintain frequent itemsets. Note that \( C_{2}^{i,j} \) is kept in main memory for the next generation of incremental mining.

Note that SWF is able to filter out false candidate itemsets in \( P_{t} \) with a hash table. Same as in [68], using a hash table to prune candidate 2-itemsets, i.e., \( C_{2} \), in each accumulative ongoing partition set \( P_{t} \) of transaction database, the CPU and memory overhead of SWF can be further reduced. As will be validated by our experimental studies, SWF indeed provides an efficient solution for incremental mining, which is, in our opinion, important for mining the record-based databases whose data are being frequently and continuously added, such as Web log records, stock market data, grocery sales data, and transactions in electronic commerce, to name a few.

### 2.3.3 Correctness of SWF

With the above two procedures described, we now examine the correctness and effectiveness of algorithm SWF. Let \( N_{pk}(I) \) be the number of transactions in partition \( P_{k} \) that contain itemset \( I \), and \( |P_{k}| \) is the number of transactions in partition \( P_{k} \). Also, let \( dB^{i,j} \) denote the part of the transaction database formed by a continuous region from partition \( P_{i} \) to partition \( P_{j} \), and \( |dB^{i,j}| = \sum_{k=i,j} |P_{k}| \). We can then define
the region ratio of an itemset as follows.

**Definition 2.1:** A region ratio of an itemset $I$ for the transaction database $db^{i,j}$, denoted by $r_{i,j}(I)$, is

$$r_{i,j}(I) = \frac{\sum_{k=i}^{j} N_{pk}(I)}{|db^{i,j}|}.$$

In essence, the region ratio of an itemset is the support of that itemset if only the part of transaction database $db^{i,j}$ is considered.

**Lemma 2.1:** An itemset $I$ remains in the CF after the processing of partition $P_j$ if and only if there exists an $i$ such that for any integer $k$ in the interval $[i, j]$, $r_{i,k}(I) \geq s$, where $s$ is the minimal support required.

**Proof of Lemma 2.1:** We shall prove the “if” condition first. Consider the following two cases. First, suppose the itemset $I$ is not in the progressive candidate set before the processing of partition $P_i$. Since $r_{i,i}(I) \geq s$, itemset $I$ will be selected as a type $\beta$ candidate itemset by SWF after the processing of partition $P_i$. On the other hand, if the itemset $I$ is already in the progressive candidate set before the processing of partition $P_i$, itemset $I$ will remain as a type $\alpha$ candidate itemset by SWF. Clearly, for the above two cases, itemset $I$ will remain in CF throughout the processing from $P_i$ to $P_j$ since for any integer $k$ in the interval $[i, j]$, $r_{i,k}(I) \geq s$.

We now prove the “only if” condition, i.e., if $I$ remains in CF after the processing of partition $P_j$ then there exists an $i$ such that for any $k$ in the interval $[i, j]$, $r_{i,k}(I) \geq s$. Note that itemset $I$ can be either type $\alpha$ or type $\beta$ candidate itemset in the CF after the processing of partition $P_j$. Suppose $I$ is a type $\beta$ candidate itemset there, then this implication follows by setting $j = i$ since $r_{i,i}(I) \geq s$. On the other hand, suppose that $I$ is a type $\alpha$ candidate itemset after the processing of $P_j$, which means itemset $I$ has become a type $\beta$ candidate itemset in a previous phase. Then, we shall trace backward the type of itemset $I$ from partition $P_j$ (i.e., looking over $P_j$, $P_{j-1}$, $P_{j-2}$ and so forth) until the partition that records itemset $I$ as a type $\beta$ candidate itemset is first encountered. (It should be noted that there could be two discontinuous regions that record itemset $I$ in the CF, which means that an itemset may get on and off the progressive candidate set through the processing of partitions. This in turn means that an itemset may appear as a type $\beta$ candidate itemset more than once. Such a scenario occurs for the itemset BE in the example in Figure 2.3.) By referring the partition identified above as partition $P_i$, we have, for any $k$ in the interval $[i, j]$, $r_{i,k}(I) \geq s$, completing the proof of this lemma. Q.E.D.

Lemma 2.1 leads to Lemma 2.2 below.

**Lemma 2.2:** An itemset $I$ remains in CF after the processing of partition $P_j$ if and only if there exists an $i$ such that $r_{i,j}(I) \geq s$, where $s$ is the minimal support required.

**Proof of Lemma 2.2:** It can be seen that the proof of “only if” condition follows directly from Lemma 1. We now prove the “if” condition of this lemma. If there exists an $i$ such that $r_{i,j}(I) \geq s$ then we let
Let $t$ be the largest $x$ such that $r_{i,x}(I) < s$. If such a $t$ does not exist, it follows from Lemma 1 that itemset $I$ will remain in $CF$ after the processing of partition $P_j$. If such a $t$ exists, we have $r_{t+1,j}(I) \geq s$ since $r_{i,t}(I) < s$ and $r_{i,j}(I) \geq s$. It again follows from Lemma 1 that itemset $I$ will remain in $CF$ after the processing of partition $P_j$. This lemma follows. Q.E.D.

Lemma 2.2 leads to the following theorem which states the correctness of algorithm SWF.

**Theorem 2.1:** If an itemset $I$ is a frequent itemset, then $I$ will be in the progressive candidate set of itemsets produced by algorithm SWF.

**Proof of Theorem 2.1:** Let $n$ be the number of partitions of the transaction database. Since the itemset $I$ is a frequent itemset, we have $r_{1,n}(I) \geq s$, which is in essence a special case of Lemma 2.2 for $i = 1$ and $j = n$, proving this theorem. Q.E.D.

It follows from Theorem 2.1 that the frequent itemsets generated by SWF are the same as those produced by existing association rule mining algorithms such as Apriori. Furthermore, we let $C^{i,j}$, where $i \leq j$, be the set of progressive candidate itemsets generated by algorithm SWF with respect to database $db^{i,j}$ after the processing of $P_j$. We then have the following lemma.

**Lemma 2.3:** For $i \leq k \leq j$, then $C^{k,j} \subseteq C^{i,j}$.

**Proof of Lemma 2.3:** Assume that there exists an itemset $I \in C^{k,j}$. From the “only if” implication of Lemma 2.2, it follows that there exists an $h$ such that $r_{h,j}(I) \geq s$, where $k \leq h \leq j$. Since $i \leq k \leq j$, we have $i \leq h \leq j$. Then, according to the “if” implication of Lemma 2.2, itemset $I$ is also in $C^{i,j}$, i.e., $I \in C^{i,j}$. The fact that $C^{k,j} \subseteq C^{i,j}$ follows. Q.E.D.

Lemma 2.3 leads to the following theorem which states the effectiveness of SWF for incremental mining.

**Theorem 2.2:** If an itemset $I$ is a frequent itemset with respect to the database $db^{i+1,j+1}$, then itemset $I$ is either in $C^{i,j}$ or will be a type $\beta$ candidate itemset after the processing of partition $P_{j+1}$.

**Proof of Theorem 2.2:** If an itemset $I$ is a frequent itemset with respect to the database $db^{i+1,j+1}$, we then have $r_{i+1,j+1}(I) \geq s$. Three cases for $r_{i+1,j+1}(I) \geq s$ are considered. The first case is $r_{i+1,j}(I) \geq s$ and $r_{j+1,j+1}(I) \geq s$, and the second one is $r_{i+1,j}(I) \geq s$ and $r_{j+1,j+1}(I) \leq s$. From Theorem 1, it follows that in the above two cases, $I \in C^{i+1,j}$, which in turn implies that $I \in C^{i,j}$ since we have $C^{i+1,j} \subseteq C^{i,j}$ by Lemma 2.3.

Consider the third case where $r_{i+1,j}(I) < s$ and $r_{j+1,j+1}(I) \geq s$. If $r_{i+1,j}(I) < s$ and $I \notin C^{i+1,j}$, then itemset $I$ will be a type $\beta$ candidate itemset after the processing of partition $P_{j+1}$ since $r_{j+1,j+1}(I) \geq s$. On the other hand, if $r_{i+1,j}(I) < s$ but $I \in C^{i+1,j}$, we also get $I \in C^{i,j}$, from Lemma 2.3. This theorem is thus proved. Q.E.D.
Note that any itemset $I$ that is a frequent itemset with respect to $db^{i+1,j+1}$ and has appeared in $C^{i,j}$ will be identified as a type $\alpha$ candidate itemset after the processing of partition $P_{j+1}$. From Theorem 2.2 and this fact, it follows that the cumulative filters of algorithm SWF can be determined in a progressive manner without missing any possible frequent itemsets even in the presence of the need of mining an ongoing time variant transaction database.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>D</td>
</tr>
<tr>
<td>$</td>
<td>\Delta^+</td>
</tr>
<tr>
<td>$</td>
<td>\Delta^-</td>
</tr>
<tr>
<td>$</td>
<td>d</td>
</tr>
<tr>
<td>$</td>
<td>T</td>
</tr>
<tr>
<td>$</td>
<td>I</td>
</tr>
<tr>
<td>$</td>
<td>L</td>
</tr>
<tr>
<td>$N$</td>
<td>Number of items</td>
</tr>
</tbody>
</table>

Table 2.2: Meanings of various parameters in the mining of SWF technique

### 2.4 Experimental Studies

To assess the performance of algorithm SWF, we performed several experiments on a computer with a CPU clock rate of 450 MHz and 512 MB of main memory. The transaction data resides in the NTFS file system and is stored on a 30GB IDE 3.5” drive with a measured sequential throughput of 10MB/second. The simulation program was coded in C++. The methods used to generate synthetic data are described in Section 2.4.1. The performance comparison of SWF, FUP$_2$ and Apriori is presented in Section 2.4.2. Section 2.4.3 shows the I/O cost among SWF, FUP$_2$ and Apriori. We conduct some experiments on examining CPU and memory overhead in Section 2.4.4. Results on scaleup experiments are presented in Section 2.4.5.

#### 2.4.1 Generation of synthetic workload

For obtaining reliable experimental results, the method to generate synthetic transactions we employed in this study is similar to the ones used in prior works [11, 26, 68, 82]. Explicitly, we generated several different transaction databases from a set of potentially frequent itemsets to evaluate the performance of SWF. These transactions mimic the transactions in the retailing environment. Note that the efficiency of algorithm SWF has been evaluated by some real databases, such as Web log records and grocery sales data. However, we show the experimental results from synthetic transaction data so that the work relevant to data cleaning, which is in fact application-dependent and also orthogonal to the incremental technique proposed, is hence omitted for clarity. Further, more sensitivity analysis can then be conducted by using the synthetic transaction data. Each database consists of $|D|$ transactions, and on the
Figure 2.4: Extensive analysis of various values of N and |L|
average, each transaction has \(|T|\) items. Table 2.2 summarizes the meanings of various parameters used in the experiments. The mean of the correlation level is set to 0.25 for our experiments.

Recall that the sizes of \(|\Delta^+|\) and \(|\Delta^-|\) are not required to be the same for the execution of SWF. Without loss of generality, we set \(|d| = |\Delta^+| = |\Delta^-|\) for simplicity. Thus, by denoting the original database as \(db^{1:n}\) and the new mining database as \(db^{i:j}\), we have \(|db^{i:j}| = |db^{1:n} - \Delta^- + \Delta^+| = |D|\), where \(\Delta^- = db^{1,i-1}\) and \(\Delta^+ = db^{n+1:j}\). In the following, we use the notation \(Tx - Iy - Dm - dn\) to represent a database in which \(D = m\) thousands, \(d = n\) thousands, \(|T| = x\), and \(|I| = y\). We compare relative performance of three methods, i.e., Apriori, FUP-based algorithms and SWF.

As mentioned before, without any provision for incremental mining, Apriori algorithm has to re-run the association rule mining algorithm on the whole updated database. As reported in [26, 27], with reducing the candidate itemsets, FUP-based algorithms outperform Apriori. As will be shown by our experimental results, with the sliding window technique that carries cumulative information selectively, the execution time of SWF is, in orders of magnitude, smaller than those required by FUP-based algorithms. In order to conduct our experiments on a database of size \(db^{i:j}\) with an increment of \(\Delta^+\) and a removal of \(\Delta^-\), a database of \(db^{i:j}\) is first generated and then \(db^{1,i-1}, db^{1:n}, db^{n+1:j}\) and \(db^{i:j}\) are produced separately.

### 2.4.2 Experiment one: Relative performance

We first conducted several experiments to evaluate the relative performance of Apriori, FUP2 and SWF. As shown in Figure 2.4, the experimental results are consistent from one to another for various values of \(|L|\) and \(N\) on dataset \(T10 - I4 - D100 - d10\). For interest of space, we only report the results on \(|L| = 2000\) and \(N = 10000\) in the following experiments. Figure 2.5 shows the relative execution times for the three algorithms as the minimum support threshold is decreased from 1% support to 0.1% support. When the support threshold is high, there are only a limited number of frequent itemsets produced. However, as the support threshold decreases, the performance difference becomes prominent in that SWF significantly outperforms both FUP2 and Apriori. As shown in Figure 2.5, SWF leads to prominent performance improvement for various sizes of \(|T|\), \(|I|\) and \(|d|\). Explicitly, SWF is in orders of magnitude faster than FUP2, and the margin grows as the minimum support threshold decreases. Note that from our experimental results, the difference between FUP2 and Apriori is consistent with that observed in [27]. In fact, SWF outperforms FUP2 and Apriori in both CPU and I/O costs, which are evaluated next.
Figure 2.5: Relative performance
2.4.3 Experiment two: Evaluation of I/O cost

To evaluate the corresponding of I/O cost, same as in [70], we assume that each sequential read of a byte of data consumes one unit of I/O cost and each random read of a byte of data consumes two units of I/O cost. Figure 2.6 shows the number of database scans and the I/O costs of Apriori, FUP$_2$H, i.e., hash-type FUP in [27], and SWF over data sets $T_{10-I4-D100-d10}$ and $T_{10-I4-D200-d20}$. As shown in Figure 2.6, SWF outperforms Apriori and FUP$_2$H where without loss of generality a hash table of 250 thousand entries is employed for those methods. Note that the large amount of database scans is the performance bottleneck when the database size does not fit into main memory. In view of that, SWF is advantageous since only one scan of the updated database is required, which is independent of the variance in minimum supports.

2.4.4 Experiment three: Reduction of CPU and memory overhead

As explained before, SWF substantially reduces the number of candidate itemsets generated. The effect is particularly important for the candidate 2-itemsets. The experimental results in Figure 2.7 show the candidate itemsets generated by Apriori, FUP$_2$H, and SWF across the whole processing on the datasets $T_{10-I4-D100-d10}$ and $T_{10-I4-D200-d20}$ with minimum support threshold $s = 0.1\%$. As shown in Figure 2.7, SWF leads to a 99% candidate reduction rate in $C_2$ when being compared to Apriori, and leads to a 93% candidate reduction rate in $C_2$ when being compared to FUP$_2$H. Similar phenomena were observed when other datasets were used. This feature of SWF enables it to efficiently
reduce the CPU and memory overhead. Note that the number of candidate 2-itemsets produced by SWF approaches to its theoretical minimum, i.e., the number of frequent 2-itemsets. Recall that the $C_3$ in either Apriori or FUP$_2H$ has to be obtained by $L_2$ due to the large size of their $C_2$. As shown in Figure 2.7, the value of $|C_k|$ ($k \geq 3$) is only slightly larger than that of Apriori or FUP$_2H$, even though SWF only employs $C_2$ to generate $C_k$s, thus fully exploiting the benefit of scan reduction.

2.4.5 Experiment four: Scaleup performance

In this experiment, we examine the scaleup performance of algorithm SWF. The scale-up results for different selected datasets are obtained. Figure 2.8 shows the scaleup performance of algorithm SWF as the values of $|D|$ and $|d|$ increase. Three different minimum supports are considered. We obtained the results for the dataset $T10 - I4 - Dm - d10$ when the number of customers increases from 100,000 to one million. The execution times are normalized with respect to the times for the 100,000 transactions dataset in the Figure 2.8a. The second scaleup experiment with the dataset $T10 - I4 - D1000 - dn$
Figure 2.8: Scaleup performance of SWF

shows the performance results of SWF when the number of transactions in the incremented dataset varies from 50 thousands to 300 thousands. The execution times are normalized with respect to the times for the 50,000 incremented transaction dataset in the Figure 2.8b. Note that, as shown in Figure 2.8b the execution time only slightly increases with the growth of the incremental size, showing good scalability of SWF.

To further understand the impact of $|D|$ and $|d|$ to the relative performance of algorithms SWF and FUP-based algorithms, we conduct the scaleup experiments for both SWF and FUP with two minimum support thresholds 0.2% and 0.4%. The results are shown in Figure 2.9 where the value in y-axis corresponds to the ratio of the execution time of SWF to that of FUP. Figure 2.9a shows the referenced ratio obtained from an updated database over datasets of $T_{10-I4-Dm-d10}$. With the value $|D|=10$, the execution-time-ratio of SWF to FUP decreases when the amount of updated database $|D|$ grows larger, meaning that the advantage of SWF over FUP increases as the database size increases. Figure 2.9b shows the execution-time-ratio for different values of $|d|$. It can be seen that since the size of $|d|$ has less influence on the performance of SWF, the execution-time-ratio becomes smaller with the growth of the incremental transaction number $|d|$. This also implies that the advantage of SWF over FUP becomes even more prominent as the amount of incremental portion increases.
Figure 2.9: Scaleup performance with the execution time ratio between SWF and FUP

2.5 Summary

We explored in this chapter an efficient sliding-window filtering algorithm for incremental mining of association rules. Under SWF, the cumulative information of mining previous partitions is selectively carried over toward the generation of candidate itemsets for the subsequent partitions. Algorithm SWF not only significantly reduces I/O and CPU cost by the concepts of cumulative filtering and scan reduction techniques but also effectively controls memory utilization by the technique of sliding-window partition. More importantly, SWF is particularly powerful for efficient incremental mining for an ongoing time-variant transaction database. The correctness of SWF is proved and some of its theoretical properties are derived. Extensive simulations have been performed to evaluate performance of algorithm SWF. Sensitivity analysis of various parameters was conducted to provide many insights into SWF. It was noted that the improvement achieved by SWF increases as the incremented portion of the dataset increases and also as the size of the database increases.
Chapter 3

Progressive Weighted Miner: An Efficient Method for Time-Constraint Mining on a Time-Variant Database

3.1 Introduction

The discovery of association relationship among the data in a huge database has been known to be useful in selective marketing, decision analysis, and business management [22, 40]. A popular area of applications is the market basket analysis, which studies the buying behaviors of customers by searching for sets of items that are frequently purchased either together or in sequence. For a given pair of confidence and support thresholds, the problem of mining association rules is to identify all association rules that have confidence and support greater than the corresponding minimum support threshold (denoted as \textit{min\_supp}) and minimum confidence threshold (denoted as \textit{min\_conf}). Association rule mining algorithms [5] work in two steps: (1) generate all frequent itemsets that satisfy \textit{min\_supp}; (2) generate all association rules that satisfy \textit{min\_conf} using the frequent itemsets.

Note that the data mining process may produce thousands of rules, many of which are uninteresting to users. Hence, several applications have called for the use of constraint-based rule mining [10, 35, 46, 45, 69, 85, 89]. Specifically, in constraint-based mining, mining is performed under the guidance of various of constraints provided by the user. The constraints addressed in the prior works include the following: (1) Knowledge type-constraints [71, 85]; (2) Data constraints [10]; (3) Dimension/level constraints [35]; (4) Interestingness constraints [46, 45]; and (5) Rule constraints [69, 85, 89]. Such constraints may be expressed as meta-rules (rule templates), as the maximum or minimum number of predicates that can occur in the rule antecedent or consequent, or as relationships among attributes, attribute values, and/or aggregates. Recently, many constraint-based mining works have focused on the use of rule constraints. This form of constraint-based mining allows users of specifying the rules to be
mined according to their need, thereby leading to much more useful mining results. According to our observation, these kinds of rule-constraint problems are based on the concept of embedding a variety of *item-constraints* in the mining process.

On the other hand, a time-variant database, as shown in Figure 3.1, consists of values or events varying with time. Time-variant databases are popular in many applications, such as daily fluctuations of a stock market, traces of a dynamic production process, scientific experiments, medical treatments, weather records, to name a few. In our opinion, the existing model of the constraint-based association rule mining is not able to efficiently handle the time-variant database due to two fundamental problems, i.e., (1) lack of consideration of the *exhibition period* of each individual transaction; (2) lack of an intelligent support counting basis for each item. Note that the traditional mining process treats transactions in different time periods indifferently and handles them along the same procedure. However, since different transactions have different exhibition periods in a time-variant database, only considering the occurrence count of each item might not lead to interesting mining results. This problem can be further explained by the examples below.

**Example 3.1.1:** When we try to analyze the past transaction data of computer and communication stores, such as sales of mobile phones and computers, we might discover little valuable information on customer buying behavior by the conventional mining methodologies. That is because that lots of mining results could be obsolete. For example, we may find such a rule as *people who buying Pentium-II computers will also like to have 4GB hard disks and 32M DRAMs*. Even though both of them have sufficient support and confidence from the mining processing of transaction database, they are not the selling items in this store anymore.

**Example 3.1.2:** In a transaction database as shown in Figure 3.2, the minimum transaction support
and confidence are assumed to be \( \text{min\_supp} = 30\% \) and \( \text{min\_conf} = 75\% \), respectively. A set of time-variant database indicates the transaction records from January 2001 to March 2001. The starting date of each transaction item is also given. Based on the traditional mining techniques, the \textit{support threshold} is denoted as \( \text{min\_S}^{T} = \lfloor 12 \times 0.3 \rfloor = 4 \) where 12 is the size of transaction set \( D \). It can be seen that only \( \{B, C, D, E, BC\} \) can be termed as frequent itemsets since their occurrences in this transaction database are all larger than the value of support threshold \( \text{min\_S}^{T} \). Thus, rule \( C \Rightarrow B \) is termed as a frequent association rule with support \( \text{supp}(C \cup B) = 41.67\% \) and confidence \( \text{conf}(C \Rightarrow B) = 83.33\% \). However, some phenomena are observed when we take the “\textit{item information}” in Figure 3.2 into consideration.

1. \textbf{An early product intrinsically possesses a higher likelihood to be determined as a frequent itemset.} As a result, the association rules we usually get will be those with long-term products such as “milk and bread are frequently purchased together”, which, while being correct by the definition, is of less interest to us in the association rule mining. In contrast, some more recent products, such as new books, which are really “frequent” and interesting in their exhibition periods are less likely to be identified as frequent ones if a traditional mining process is employed.

2. \textbf{Some discovered rules may be expired from users’ interest.} As shown in Example 3.1.1, some discovered knowledge may be obsolete and of little use. This is especially true when we perform the mining schemes on a transaction database of short life cycle products such as CPU, RAM, etc.

   \textbf{Example 3.1.3:} Considering the generated rule \( C \Rightarrow B \), both \( B \) and \( C \) were available from the very early dates of this mining transaction database. This information is very likely to have been explored in the previous mining database, such as the one from January 1997 to December 1999. Such mining results could be of less interest to our on-going mining works. For example, most researchers tend to pay more attention to the recently published papers.

3. \textbf{Different transactions are usually of different importance to the user.} From the above discussions, it is noted that mining long period transaction data could have less contribution to making future business decisions because, for example, the selling items could be out of date. Since a new coming data is usually viewed more important than an old one, without fully considering this aspect, the knowledge discovered from the traditional mining framework may lead to wrong decisions.

   Since the early work in [5], several efficient algorithms to mine association rules have been developed. These studies cover a broad spectrum of topics including: (1) fast algorithms based on the level-wise Apriori framework [7, 68, 23], partitioning [53, 75], sampling [83], parallel methods [6, 67], TreeProjection
One straightforward approach to addressing the above issues is to employ the item-constraints and/or multiple supports strategies, i.e., new coming items have higher weights for their item occurrences. However, as noted in [55, 87] these approaches will encounter another problem, i.e., there is no proper confidence threshold in such cases for the corresponding rule generation.

To remedy this, we broaden in this chapter the horizon of frequent pattern mining by introducing a weighted model of transaction-weighted association rules (abbreviated as weighted association rules) in a time-variant database. Specifically, we propose an efficient Progressive Weighted Miner (abbreviated as PWM) algorithm to perform the mining for this problem as well as conduct the corresponding performance studies. In algorithm PWM, the importance of each transaction period is first reflected by a proper weight assigned by the user. Then, PWM partitions the time-variant database in light of weighted periods of transactions and performs weighted mining. Explicitly, algorithm PWM explores
the mining of weighted association rules, denoted by \((X \Rightarrow Y)^W\), which is produced by two newly defined concepts of weighted – support and weighted – confidence in light of the corresponding weights in individual transactions. Basically, an association rule \(X \Rightarrow Y\) is termed to be a frequent weighted association rule \((X \Rightarrow Y)^W\) if and only if its weighted support is larger than minimum support required, i.e., \(\text{supp}^W(X \cup Y) > \text{min}_-\text{supp}\), and the weighted confidence \(\text{conf}^W(X \Rightarrow Y)\) is larger than minimum confidence needed, i.e., \(\text{conf}^W(X \Rightarrow Y) > \text{min}_-\text{conf}\). Instead of using the traditional support threshold \(\text{min}_-_S^T = \lceil |D| \times \text{min}_-\text{supp}\rceil\) as a minimum support threshold for each item, a weighted minimum support, denoted by \(\text{min}_-_S^W = \{\Sigma|P_i| \times W(P_i)\} \times \text{min}_-\text{supp}\), is employed for the mining of weighted association rules, where \(|P_i|\) and \(W(P_i)\) represent the amount of partial transactions and their corresponding weight values by a weighting function \(W(\cdot)\) in the weighted period \(P_i\) of the database \(D\). Let \(N_{P_i}(X)\) be the number of transactions in partition \(P_i\) that contain itemset \(X\). The support value of an itemset \(X\) can then be formulated as \(S^W(X) = \Sigma N_{P_i}(X) \times W(P_i)\). As a result, the weighted support ratio of an itemset \(X\) is \(\text{supp}^W(X) = \frac{S^W(X)}{\Sigma|P_i| \times W(P_i)}\).

Example 3.1.4: Let us follow Example 3.1.2 with the given \(\text{min}_-\text{supp} = 30\%\) and \(\text{min}_-\text{conf} = 75\%\). Consider \(W(P_1) = 0.5\), \(W(P_2) = 1\), and \(W(P_3) = 2\), we have this newly defined support threshold as \(\text{min}_-_S^W = \{4 \times 0.5 + 4 \times 1 + 4 \times 2\} \times 0.3 = 4.2\), we have weighted association rules, i.e., \((C \Rightarrow B)^W\) with relative weighted support \(\text{supp}^W(C \cup B) = 35.7\%\) and confidence \(\text{conf}^W(C \Rightarrow B) = \frac{\text{supp}^W(C \cup B)}{\text{supp}^W(C)} = 83.3\%\) and \((F \Rightarrow B)^W\) with relative weighted support \(\text{supp}^W(F \cup B) = 42.8\%\) and confidence \(\text{conf}^W(F \Rightarrow B) = \frac{\text{supp}^W(F \cup B)}{\text{supp}^W(F)} = 100\%\). More details can be found in a complete example in Section 3.3.2.

Explicitly, PWM first partitions the transaction database in light of weighted periods of transactions and then progressively accumulates the occurrence count of each candidate 2-itemset based on the intrinsic partitioning characteristics. With this design, algorithm PWM is able to efficiently produce weighted association rules for applications where different time periods are assigned with different weights. Algorithm PWM is also designed to employ a filtering threshold in each partition to early prune out those cumulatively infrequent 2-itemsets. The feature that the number of candidate 2-itemsets generated by PWM is very close to the actual number of frequent 2-itemsets allows us of employing the scan reduction technique by generating \(C_k\)'s from \(C_2\) directly to effectively reduce the number of database scans. Experimental results show that PWM produces a significantly smaller amount of candidate 2-itemsets than \(\text{Apriori}^W\), i.e., an extended version of \text{Apriori} algorithm. In fact, the number of the candidate itemsets \(C_k\)'s generated by PWM approaches to its theoretical minimum, i.e., the number of actual frequent \(k\)-itemsets, as the value of the minimal support increases. Specifically, the execution time of PWM is, in orders of magnitude, smaller than those required by \(\text{Apriori}^W\). Sensitivity analysis on various parameters of the database is also conducted to provide many insights into algorithm PWM.
We mention in passing that the works in [53, 75] which are essentially based on a partition-based heuristic were not designed for handling the time-constraints on mining association rules. In addition, the Frequent Pattern growth (FP-growth), which constructs a highly compact data structure (an FP-tree) to compress the original transaction database, is a method of mining frequent itemsets without candidate generation [36, 38]. However, in our opinion, FP-growth algorithms do not have obvious extensions to deal with the problem of mining weighted association rules. Note that those constraint-based rule mining methods that allow users of deriving rules of interest by providing meta-rules and item-constraints [35, 46, 45, 69, 85, 89] are not applicable to solving the weighted mining problem addressed in this chapter since the constraints we consider are on individual transactions rather than on items. Note that the problem of mining weighted association rules will be degenerated to the traditional one of mining association rules explored in previous works if the weighting function is assigned to be \(W(\cdot) = 1\), meaning that the model we consider can be viewed as a general framework of prior studies.

In this chapter, we not only explore the new model of weighted association rules in a time-variant database, but also propose an efficient Progressive Weighted Miner methodology to perform the mining for this problem as well as conduct the corresponding performance studies. These features distinguish this dissertation from others.

The rest of this chapter is organized as follows. Problem description is given in Section 3.2. Algorithm PWM is described in Section 3.3 with its correctness proved. Performance studies on various schemes are conducted in Section 3.4. This chapter concludes with Section 3.5.

### 3.2 Problem Description

Let \(n\) be the number of partitions with a time granularity, e.g., business-week, month, quarter, year, etc., in database \(D\). In the model considered, \(P_i\) denotes the part of the transaction database where \(P_i \subseteq D\). Explicitly, we explore in this chapter the mining of transaction-weighted association rules (abbreviated as weighted association rules), i.e., \((X \Rightarrow Y)^W\), where \(X \Rightarrow Y\) is produced by the concepts of weighted − support and weighted − confidence. Further, instead of using the traditional support threshold \(min\_S^T = [|D| \times min\_supp]\) as a minimum support threshold for each item in Figure ??, a weighted minimum support for mining an association rules is determined by \(min\_S^W = \{\Sigma|P_i| \times W(P_i)\} \times min\_supp\) where \(|P_i|\) and \(W(P_i)\) represent the amount of partial transactions and their corresponding weight values by a weighting function \(W(\cdot)\) in the weighted period \(P_i\) of the database \(D\). Formally, we have the following definitions.

**Definition 3.1:** Let \(N_{P_i}(X)\) be the number of transactions in partition \(P_i\) that contain itemset \(X\). Consequently, the weighted support value of an itemset \(X\) can be formulated as \(S^W(X) = \Sigma N_{P_i}(X) \times W(P_i)\). As a result, the weighted support ratio of an itemset \(X\) is \(supp^W(X) = \frac{S^W(X)}{|P_i| \times W(P_i)}\).
In accordance with Definition 3.1, an itemset $X$ is termed to be frequent when the weighted occurrence frequency of $X$ is larger than the value of $\text{min supp}$ required, i.e., $\text{supp}_W(X) > \text{min supp}$, in transaction set $D$. The weighted confidence of a weighted association rule $(X \Rightarrow Y)^W$ is then defined below.

**Definition 3.2:** $\text{conf}_W^W(X \Rightarrow Y) = \frac{\text{supp}_W(X \cup Y)}{\text{supp}_W(X)}$.

**Definition 3.3:** An association rule $X \Rightarrow Y$ is termed a frequent weighted association rule $(X \Rightarrow Y)^W$ if and only if its weighted support is larger than minimum support required, i.e., $\text{supp}_W(X \cup Y) > \text{min supp}$, and the weighted confidence $\text{conf}_W^W(X \Rightarrow Y)$ is larger than minimum confidence needed, i.e., $\text{conf}_W^W(X \Rightarrow Y) > \text{min conf}$.

**Example 3.2.1:** Recall the illustrative weighted association rules, e.g., $(F \Rightarrow B)^W$ with relative weighted support $\text{supp}_W(F \cup B) = 42.8\%$ and confidence $\text{conf}_W^W(F \Rightarrow B) = 100\%$, in Example 3.1.4. In accordance with Definition 3.3, the implication $(F \Rightarrow B)^W$ is termed as a frequent weighted association rule if and only if $\text{supp}_W(F \cup B) > \text{min supp}$ and $\text{conf}_W^W(F \Rightarrow B) > \text{min conf}$. Consequently, we have to determine if $\text{supp}_W(F) > \text{min supp}$ and $\text{supp}_W(FB) > \text{min supp}$ for discovering the newly identified association rule $(F \Rightarrow B)^W$.

Note that with this weighted association rule $(F \Rightarrow B)^W$, we are able to discover that a new coming product, e.g., a high resolution digital camera, may promote the selling of an existing product, e.g., a color printer. This important information, as pointed out earlier, will not be discovered by the traditional mining schemes, showing the usefulness of this novel model of weighted association rule mining. Once, $\mathcal{F}^W = \{ X \subseteq \mathcal{I} \mid X \text{ is frequent} \}$, the set of all frequent itemsets together with their support values is known, deriving the desired weighted association rules is straightforward. For every $X \in \mathcal{F}^W$, one can simply check the confidence of all rules $(X \Rightarrow Y)^W$ and drop those whose $\frac{\text{supp}_W(X \cup Y)}{\text{supp}_W(X)} < \text{min conf}$ for rule generation. Therefore, in the rest of this chapter we concentrate our discussion on the algorithms for mining frequent itemsets.

It is worth mentioning that there is no restriction imposed on the weighting functions assigned by users. In fact, in addition to the time periods of transactions, other attributes of transactions, such as ownership, transaction length, etc., can also be incorporated into the determination of weights for individual transactions. Further, the application domain of this study is not limited to the association rule mining of a time-variant database. With proper provisions, this technique of mining frequent itemsets can be incorporated into the methods for sequential pattern mining, Web traversal pattern mining, episode mining, etc., to achieve the respective weighted mining.
3.3 Progressive Weighted Mining

It is noted that most of the previous studies, including those in [5, 19, 26, 27, 68, 79, 83], belong to Apriori-like approaches. Basically, an Apriori-like approach is based on an anti-monotone Apriori heuristic [5], i.e., if any itemset of length \( k \) is not frequent in the database, its length \((k + 1)\) super-itemset will never be frequent. The essential idea is to iteratively generate the set of candidate itemsets of length \((k+1)\) from the set of frequent itemsets of length \( k \) (for \( k \geq 1 \)), and to check their corresponding occurrence frequencies in the database. As a result, if the largest frequent itemset is a \( j \)-itemset, then an Apriori-like algorithm may need to scan the database up to \((j + 1)\) times. This is the basic concept of an extended version of Apriori-based algorithm, referred to as \( Apriori^W \), whose performance will be comparatively evaluated with algorithm \( PWM \) in our experimental studies later. In fact, as will be validated by experimental results later, the increase of candidates often causes a drastic increase of execution time and a severe performance degradation, meaning that without utilizing the partitioning and progressive support counting techniques proposed, a direct extension to priori work is not able to handle the weighted association rule mining efficiently.

In [23], the technique of scan-reduction was proposed and shown to result in prominent performance improvement. By scan reduction, \( C_k \) is generated from \( C_{k-1} \times C_{k-1} \) instead of from \( L_{k-1} \times L_{k-1} \). Clearly, a \( C'_3 \) generated from \( C_2 \times C_2 \), instead of from \( L_2 \times L_2 \), will have a size greater than \( |C_3| \) where \( C_3 \) is generated from \( L_2 \times L_2 \). However, if \( |C'_3| \) is not much larger than \( |C_3| \), and both \( C_2 \) and \( C_3 \) can be stored in main memory, we can find \( L_2 \) and \( L_3 \) together when the next scan of the database is performed, thereby saving one round of database scan. It can be seen that using this concept, one can determine all \( L_k \)'s by as few as two scans of the database (i.e., one initial scan to determine \( L_1 \) and a final scan to determine all other frequent itemsets), assuming that \( C'_k \) for \( k \geq 3 \) is generated from \( C'_{k-1} \) and all \( C'_k \) for \( k > 2 \) can be kept in the memory. It will be seen that the progressive mining technique used in algorithm \( PWM \) will enable \( PWM \) to obtain candidate set \( C_k \) with the size very close to that of \( L_k \). This feature of \( PWM \) allows itself of fully utilizing the technique of scan reduction and leads to prominent performance improvement over \( Apriori^W \).

A detailed description of algorithm \( PWM \) is given in Section 3.3.1. We present an illustrative example for the operations of \( PWM \) in Section 3.3.2. The correctness of algorithm \( PWM \) is proved in Section 3.3.3.

3.3.1 Algorithm of \( PWM \)

In general, databases are too large to be held in main memory. Thus, the data mining techniques applied to very large databases have to be highly scalable for efficient execution. As mentioned above,
by partitioning a transaction database into several partitions, algorithm PWM is devised to employ a progressive filtering scheme in each partition to deal with the candidate itemset generation and process one partition at a time. For ease of exposition, the processing of a partition is termed a phase of processing. Explicitly, a progressive candidate set of itemsets is composed of the following two types of candidate itemsets, i.e., (1) the candidate itemsets that were carried over from the previous progressive candidate set in the previous phase and remain as candidate itemsets after the current partition is included into consideration (Such candidate itemsets are called type $\alpha$ candidate itemsets); and (2) the candidate itemsets that were not in the progressive candidate set in the previous phase but are newly identified after only taking the current data partition into account (Such candidate itemsets are called type $\beta$ candidate itemsets). Under PWM, the cumulative information in the prior phases is selectively carried over toward the generation of candidate itemsets in the subsequent phases. After the processing of a phase, algorithm PWM outputs a progressive candidate set of itemsets, their occurrence counts and the corresponding partial supports required.

Initially, a time-variant database $D$ is partitioned into $n$ partitions based on the weighted periods of transactions. The procedure of algorithm PWM is outlined below, where algorithm PWM is decomposed into four sub-procedures for ease of description. $C_2$ is the set of progressive candidate 2-itemsets generated by database $D$. Recall that $N_{P_i}(X)$ is the number of transactions in partition $P_i$ that contain itemset $X$ and $W(P_i)$ is the corresponding weight of partition $P_i$.

**Algorithm PWM ($n$, min_supp) : Progressive Weighted Miner**

**Procedure I: Initial Partition**
1. $|D| = \sum_{i=1,n} |P_i|$;
2. $C_2 = \emptyset$;

**Procedure II: Candidate 2-Itemset Generation**
1. begin for $i = 1$ to $n$  
   // 1st scan of $D$
2. begin for each 2-itemset $X_2 \in P_i$
3. if ( $X_2 \notin C_2$ )
4.   $X_2.count = N_{P_i}(X_2) \times W(P_i)$;
5.   $X_2.start = h$;
6.   if ( $X_2.count \geq \text{min}_\text{supp} \times |P_i| \times W(P_i)$ )
7.     $C_2 = C_2 \cup X_2$;
8.   if ( $X_2 \in C_2$ )
9.     $X_2.count = X_2.count + N_{P_i}(X_2) \times W(P_i)$;
10.    if ( $X_2.count < \text{min}_\text{supp} \times \sum_{m=X_2.start,i}^{m=X_2.start,i+|P_m| \times W(P_m)}$ )
11.       $C_2 = C_2 - X_2$;
12. end
13. end

**Procedure III: Candidate k-Itemset Generation**
1. begin while ($C_k \neq \emptyset$ & $k \geq 2$)
2.   $C_{k+1} = C_k \times C_k$;
3.   $k = k + 1$;
4. end

Procedure IV: Frequent Itemset Generation
1. begin for $i = 1$ to $n$
2. begin for each itemset $X_k \in C_k$
3. $X_k.count = X_k.count + N_{P_i}(X_k) \times W(P_i)$;
4. end
5. begin for each itemset $X_k \in C_k$
6. if $(X_k.count \geq \min _ {supp} \times \sum _ {m=1,n}(|P_m| \times W(P_m)))$
7. $L_k = L_k \cup X_k$;
8. end
9. return $L_k$;

In essence, Procedure II (Candidate 2-Itemset Generation) first scans partition $P_i$, for $i = 1$ to $n$, to find the set of all local frequent 2-itemsets in $P_i$. Note that $C_2$ is a superset of the set of all frequent 2-itemsets in $D$. Algorithm $PWM$ constructs $C_2$ incrementally by adding candidate 2-itemsets to $C_2$ as well as counting the number of occurrences for each candidate 2-itemset $X_2$ in $C_2$. If the cumulative occurrences of a candidate 2-itemset $X_2$ does not meet the partial minimum support required, $X_2$ is removed from the progressive $C_2$. From Step 1 to Step 13 of Procedure II (Candidate 2-Itemset Generation), algorithm $PWM$ processes one partition at a time for all partitions. The number of occurrences of an itemset $X_2$ and its starting partition which keeps its first occurrence in $C_2$ are recorded in $X_2.count$ and $X_2.start$, respectively. As such, in the end of processing $P_i$, an itemset $X_2$ will be kept in $C_2$ only if $X_2.count > \min _ {supp} \times \sum _ {m=1,n}(|P_m| \times W(P_m))$. Next, in Procedure III (Candidate k-Itemset Generation), with the scan reduction scheme [68], $C_2$ produced by the first scan of database is employed to generate $C_k$s in main memory.

Then, from Step 1 to Step 9 of Procedure IV (Frequent Itemset Generation) we begin the second database scan to calculate the support of each itemset in $C_k$ and to find out which candidate itemsets are really frequent itemsets in database $D$. As a result, those itemsets whose $X_k.count \geq \min _ {supp} \times \sum _ {m=1,n}(|P_m| \times W(P_m))$ are the frequent itemsets $L_k$s. As will be proved in Section 3.3.3, the output of algorithm $PWM$ consists of frequent itemsets $L_k$s of database $D$. Finally, according to these output $L_k$s in Step 9, all kinds of weighted association rules implied in database $D$ can be generated in a straightforward manner.

Note that $PWM$ is able to filter out false candidate itemsets in $P_i$ with a hash table. Same as in [68], using a hash table to prune candidate 2-itemsets, i.e., $C_2$, in each accumulative ongoing partition set $P_i$ of transaction database, the CPU and memory overhead of $PWM$ can be further reduced. Owing to the small number of candidate sets generated, the scan reduction technique can be applied efficiently. As a result, only two scans of the database are required.
3.3.2 An illustrative example of algorithm PWM

Recall the transaction database shown in Figure 3.2 where the transaction database $D$ is assumed to be segmented into three partitions $P_1$, $P_2$ and $P_3$, which correspond to the three time granularities from January 2001 to March 2001. Suppose that $\text{min}\_\text{supp} = 30\%$ and $\text{min}\_\text{conf} = 75\%$. In addition, the weight value of each partition is given as follows: $W(P_1) = 0.5$, $W(P_2) = 1$, and $W(P_3) = 2$. The operation of algorithm PWM can be best understood by an illustrative example described below. The flowchart of PWM is given in Figure 3.3.

Specifically, each partition is scanned sequentially for the generation of candidate 2-itemsets in the first scan of the database $D$. After scanning the first segment of 4 transactions, i.e., partition $P_1$, 2-itemsets $\{BD, BC, CD, AD\}$ are sequentially generated as shown in Figure 3.4. In addition, each potential candidate itemset $c \in C_2$ has two attributes: (1) $c.\text{start}$ which contains the partition number of the corresponding starting partition when $c$ was added to $C_2$, and (2) $c.\text{count}$ which contains the number of weighted occurrences of $c$. Since there are four transactions in $P_1$, the partial weighted minimal support is $\text{min}\_\text{SW}(P_1) = 4 \times 0.3 \times 0.5 = 0.6$. Such a partial weighted minimal support is called the filtering threshold. Itemsets whose occurrence counts are below the filtering threshold are removed. Then, as shown in Figure 3.4, only $\{BD, BC\}$, marked by “○”, remain as candidate itemsets (of type $\beta$ in this phase since they are newly generated) whose information is then carried over to the next phase $P_2$ of processing.

Similarly, after scanning partition $P_2$, the occurrence counts of potential candidate 2-itemsets are
After 1st scan database D, we have candidate itemsets: \{B\}, \{C\}, \{E\}, \{F\}, \{BC\}, \{BF\}, \{CE\}

<table>
<thead>
<tr>
<th>Candidates</th>
<th>(N^0(X)) count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{B}</td>
<td>(3\times0.5+2\times1+3\times2=9.5)</td>
</tr>
<tr>
<td>{C}</td>
<td>(2\times0.5+3\times1+1\times2=6)</td>
</tr>
<tr>
<td>{E}</td>
<td>(3\times1+1\times2=5)</td>
</tr>
<tr>
<td>{F}</td>
<td>(3\times2=6)</td>
</tr>
<tr>
<td>{BC}</td>
<td>(2\times0.5+2\times1+1\times2=5)</td>
</tr>
<tr>
<td>{BF}</td>
<td>(3\times2=6)</td>
</tr>
<tr>
<td>{CE}</td>
<td>(2\times1+1\times2=4)</td>
</tr>
</tbody>
</table>

After 2nd scan database D, we have candidate itemsets: \{B\}, \{C\}, \{E\}, \{F\}, \{BC\}, \{BF\}

<table>
<thead>
<tr>
<th>Candidates</th>
<th>(N^0(X)) count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{B}</td>
<td>(9.5)</td>
</tr>
<tr>
<td>{C}</td>
<td>(6)</td>
</tr>
<tr>
<td>{E}</td>
<td>(5)</td>
</tr>
<tr>
<td>{F}</td>
<td>(6)</td>
</tr>
<tr>
<td>{BC}</td>
<td>(5)</td>
</tr>
<tr>
<td>{BF}</td>
<td>(6)</td>
</tr>
</tbody>
</table>

Figure 3.4: Frequent itemsets generation for mining weighted association rules by PWM

recorded (of type \(\alpha\) and type \(\beta\)). From Figure 3.4, it is noted that since there are also 4 transactions in \(P_2\), the filtering threshold of those itemsets carried out from the previous phase (that become type \(\alpha\) candidate itemsets in this phase) is \(min_{SW}(P_1+P_2) = 4 \times 0.3 \times 0.5 + 4 \times 0.3 \times 1 = 1.8\) and that of newly identified candidate itemsets (i.e., type \(\beta\) candidate itemsets) is \(min_{SW}(P_2) = 4 \times 0.3 \times 1 = 1.2\). It can be seen that we have 3 candidate itemsets in \(C_2\) after the processing of partition \(P_2\), and one of them is of type \(\alpha\) and two of them are of type \(\beta\).

Finally, partition \(P_3\) is processed by algorithm PWM. The resulting candidate 2-itemsets are \(C_2 = \{BC, CE, BF\}\) as shown in Figure 3.4. Note that though appearing in the previous phase \(P_2\), itemset \{DE\} is removed from \(C_2\) once \(P_3\) is taken into account since its occurrence count does not meet the filtering threshold then, i.e., \(2 < 3.6\). However, we do have one new itemset, i.e., \{BF\}, which joins the \(C_2\) as a type \(\beta\) candidate itemset. Consequently, we have 3 candidate 2-itemsets generated by PWM, and two of them are of type \(\alpha\) and one of them is of type \(\beta\). Note that only 3 candidate 2-itemsets are generated by PWM.

After generating \(C_2\) from the first scan of database \(D\), we employ the scan reduction technique [68]
and use $C_2$ to generate $C_k$. As discussed earlier, since the $|C_2|$ generated by $PWM$ is very close to the theoretical minimum, i.e., $|L_2|$, the $|C_2'|$ is not much larger than $|C_3|$. Similarly, the $|C_k'|$ is close to $|C_k|$. Since $C_2 = \{BC, CE, BF\}$, no candidate $k$-itemset is generated in this example for $k \geq 3$. Thus, $C_1' = \{B, C, E, F\}$ and $C_2' = \{BC, CE, BF\}$, where all $C_k'$s can be stored in main memory. Then, we can find $L_k$'s ($k = 1, 2, ..., m$) together when the second scan of the database $D$ is performed.

Note that since there is no candidate $k$-itemset ($k \geq 2$) containing $A$ or $D$ in this example, $A$ and $D$ are not necessary taken as potential itemsets for generating weighted association rules. In other words, we can skip them from the set of candidate itemsets $C_k'$s. Finally, all occurrence counts of $C_k'$s can be calculated by the second database scan. As shown in Figure 3.5, after all frequent $k$-itemsets are identified, the corresponding weighted association rules can be derived in a straightforward manner. Explicitly, the weighted association rule of $(X \Rightarrow Y)^W$ holds if $conf^W(X \Rightarrow Y) \geq \text{min}\_conf$.

### 3.3.3 Correctness of $PWM$

We now prove the correctness of algorithm $PWM$. Let $db^{i,j}$ denote the part of the transaction database formed by a continuous region from partition $P_i$ to partition $P_j$, and the number of transactions in $db^{i,j}$ is $|db^{i,j}| = \sum_{h=i,j} |P_h|$. We can then define the region ratio of an itemset as follows.

**Definition 3.4:** A region ratio of an itemset $X$ for the transaction database $db^{i,j}$, denoted by $r_{i,j}(X)$, is:

$$r_{i,j}(X) = \frac{\sum_{h=i,j} N_{P_h}(X)}{|db^{i,j}| \times W(P_h)}.$$

In essence, the region ratio of an itemset is the support of that itemset if only the part of transaction database $db^{i,j}$ is considered.

**Lemma 3.1:** A 2-itemset $X_2$ remains in the $C_2$ after the processing of partition $P_j$ if and only if there exists an $i$ such that for any integer $t$ in the interval $[i, j]$, $r_{i,t}(X_2) \geq \text{min}_SW(db^{i,t})$, where $\text{min}_SW(db^{i,t})$ is the minimal weighted support required.

**Proof of Lemma 1:** We shall prove the “if” condition first. Consider the following two cases. First, suppose the 2-itemset $X_2$ is not in the progressive candidate set before the processing of partition $P_i$. Since $r_{i,t}(X_2) \geq \text{min}_SW(P_i)$, itemset $X_2$ will be selected as a type $\beta$ candidate itemset by $PWM$ after the processing of partition $P_i$. On the other hand, if the itemset $X_2$ is already in the progressive candidate set before the processing of partition $P_i$, itemset $X_2$ will remain as a type $\alpha$ candidate itemset by $PWM$. Clearly, for the above two cases, itemset $X_2$ will remain in $C_2$ throughout the processing from $P_i$ to $P_j$ since for any integer $t$ in the interval $[i, j]$, $r_{i,t}(X_2) \geq \text{min}_SW(db^{i,t})$.

We now prove the “only if” condition, i.e., if $X_2$ remains in $C_2$ after the processing of partition $P_j$ then there exists an $i$ such that for any $t$ in the interval $[i, j]$, $r_{i,t}(X_2) \geq \text{min}_SW(db^{i,t})$. Note that itemset $X_2$ can be either type $\alpha$ or type $\beta$ candidate itemset in the $C_2$ after the processing of partition $P_j$. Suppose $X_2$ is a type $\beta$ candidate itemset there, then this implication follows by setting $j = i$ since

55
Figure 3.5: The weighted association rule generation from frequent itemsets

Let $r_{i,j}(X_2) \geq \min_S W (db^{i,j})$. On the other hand, suppose that $X_2$ is a type $\alpha$ candidate itemset after the processing of $P_j$, which means itemset $X_2$ has become a type $\beta$ candidate itemset in a previous phase. Then, we shall trace backward the type of itemset $X_2$ from partition $P_j$ (i.e., looking over $P_j$, $P_{j-1}$, $P_{j-2}$ and so forth) until the partition that records itemset $X_2$ as a type $\beta$ candidate itemset is first encountered. (It should be noted that there could be two discontinuous regions that record itemset $X_2$ in the $C_2$, which means that an itemset may get on and off the progressive candidate set through the processing of partitions. This in turn means that an itemset may appear as a type $\beta$ candidate itemset more than once.) By referring the partition identified above as partition $P_i$, we have, for any $t$ in the interval $[i, j]$, $r_{i,t}(X_2) \geq \min_S W (db^{i,t})$, completing the proof of this lemma. Q.E.D.

Lemma 3.2 leads to the following theorem which states the correctness of algorithm PWM.

**Theorem 3.1:** If an itemset $X$ is a frequent itemset, then $X$ will be in the candidate set of itemsets produced by algorithm PWM.

**Proof of Theorem 3.1:** Let $n$ be the number of partitions of the transaction database $D$. Since the itemset $X$ is a frequent itemset, we have $r_{1,n}(X) \geq \min_S W (db^{i,j})$, which is in essence a special case of Lemma 3.2 for $i = 1$ and $j = n$, proving this theorem. Q.E.D.
Figure 3.6: Relative performance studies

It follows from Theorem 3.1 that when $W(\cdot) = 1$, the frequent itemsets generated by $PWM$ will be the same as those produced by the association rule mining algorithms.

3.4 Performance Studies

To assess the performance of algorithm $PWM$, we performed several experiments on a computer with a CPU clock rate of 450 MHz and 512 MB of main memory. The methods used to generate synthetic data are described in Section 3.4.1. The performance comparison of $PWM$ and $Apriori^W$ is presented in Section 3.4.2. Section 3.4.3 shows the I/O cost and CPU overhead for $PWM$ and $Apriori^W$. Results on scaleup experiments are presented in Section 3.4.4.

3.4.1 Generation of synthetic workload

For obtaining reliable experimental results, the method to generate synthetic transactions we employed in this study is similar to the ones used in prior works [7, 68]. Explicitly, we generated several different
transaction databases from a set of potentially frequent itemsets to evaluate the performance of PWM. Note that the efficiency of algorithm PWM has been evaluated by some real databases, such as bookstore transaction databases and grocery sales data. However, we show the experimental results from synthetic transaction data so as to obtain results of different workload parameters. Each database consists of $|D|$ transactions, and on the average, each transaction has $|T|$ items. The average size of maximal potentially frequent itemsets is given by the value of $|I|$. In accordance with the weighted periods of transactions, database $D$ is assumed to be divided into $n$ partitions. We use the notation $T_x - I_y - D_m$ to represent a database in which $D = m$ thousands, $|T| = x$, and $|I| = y$. We compare relative performance of Apriori$^W$ and PWM.

As mentioned before, Apriori$^W$ algorithm is an extended version of Apriori-like algorithms to deal with the mining problem in a sequence database. As will be shown by our experimental results, with the progressive filtering technique that carries cumulative information selectively, the execution time of PWM is, in orders of magnitude, smaller than that required by Apriori$^W$.

3.4.2 Experiment one: Relative performance

We first conducted several experiments to evaluate the relative performance of Apriori$^W$ and PWM. Since the experimental results are consistent for various values of $n$, $|L|$ and $N$ on datasets, we only report the results on $n = 10$, $|L| = 2000$ and $N = 10000$ in the following experiments for interest of space. Without loss of generality, the weighting function is assumed to be $W(i) = 1 + \frac{i}{n}$ where $1 \leq i \leq n$. Note that as pointed out earlier, there is essentially no restriction on the form of weighting functions. Figure 3.6 shows the relative execution times for both two algorithms as the minimum support threshold decreases from 1% support to 0.1% support. When the support threshold is high, there are only a limited number of frequent itemsets produced. However, as the support threshold decreases, the performance difference becomes prominent in that PWM significantly outperforms Apriori$^W$. Explicitly, PWM is in orders of magnitude faster than Apriori$^W$, and the margin grows as the minimum support threshold decreases. In fact, PWM outperforms Apriori$^W$ in both CPU and I/O costs, which are evaluated next.

3.4.3 Experiment two: Evaluation of I/O cost and CPU overhead

To evaluate the corresponding of I/O cost, same as in [71], we assume that each sequential read of a byte of data consumes one unit of I/O cost and each random read of a byte of data consumes ten units of I/O cost. The I/O costs of Apriori$^W$ and PWM over the data set $T10 - I4 - D200$ are shown in Figure 3.7a, where it can be seen that PWM significantly outperforms Apriori$^W$. Note that the large amount of database scans causes the performance bottleneck when the database size does not fit into main memory. In view of this, PWM is advantageous since only two scans of the transaction database
3.4.4 Experiment three: Scaleup performance

In this experiment, we examine the scaleup performance of algorithm PWM. The scale-up results for different selected datasets are obtained. Figure 3.8 shows the scaleup performance of algorithm PWM as the values of \(|D|\) increase. Three different minimum supports are considered. We obtained the results for the dataset T10 – I4 – Dm when the number of customers increases from 100,000 to one million. The execution times are normalized with respect to the times for the 100,000 transactions dataset in the Figure 3.8a. Note that, as shown in Figure 3.8a the execution time only slightly increases with the growth of the database size, showing good scalability of PWM.

To further understand the impact of \(|D|\) to the relative performance of algorithms AprioriW and
Figure 3.8: Scaleup performance of $PWM$ and the execution time ratio between $PWM$ and $Apriori^W$. 

We conduct the scaleup experiments for both $PWM$ and $Apriori^W$ with two minimum support thresholds 0.1% and 0.2%. The results are shown in Figure 3.8b where the value in $y$-axis corresponds to the ratio of the execution time of $PWM$ to that of $Apriori^W$. Figure 3.8b shows the referenced ratio obtained from a time-variant database over datasets of $T_{10-I4-Dm}$. The execution-time-ratio of $PWM$ to $Apriori^W$ decreases when the amount of database $|D|$ grows larger, meaning that the advantage of $PWM$ over $Apriori^W$ increases as the database size increases.

### 3.5 Summary

In this chapter, we not only explored a new model of mining weighted association rules, i.e., $(X \Rightarrow Y)^W$, in a transaction database but also developed algorithm $PWM$ to generate the weighted association rules as well as conducted related performance studies. In algorithm $PWM$, the importance of each transaction period was first reflected by a proper weight assigned by the user. Then, $PWM$ partitioned the time-variant database in light of weighted periods of transactions and performed weighted mining. Algorithm $PWM$ was designed to progressively accumulate the itemset counts based on the intrinsic partitioning characteristics and employed a filtering threshold in each partition to early prune out those cumulatively infrequent 2-itemsets. With this design, algorithm $PWM$ was able to efficiently produce weighted association rules for applications where different time periods were assigned with different weights, leading to more interesting results. The correctness of $PWM$ was proved and some of its theoretical properties were derived. Extensive experimental studies have been conducted to evaluate performance of algorithm $PWM$. 
Chapter 4

Progressive Partition Miner: An Efficient Algorithm for Mining General Temporal Association Rules

4.1 Introduction

The discovery of association relationship among a huge database has been known to be useful in selective marketing, decision analysis, and business management [22, 40]. A popular area of applications is the market basket analysis, which studies the buying behaviors of customers by searching for sets of items that are frequently purchased together (or in sequence). Let $\mathcal{I} = \{x_1, x_2, ..., x_m\}$ be a set of items. A set $X \subseteq \mathcal{I}$ with $k = |X|$ is called a $k$-itemset or simply an itemset. Let a database $D$ be a set of transactions, where each transaction $T$ is a set of items such that $T \subseteq \mathcal{I}$. A transaction $T$ is said to support $X$ if and only if $X \subseteq T$. Conventionally, an association rule is an implication of the form $X \implies Y$, meaning that the presence of the set $X$ implies the presence of another set $Y$ where $X \subset \mathcal{I}$, $Y \subset \mathcal{I}$ and $X \cap Y = \phi$. The rule $X \implies Y$ holds in the transaction set $D$ with confidence $c$ if $c\%$ of transactions in $D$ that contain $X$ also contain $Y$. The rule $X \implies Y$ has support $s$ in the transaction set $D$ if $s\%$ of transactions in $D$ contain $X \cup Y$.

For a given pair of confidence and support thresholds, the problem of mining association rules is to identify all association rules that have confidence and support greater than the corresponding minimum support threshold (denoted as \textit{min\_supp}) and minimum confidence threshold (denoted as \textit{min\_conf}). Association rule mining algorithms [5] work in two steps: (1) generate all frequent itemsets that satisfy \textit{min\_supp}; (2) generate all association rules that satisfy \textit{min\_conf} using the frequent itemsets. This problem can be reduced to the problem of finding all frequent itemsets for the same support threshold.

Since the early work in [5], several efficient algorithms to mine association rules have been developed in recent years. These studies cover a broad spectrum of topics including: (1) fast algorithms based
on the level-wise Apriori framework [7, 68], partitioning [53, 75], and sampling [83]; (2) TreeProjection [1] and FP-growth algorithms [36, 37, 38, 70]; (3) incremental updating [11, 26, 49, 82] and parallel algorithms [6, 67]; (4) mining of generalized and multi-level rules [33, 79]; (5) mining of quantitative rules [80]; (6) mining of multi-dimensional rules [64, 88, 90]; (7) constraint-based rule mining [35, 46, 45, 69, 85] and multiple minimum supports issues [55, 87]; (8) associations among correlated or infrequent items [31]; and (9) temporal association rule discovery [9, 14, 25, 24, 81].

While these are important results toward enabling the integration of association mining and fast searching algorithms, e.g., BFS and DFS which are classified in [40], we note that these mining methods cannot effectively be applied to the mining of a publication-like database which is of increasing popularity recently. In essence, a publication database is a set of transactions where each transaction \( T \) is a set of items of which each item contains an individual exhibition period. The current model of association rule mining is not able to handle the publication database due to the following fundamental problems, i.e., (1) lack of consideration of the exhibition period of each individual item; (2) lack of an equitable support counting basis for each item. Note that the traditional mining process takes the same task-relevant tuples, i.e., the size of transaction set \( D \), as a counting basis. Recall that the task of support specification is to specify the minimum transaction support for each itemset. However, since different items have different exhibition periods in a publication database, only considering the occurrence count of each item might not lead to a fair measurement. This problem can be further explained by the two illustrative examples below.

**Example 4.1.1:** Consider an illustrative database of publications in a bookstore as shown in Figure 4.1. Note that items \( A \) and \( B \) are exhibited from 1990 to 2001. However, item \( C \) is exhibited from 1992 to 2001 and item \( D \) is from 1994 to 2001. Even though each transaction item has a unique exhibition period, conventional mining algorithms, without further provision, tend to ignore such differences and determine frequent association rules with the same counting basis \( D \).

**Example 4.1.2:** In a bookstore transaction database as shown in Figure 4.2, the minimum transaction support and confidence are assumed to be \( \text{min\_supp} = 30\% \) and \( \text{min\_conf} = 75\% \), respectively. A set of time series database indicates the transaction records from January 2001 to March 2001. The publication date of each transaction item is also given. Based on the traditional mining techniques, the absolute support threshold is denoted as \( S^A = [12 \times 0.3] = 4 \) where 12 is the size of transaction set \( D \). It can be seen that only \( \{B, C, D, E, BC\} \) can be termed as frequent itemsets since the amounts of their occurrences in this transaction database are respectively larger than the absolute value of support threshold. Thus, only rule \( C \rightarrow B \) is termed as a frequent association rule with support \( s = 41.67\% \) and confidence \( c = 83.33\% \). However, some phenomena are observed when we take the “item information” in Figure 4.2 into consideration.
Figure 4.1: An illustrative publication database where different items may have different publication dates

<table>
<thead>
<tr>
<th>Date</th>
<th>TID</th>
<th>Itemset</th>
<th>Publication Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan-01</td>
<td>t₁</td>
<td>B D</td>
<td>Jan-95</td>
</tr>
<tr>
<td></td>
<td>t₂</td>
<td>B C D</td>
<td></td>
</tr>
<tr>
<td></td>
<td>t₃</td>
<td>B C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>t₄</td>
<td>A D</td>
<td></td>
</tr>
<tr>
<td>Feb-01</td>
<td>t₅</td>
<td>B C E</td>
<td></td>
</tr>
<tr>
<td></td>
<td>t₆</td>
<td>D E</td>
<td></td>
</tr>
<tr>
<td></td>
<td>t₇</td>
<td>A B C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>t₈</td>
<td>C D E</td>
<td></td>
</tr>
<tr>
<td>Mar-01</td>
<td>t₉</td>
<td>B C E F</td>
<td></td>
</tr>
<tr>
<td></td>
<td>t₁₀</td>
<td>B F</td>
<td></td>
</tr>
<tr>
<td></td>
<td>t₁₁</td>
<td>A D</td>
<td></td>
</tr>
<tr>
<td></td>
<td>t₁₂</td>
<td>B D F</td>
<td></td>
</tr>
</tbody>
</table>

Transaction Database

Item Information

<table>
<thead>
<tr>
<th>Item</th>
<th>Publication Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Jan-95</td>
</tr>
<tr>
<td>B</td>
<td>Apr-96</td>
</tr>
<tr>
<td>C</td>
<td>Jul-97</td>
</tr>
<tr>
<td>D</td>
<td>Aug-00</td>
</tr>
<tr>
<td>E</td>
<td>Feb-01</td>
</tr>
<tr>
<td>F</td>
<td>Mar-01</td>
</tr>
</tbody>
</table>

Figure 4.2: An illustrative transaction database and the corresponding item information
1. **An early publication intrinsically possesses a higher likelihood to be determined as a frequent itemset.** For example, the sales volume of an early product, such as A, B, C or D, is likely to be larger than that of a newly exhibited product, e.g., E or F, since an early product has a longer exhibition period. As a result, the association rules we usually get will be those with long-term products such as “milk and bread are frequently purchased together”, which, while being correct by the definition, is of less interest to us in the association rule mining. In contrast, some more recent products, such as new books, which are really “frequent” and interesting in their exhibition periods are less likely to be identified as frequent ones if a traditional mining process is employed.

2. **Some discovered rules may be expired from users’ interest.** Considering the generated rule \(C \Rightarrow B\), both B and C were published from the very early dates of this mining transaction database. This information is very likely to have been explored in the previous mining database, such as the one from January 1996 to December 1997. Such mining results could be of less interest to our on-going mining works. For example, most researchers tend to pay more attention to the latest published papers.

Note that one straightforward approach to addressing the above issues is to lower the value of the minimum support threshold required. However, this naive approach will cause another problem, i.e., those interesting rules with smaller supports may be overshadowed by lots of less important information with higher supports. As a consequence, we introduce the notion of *exhibition period* for each transaction item in this chapter and develop an algorithm, Progressive Partition Miner (abbreviated as PPM), to address this problem. It is worth mentioning that the application domain of this study is not limited to the mining of a publication database. Other application domains include bookstore transaction databases, video and audio rental store records, stock market data, and transactions in electronic commerce, to name a few.

Explicitly, we explore in this chapter the mining of *general temporal association rules*, i.e., \((X \Rightarrow Y)^{t,n}\), where \(t\) is the *latest-exhibition-start time* of both itemsets \(X\) and \(Y\), and \(n\) denotes the *end time* of the publication database. In other words, \((t, n)\) is the *maximal common exhibition period* of itemsets \(X\) and \(Y\). An association rule \(X \Rightarrow Y\) is termed to be a frequent general temporal association rule \((X \Rightarrow Y)^{t,n}\) if and only if its probability is larger than minimum support required, i.e., \(P(X^{t,n} \cup Y^{t,n}) > \text{min\_supp}\), and the conditional probability \(P(Y^{t,n} | X^{t,n})\) is larger than minimum confidence needed, i.e., \(P(Y^{t,n} | X^{t,n}) > \text{min\_conf}\). Instead of using the absolute support threshold \(S^A = |D| * \text{min\_supp}\) as a minimum support threshold for each item in Figure 4.2, a *relative* minimum support, denoted by \(S^R_X = |D_X| * \text{min\_supp}\) where \(|D_X|\) indicates the amount of partial transactions.
in the exhibition period of itemset $X$, is given to deal with the mining of temporal association rules.

**Example 4.1.3:** Let us follow Example 4.1.2 and the given minimum support and confidence thresholds. According to this newly identified support threshold $S_X^R$, we have temporal association rules as follows:

1. $(C \Rightarrow E)^{2,3}$ with relative support 37.5% and confidence 75%;
2. $(E \Rightarrow C)^{2,3}$ with relative support 37.5% and confidence 75%;
3. $(B \Rightarrow F)^{3,3}$ with relative support 75% and confidence 100%;
4. $(F \Rightarrow B)^{3,3}$ with relative support 75% and confidence 100%.

To deal with the mining of general temporal association rule $(X \Rightarrow Y)^{t,n}$, an efficient algorithm, **Progressive Partition Miner**, is devised. The basic idea of **PPM** is to first partition the publication database in light of exhibition periods of items and then progressively accumulate the occurrence count of each candidate 2-itemset based on the intrinsic partitioning characteristics. Algorithm **PPM** is also designed to employ a filtering threshold in each partition to early prune out those cumulatively infrequent 2-itemsets. The feature that the number of candidate 2-itemsets generated by **PPM** is very close to the number of frequent 2-itemsets allows us to employ the scan reduction technique by generating $C_k$s from $C_2$ directly to effectively reduce the number of database scan. Experimental results show that **PPM** produces a significantly smaller amount of candidate 2-itemsets than **Apriori**+, i.e., an extended version of **Apriori** algorithm. In fact, the number of the candidate itemsets $C_k$s generated by **PPM** approaches to its theoretical minimum, i.e., the number of frequent $k$-itemsets, as the value of the minimal support increases. Explicitly, the execution time of **PPM** is, in orders of magnitude, smaller than those required by **Apriori**+. Sensitivity analysis on various parameters of the database is also conducted to provide many insights into algorithm **PPM**. The advantage of **PPM** over **Apriori**+ becomes even more prominent as the size of the database increases. This is indeed an important feature for **PPM** to be practically used for the mining of a time series database in the real world.

It is worth mentioning that the problem of mining general temporal association rules will be degenerated to the one of mining temporal association rules explored in prior works [9, 14, 24, 25, 81] if the exhibition period $(t, n)$ of association rule $(X \Rightarrow Y)^{t,n}$ is applied to a none-maximal exhibition period of $X \Rightarrow Y$, such as $(j, n)$ where $j > t$. Consider for example the database in Figure 4.2 where $(C \Rightarrow B)^{1,3}$ and $(C \Rightarrow E)^{2,3}$ are two general temporal association rules in database $D$ while the temporal subset of $(C \Rightarrow B)^{1,3}$, e.g., $(C \Rightarrow B)^{2,3}$, can also be a temporal association rule as defined before [9, 14, 24, 25, 81], showing that the model we consider can be viewed as a general framework of prior studies. This is the very reason we use the term “general temporal association rule” in this chapter.

We mention in passing that the works in [53, 62, 75] are essentially based on a partition-based heuristic, i.e, *if $X$ is a frequent itemset in database $D$, which is divided into $n$ partitions $p_1, p_2, ..., p_n$,*
then $X$ must be a frequent itemset in at least one of the $n$ partitions. However, these works were not applicable to handling the exhibition period of transaction items on mining association rules. In addition, the Frequent Pattern growth (FP-growth), which constructs a highly compact data structure (an FP-tree) to compress the original transaction database, is a method of mining frequent itemsets without candidate generation [36, 37, 38, 70]. However, in our opinion, FP-growth algorithms do not have obvious extensions to deal with this publication database problem, nor do those constraint-based rule mining methods that allow users to focus the search for rules by providing meta-rules [35, 46, 45, 69, 85]. Further, some methodologies were proposed to explore the problem of discovering temporal association relationship in the partial of database retrieved [9, 14, 24, 25, 81], i.e., to determine association rules from a given subset of database specified by time. These works, however, do not consider the individual exhibition period of each transaction item, and are thus not applicable to solving the mining problems in a publication database. On the other hand, some techniques were devised to use multiple minimum supports for frequent itemsets generation [55, 87]. However, it remains an open issue for how these techniques to be coupled with the corresponding minimum confidence thresholds when general temporal association rules we consider in this chapter in a publication database are being generated. In this chapter, we not only explore the new model of general temporal association rules in a publication database, but also propose an efficient Progressive Partition Miner methodology to perform the mining for this problem as well as conduct the corresponding performance studies. These features distinguish this dissertation from others.

The rest of this chapter is organized as follows. Problem description is given in Section 4.2. Algorithm PPM is described in Section 4.3 with its correctness proved. Performance studies on various schemes are conducted in Section 4.4. This chapter concludes with Section 4.5.

### 4.2 Problem Description

To facilitate our presentation, some definitions and symbols used are presented in Section 4.2.1. For further looking into the proposed problem of mining temporal association rules, the traversing of the search space is examined in Section 4.2.2. In addition, to assess the performance of PPM, we also present in Section 4.2.2 the concept of an extended version of Apriori algorithm, called Apriori$^+$, which will be employed in Section 4.4 later for performance comparison.

#### 4.2.1 Preliminaries

Let $n$ be the number of partitions with a time granularity, e.g., business-week, month, quarter, year, to name a few, in database $D$. In the model considered, $db^{t,n}$ denotes the part of the transaction database formed by a continuous region from partition $P_t$ to partition $P_n$, and $|db^{t,n}|=\sum_{h=t,n}|P_h|$
where \( \text{db}^{t,n} \subseteq D \). An item \( x^{t,\text{start},n} \) is termed as a temporal item of \( x \), meaning that \( P_{x,\text{start}} \) is the starting partition of \( x \) and \( n \) is the partition number of the last database partition retrieved.

**Example 4.2.1:** Consider the database in Figure 4.2. Since database \( D \) records the transaction data from January 2001 to March 2003, database \( D \) is intrinsically segmented into three partitions \( P_1, P_2 \) and \( P_3 \) in accordance with the “month” granularity. As a consequence, a partial database \( \text{db}^{2,3} \subseteq D \) consists of partitions \( P_2 \) and \( P_3 \). A temporal item \( E^{2,3} \) denotes that the exhibition period of \( E^{2,3} \) is from the beginning time of partition \( P_2 \) to the end time of partition \( P_3 \).

As such, we can define a maximal temporal itemset \( X^{t,n} \) as follows.

**Definition 4.1:** An itemset \( X^{t,n} \) is called a maximal temporal itemset in a partial database \( \text{db}^{t,n} \) if \( t \) is the latest starting partition number of all items belonging to \( X \) in database \( D \) and \( n \) is the partition number of the last partition in \( \text{db}^{t,n} \) retrieved.

For example, as shown in Figure 4.2, itemset \( DE^{2,3} \) is deemed a maximal temporal itemset whereas \( CD^{2,3} \) is not. In view of this, the exhibition period of an itemset is expressed in terms of Maximal Common exhibition Period (MCP) of the items that appear in the itemset. Let \( \text{MCP}(x) \) denote the MCP value of item \( x \). The MCP value of an itemset \( X \) is the shortest MCP among the items in itemset \( X \). Consider three items \( C, E \) and \( F \) in Figure 4.2 for example. Their exhibition periods are as follows: \( \text{MCP}(C) = (1, 3) \), \( \text{MCP}(E) = (2, 3) \) and \( \text{MCP}(F) = (3, 3) \). Since itemset \( CEF \) is termed to be \( CEF^{3,n} = (CEF)^{3,n} \) with considering the exhibition of \( CEF \), we have \( \text{MCP}(CEF) = (3, 3) \).

In addition, \( |\text{db}^{t,n}| \) is the number of transactions in the partial database \( \text{db}^{t,n} \). The fraction of transaction \( T \) supporting an itemset \( X \) with respect to partial database \( \text{db}^{t,n} \) is called the support of \( X^{t,n} \), i.e., \( \text{supp}(X^{\text{MCP}(X)}) = \frac{|\{T \in \text{db}^{\text{MCP}(X)}|X \subseteq T\}|}{|\text{db}^{\text{MCP}(X)}|} \). The support of a rule \( (X \Rightarrow Y)^{\text{MCP}(XY)} \) is defined as \( \text{supp}((X \Rightarrow Y)^{\text{MCP}(XY)}) = \text{supp}((X \cup Y)^{\text{MCP}(XY)}) \). The confidence of this rule is defined as \( \text{conf}((X \Rightarrow Y)^{\text{MCP}(XY)}) = \frac{\text{supp}(X \cup Y)^{\text{MCP}(XY)}}{\text{supp}(X^{\text{MCP}(XY)})} \). Consequently, a general temporal association rule \( (X \Rightarrow Y)^{\text{MCP}(XY)} \) which holds in the transaction set \( D \) can be defined as follows.

**Definition 4.2:** An association rule \( (X \Rightarrow Y)^{\text{MCP}(XY)} \) is called a general temporal association rule in the transaction set \( D \) with \( \text{conf}((X \Rightarrow Y)^{\text{MCP}(XY)}) = c \) and \( \text{supp}((X \Rightarrow Y)^{\text{MCP}(XY)}) = s \) if \( c \% \) of transactions in \( \text{db}^{\text{MCP}(XY)} \) that contain \( X \) also contain \( Y \) and \( s \% \) of transactions in \( \text{db}^{\text{MCP}(XY)} \) contain \( X \cup Y \) while \( X \cap Y = \phi \).

For a given pair of \( \text{min}_\text{conf} \) and \( \text{min}_\text{supp} \) as the minimum thresholds required in the maximal common exhibition period of each association rule, the problem of mining general temporal association rules is to determine all frequent general temporal association rules, e.g., \( (X \Rightarrow Y)^{\text{MCP}(XY)} \in \text{db}^{\text{MCP}(XY)} \) which transaction itemsets \( X \) and \( Y \) have “relative” support and confidence greater than
the corresponding thresholds. Thus, we have the following definition to identify the frequent general temporal association rules.

**Definition 4.3:** A general temporal association rule \((X \implies Y)^{MCP(XY)}\) is termed to be frequent if and only if \(supp((X \implies Y)^{MCP(XY)}) > \text{min\_supp}\) and \(conf((X \implies Y)^{MCP(XY)}) > \text{min\_conf}\).

Consequently, this rule mining of general temporal association can also be decomposed into three steps:

1. Generate all frequent maximal temporal itemsets \((TI)\) with their support values.
2. Generate the support values of all corresponding temporal sub-itemsets \((SI)\) of frequent \(TI\)s.
3. Generate all temporal association rules that satisfy \text{min\_conf}\) using the frequent \(TI\)s and/or \(SI\)s.

**Example 4.2.2:** Recall the illustrative general temporal association rules, e.g., \((C \implies E)^{2,3}\) with relative support 37.5% and confidence 75%, in Example 4.1.3. In accordance with Definition 4.3, the implication \((C \implies E)^{2,3}\) is termed as a general temporal association rule if and only if \(supp((C \implies E)^{2,3}) > \text{min\_supp}\) and \(conf((C \implies E)^{2,3}) > \text{min\_conf}\). Consequently, we have to determine if \(supp(CE^{2,3}) > \text{min\_supp}\) and \(supp(C^{2,3}) > \text{min\_supp}\) for discovering the newly identified association rule \((C \implies E)^{2,3}\). It is worth mentioning that though \(CE^{2,3}\) has to be a maximal temporal itemset, called \(TI\), \(C^{2,3}\) may not be a \(TI\). We call \(C^{2,3}\) is one of corresponding temporal sub-itemsets, i.e., \(SI\), of itemset \(CE^{2,3}\).

For better readability, a list of symbols used is given in Table 4.1. Then, the definition of a frequent maximal temporal itemset and the property of its corresponding sub-itemsets are given below.

**Definition 4.4:** A maximal temporal itemset \(X^{MCP(X)}\) is termed to be frequent when the occurrence frequency of \(X^{MCP(X)}\) is larger than the value of \(\text{min\_supp}\) required, i.e., \(supp(X^{MCP(X)}) > \text{min\_supp}\), in transaction set \(db^{MCP(X)}\).

**Property 4.1:** When a maximal temporal \(k\)-itemset \(X^{MCP(X_k)}\) is frequent in data set \(db^{MCP(X_k)}\), each of its corresponding sub-itemset \(X^{MCP(X_k)}_i\) \((1 \leq i < k)\) is also frequent in \(db^{MCP(X_k)}\).

Once, \(\mathcal{F} = \{X^{MCP(X)} \subseteq \mathcal{I} | X^{MCP(X)}\text{ is frequent}\}\), the set of all frequent \(TI\)s and \(SI\)s together with their support values is known, deriving the desired association rules is straightforward. For every \(X^{MCP(X)} \in \mathcal{F}\), check the confidence of all rules \((X \implies Y)^{MCP(XY)}\) and drop those that do not satisfy \(s(XY^{MCP(XY)})/s(X^{MCP(XY)}) \geq \text{min\_conf}\). This problem can also be reduced to the problem of finding all frequent maximal temporal itemsets first and then generating their corresponding frequent sub-itemsets for the same support threshold. Therefore, in the rest of this chapter we concentrate our
The partial database of $D$ formed by a continuous region from $P_i$ to $P_n$.

$|db_{i,n}|$ Number of transactions in $db_{i,n}$

$X_{i,n}$ A temporal itemset in partial database $db_{i,n}$

$MCP(\cdot)$ The maximal common exhibition period of an itemset

$(X \implies Y)_{MCP(XY)}$ A general temporal association rule

$supp((X \implies Y)_{t,n})$ The support of $X \implies Y$ in partial database $db_{t,n}$

$conf((X \implies Y)_{t,n})$ The support of $X \implies Y$ in partial database $db_{t,n}$

$min\_supp$ Minimum support threshold required

$min\_conf$ Minimum confidence threshold required

$min\_leng$ Minimum length of exhibition period required

$TI$ A maximal temporal itemset

$SI$ A corresponding temporal sub-itemset of $TI$

### Table 4.1: Meanings of symbols used in the mining of general temporal association rules

In fact, the process steps of generating frequent TIs and SIs can be further merged to one step in our proposed algorithm $PPM$.

In addition, it is noted that users are likely to be interested in association rules whose exhibition periods are longer than a certain period. In view of this, we introduce a parameter of the minimum length of the exhibition period, denoted by $min\_leng$, as a constraint in rule generation to reflect such a users’ requirement in the exhibition period. In other words, for each general temporal association rule $(X \implies Y)_{t,n}^{MCP(XY)}$ produced, the value of $MCP(XY)$ should be larger than that of $min\_leng$ required, i.e., $MCP(XY) > min\_leng$.

#### 4.2.2 Traversing the Search Space

As explained, we have to find all maximal temporal itemsets that satisfy $min\_supp$ first and then to calculate the occurrences of their corresponding sub-itemsets for producing all temporal association rules hidden in database $D$. However, if we use an existing algorithm to find all frequent TIs for this new problem, the downward closure property, which Apriori-based algorithms are based on, no longer holds. In addition, the candidate generation process is not intuitive at all. Note that, even though itemset $X_{t,n}$ is not a frequent itemset, it does not imply that $X_{t+1,n}$, i.e., a temporal sub-itemset of $X_{t,n}$, is not a frequent itemset. In other words, even knowing $X_{t,n}$ is not frequent in $db_{t,n}$ where $MCP(X) = (t, n)$, we are not able to assert whether $XY_{t+1,n}$ is frequent or not when $MCP(Y) = (t + 1, n)$. Specifically, to determine whether a general temporal association rule $(X \implies Y)_{t+1,n}^{MCP(XY)}$ is frequent, we have to find out the support values of $X_{t+1,n}$ and $XY_{t+1,n}$ where $MCP(XY) = MCP(Y) = (t + 1, n)$.

**Example 4.2.3:** Consider $MCP(x_1) = (1, n)$, $MCP(x_2) = (2, n)$ and $MCP(x_3) = (3, n)$. If we find that item $x_1$ is not frequent at exhibition period $(1, n)$, then it does not satisfy $min\_supp$ requirement at level 1. Under a conventional Apriori-based association rule mining algorithm, this itemset is discarded.
Figure 4.3: Traversing the search space for existing algorithms, e.g., Apriori-like algorithms

since it will not be frequent. The potentially frequent itemsets \( x_1x_2 \) and \( x_1x_3 \) will then not be generated at level 2 for consideration. Clearly, this disposition is incorrect in mining general temporal association rules since \( x_1 \) is still possible to be frequent at \((2, n)\) and \((3, n)\), indicating that the downward property is not valid in mining general temporal association rules.

It is worth mentioning that one may deal with the problem we consider with two naive procedures. The first one is to process the conventional mining algorithms in all kinds of combinatorial sub-databases, e.g., \( db_{1,3}, db_{2,3} \) and \( db_{3,3} \) in the foregoing example in Figure 4.2, separately. However, due to the huge search space involved, looking at all subsets of \( I, i.e., db_{t,n} \) for \( 1 \leq t \leq n \), is too costly for this approach to be practically used.

Further, since the downward level-wise property, which holds for Apriori-like algorithms, is not valid in this general temporal association rule mining problem, the second method is to expand each transaction item to be its combination with different exhibition periods. For instance, all temporal sub-itemsets of \( X_{t,n}^k \) at level \( k \) with different exhibition periods, i.e., \( X_{1,n}^k, X_{2,n}^k, X_{t+1,n}^k, X_{k+2,n}^k, ..., X_{n,n}^k \), are taken as “temporal candidate k-itemsets” for producing any possible combination of general temporal association rules. Using this approach, the problem of mining temporal association rules can be implemented on an anti-monotone Apriori-like heuristic. As in most previous works, the essential idea is to iteratively generate the set of candidate itemsets of length \((k + 1)\), i.e., \( X_{k+1,n}^n \), from the set of frequent itemsets of length \( k \), i.e., \( X_{k,n}^n \), (for \( k \geq 1 \)), and to check their corresponding occurrence frequencies in the database \( db_{t,n} \). This is the basic concept of an extended version of Apriori-based algorithm, called \textit{Apriori+}, whose performance will be comparatively evaluated with algorithm \textit{PPM} in our experimental studies.
later.

We next describe the search scenario of Apriori+. For the special case \( I = \{A^{1,n}, B^{1,n}, X^{2,n}, Y^{3,n}\} \) we visualize the search space that forms a lattice in Figure 4.3. The frequent itemsets are located in the upper part of the figure whereas the infrequent ones are located in the lower part. Assume that the bold border separates the frequent itemsets from the infrequent ones. The basic principle of Apriori+ is to employ this border to efficiently prune the search space. As soon as the border line is found, we are able to restrict ourselves on determining the support values of the itemsets above the border and to ignore the itemsets below. However, it should be noted that a linearly growing number of temporal items still implies an exponential growing number of itemsets to be considered. In fact, as will be validated by experimental results later, the increase of candidates often causes a huge increase of execution time and a drastic performance degradation, meaning that without utilizing the partitioning and progressive support counting techniques we propose, a direct extension to priori work is not able to handle the general temporal association rule mining efficiently.

### 4.3 Mining General Temporal Association Rules

A detailed description of algorithm \( PPM \) is given in Section 4.3.1. We present an illustrative example for the operations of \( PPM \) in Section 4.3.2. The correctness of algorithm \( PPM \) is proved in Section 4.3.3.

#### 4.3.1 Algorithm of \( PPM \)

As explained above, a naive adoption of conventional methods to mine general temporal association rules will be prohibitively expensive. To remedy this, by partitioning a transaction database into several partitions, algorithm \( PPM \) is devised to employ a filtering threshold in each partition to deal with the candidate itemset generation and process one partition at a time. For ease of exposition, the processing of a partition is termed a phase of processing. Explicitly, a progressive candidate set of itemsets is composed of the following two types of candidate itemsets, i.e., (1) the candidate itemsets that were carried over from the previous progressive candidate set in the previous phase and remain as candidate itemsets after the current partition is included into consideration (Such candidate itemsets are called type \( \alpha \) candidate itemsets); and (2) the candidate itemsets that were not in the progressive candidate set in the previous phase but are newly selected after only taking the current data partition into account (Such candidate itemsets are called type \( \beta \) candidate itemsets). Under \( PPM \), the cumulative information in the prior phases is selectively carried over toward the generation of candidate itemsets in the subsequent phases. After the processing of a phase, algorithm \( PPM \) outputs a progressive screen, denoted by \( PS \), which consists of a progressive candidate set of itemsets, their occurrence counts and
the corresponding partial supports required.

Initially, a publication database $D$ is partitioned into $n$ partitions based on the exhibition periods of items, and $PS$, i.e., progressive screen, is empty. Let $C_2$ be the set of progressive candidate 2-itemsets generated by database $D$. Three parameters, i.e., $n$, $min\_supp$ and $min\_leng$, are taken as the input values into algorithm $PPM$. As mentioned above, the minimum support threshold required is denoted as $min\_supp$. In the process of general temporal association rule generation, we employ the parameter $min\_leng$ to be a filtering threshold for frequent itemsets to satisfy the minimal length required for the exhibition period. The procedure of algorithm $PPM$ is outlined below, where algorithm $PPM$ is decomposed into five sub-procedures for ease of description.

**Algorithm $PPM$ ($n$, $min\_supp$, $min\_leng$): Progressive Partition Miner**

**Initial Sub-procedure:** The database $D$ is partitioned into $n$ partitions and set $PS = \emptyset$.

1. $|db^{1,n}| = \sum_{h=1,n} |P_h|$;
2. $PS = \emptyset$;

**Sub-procedure I:** Generate 2nd level candidate TIs with progressive screen

3. begin for $h = 1$ to $n$  
   // 1st scan of $db^{1,n}$
4. begin for each 2-itemset $X_2^{1,n} \in P_h$ where $n - t > min\_leng$
5. if ( $X_2 \notin PS$ )
6.   $X_2.count = N_{ph}(X_2)$;
7.   $X_2.start = h$;
8.   if ( $X_2.count \geq min\_supp \times |P_h|$ )
9.     $PS = PS \cup X_2$;
10. if ( $X_2 \in PS$ )
11.   $X_2.count = X_2.count + N_{ph}(X_2)$;
12.   if ( $X_2.count < \left[ min\_supp \times \sum_{m=X_2.start,h} |P_m| \right]$ )
13.     $PS = PS - X_2$;
14. end
15. end
16. select $C_2$ from $X_2$ where $X_2 \in PS$;
17. $PS = \emptyset$;

**Sub-procedure II:** Generate candidate TIs and SIs with the scheme of database scan reduction

18. begin while ( $C_k \neq \emptyset$ & $k \geq 2$ )
19.     $C_{k+1} = C_k \times C_k$;
20. \[ k = k + 1; \]
21. end
22. \[ \mathcal{T}I = \{ X^{t,n}_k \subseteq X_k \mid X_k \in C_k \} ; \quad \text{// Candidate TIs generation} \]
23. \[ \mathcal{S}I = \{ X^{t,n}_j \subseteq \text{subset of } \mathcal{T}I \}; \quad \text{// Candidate SIs of TIs generation} \]
24. \[ \mathcal{P}S = \mathcal{T}I \cup \mathcal{S}I; \]
25. select \( X^{t,n}_k \) into \( C_k \) where \( X^{t,n}_k \in \mathcal{P}S \);

Sub-procedure III: Generate all frequent TIs and SIs with the 2nd scan of database \( D \)
26. begin for \( h = 1 \) to \( n \)
27. for each itemset \( X^{t,n}_k \in C_k \)
28. \( X^{t,n}_k.\text{count} = X^{t,n}_k.\text{count} + N_{ph}(X^{t,n}_k); \)
29. end
30. for each itemset \( X^{t,n}_k \in C_k \)
31. if ( \( X^{t,n}_k.\text{count} \geq [\text{min\_supp} \ast |db^{t-n}|] \) )
32. \( L_k = L_k \cup X^{t,n}_k; \)
33. end

Sub-procedure IV: Prune out the redundant frequent SIs from \( L_k \)
34. for each SI itemset \( X^{t,n}_k \in L_k \)
35. if ( \( \exists \) TI itemset \( X^{t,n}_j \subseteq L_j \mid j > k \) )
36. \( L_k = L_k - X^{t,n}_k; \)
37. end
38. return \( L_k; \)

In essence, Sub-procedure I first scans partition \( p_i \), for \( i = 1 \) to \( n \), to find the set of all local frequent 2-itemsets in \( p_i \). Note that \( \mathcal{P}S \) is a superset of the set of all frequent 2-itemsets in \( D \). Algorithm PPM constructs \( \mathcal{P}S \) incrementally by adding candidate 2-itemset to \( \mathcal{P}S \) and starts counting the number of occurrences for each candidate 2-itemset \( X_2 \) in \( \mathcal{P}S \) whenever \( X_2 \) is added to \( \mathcal{P}S \). If the cumulative occurrences of a candidate 2-itemset \( X_2 \) does not meet the partial minimum support required, \( X_2 \) is removed from the progressive screen \( \mathcal{P}S \). From Step 3 to Step 15 of Sub-procedure I, algorithm PPM processes one partition at a time for all partitions. When processing partition \( P_i \), each potential candidate 2-itemset \( X_2 \) is read and saved to \( \mathcal{P}S \) where its exhibition period, i.e., \( n - t \), should be larger than the minimum constraint exhibition period \( \text{min\_leng} \) required. The number of occurrences of an itemset \( X_2 \) and its starting partition which keeps its first occurrence in \( \mathcal{P}S \) are recorded in \( X_2.\text{count} \) and \( X_2.\text{start} \), respectively. As such, in the end of processing \( db^{1,h} \), only an itemset, whose \( X_2.\text{count} \geq [\text{min\_supp} \ast \sum_{m=X_2.\text{start},h} |P_m|] \), will be kept in \( \mathcal{P}S \). Note that a large amount of infrequent
TI candidates will be further reduced with the early pruning technique by this progressive portioning processing. Next, in Step 16 we select \( C_2 \) from \( X_2 \in PS \) and set \( PS = \emptyset \) in Step 17.

In Sub-procedure II, with the scan reduction scheme [68], \( C_2 \) produced by the first scan of database is employed to generate \( C_k \)s \( (k \geq 3) \) in main memory from Step 18 to Step 21. Recall that \( X_k^{t,n} \) is a maximal temporal k-itemset in a partial database \( db^{t,n} \). In Step 22, all candidate TIs, i.e., \( X_k^{t,n} \)s, are generated from \( X_k \in C_k \) with considering the maximal common exhibition period of itemset \( X_k \) where \( MCP(X_k) = (t, n) \). After that, from Step 23 to Step 25 we generate all corresponding temporal sub-itemsets of \( X_k^{t,n} \), i.e., \( SI(X_k^{t,n}) \), to join into \( PS \).

Then, from Step 26 to Step 33 of Sub-procedure III we begin the second database scan to calculate the support of each itemset in \( PS \) and to find out which candidate itemsets are really frequent TIs and SIs in database \( D \). As a result, those itemsets whose \( X_k^{t,n}.count \geq \text{min\_supp} \times |db^{t,n}| \) are the frequent temporal itemsets \( L_k \)s.

Finally, in Sub-procedure IV, we have to prune out those redundant frequent SIs whose TI itemsets are not frequent in database \( D \) from the \( L_k \)s. As will be proved in Section 3.3, the output of algorithm PPM consists of frequent itemsets \( L_k \)s of database \( D \). According to these output \( L_k \)s in Step 38, all kinds of general temporal association rules implied in database \( D \) can be generated in a straightforward method.

Note that PPM is able to filter out false candidate itemsets in \( P_i \) with a hash table. Same as in [68], using a hash table to prune candidate 2-itemsets, i.e., \( C_2 \), in each accumulative ongoing partition set \( P_i \) of transaction database, the CPU and memory overhead of PPM can be further reduced. As will be validated by our experimental studies, PPM provides very efficient solutions for mining general temporal association rules. This feature is, as described earlier, very important for mining the publication-like databases whose data are being exhibited from different starting times.

In addition, the progressive screen produced in each processing phase constitutes the key component to realize the mining of general temporal association rules. Note that algorithm PPM proposed has several important advantages, including (1) with judiciously employing progressive knowledge in the previous phases, PPM is able to reduce the amount of candidate itemsets efficiently which in turn reduces the CPU and memory overhead; (2) owing to the small number of candidate sets generated, the scan reduction technique can be applied efficiently. As a result, only two scan of the time series database is required.

4.3.2 An illustrative example of algorithm PPM

The operation of algorithm PPM can be best understood by an illustrative example described below and its corresponding flowchart is depicted in Figure 4.4. Recall the transaction database shown in Figure
Partition database based on exhibition periods

1st Scan database

Produce candidate 2-TIs

2nd Scan database

Use candidate 2-TIs to produce candidate k-TIs and k-SIs

Generate frequent k-TIs and k-SIs

Rule generation

Figure 4.4: The flowchart of PPM

4.2 where the transaction database $db^{1,3}$ is assumed to be segmented into three partitions $P_1$, $P_2$ and $P_3$, which correspond to the three time granularities from January 2001 to March 2001. Suppose that $\text{min\_supp} = 30\%$ and $\text{min\_conf} = 75\%$. Each partition is scanned sequentially for the generation of candidate 2-itemsets in the first scan of the database $db^{1,3}$. After scanning the first segment of 4 transactions, i.e., partition $P_1$, 2-itemsets $\{BD, BC, CD, AD\}$ are sequentially generated as shown in Figure 4.5. In addition, each potential candidate itemset $c \in C_2$ has two attributes: (1) $c\text{.start}$ which contains the partition number of the corresponding starting partition when $c$ was added to $C_2$, and (2) $c\text{.count}$ which contains the number of occurrences of $c$ since $c$ was added to $C_2$. Since there are four transactions in $P_1$, the partial minimal support is $[4 \times 0.3] = 2$. Such a partial minimal support is called the filtering threshold in this chapter. Itemsets whose occurrence counts are below the filtering threshold are removed. Then, as shown in Figure 4.5, only $\{BD, BC\}$, marked by “$\circ$”, remain as candidate itemsets (of type $\beta$ in this phase since they are newly generated) whose information is then carried over to the next phase $P_2$ of processing.

Similarly, after scanning partition $P_2$, the occurrence counts of potential candidate 2-itemsets are recorded (of type $\alpha$ and type $\beta$). From Figure 4.5, it is noted that since there are also 4 transactions in $P_2$, the filtering threshold of those itemsets carried out from the previous phase (that become type $\alpha$ candidate itemsets in this phase) is $[(4 + 4) \times 0.3] = 3$ and that of newly identified candidate itemsets (i.e., type $\beta$ candidate itemsets) is $[4 \times 0.3] = 2$. It can be seen that we have 3 candidate itemsets in $C_2$ after the processing of partition $P_2$, and one of them is of type $\alpha$ and two of them are of type $\beta$.

Finally, partition $P_3$ is processed by algorithm $PPM$. The resulting candidate 2-itemsets are $C_2 = \{BC, CE, BF\}$ as shown in Figure 4.5. Note that though appearing in the previous phase $P_2$, itemset $\{DE\}$ is removed from $C_2$ once $P_3$ is taken into account since its occurrence count does not meet the
<table>
<thead>
<tr>
<th>P1</th>
<th>P1 + P2</th>
<th>P1 + P2 + P3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C2 start count</td>
<td>C2 start count</td>
</tr>
<tr>
<td>BD</td>
<td>1 2</td>
<td>BD</td>
</tr>
<tr>
<td>BC</td>
<td>1 2</td>
<td>BC</td>
</tr>
<tr>
<td>CD</td>
<td>1 1</td>
<td>BE</td>
</tr>
<tr>
<td>AD</td>
<td>1 1</td>
<td>BE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

After 1st scan database D, we have candidate itemsets (relative support = 30%) as follows:
{B1,3}, {B3,3}, {C1,3}, {C2,3}, {E1,3}, {E2,3}, {BC1,3}, {BF3,3}, {CE2,3}

After 2nd scan database D, we have frequent itemsets (relative support = 30%) as follows:
{B1,3}, {B3,3}, {C1,3}, {C2,3}, {E1,3}, {E2,3}, {BC1,3}, {BF3,3}, {CE2,3}

<table>
<thead>
<tr>
<th>Candidate Itemsets</th>
<th>count</th>
<th>SPK</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{B1,3}</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>{B3,3}</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>{C1,3}</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>{C2,3}</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>{E1,3}</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>{E2,3}</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>C2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{BC1,3}</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>{BF3,3}</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>{CE2,3}</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

After 2nd scan database D, we have frequent itemsets (relative support = 30%) as follows:
{B1,3}, {B3,3}, {C1,3}, {C2,3}, {E1,3}, {E2,3}, {BC1,3}, {BF3,3}, {CE2,3}

<table>
<thead>
<tr>
<th>Rules</th>
<th>Support</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>(B =&gt; C)1,3</td>
<td>41.67%</td>
<td>62.50%</td>
</tr>
<tr>
<td>(C =&gt; B)1,3</td>
<td>41.67%</td>
<td>83.33%</td>
</tr>
<tr>
<td>(B =&gt; F)3,3</td>
<td>75.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>(F =&gt; B)3,3</td>
<td>75.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>(C =&gt; E)2,3</td>
<td>37.50%</td>
<td>75.00%</td>
</tr>
<tr>
<td>(E =&gt; C)2,3</td>
<td>37.50%</td>
<td>75.00%</td>
</tr>
</tbody>
</table>

Figure 4.5: Frequent temporal itemsets generation for mining general temporal association rules by PPM
filtering threshold then, i.e., $2 < 3$. However, we do have one new itemset, i.e., $BF$, which joins the $C_2$ as a type $\beta$ candidate itemset. Consequently, we have 3 candidate 2-itemsets generated by $PPM$, and two of them are of type $\alpha$ and one of them is of type $\beta$. Note that only 3 candidate 2-itemsets are generated by $PPM$. The correctness of algorithm $PPM$ will be formally proved later.

After generating $C_2$ from the first scan of database $db^{1,3}$, we employ the scan reduction technique [68] and use $C_2$ to generate $C_k$ ($k = 2, 3, \ldots, m$), where $C_m$ is the candidate last-itemsets. Instead of generating $C_3$ from $L_2 \times L_2$, a $C_2$ generated by $PPM$ can be used to generate the candidate 3-itemsets and its sequential $C_{k-1}$ can be utilized to generate $C_k$. Clearly, a $C_3$ generated from $C_2 \times C_2$, instead of from $L_2 \times L_2$, will have a size greater than $|C_3|$ where $C_3$ is generated from $L_2 \times L_2$. However, since the $|C_2|$ generated by $PPM$ is very close to the theoretical minimum, i.e., $|L_2|$, the $|C_3|$ is not much larger than $|C_3|$. Similarly, the $|C_k|$ is close to $|C_k|$. Since $C_2 = \{BC, CE, BF\}$, no candidate $k$-itemset is generated in this example where $k \geq 3$. Thus, $C_k' = \{BC, CE, BF\}$ and all $C_k'$ can be stored in main memory. Then, we can find $L_k$ ($k = 1, 2, \ldots, m$) together when the second scan of the database $db^{1,3}$ is performed. Note that those generated itemsets $C_k'$ are termed as frequent maximal temporal itemsets ($TIs$), i.e., $BC^{1,3}$, $CE^{2,3}$ and $BF^{3,3}$, with a maximal exhibition period of each candidate.

Before we process the second scan of the database $db^{1,3}$ to generate $L_k$s, all candidate $SIs$ of candidate $TIs$ can be propagated based on Property 1, and then added into $C_k'$. For instance, as shown in Figure 4.5, both candidate 1-itemsets $B^{1,3}$ and $C^{1,3}$ are derived from $BC^{1,3}$. Moreover, since $BC^{1,3}$, for example, is a candidate 2-itemset, its subsets, i.e., $B^{1,3}$ and $C^{1,3}$, should potentially be candidate itemsets. As a result, 9 candidate itemsets, i.e., $\{B^{1,3}, B^{3,3}, C^{1,3}, C^{2,3}, E^{2,3}, F^{3,3}, BC^{1,3}, BF^{3,3}, CE^{2,3}\}$ as shown in Figure 4.5, are generated. Note that since there is no candidate $TI$ $k$-itemset ($k \geq 2$) containing $A$ or $D$ in this example, $A^{i,3}$ and $D^{i,3}$ ($1 \leq i \leq 3$) are not necessary to be taken as $SI$ itemsets for generating general temporal association rules. In other words, we can skip them from the set of candidate itemsets $C_k'$. Finally, all occurrence counts of $C_k'$s can be calculated by the second database scan. Note that itemsets $BC^{1,3}$, $BF^{3,3}$ and $CE^{2,3}$ are termed as frequent $TIs$, while $B^{1,3}$, $B^{3,3}$, $C^{1,3}$, $C^{2,3}$, $E^{2,3}$ and $F^{3,3}$ are frequent $SIs$ in this example.

As shown in Figure 4.5, after all frequent $TI$ and $SI$ itemsets are identified, the corresponding general temporal association rules can be derived in a straightforward manner. Explicitly, the general temporal association rule of $(X \Rightarrow Y)^{i,n}$ holds if $conf((X \Rightarrow Y)^{i,n}) \geq min\_conf$.

### 4.3.3 Correctness of $PPM$

We now prove the correctness of algorithm $PPM$. Let $N_{p_h}(X)$ be the number of transactions in partition $P_h$ that contain itemset $X$, and $|P_h|$ is the number of transactions in partition $P_h$. Also, let $db^{i,j}$ denote
the part of the transaction database formed by a continuous region from partition \( P_i \) to partition \( P_j \), and 
\[ |db^{i,j}| = \sum_{h=i,j} |P_h|. \]
We can then define the region ratio of an itemset as follows. For better readability, proofs of lemmas and theorems are given in the Appendix for interested readers.

**Definition 4.5:** A region ratio of an itemset \( X \) for the transaction database \( db^{i,j} \), denoted by 
\[ r_{i,j}(X) = \frac{\sum_{h=i,j} N_{ph}(X)}{|db^{i,j}|}. \]

In essence, the region ratio of an itemset is the support of that itemset if only the part of transaction database \( db^{i,j} \) is considered.

**Lemma 4.1:** A 2-itemset \( X_2 \) remains in the PS after the processing of partition \( P_j \) if and only if there exists an \( i \) such that for any integer \( t \) in the interval \([i, j]\), 
\[ r_{i,t}(X_2) \geq \text{min\_supp}, \] where \( \text{min\_supp} \) is the minimal support required.

**Proof of Lemma 4.1:** We shall prove the “if” condition first. Consider the following two cases. First, suppose the 2-itemset \( X_2 \) is not in the progressive candidate set before the processing of partition \( P_i \). Since \( r_{i,j}(X_2) \geq \text{min\_supp} \), itemset \( X_2 \) will be selected as a type \( \beta \) candidate itemset by PPM after the processing of partition \( P_i \). On the other hand, if the itemset \( X_2 \) is already in the progressive candidate set before the processing of partition \( P_i \), itemset \( X_2 \) will remain as a type \( \alpha \) candidate itemset by PPM. Clearly, for the above two cases, itemset \( X_2 \) will remain in PS throughout the processing from \( P_i \) to \( P_j \) since for any integer \( t \) in the interval \([i, j]\), 
\[ r_{i,t}(X_2) \geq \text{min\_supp}. \]

We now prove the “only if” condition, i.e., if \( X_2 \) remains in PS after the processing of partition \( P_j \) then there exists an \( i \) such that for any \( t \) in the interval \([i, j]\), 
\[ r_{i,t}(X_2) \geq \text{min\_supp}. \]
Note that itemset \( X_2 \) can be either type \( \alpha \) or type \( \beta \) candidate itemset in the PS after the processing of partition \( P_j \). Suppose \( X_2 \) is a type \( \beta \) candidate itemset there, then this implication follows by setting \( j = i \) since 
\[ r_{i,i}(X_2) \geq \text{min\_supp}. \]
On the other hand, suppose that \( X_2 \) is a type \( \alpha \) candidate itemset after the processing of \( P_j \), which means itemset \( X_2 \) has become a type \( \beta \) candidate itemset in a previous phase. Then, we shall trace backward the type of itemset \( X_2 \) from partition \( P_j \) (i.e., looking over \( P_j, P_{j-1}, P_{j-2} \) and so forth) until the partition that records itemset \( X_2 \) as a type \( \beta \) candidate itemset is first encountered. (It should be noted that there could be two discontinuous regions that record itemset \( X_2 \) in the PS, which means that an itemset may get on and off the progressive candidate set through the processing of partitions. This in turn means that an itemset may appear as a type \( \beta \) candidate itemset more than once.) By referring the partition identified above as partition \( P_i \), we have, for any \( t \) in the interval \([i, j]\), 
\[ r_{i,t}(X_2) \geq \text{min\_supp}, \] completing the proof of this lemma. **Q.E.D.**

Lemma 4.1 leads to Lemma 4.2 below.

**Lemma 4.2:** An itemset \( X_2 \) remains in PS after the processing of partition \( P_j \) if and only if there exists an \( i \) such that 
\[ r_{i,j}(X_2) \geq \text{min\_supp}, \] where \( \text{min\_supp} \) is the minimal support required.
Proof of Lemma 4.2: It can be seen that the proof of “only if” condition follows directly from Lemma 4.1. We now prove the “if” condition of this lemma. If there exists an \( i \) such that \( r_{i,j}(X_2) \geq \text{min}_\text{supp} \) then we let \( u \) be the largest \( v \) such that \( r_{i,u}(X_2) < \text{min}_\text{supp} \). If such a \( u \) does not exist, it follows from Lemma 4.1 that itemset \( X_2 \) will remain in \( \text{PS} \) after the processing of partition \( P_j \). If such a \( u \) exists, we have \( r_{u+1,j}(X_2) \geq \text{min}_\text{supp} \) since \( r_{i,u}(X_2) < \text{min}_\text{supp} \) and \( r_{i,j}(X_2) \geq \text{min}_\text{supp} \). It again follows from Lemma 4.1 that itemset \( X_2 \) will remain in \( \text{PS} \) after the processing of partition \( P_j \). This lemma follows. Q.E.D.

Lemma 4.2 leads to the following theorem which states the completeness of candidates 2-itemsets generated by the first scan of transaction database \( \text{db}^{1,n} \) with algorithm PPM.

**Theorem 4.1**: If there exists a frequent itemset \( X_{t,n}^{2} \) in the transaction database \( \text{db}^{t,n} \) such that \( r_{t,n}(X_2) \geq \text{min}_\text{supp} \), then \( X_{t,n}^{2} \) will be in the progressive candidate set of itemsets produced by algorithm PPM.

**Proof of Theorem 4.1**: Let \( n \) be the number of partitions of the transaction database. Since the itemset \( X_{2}^{t,n} \) is a frequent itemset, we have \( r_{t,n}(X_2) \geq \text{min}_\text{supp} \), which is in essence a special case of Lemma 4.2 for \( i = t \) and \( j = n \), proving this theorem. Q.E.D.

Furthermore, we let \( C_{i,j} \), \( i \leq j \), be the set of progressive candidate itemsets generated by algorithm PPM with respect to database \( \text{db}^{i,j} \) after the processing of \( P_j \). We then have the following lemma.

**Lemma 4.3**: For \( i \leq t \leq j \), then \( C_{t,j} \subset C_{i,j} \).

**Proof of Lemma 4.3**: Assume that there exists a 2-itemset \( X_2 \in C_{t,j} \). From the “only if” implication of Lemma 4.2, it follows that there exists an \( h \) such that \( r_{h,j}(X_2) \geq s \), where \( t \leq h \leq j \). Since \( i \leq t \leq j \), we have \( i \leq h \leq j \). Then, according to the “if” implication of Lemma 4.2, itemset \( X_2 \) is also in \( C_{i,j} \), i.e., \( X_2 \in C_{i,j} \). The fact that \( C_{t,j} \subset C_{i,j} \) follows. Q.E.D.

Theorem 4.1 and Lemma 4.3 lead to the following theorem which states the correctness of algorithm PPM.

**Theorem 4.2**: If there exists a frequent \( k \)-itemset \( X_{k}^{t,n} \) in the transaction database \( \text{db}^{t,n} \) such that \( r_{t,n}(X_k) \geq s \), then \( X_{k}^{t,n} \) will be produced by algorithm PPM.

**Proof of Theorem 4.2**: Since itemset \( X_{k}^{t,n} \) is frequent, we have \( r_{t,n}(X_k) \geq \text{min}_\text{supp} \). As mentioned above, all of its sub-itemsets \( X_{h}^{t,n} \)s \( h < k \) will be frequent with \( r_{t,n}(X_h) \geq s \). Specifically, \( X_2^{t,n} \)s are in essence special cases of \( X_{h}^{t,n} \)s with \( h = 2 \). Consequently, according to the implication of Theorem 1, \( X_2 \)s will be in the progressive candidate set of itemsets, i.e., \( \text{PS} \), produced by algorithm PPM. As such, based on an anti-monotone Apriori-like heuristic, i.e., if any length \( k \) itemset \( X_{k}^{t,n} \) is not frequent
in the database, its length \((k + 1)\) super-itemset \(X^t_{k+1}\) will never be frequent, the super-itemset \(X^t_k\) of \(X^t_2\) will be produced by algorithm PPM, proving this theorem. Q.E.D.

Further, if there exists a frequent TI 3-itemset \(ABC^t\), for example, in the transaction database \(db^t\), meaning that \(r_{t,n}(ABC) \geq \min\supp\), then we have \(r_{t,n}(AB) \geq \min\supp\), \(r_{t,n}(AC) \geq \min\supp\), and \(r_{t,n}(BC) \geq \min\supp\). According to Theorem 4.1, we learn that all SIs of \(ABC^t\), i.e., \(AB^t\), \(AC^t\), \(BC^t\), \(A^t\), \(B^t\) and \(C^t\), will be in the progressive candidate set of itemsets produced by algorithm PPM. Consequently, Theorem 4.2 states the correctness of algorithm PPM.

4.4 Experimental Studies

To assess the performance of algorithm PPM, we performed several experiments on a computer with a CPU clock rate of 450 MHz and 512 MB of main memory. The transaction data resides in the NTFS file system and is stored on a 30GB IDE 3.5” drive with a measured sequential throughput of 10MB/second. The simulation program was coded in C++. The methods used to generate synthetic data are described in Section 4.4.1. The performance comparison of PPM and Apriori+ is presented in Section 4.4.2. Section 4.4.3 shows the I/O cost and CPU overhead for PPM and Apriori+. Results on scaleup experiments are presented in Section 4.4.4.

4.4.1 Generation of synthetic workload

For obtaining reliable experimental results, the method to generate synthetic transactions we employed in this study is similar to the ones used in prior works [7, 68]. Explicitly, we generated several different transaction databases from a set of potentially frequent itemsets to evaluate the performance of PPM. These transactions mimic the publication items in a publication database. Note that the efficiency of algorithm PPM has been evaluated by some real databases, such as bookstore transaction databases and grocery sales data. However, we show the experimental results from synthetic transaction data so as to obtain results of different workload parameters. Each database consists of \(|D|\) transactions, and on the average, each transaction has \(|T|\) items. To simulate the characteristic of the exhibition period in each item, transaction items are uniformly distributed into database \(D\) with a random selection. In accordance with the exhibition periods of items, database \(D\) is divided into \(n\) partitions. Table 4.2 summarizes the meanings of various parameters used in the experiments. The mean of the correlation level is set to 0.25 for our experiments. Without loss of generality, we use the notation \(Tx - Iy - Dm\) to represent a database in which \(D = m\) thousands, \(|T| = x\), and \(|I| = y\). We compare relative performance of Apriori+ and PPM.

As mentioned before, Apriori+ algorithm is an extended version of Apriori-like algorithms to deal
| $|D|$ | Number of transactions in the database |
| $|T|$ | Average size of the transactions |
| $|L|$ | Average size of the maximal potentially frequent itemsets |
| $|L|$ | Number of maximal potentially frequent itemsets |
| $N$ | Number of items |
| $|P_i|$ | Number of transactions in the partition database $P_i$ |

Table 4.2: Meanings of various parameters in the mining of general temporal association rules with the mining problem in publication databases. As will be shown by our experimental results, with the progressive partition technique that carries cumulative information selectively, the execution time of PPM is, in orders of magnitude, smaller than that required by Apriori$^+$.  

4.4.2 Experiment one: Relative performance

We first conducted several experiments to evaluate the relative performance of Apriori$^+$ and PPM. As shown in Figure 4.6, the experimental results are consistent for various values of $n$, $|L|$ and $N$ on dataset T10-I4-D100, e.g., T10-I4-D100(N20-L4-n20). For interest of space, we only report the results on $|L| = 2000$ and $N = 10000$ in the following experiments. In addition, the number of partitions in the database is selected as $n = 10$. Figure 4.6 shows the relative execution times for both two algorithms as the minimum support threshold is decreased from 1% support to 0.1% support. When the support threshold is high, there are only a limited number of frequent itemsets produced. However, as the support threshold decreases, the performance difference becomes prominent in that PPM significantly outperforms Apriori$^+$. Explicitly, PPM is in orders of magnitude faster than Apriori$^+$, and the margin grows as the minimum support threshold decreases. In fact, PPM outperforms Apriori in both CPU and I/O costs, which are evaluated next.

4.4.3 Experiment two: Evaluation of I/O cost and CPU overhead

To evaluate the corresponding of I/O cost, same as in [70], we assume that each sequential read of a byte of data consumes one unit of I/O cost and each random read of a byte of data consumes two units of I/O cost. Figure 4.7a shows the number of database scans and the I/O costs of Apriori$^+$ and PPM over the data set T10-I4-D100. As shown in Figure 4.7a, PPM outperforms Apriori$^+$. Note that the large amount of database scans is the performance bottleneck when the database size does not fit into main memory. In view of that, PPM is advantageous since only two scan of the publication database is required, which is independent of the variance in minimum supports.

As explained before, PPM substantially reduces the number of candidate itemsets generated. The effect is particularly important for the candidate 2-itemsets. The experimental results in Figure 4.7b
Figure 4.6: Relative performance studies
Figure 4.7: I/O cost and CPU overhead performance

show the candidate itemsets generated by Apriori$^+$ and PPM across the whole processing on the dataset T10-I4-D100 with minimum support threshold min$_{supp} = 0.2\%$. As shown in Figure 4.7b, PPM leads to a 96% candidate reduction rate in $C_2$ when being compared to Apriori$^+$. This feature of PPM enables it to efficiently reduce the CPU and memory overhead. Note that the number of candidate 2-itemsets produced by PPM approaches to its theoretical minimum, i.e., the number of large 2-itemsets. Recall that the $C_3$ in Apriori$^+$ has to be obtained by $L_2$ due to the large size of their $C_2$. As shown in Figure 4.7b, the value of $|C_k|$ ($k \geq 3$) is only slightly larger than that of Apriori$^+$, even though PPM only employs $C_2$ to generate $C_k$s, thus fully exploiting the benefit of scan reduction.

4.4.4 Experiment three: Scaleup performance

In this experiment, we examine the scaleup performance of algorithm PPM. The scale-up results for different selected datasets are obtained. Figure 4.8 shows the scaleup performance of algorithm PPM as the values of $|D|$ increase. Three different minimum supports are considered. We obtained the results for the dataset T10-I4-Dm when the number of customers increases from 100,000 to one million. The execution times are normalized with respect to the times for the 100,000 transactions dataset in the Figure 4.8a. Note that, as shown in Figure 4.8a the execution time only slightly increases with the growth of the database size, showing good scalability of PPM.

To further understand the impact of $|D|$ to the relative performance of algorithms PPM and Apriori$^+$ algorithms, we conduct the scaleup experiments for both PPM and Apriori$^+$ with two minimum support thresholds 0.2% and 0.4%. The results are shown in Figure 4.8b where the value in y-axis corresponds
Figure 4.8: Scaleup performance of PPM and the execution time ratio between PPM and Apriori+. Figure 4.8b shows the referenced ratio obtained from an publication-like database over datasets of T10-I4-Dm. The execution-time-ratio of PPM to Apriori+ decreases when the amount of database |D| grows larger, meaning that the advantage of PPM over Apriori+ increases as the database size increases.

4.5 Summary

in this chapter, we not only explored a new model of mining general temporal association rules, i.e., \((X \Rightarrow Y)^{MCP(XY)}\), in a publication database but also developed algorithm PPM to generate the temporal association rules as well as conducted related performance studies. Under PPM, the cumulative information of mining previous partitions is selectively carried over toward the generation of candidate itemsets for the subsequent partitions. Algorithm PPM not only significantly reduced I/O and CPU cost by the concepts of progressive counting and scan reduction techniques but also effectively controlled memory utilization by proper partitioning. Algorithm PPM is particularly powerful for efficient mining for a publication-like transaction database, such as bookstore transaction databases, video rental store records, library-book rental records, and transactions in electronic commerce. The correctness of PPM is proved and some of its theoretical properties are derived. Extensive simulations have been performed to evaluate performance of algorithm PPM. Sensitivity analysis of various parameters was conducted to provide many insights into algorithm PPM. It was noted that the improvement achieved by PPM increases as the size of the database increases.
Chapter 5

Causality Rules: Exploring the Relationship between Triggering and Consequential Events in a Database of Short Transactions

5.1 Introduction

Since the earlier work in [5], a broad variety of data mining capabilities has been developed. These studies cover a broad spectrum of topics including: (1) association rule mining [1, 7, 37, 38, 53, 68]; (2) incremental updating [10, 26, 49]; (3) mining of generalized [79], multi-level [33], quantitative [80], multi-dimensional rules [88, 90]; (4) constraint-based rule mining [35, 46, 69] and multiple minimum supports issues [55, 87]; (5) temporal association rule [9, 24, 48, 86]; (6) frequent episodes discovery [58, 59, 60]; (7) sequential patterns mining [8, 37, 70]; and (8) Web log mining [23].

Among others, mining association rules and sequential patterns received a significant amount of research attention. A popular area of applications is the market basket analysis, which studies the buying behaviors of customers by searching for sets of items that are frequently purchased either together or in sequence. While the discovery of association relationship among the data in a huge database has been known to be useful in selective marketing, decision analysis, and business management [22, 40], it is noted that the existing models of rule mining might not be able to discover user preferred frequent patterns efficiently due to the following fundamental problems.

1. **The puzzle of mining association rules on a short transaction database:** Consider the knowledge discovery in a transaction database of a convenience store where customers usually visit frequently and the number of items purchased in each transaction is usually small. Such a database is then composed of short transactions. Previous mining works in association rules, however, do not fully explore the inter-transaction relationship, and are thus apt to provide the
limited knowledge in the sales patterns (as discovered from intra-transactions). Note that such purchasing scenarios for short transactions also occur in electronic commerce purchasing records, pharmacy purchasing databases, bookstore transaction records, and so on, thereby unavoidably reducing the usefulness of rule mining in these applications.

**Example 5.1.1:** Consider the database shown in Figure 5.1, e.g., the customer purchasing records of a furniture store. Let the minimum transaction support threshold required for generating frequent patterns be two transactions (denoted by TIDs in Figure 5.1). Note that since fewer items will be bought at the same time, as shown in Figure 5.1, the association rules we obtain will be those such as “TV and TV set are frequently purchased together”, which, while being correct by the definition, is of less interest to us in the association rule mining.

2. **Lack of long patterns for sequential pattern mining:** On the other hand, due to the imposition of a strict order, e.g., people would buy B after the purchase of A, followed by shopping C, the mining of sequential patterns tends to suffer from the drawback of having very low supports in long sequential patterns [8, 37, 70]. Otherwise, rules in long sequences will rarely be discovered. For instance, the occurrence probability of a seven-item sequential pattern could be in proportion to \( p^6 \) where \( p \) is the probability of a certain product being bought after a given one. In fact, such a strict order in the sequential pattern mining might not be justifiable in some real applications, since after the purchase of a TV, one may have less interest in the exact subsequent buying order of products, say, TV set, sofa, end table, lamp, carpet, coffee table, than in exploring what set of products in general would be inspired for subsequent purchases.

**Example 5.1.2:** Recall the illustrative example in Figure 5.1. The minimum support (denoted by \( \text{min\_supp} \)) threshold for analyzing the customer purchasing behaviors is assumed to be \( \text{min\_supp} = 40\% \), meaning that a frequent pattern should appear in at least two customer’ purchasing behaviors (denoted by CIDs in Figure 5.1). As shown in Figure 5.1, there is no more frequent 3-items sequential pattern in this example.

Consequently, this chapter explores the mining of causality rules with the triggering and consequential events for a database of short transactions. Such a short transaction database is, in our opinion, common in many real applications. Explicitly, we shall conduct in this chapter the mining of causality rules from a transaction database, where each event may belong to multiple categories and the causality rule consists of (a) a sequence of triggering events and (b) a set of consequential events. The causality rule mining capability can be applied to various applications. For example, one can improve electronic commerce applications by first identifying consumer telephone calling patterns and consumer buying patterns, and then using the information discovered to attain more effective on-line advertising and
Table 5.1: An illustrative example for the mining on a short transaction database

<table>
<thead>
<tr>
<th>Customer</th>
<th>Buying sequences</th>
<th>Item Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>CID TID1</td>
<td>TID2</td>
<td>TID3</td>
</tr>
<tr>
<td>S1</td>
<td>(B, C)</td>
<td>A</td>
</tr>
<tr>
<td>S2</td>
<td>D</td>
<td>B</td>
</tr>
<tr>
<td>S3</td>
<td>(A, B)</td>
<td>(C, D)</td>
</tr>
<tr>
<td>S4</td>
<td>(A, B, E)</td>
<td></td>
</tr>
<tr>
<td>S5</td>
<td>E</td>
<td>D</td>
</tr>
</tbody>
</table>

Figure 5.1: An illustrative example for the mining on a short transaction database

offerings to the consumers. Specifically, transaction patterns can be derived from collected data by observing the customer behavior in terms of *cause* and *effect*, or what will be referred to as triggering events and consequential events. In this chapter, the term “causality rule(s),” denoted by $X \rightarrow Y$, will refer to a rule of describing certain customer behavior where some triggering events, i.e., $X$, lead to a set of consequential events, i.e., $Y$. This problem can be further described by the example below.

**Example 5.1.3:** The purchase of a new set of the living room furniture may start with buying sofa sets and TVs, followed by shopping for TV sets, coffee tables, lamps and carpets for various accessories. Hence, the triggering event would be buying sofas/TVs, and the consequential events are the behaviors of buying TV sets, coffee tables, lamps, carpets and accessories. Consider the database in Figure 5.1 with $min\_supp = 40\%$ again. With the concept of the triggering and consequential events, plenty of useful causality rules will be discovered from this transaction database. For interest of space, we only show some interesting causality rules below. A detailed process of the causality rule mining will be given in Section 5.2.3.

1. $\{TV, TV\ set, Sofa\} \rightarrow \{Lamp\}$ with $min\_supp = 40\%$;
2. $\{TV, TV\ set\} \rightarrow \{End\ table, Lamp\}$ with $min\_supp = 40\%$;

Note that no specific order is assumed among the triggering/consequential events. In essence, the problem of mining causality rules can be mapped from an event sequence database into a problem of counting large event sets. In this chapter, we decompose the problem of mining causality rules into two phases, i.e., the phase of discovering one-triggering causality rules and the phase of generating multi-triggering causality rules:

1. **In the phase of discovering one-triggering causality rules**, we use iterative approaches to deriving one-triggering event causality rules which contain only single triggering events. To count the occurrences in the event sequence database of each candidate $k$-event rule in $C_k$, it is necessary to scan through the sequence database to do a sub-sequence matching. This procedure is very
costly in the presence of a huge number of candidate sets and a large event database, and in our opinion, cannot be dealt with by direct extensions from existing rule mining methods, including GSP [8], FP-tree [36], Free-Span [38], PrefixSpan [70], episode mining algorithms [58, 59, 60], and so on. Hence, instead of comparing each candidate rule in $C_k$ directly with the event sequence in the database, the event sequence is transformed into a simple event sequence based on the concept of hierarchical sub-sequence matching. With the hierarchical matching methodologies, the detection of an occurrence of a causality rule in an event sequence can be greatly facilitated.

2. **In the phase of generating multi-triggering causality rules**, newly identified one-triggering event causality rules are used to generate the next set of candidate rules to be evaluated, by increasing either (1) the size of the set of consequential events triggered by triggering events or (2) the number of triggering events. For example, *the causality rule of “A and B triggering C” can hold true only if both rules “A triggers C” and “B triggers C” hold.*

Since the corresponding causality rules can be derived in a straightforward manner in the phase of generating multi-triggering causality rule, the overall performance of mining causality rules is in fact determined by the first phase, i.e., the phase of discovering one-triggering causality rules. To minimize the corresponding computational cost in the first phase, we develop, in light of the concept of hierarchical matching, three algorithms, namely candidate-sets-based hierarchical matching (referred to as algorithm $HM_C$), data-sets-based hierarchical matching (referred to as algorithm $HM_D$) and adaptive hierarchical matching (referred to as algorithm $HM_A$), to explore the mining of causality rules. Extensive experiments are performed to assess the performance of the proposed algorithms. Sensitivity analysis on various parameters of the event database is also conducted to provide many insights into algorithms proposed. According to the experimental results, it is shown that algorithm $HM_C$ is effective in generating the higher order large event sets whereas algorithm $HM_D$ is good at dealing with the huge numbers of the lower order event sets. The adaptive matching algorithm $HM_A$ is shown to outperform algorithms $HM_C$ and $HM_D$, by adaptively employing matching techniques of algorithms $HM_C$ and $HM_D$. These experimental results conform with the complexity analysis of algorithms proposed. Scale-up experiments show that all three algorithms scale linearly with the number of customer transactions. They also have good scale-up properties with respect to the number of transactions per customer and the number of items in a transaction.

We mention in passing that association rules deal with intra-transaction information, in which the events have occurred effectively simultaneously, with no regard for cause and effect or trigger and consequence. Causality rules, in contrast, deal with inter-transaction behavior explicitly. The triggering event must occur earlier in time than all of the consequential events where no specific order is assumed.
among the triggering/consequential events. On the other hand, sequential events are inter-transaction events which are necessarily ordered in time [8, 37, 70], such that a second event always follows the first, and a third event follows the second, but the third would never directly follow the first. Further, the works in [58, 59, 60] consider frequent episodes discovered from a long event sequence such as the one composed of signals in a telecommunication database. With the use of a moving window, the episode mining algorithms explore the temporal relationship (parallel or sequence) of signals in individual long transactions. The inter-transaction behavior is, however, not addressed. It is worth mentioning that since the causality relationship can only be captured by the mining on non-sequential, inter-transaction information across multiple categories, the causality rule can be viewed as a more general framework than those in prior studies. To the best of our knowledge, neither of the prior works has fully addressed the causality relationship, let alone devising algorithms to conduct causality rule mining for a database of short transactions. These features distinguish this dissertation from others.

The rest of this chapter is organized as follows. The problem description of mining causality rules is given in Section 5.2. Section 5.3 examines three different procedures, i.e., $HM_C$, $HM_D$, and $HM_A$, in detail. We empirically evaluate the performance of these algorithms and study their scale-up properties in Section 5.4. This chapter concludes with Section 5.5.

### 5.2 Problem Description

Specifically, a rule “$\{c_1\}$ triggers $\{r_1, \ldots, r_k\}$”, denoted by $\{c_1\} \rightarrow \{r_1, \ldots, r_k\}$, is called a one-triggering causality rule, if there is a sufficient number of sequences in the database containing $c_1$ followed by the $r_j$’s, ($1 \leq j \leq k$) in any order and the $c_1, r_1, \ldots, r_k$, satisfy some pre-specified constraints (e.g., time constraint). This causality rule can be generalized to allow for a sequence of triggering events to lead to a set of consequential events. In other words, “$\{c_1, \ldots, c_m\}$ triggering $\{r_1, \ldots, r_k\}$”, denoted by $\{c_1, \ldots, c_m\} \rightarrow \{r_1, \ldots, r_k\}$, may be regarded as a multi-triggering causality rule, if there is a sufficient number of sequences containing $\{c_1, \ldots, c_m\}$ followed by the $r_j$’s, ($1 \leq j \leq k$) in any order, and furthermore that $\{c_1, \ldots, c_m\}$ and $\{r_1, \ldots, r_k\}$ satisfy some pre-specified constraints. We then have the following definition.

**Definition 5.1**: A causality rule $X \rightarrow Y$ is termed to be frequent if and only if its support is larger than the minimum support required, i.e., $\text{supp}(X \rightarrow Y) > \text{min}_\text{supp}$.

It is noted that in accordance with Definition 5.1, one may find out both $X \rightarrow Y$ and $Y \rightarrow X$ causality rules in the same transaction database. In our opinion, such kinds of causality rules explore the “Nature Puzzles”, e.g., both “Chicken $\rightarrow$ Egg” and “Egg $\rightarrow$ Chicken” are termed as nature causality rules.
In this section, we concentrate the discussion on the basic concepts of mining causality rules. Consequently, as mentioned above, we decompose the discussion into two phases: (1) discovering one-triggering causality rules and (2) generating multi-triggering causality rules. Section 5.2.1 introduces the design principle and the hierarchical matching methodology for the phase of discovering one-triggering causality rules. Section 5.2.2 develops the scheme of generating all general causality rules which contain multiple triggering events. In Section 5.2.3, an illustrative example of mining causality rules is presented.

5.2.1 First Phase: Discovering One-Triggering Causality Rules

A sequence of \( k \) simple events, \( c_1, r_1, ..., r_{k-1} \), corresponds to a large \( k \)-event rule, if there are sufficient numbers of sequences containing \( c_1 \) followed by the \( r_j \)'s where \( (1 \leq j \leq k - 1) \) in any order (i.e., its fraction of appearances in the sequence database exceeds a minimum threshold). The set of all large \( k \)-event rules forms the large \( k \)-event set. A large \( k \)-event rule, \( \{c_1\} \rightarrow \{r_1, ..., r_{k-1}\} \), represents a causality rule if the number of sequences containing \( c_1, r_1, ..., r_{k-1} \), exceeds some pre-specified fraction of the number of sequences containing \( c_1 \). The foregoing fraction is referred to as the confidence requirement of the causality rule \( \{c_1\} \rightarrow \{r_1, ..., r_{k-1}\} \).

Let \( L_k \) represent the large \( k \)-event set. Here, without loss of generality, it is assumed that for each large \( k \)-event rule, \( \{c_1\} \rightarrow \{r_1, ..., r_{k-1}\} \), in \( L_k \), letters in \( \{r_1, ..., r_{k-2}, r_{k-1}\} \) are in lexicographic order. It is noted that \( L_1 \) denotes a degenerated case, where each element only represents a frequent event, not a causality rule. Only elements in \( L_1 \) can be triggering events or consequential events of a causality rule. First, \( L_1 \) is found by scanning the sequence database and keeping a count of the occurrence of each event. For each composite event which is a set of basic events, each of the basic events in the set receives an increment on its count.

Starting with \( k = 2 \), we can then generate a candidate \( k \)-event set, \( C_k \) from \( L_{k-1} \). This can be done by joining \( L_{k-1} \) with \( L_{k-1} \) to derive \( C_k \). Specifically, for \( k = 2 \), \( C_2 \) is the cross-product of \( L_1 \) by itself. For \( k > 2 \), any two large \((k-1)\)-event rules in \( L_{k-1} \), with the same starting event and matching \( k - 3 \) of the remaining events, can be joined together to form a candidate \( k \)-event rule in \( C_k \). The candidate \( k \)-event set will contain the large \( k \)-event set as its subset.

5.2.2 Second Phase: Generating Multi-Triggering Causality Rules

Let \( L_i^k \) be the large \( i \)-event set with \( k \) triggering events at the beginning of each sequence followed by \( i - k \) consequential events. Then \( L_1^k \) is equal to \( L_1 \). Starting with \( j = 2 \) and \( i = j + 1 \), we obtain \( L_i^j \) from \( L_i^{j-1} \). Specifically, by joining any two rules in \( L_i^{j-1} \) with (1) the same consequential set, and (2) the first \( j - 2 \) triggering events in one of the sequences that matches the last \( j - 2 \) triggering events in the other sequence, we can obtain a candidate set \( C_i^j \) with \( j \) triggering events.
Example 5.2.1: Consider two sequences, $S_1$ and $S_2$ in $L_5^2$. Assuming that $S_1 = \{A, E\} \Rightarrow \{F, H, Y\}$ and $S_2 = \{E, G\} \Rightarrow \{F, H, Y\}$, the two rules can be joined to form a new candidate rule $\{A, E, G\} \Rightarrow \{F, H, Y\}$ in $C_6^3$ where $A, E$ and $G$ are triggering events and $F, H$ and $Y$ form the consequential set of events. Similarly, the occurrences of each $i$-event rule in $C_i^j$ can be obtained. When matching an $i$-event rule with a sequence in the database, the triggering sub-sequence will be identified first. Then, matching of the consequential set is conducted by using the hierarchical matching approach. The candidate $i$-event rules with occurrences exceeding the given threshold form $L_i^j$.

5.2.3 An Illustrative Example for the Mining of Causality Rules

The mining process of causality rules can be best understood by an illustrative example described below. Consider again the database with the customer-sequences shown in Figure 5.1. The minimum support is assumed to be 40% (i.e., two customer sequences). The corresponding flowchart is depicted in Figure 5.2. For ease of exhibition, we employ the symbol \{A, B\} as a non-ordered event sequence of event A and event B. On the other hand, $<\{A, B\}, \{D, E\}>$ indicates event sequence \{A, B\} occurring before event sequence \{D, E\} as a causality rule which \{A, B\} triggers \{D, E\}, i.e., \{A, B\} $\Rightarrow$ \{D, E\}. Figure 5.3 shows the profile of candidate $k$-event sets $C_k$ and large $k$-event sets $L_k$ in each pass.

In the phase of discovering one-triggering causality rules, after the first pass of scanning over the database, we determine the large one-triggering event set. The large event sets together with their supports at the end of the second and the third passes are shown in Figure 5.3. No candidate is generated in the fourth pass. As a result, after pruning the redundant rules from $L_k$, the $R^1$ set of one-triggering causality rules contains the three event rules, i.e., \{C\} $\Rightarrow$ \{E\}, \{A\} $\Rightarrow$ \{D, E\} and \{B\} $\Rightarrow$ \{D, E\}, shown in Figure 5.3.

The phase of generating multi-triggering causality rules is processed to generate the causality rules with higher order triggering events, i.e., $R^j$ where $j \geq 2$. According to the mining results of large 1 triggering $k$ event sets, i.e., $L_k^1$, in the first phase, candidate $i$ triggering $j$ event sets, i.e., $C_i^j$, can
be sequentially generated by joining any two $L_{i-1}^{j-1}$ where $j \geq 2$ and $i \geq 3$. Similarly, the database occurrences of each $j$-event rule in $C_j^i$ are counted. As a result, the total causality rules from the database in Figure 5.1 are shown in Figure 5.4. Note that in a set of event rules, an event rule, e.g., $X \rightarrow Y$, is redundant if $X \rightarrow Y$ is contained in any other event sequence. It is worth mentioning that the event rules $\{A, B\} \rightarrow \{D\}$, $\{A, B\} \rightarrow \{E\}$, $\{A, C\} \rightarrow \{E\}$ and $\{B, C\} \rightarrow \{E\}$ shown in Figure 5.4a, though having minimum supports, do not appear in the answer in Figure 5.4b because they are redundant.

Furthermore, in mining causality rules, certain constraints such as time constraint may additionally be imposed. For example, it is reasonable to require that all consequential events occur within a certain period after the triggering event has transpired. One can view the time constraint as a moving window on deciding causality rules or as the setting of an upper bound on the time gap between two successive events in a causality rule. Also, the causality-based mining capability can be extended to the one with multiple steps, e.g., a two-step causality rule such as $X \rightarrow Y \rightarrow Z$. Proper applications of causality rule mining include Web-log analysis, telecommunication, bio-medical research and DNA analysis.

As mentioned above, since the overall performance of mining causality rules is determined by the first phase, this problem can be reduced to the problem of discovering all one-triggering causality rules for the same support threshold. For interest of space, we concentrate our presentation on mining one-triggering causality rules in the following sections.

### 5.3 Algorithms for Mining One-Triggering Causality Rules

As mentioned earlier, to count the occurrences in the event sequence database of each candidate $k$-event rule in $C_k$, it is necessary to scan through the sequence database to do a sub-sequence matching. It is costly to handle the matching in the presence of a huge number of candidate sets and a large event

<table>
<thead>
<tr>
<th>Large k-sequence</th>
<th>Supp.</th>
<th>Large k-sequence</th>
<th>Supp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_1$</td>
<td></td>
<td>$L_2$</td>
<td></td>
</tr>
<tr>
<td>${A}$</td>
<td>4</td>
<td>${A}, {D}$</td>
<td>2</td>
</tr>
<tr>
<td>${B}$</td>
<td>4</td>
<td>${A}, {E}$</td>
<td>2</td>
</tr>
<tr>
<td>${C}$</td>
<td>2</td>
<td>${B}, {D}$</td>
<td>2</td>
</tr>
<tr>
<td>${D}$</td>
<td>4</td>
<td>${B}, {E}$</td>
<td>2</td>
</tr>
<tr>
<td>${E}$</td>
<td>4</td>
<td>${C}, {E}$</td>
<td>2</td>
</tr>
<tr>
<td>$L_3$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>${A}, {D, E}$</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>${B}, {D, E}$</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.3: An example of generating large one-triggering event sequences with $\text{min supp} = 40\%$
Supp. \{A, B\} → \{E\} 2 Pruning
Supp. \{A, B\} → \{D\} 2 reductant rules R
Supp. \{A, C\} → \{E\} 2 R
Supp. \{B, C\} → \{E\} 2 R
Supp. \{A, B\} → \{D, E\} 2
Supp. \{A, B\} → \{D, E\} 2
Supp. \{A, C\} → \{E\} 2 R
Supp. \{B, C\} → \{E\} 2 R
Supp. \{A, B, C\} → \{E\} 2

(a) multi-triggering causality rules before pruning
(b) multi-triggering causality rules after pruning

Figure 5.4: $R^j$ sets of multi-triggering causality rules on mining the database in Figure 5.1 where $j \geq 2$

<table>
<thead>
<tr>
<th>multi-triggering rules ($R^j$)</th>
<th>Supp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A, B} → {E}</td>
<td>2</td>
</tr>
<tr>
<td>{A, B} → {D}</td>
<td>2</td>
</tr>
<tr>
<td>{A, C} → {E}</td>
<td>2</td>
</tr>
<tr>
<td>{B, C} → {E}</td>
<td>2</td>
</tr>
<tr>
<td>{A, B} → {D, E}</td>
<td>2</td>
</tr>
<tr>
<td>{A, B} → {D, E}</td>
<td>2</td>
</tr>
<tr>
<td>{A, C} → {E}</td>
<td>2</td>
</tr>
<tr>
<td>{B, C} → {E}</td>
<td>2</td>
</tr>
<tr>
<td>{A, B, C} → {E}</td>
<td>2</td>
</tr>
</tbody>
</table>

Database. In view of this, we devise three candidate-matching algorithms, denoted by $HM_C$, $HM_D$, and $HM_A$, to minimize the computing cost needed by the first phase of discovering all one-triggering causality rules. By utilizing the concept of hierarchical sub-sequence matching, which will be introduced in Section 5.3.1, these three algorithms present good efficiency and salability in the mining of one-triggering causality rules. In essence, instead of comparing each candidate rule in $C_k$ directly with the event sequence in the database, the proposed algorithms transform every composite event sequence into basic ones. As will be seen later, the use of hierarchical matching techniques will improve the execution efficiency.

Basically, algorithm $HM_C$ demonstrates the matching processing driven by candidate sets in Section 5.3.2 while $HM_D$ explores a matching technique which is particularly powerful in the presence of a huge number of low order large event sets in Section 5.3.3. As will be verified by our experimental results, by combining the advantages of both $HM_C$ and $HM_D$, the adaptive algorithm $HM_A$, presented in Section 5.3.4, will outperform $HM_C$ and $HM_D$ for generating any order of large event sets. For better readability, a list of symbols used is given in Table 5.1.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_i$</td>
<td>the $i^{th}$ sequence in an event sequence database $D$</td>
</tr>
<tr>
<td>$a_{ij}$</td>
<td>a potential triggering event in the event sequence $S_i$</td>
</tr>
<tr>
<td>$P_{ij}$</td>
<td>the sub-sequence of $S_i$ that contains all of the following events of $a_{ij}$</td>
</tr>
<tr>
<td>$\Phi_{ij}$</td>
<td>the sorted sub-sequence of $P_{ij}$ that contains all basic events in $P_{ij}$</td>
</tr>
<tr>
<td>$C_i^j$</td>
<td>candidate $i$ event rule with $j$ triggering events</td>
</tr>
<tr>
<td>$L_i^j$</td>
<td>large $i$ event rule with $j$ triggering events</td>
</tr>
</tbody>
</table>

Table 5.1: Meanings of symbols used in the mining of causality rules
5.3.1 Hierarchical Matching

While deferring the description of details, we first highlight the basic idea of hierarchical matching below. For a sequence \( S_i \) in the database, we examine whether each \( a_{ij} \) in the sequence can be a potential triggering event. First, a maximum potential set of possible consequential events is determined, including all possible consequential events which can be triggered by \( a_{ij} \) under the pre-specified constraints. (For example, if a time constraint from the triggering event is given, the sub-sequence \( P_{ij} \) will consist of all of the following events in \( S_i \) occurring within the time constraint.) Each composite event in \( P_{ij} \) is replaced by the basic events comprising it. After replacing the composite events, the basic events are then sorted in lexicographic order and the resulting sequence is denoted as \( \Phi_{ij} \).

Next, within \( C_k \), those rules which start with \( a_{ij} \) and have their remaining portions matching some sub-sequences contained in \( \Phi_{ij} \) are determined. This subset of \( C_k \) is referred to as \( \Delta_{ij} \). Set \( \Phi_{ij} \) is now examined to assure that, after discarding its first event, the remaining portion of each candidate rule in \( \Delta_{ij} \) is indeed a legitimate sub-sequence in \( \Phi_{ij} \). Then, the count of the corresponding candidate \( k \)-event rule in \( C_k \) will be incremented. After the scan of the sequence database is completed, those candidate rules in \( C_k \) with counts exceeding the threshold will become \( L_k \). If \( L_k \) is non-empty, the iteration continues with \( k \) incremented by one.

Example 5.3.1: Assume that \( S_i = A, (B, W), D, (U, B), (H, G) \). With \( a_{i1} = A \), we have \( P_{i1} = (B, W), D, (U, B), (H, G) \). Then \( \Phi_{i1} = \{B, D, G, H, U, W\} \). Consider \( C_3 = \{(A, B, H), (A, G, H), (A, E, W)\} \), for instance. Both \((A, B, H)\) and \((A, G, H)\) start with \( A \), with the remaining portions matching some sub-sequences of \( \Phi_{i1} \). On the other hand, \((A, E, W)\) does not find a match.

The reason for performing the initial matching of \( C_k \) with \( \Phi_{ij} \) is that matching of an ordered list is much more efficient than matching of a non-ordered list. The presence of composite events inhibits ordering of the lists. That is very reason we decompose each composite event into its basic events.

5.3.2 Algorithm \( HMC_k \): Candidate-Sets-Based Hierarchical Matching

For ease of exposition, the transaction database is pre-processed to create an event sequence database \( \mathcal{D} \) of \( N \) sequences, with the \( i^{th} \) sequence consisting of \( g_i \) events. We present the first proposed algorithm \( HMC_k \) together with the candidate-sets-based hierarchical matching methodology. The algorithm of \( HMC_k \) is outlined below. To count the occurrences in the event sequence database of each candidate \( k \)-event rule in \( C_k \), one will scan through the sequence database to do a two step hierarchical matching. Instead of comparing each candidate rule in \( C_k \) directly with the event sequence in the database, we transform the event sequence into a simple event sequence. It can be verified that the complexity of algorithm \( HMC_k \) is \( O(|C_k| \times n \times k) \) for generating a large \( k \)-event set, where \( |C_k| \) denotes the amount of
candidate $k$-event sequences and $n$ is the average number of transactions for each customer.

**Algorithm HMC**: Candidate-sets based hierarchical matching

1. begin;
2. obtain $L_1$ by counting occurrences;
3. set $k$ to 2;
4. set $C_k$ to $L_{k-1} \ast L_{k-1}$;
5. set $i$ to 1;
6. while ($i \leq N$) {
   7. while ($C_k \neq \phi$) {
      8. denote the $i^{th}$ sequence as $S_i = \{a_{i1}, a_{i2}, ..., a_{ig_i}\}$;
      9. set $j$ to 1;
      10. while ($j \leq g_i - k + 1$) {
         11. set $y$ to $a_{ij}$;
         12. derive $P_{ij}$;
         13. map $P_{ij}$ into $\Phi_{ij}$;
         14. denote $D_{y}^{C}$ as the subset of $C_k$;
         15. set $m$ to 1;
         16. while ($m \leq |D_{y}^{C}|$) {
            17. denote the $m^{th}$ candidate rule in $D_{y}^{C}$ as $y_{w1}...w_{k-1}$;
            18. if ($w_{1}...w_{k-1}$ is a sub-sequence of $\Phi_{ij}$) {
               19. increment count of $y_{w1}...w_{k-1}$ by 1;
            }
         }
      }
   }
   20. set $m$ to $m + 1$;
   21. set $j$ to $j + 1$;
   22. set $i$ to $i + 1$;
   23. derive $L_k$ and set $k$ to $k + 1$;
24. generate rule;
25. eliminate redundant rules;
26. end;

In Step 2, the number of occurrences of each event is counted across all user sequences. Those events with occurrence counts exceeding a given threshold requirement will be included into $L_1$, the large 1-event set. The next step is to determine $L_k$ for $k > 1$. In Step 3, $k$ is set to 2. In Step 4, the candidate large $k$-event set $C_k$, which is a superset of $L_k$ is derived. As mentioned before, this can be done by joining $L_{k-1}$ with $L_{k-1}$. Specifically, for $k = 2$, $C_2$ is the cross-product of $L_1$ with itself. For $k > 2$, any two rules in $L_{k-1}$ with the same starting event and matching $k - 3$ of the remaining events can be joined together to form a candidate $k$-event rule.

The next step is scanning through the event sequence database to determine the number of occurrences of each candidate rule in $C_k$. In Step 5, $i$ is set to 1, where $i$ is the index to scan through the event sequence database. In Step 6, $i$ is compared with $N$, i.e., the number of event sequences in the database. If it is smaller than $N$, there are more sequences to scan. In Step 8, the $i^{th}$ sequence in the database is denoted as $S_i = \langle a_{i1}, a_{i2}, ..., a_{ig_i} \rangle$. We next scan through the sequence using an index $j$. In Step 9, the index $j$ is set to 1. At Step 10, $j$ is compared with $g_i - k + 1$, where $g_i$ is the length of
the $i$-th sequence and $g_i - k + 1$ is the last event in the sequence that can start a sub-sequence of length $k$. If $j$ is smaller than $g_i - k + 1$, $a_{ij}$ may start a causality rule of length $k$.

In the next steps, candidate rules in $C_k$ are matched with a sub-sequence starting with $a_{ij}$. In Step 11, $y$ is set to equal $a_{ij}$. In Step 12, $P_{ij}$ is derived, with $P_{ij}$ being the maximum potential set of consequential events that can be triggered by $a_{ij}$. In Step 13 which becomes necessary if any of the consequential events are composite events, $\Phi_{ij}$ is derived by (1) replacing each composite event in $P_{ij}$ by the basic events comprising it; and (2) re-ordering the basic events in lexicographic order. In Step 14, $D_y$ is denoted as the subset of $C_k$ rules that start with $a_{ij}$.

Next, a scan is made through $D_y$ with an index of $m$ to determine which candidate rules in $D_y$ have a match with $S_i$. In Step 15, $m$ is set to 1. In Step 16, $m$ is compared with the size of $D_y$. If $m$ is larger, meaning the scanning of $D_y$ is complete, $j$ is incremented by 1 at Step 17 and the system proceeds to Step 10. If $m$ is smaller, however, the $m$-th candidate rule in $D_y$ is designated as $yw_1w_2...w_{k-1}$, at Step 17. In Step 18, a determination is made as to whether $w_1w_2...w_{k-1}$ is a sub-sequence of $\Phi_{ij}$. If it is not a sub-sequence, the system proceeds to Step 20. If it is determined that $w_1w_2...w_{k-1}$ is a sub-sequence in $P_{ij}$, the occurrence count of $yw_1w_2...w_{k-1}$ is incremented by 1. At Step 20, $m$ is incremented by 1 and the system proceeds to Step 16.

A comparison is made to determine if $j \leq g_i - k + 1$, at Step 10. If the result of the comparison is “no”, $i$ is incremented by 1 at Step 22 and the system returns to Step 6. At Step 6, the value for $i$ is compared to $N$. If the comparison yields a “no”, $C_k$ is checked to see if it is empty, at Step 7. If not empty, $L_k$ is derived from $C_k$ at Step 23, where $L_k$ is set to include the candidate rules in $C_k$ with occurrence counts exceeding a pre-set threshold and $k$ is incremented by 1. If it is determined that $C_k$ is empty, then all of the large event rules have been determined.

In Step 24, from the large event sets $L_2, ..., L_{k-1}$, causality rules are generated based on the confidence requirement. Finally, all redundant rules are eliminated at Step 25; such that for any rule, if there exists another rule with the same triggering event and a larger consequential set containing its consequential set, (i.e., a more comprehensive rule exists) the less comprehensive rule will be eliminated.

**Example 5.3.2**: For the demonstration of counting occurrence of $C_3$. Consider the customer sequence $S_1 = \langle (A, D), B, D, C \rangle$. After setting $y$ to $a_{11} = A$ at Step 11, we derive $P_{ij}$ as $P_{11} = \{B, D, C\}$. Also at Step 13, $\Phi_{11} = \{B, C, D\}$. Then, we denote $D_y^C$ as the subset of $C_3$ and $D_y^C = \{(A, B, C), (A, B, D), (A, C, D), (A, D, E)\}$, for instance. Since $|D_y^C| = 4$ and set $m$ to 1 at Step 15, we have $m \leq 4$ and denote 1st candidate rule in $D_y^C$ as $(A, B, C)$ at Step 17. At Step 18, we identify that $(B, C)$ is a sub-sequence of $\Phi_{11}$ and it is also a sub-sequence in $P_{11}$. Thus, we increment the count of $(A, B, C)$ by 1 at Step 19. Then, set $m = 2$ at Step 20 and redo the process again for determining if $(B, D), (C, D)$, and $(D, E)$ are sub-sequences of $\Phi_{11}$. When $m > |D_y^C|$, we set $j$ to $j + 1$ (i.e., $j = 2$ in this case) and
go back to Step 10. Therefore, in the second round we redo the candidate-matching phase for \( y = a_{1j} \) until we finish the process for customer sequence \( S_1 \). Then, we escape from candidate-matching phase and go to Step 8 to retrieve the next sequence from the event database.

### 5.3.3 Algorithm \( HMD \): Data-Sets-Based Hierarchical Matching

On the other hand, the procedure \( HMD \) utilizes another efficient concept, i.e., data-sets-based hierarchical matching, to deal with the mining one-triggering event causality rules. *Instead of generating \( D_y^C \) from \( C_k \) to match \( P_{ij} \) and \( \Phi_{ij} \) as in algorithm \( HMC \), algorithm \( HMD \) will have \( w_1...w_{k-1} \) be generated from \( \Phi_{ij} \) and then determine if \( yw_1...w_{k-1} \) is in \( C_k \).*

**Example 5.3.3:** Recall the sequence \( S_1 = \langle (A, D), B, D, C \rangle \) in Example 5.3.2, consider the case of \( y = a_{11} \) (i.e., \( y = A \)) and \( \Phi_{11} = \{B, C, D\} \). Thus, the generating set of \( D_y^D \) will be \( D_y^D = \{(A, B, C), (A, B, D), (A, C, D)\} \). After checking if the candidate rules in \( D_y^D \) are members of \( C_3 \), we set \( y = a_{12} \) and redo the process again.

Consequently, some steps from Step 14a to Step 20a are re-organized to replace steps from Step 14 to Step 20 of algorithm \( HMC \). Part of algorithm of \( HMD \) is outlined below.

**Code Segment of Algorithm \( HMD \):** Data-sets-based hierarchical matching

```plaintext
14a. if \((a_{ij} \text{ is a triggering event in } C_k)\)
15a. 
16a. while \((m \leq |D_y^D|)\)
17a. 
18a. 
19a. 
20a. set \( m \) to \( m + 1 \);
```

Consequently, the complexity of algorithm \( HMD \) is approximately \( O\left(\frac{n!}{(n-k)!}\right) \) for generating a large \( k \)-event set. Recall the corresponding complexity, i.e., \( O(|C_k| \times n \times k) \), of algorithm \( HMC \) as mentioned in Section 5.3.2. Note that the efficiency of algorithms \( HMC \) and \( HMD \) highly depends on the amount of candidate \( k \)-event sequences \( |C_k| \), the average number of transactions for each customer \( n \) and the size of large event set \( k \). It can be verified that algorithm \( HMD \) will perform better in the case of low order candidate \( k \)-event sets, where \( |C_k| \) is very huge. On the other hand, algorithm \( HMC \) is powerful for generating higher order of large \( k \)-event sets, where \( |C_k| \) is very small. By combining the advantages of \( HMC \) and \( HMD \), we next present the adaptive algorithm \( HMA \) in Section 5.3.4 for the overall performance improvement.

### 5.3.4 Algorithm \( HMA \): Adaptive Hierarchical Matching

Since fewer candidate rules usually incur shorter execution time needed, with combining two algorithms \( HMC \) and \( HMD \), algorithm \( HMA \) will reduce the execution times for generating any large \( k \)-event set.
Specifically, when \(|C_k| \times n \times k > \frac{n!}{(n-k)!} \), i.e. \(|C_k| \times n \times k > \frac{(n-1)!}{(n-k)!} \), the adaptive algorithm \(HMA\) is similar to algorithm \(HMD\) and \(D_y^A = D_y^D\). For higher order candidate \(k\)-event sets, however, since the size of \(|C_k|\) is usually very small and \(|(|C_k| \times k) < \frac{(n-1)!}{(n-k)!} \), appropriately using the \(HMC\)-like procedure by setting \(D_y^A = D_y^C\) in the matching phase will greatly improve the program execution.

It is noted that for mining causality rules, some techniques, such as hash [68], partitioning [53], sampling [83] and scan reduction methodologies [68], can also be applied to further reduce the CPU and I/O overhead needed in our performance studies.

5.4 Experimental Studies

Explicitly, we generated several different transaction databases from a set of potentially frequent item-sets to evaluate the performance of the three proposed algorithms, i.e., \(HM_C\), \(HMD\) and \(HMA\). Note that the efficiency of the three algorithms has been evaluated by some real databases, such as bookstore transaction databases and grocery sales data. However, we show the experimental results from synthetic transaction data so as to obtain results of different workload parameters. To assess the relative performance of the algorithms and study their scale-up properties, we perform several experiments on a computer with a CPU clock rate of 450 MHz and 512 MB of main memory. The methods used to generate synthetic data are described in Section 5.4.1. The relative performance of algorithms is presented in Section 5.4.2. We conduct the scale-up performance evaluation in Section 5.4.3. The scenario of causality rule generation with multiple triggering events is explored in Section 5.4.4.

5.4.1 Generation of Synthetic Data

For obtaining reliable experimental results, the method to generate synthetic transactions we employed in this study is similar to the ones used in prior works [8, 37, 70]. Explicitly, we generated synthetic customer transactions to evaluate the performance of the proposed algorithms. These transactions mimic the transactions in the retailing environment where people buy sequences of sets of items. Without loss of generality, we use the notation \(Cx - Ty - Sm - In\) to represent a database in which the average number of transactions per customer \(|C| = x\), the average number of items per transaction \(|T| = y\), the average length of maximal potentially large sequences \(|S| = m\), and the average size of itemsets in maximal potentially large sequences \(|I| = n\).

For interest of space, we generate datasets by setting the number of customers (size of database) \(|D| = 100,000\), the number of maximal potentially large sequences \(N_S = 5,000\), the number of maximal potentially large itemsets \(N_I = 25,000\) and the number of different items \(N = 2,000\). To reflect the reality of mining environment, the mining datasets are generated as ASCII format in the following performance evaluation.
Figure 5.5: The execution times of three hierarchical matching algorithms on $C_{10} - T_{2.5} - S_{4} - I_{1.25}$ with $\text{min\_supp} = 0.5\%$.

### 5.4.2 Relative Performance

Figure 5.5 shows the relative execution times for the three algorithms with the database $C_{10} - T_{2.5} - S_{4} - I_{1.25}$ as the minimum support is 0.5%. As shown in Figure 5.5b, the experimental results show that $HMD$ outperforms $HMC$ when generating the large 2-event set and $HMC$ has better performance in generating $L_k$ as $k > 2$. As a result of combining the advantages of both $HMC$ and $HMD$, the adaptive algorithm $HMA$ outperforms algorithms $HMC$ and $HMD$ for generating any order of large event sets. As shown in Figure 5.5a, algorithm $HMA$ significantly reduces the total execution time for mining of $L_k$.

In addition, Figure 5.6 shows the relative execution times for the three algorithms as the minimum support is decreased from 3% to 1%. We did not plot the execution time of $HMD$ for low values of minimum support since its execution time in those cases is prohibitively large. This essentially agrees with the complexity of algorithm $HMD$. As expected, the execution times of all the algorithms increase as the support is decreased due to the large increase in the number of corresponding sequences.

In the low values of minimum support, $HMD$ performs worse than the other two algorithms because it generates and counts a much larger number of potential candidate, $D^D_y$, from datasets to match the candidates in candidate-sets. On the other hand, $HMC$ requires a longer execution time to match large amounts of candidates when generating low order of large event sets, such as $L_2$ and $L_3$. Thus, relatively, in the high values of minimum support, $HMC$ does not perform as well as the other two algorithms. With aggregating the advantages of $HMD$ and $HMC$, $HMA$ outperforms in any value of minimum support. These experimental results conform with the complexity analysis of the proposed
Figure 5.6: Relative performance of execution time
5.4.3 Scale-up

We next present in this subsection the results of scale-up experiments for the algorithm $HMA$. We also performed the same experiments for the other two algorithms, and chose not to report their results for interest of space since no additional insights were provided. Instead, we will present the scale-up results for some selected datasets. Figure 5.7 shows how algorithm $HMA$ scales up as the number of customers is increased from 10,000 to 1 million. We show the results for the dataset $C10 - T2.5 - S4 - I1.25$ with three levels of minimum support. The size of the dataset for 1 million customers was 703 MB in ASCII format. The execution times are normalized with respect to the times for the 10,000 customers dataset in the first graph, and with respect to the 100,000 customer dataset in the second. As shown in Figure 5.7 the execution times of algorithm $HMA$ scale very linearly. For the physical memory limitation and I/O throughput boundary, the execution times slightly increase beyond the linear proportion in the mining of large amount of transaction database. Obviously, with the growth of database size, less portion of database can be cached in the physical memory. Thus, a longer execution time will be required for scanning database.

Next, we investigate the scale-up as we increase the total number of items in a customer sequence. This increase was achieved in two different ways: (1) by increasing the average number of transactions per customer, keeping the average number of items per transaction the same; and (2) by increasing the average number of items per transaction, keeping the average number transactions per customer the same. We kept the size of the database roughly the same and fixed the minimum support in terms of the number of transactions in this experiment. All the experiments had the large sequence length set
The results are shown in Figure 5.8 where three cases for the minimum supports being 50, 100, 200 are evaluated. The average transaction size was set to 2.5 in the first graph, while the number of transactions per customer was set to 10 in the second. As shown, the execution times increase with the customer-sequence size. The main reason for the increase was that in spite of setting the minimum support in terms of the number of customers, the number of large sequences increase as the customer-sequence size increases. Another reason was that finding the candidates present in a customer sequence took a little more time. As the above discussion, the algorithm $HMA$ adaptively combines the techniques of $HMC$ and $HMD$ for matching candidate large $k$-event sequences. For generating a large $k$-event set, the complexity of $HMC$ is $O(|C_k| \times n \times k)$, where $|C_k|$ denotes the amount of candidate $k$-event sequences and $n$ is the average number of transactions for each customer, and that of $HMD$ is $O\left(\frac{n!}{(n-k)!}\right)$. Actually, with the boundary condition of whether $|C_k| \times k < \frac{(n-1)!}{(n-k)!}$ or not, algorithm $HMA$ will utilize the merit of $HMD$ for generating lower order of $L_k$s and employ $HMC$ to handle most higher order of $L_k$s.

Finally, we increase the size of the maximal large sequences (in terms of the total number of items in them) while keeping other factors the same. Again, we increase this size in two ways: (1) increasing the average length of the potential large sequences (i.e., the number of elements in the sequences); and (2) increasing the number of items in the large itemsets in the large sequences (i.e., the size of the elements). The first experiment was run with an average of 20 transactions per customer, 2.5 items per transaction and the large itemset size of 1.25. The second was run with an average of 10 transactions per customer, 5 items per transaction and the large sequence length of 4.
The results of this experiment are shown in 5.9 where the execution times decrease as the size of maximal large sequences increases. Since the probability that a specific sequence is included in a customer-sequence is proportional to the average size of the potential large sequences, increasing the total number of items in the potential large sequences leads to fewer maximal large sequences and hence shorter execution time. Especially, for a low minimum support, e.g. 1%, since large amounts of maximal large sequences are generated in $L_2$ and $L_3$ when $|S| = 2$, the execution time is significantly lengthened for matching the candidates of $C_3$ and $C_4$ with customer sequence datasets. Therefore, the relative execution time drops down from the case of $|S| = 2$ to that of $|S| = 4$.

5.4.4 Causality Rules Generation

Recall that we decompose the problem of mining causality rules into two phases. The first one is to discover one-triggering causality rules which contain only single triggering events. Then, in the second phase, we can use one-triggering causality rules to generate those rules in which a sequence of triggering events is allowed in each rule.

Here, we investigate the number of resulting causality rules on the database $C_{10−T2.5−S4−I1.25}$ as shown in Figure 5.10. Similarly to the above subsection of relative performance studies, we generate datasets with $N = 2,000$. Recall that $L^j_i$ denotes the large $i$ event set with $j$ triggering events at the beginning of each sequence followed by $i − j$ consequential events. $\sum L^j_i$ in Figure 5.10 is the total amount of causality rules with $j$ triggering events discovered from the database $C_{10−T2.5−S4−I1.25}$, e.g., $\sum L^2_i = 1,383$ with the support threshold $s\% = 0.4\%$. As in most previous mining techniques, the decrease of the support threshold leads to the value of $\sum L^j_i$ increases significantly. On the other hand, the value of $\sum L^j_i$ decreases with the growth of the number of trigger events $j$ as a whole.
<table>
<thead>
<tr>
<th>Support s%</th>
<th>1.0%</th>
<th>0.5%</th>
<th>0.4%</th>
<th>0.3%</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\sum L_i^1)</td>
<td>1,110</td>
<td>2,720</td>
<td>4,133</td>
<td>9,376</td>
</tr>
<tr>
<td>(\sum L_i^2)</td>
<td>179</td>
<td>719</td>
<td>1,383</td>
<td>4,534</td>
</tr>
<tr>
<td>(\sum L_i^3)</td>
<td>0</td>
<td>12</td>
<td>122</td>
<td>1,119</td>
</tr>
<tr>
<td>(\sum L_i^4)</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>281</td>
</tr>
<tr>
<td>(\sum L_i^5)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>31</td>
</tr>
</tbody>
</table>

Figure 5.10: Causality rules generation on the database C10-T2.5-S4-I1.25

### 5.5 Summary

As explored in this chapter, causality rule mining provides a more general framework for discovering useful knowledge hidden behind a short transaction database. To count the occurrence in the sequence database of each candidate \(k\)-event rule for causality rule mining, it is necessary to scan through the sequence database to do a sub-sequence matching. This procedure is very costly particularly in the presence of a huge number of candidate sets and a large event database. For efficient execution, the detection of an occurrence of a causality rule in an event sequence was handled as a sub-sequence matching problem using a hierarchical matching method. With this technique, we developed three algorithms to solve this problem, and conducted extensive experimental studies to evaluate the performance of algorithms proposed. The proposed adaptive algorithm, \(HMA\), was shown to outperform the other two algorithms, \(HMC\) and \(HMD\), which are based on candidate-sets-based and data-sets-based hierarchical matchings, respectively. Scale-up experiments showed that all three algorithms scale linearly with the number of customer transactions, the number of transactions per customer, and also the number of items in a transaction.
Chapter 6

Efficient Algorithms for On-line Web Log Mining with Dynamic Thresholds

6.1 Introduction

Data mining is an emerging interdisciplinary field of computer science research, encompassing various techniques for analyzing large datasets, including those on theory, statistics, databases and machine learning. The goal of data mining is mainly to discover interesting, important and previously unknown information. Various data mining capabilities have been explored in the literature [8, 12, 22, 34]. Among them, the one receiving a significant amount of research attention is on mining association rules [5]. Given a database of sales transactions, the goal of mining an association rule is to discover the relationship that the presence of some items in a transaction will imply the presence of other items in the same transaction. Since the earlier work in [5], several technologies on association rule mining have been developed, including: (1) improvement on association rule mining [7, 39, 53, 68]; (2) incremental updating [10, 26, 49]; (3) mining of generalized [79], multi-level [33], quantitative [80], multi-dimensional rules [88, 90]; (4) constraint-based rule mining [35, 46, 69] and multiple minimum supports issues [55, 87]; (5) temporal association rule mining [9, 24, 43, 48, 86]; (6) frequent episodes discovery [58, 60]; (7) causality rules discovery [50]; and (8) Web log mining [23, 56].

With the fast increase in Web activities, Web data mining has recently become an important research topic and is receiving an increasing amount of research interest from both academic and industrial environments [16, 28, 63, 72]. Among others, mining of path traversal patterns plays an essential role in the Web mining. Several prototypes and implemented systems are using mining techniques on path traversal patterns to explain aspects of behavior associated with the implicit time-variant nature of page viewers. In essence, log data which are collected by Web servers contain information about user access to the Web documents of the site. The size of logs increases rapidly due to two reasons: the rate that data are collected and the increase in the number of Web sites themselves. The analysis of these large
volumes of log data requires the employment of data mining methods. Following the paradigm of mining association rules [7], mined patterns are those access sequences of frequent occurrences. An example of this kind of pattern is a sequence \(<D_1, ..., D_n>\) of visited pages in a Web site. If such a sequence appears frequently enough, then this sequence indicates a frequent traversal pattern. Understanding user access patterns in such a Web environment will not only help improve the Web site design but also be able to lead to better marketing decisions.

While existing methods are efficient for the mining of frequent path traversal patterns from the access information contained in a log file, these approaches are likely to over evaluate associations. Explicitly, most previous studies of path traversal pattern mining are based on the model of a uniform support threshold, where a single support threshold is used to determine frequent traversal patterns without taking into consideration such important factors as the length of the pattern, the positions of Web pages, and the importance of a particular pattern, etc. As a result, a low support threshold will lead to lots of uninteresting patterns derived while a high support threshold may cause some interesting patterns with lower supports to be ignored. Hence, different support thresholds are deemed necessary for Web pages at different levels of Web sites. More specifically, we have the following observations to justify the model of mining traversal patterns with different support thresholds considered in this chapter.

1. A Web page at lower level of a Web site, e.g., the Web page \(H\) as shown in Figure 6.1, will naturally have a lower occurrence frequency than their corresponding higher level concepts, e.g., the portal Web page \(A\). Thus, if the minimum support is set to too high, those patterns that involve Web pages at lower levels of a Web site will not be found. On the other hand, if the minimum support
is set too low, a lot of uninteresting patterns will be produced. This phenomenon can be further described by the illustrate example below.

**Example 6.1.1:** Consider a Web site shown in Figure 6.1 and a database that contains four sequences: \(ABC\), \(AB\), \(AO\), \(ABEGH\). Suppose the minimum support (abbreviated as \(\text{min\_sup}\)) is \(\text{min\_sup} = 2\). Web pages \(A\) and \(B\) at higher levels of Web site are deemed frequent in this case, which might, however, be due more to their locations than to their contents.

2. The access design of Web pages is not the same. A *single-linked* Web page usually occurs less frequently than a *multiple-linked* Web page. An example of a multiple-linked Web page is a logo page of a company that is included in many Web documents. It is therefore undesirable to use the same support threshold for mining both single-linked and multiple-linked patterns.

3. The importance of Web pages is not the same. A free information Web page tends to attract more attention than Web pages designed for sales transactions. For Web sites owners, however, analyzing the access behavior of sales Web pages may be more important than that of free information Web pages.

4. The natural occurrence frequencies of items often vary greatly in real world \([55, 87]\). For instance, people access more Web pages of popular musics than those of classical musics. It is natural to have different support constraints on different groups of data items, or to set certain constraints on items of particular interest.

A naive way to handle the non-uniform supports is to apply existing algorithms with a very small support threshold and filter the results using higher minimum supports. This method is referred to as *Uniform Threshold Miner* (abbreviated as \(UTM\)) in this chapter. As will be validated by our experimental results later, \(UTM\) suffers from the drawback that many candidates have to be generated and later discarded, which implies not only knowledge of low interest produced but also excessive execution time involved.

Consequently, this chapter broadens the horizon of frequent path traversal pattern mining by introducing a flexible model of mining Web traversal patterns with dynamic thresholds. Specifically, we explore a new data mining capability which involves mining path traversal patterns with the concept of dynamic thresholds in a time-variant Web environment. Such a time-variant database is very popular in many applications, including daily fluctuations of a stock market, traces of a dynamic production process, scientific experiments, medical treatments, Web log data, weather records, to name a few. By properly employing some effective techniques devised for joining reference sequences, the proposed algorithm *DTM* (standing for Dynamic Threshold Miner) not only possesses the capability of mining with
dynamic thresholds, but also significantly improves the execution efficiency of mining Web traversal patterns. In addition, an innovative hybrid hash method with multiple hash tables is designed as an efficient technique for the mining with dynamic support thresholds.

Note that time advances, one has to include new data (e.g., data in Oct 2001 as shown in Figure 6.2) and $D' = D + D^+$ for mining. This scenario calls for the incremental mining capability. Consequently, an incremental version of DTM (referred to as incremental DTM or IDTM) is also developed. It is noted that since the occurrence frequency, i.e., $frequency(p, t)$, of Web page $p$ might be variant as the time, i.e., $t$, advances, the support threshold, e.g., $min\_sup(p, t_i)$, of Web page $p$ at the time point $t_i$ might be different from the one, e.g., $min\_sup(p, t_j)$, at another time point $t_j$. This is the very reason we use the term “dynamic support threshold” in this chapter. Further, it is worth mentioning that the problem of mining Web path traversal patterns with dynamic support threshold will be degenerated to the traditional one of mining Web path traversal patterns explored in previous works if the minimum support threshold function $min\_sup(\cdot)$ is assigned to be a uniform support threshold $min\_sup$, meaning that the model we consider can be viewed as a general framework of prior studies. Performance of algorithm DTM and the extension of existing methods is comparatively analyzed. It is shown that the option of algorithm DTM is very advantageous and leads to prominent performance improvement. Also, algorithm IDTM is shown to possess very good scalability as the data size increases. Sensitivity analysis on various parameters is conducted.

It is noted that there are some studies conducted on the incorporation of relatively flexible models in frequent pattern mining, including mining association rules by incorporation of multi-level (or taxonomy) concepts [33], mining of frequent patterns adaptive to user-specified support constraints [55, 87] and mining of frequent patterns in multi-dimensional circumstances [88]. Recently, the work in [73] developed some FP-tree based algorithms to deal with the flexible support constraints on mining multi-dimensional
frequent patterns. However, these works were mainly conducted to deal with individual issues and are not designed to deal with incremental mining, thus not providing a general framework for the mining with dynamic thresholds in a Web environment. These features distinguish this dissertation from others.

The rest of this chapter is organized as follows. Problem description is given in Section 6.2. Algorithms proposed are described in Section 6.3. Performance studies on various schemes are conducted in Section 6.4. This chapter concludes with Section 6.5.

6.2 Basic Principles

The problem addressed in this chapter is described in Section 6.2.1 with some novel notion introduced. The search space of algorithm $UTM$ which is devised for comparison purposes with $DTM$ is explored in Section 6.2.2.

6.2.1 Problem Description

A Web site can be abstractly viewed as a set of Web documents connected with hypertext links. The site can be represented by a simple, unweighted, directed graph which is a finite set of vertices and edges. A vertex corresponds to a document and an edge to a link. A description of Web path traversal patterns is given in Section 6.2.1.1. The mining of dynamic support thresholds is presented in Section 6.2.1.2.

6.2.1.1 Web path traversal patterns

A Web traversal in a Web site is a sequence of pages visited. The traversals are contained by a log file. Each entry in the log is of the form $(userID, s, d)$, which denotes the user identification number, the starting position $s$ and the destination position $d$. Although the actual representation of a log may contain additional information, the log can be abstractly viewed as described above for the purpose of mining traversal patterns. The beginning of a new traversal is marked with a pair $(userID, null, d)$. All pairs corresponding to the same user identification number are grouped together to form a traversal.

As pointed in [23], in an information providing environment where objects are linked together, users are apt to traverse objects back and forth in accordance with the links and icons provided. Suppose the traversal log contains the following traversal path for a user: $\{A, B, C, D, C, B, E, G, H, G, W, A, O, U, O, V\}$, based on the Web structure depicted in Figure 6.1. Then, it can be verified that the set of maximal forward references for this user is $\{ABCD, ABEGH, ABEGW, AOU, AOV\}$. After maximal forward references for all users are obtained, we then map the problem of finding frequent traversal patterns into the one of finding frequent occurring consecutive subsequences among all maximal forward references. For interest of brevity, our study in this chapter concentrates on the mining process.
after the point that maximal forward references are obtained. Readers interested in the details of obtaining maximal forward references are referred to [23].

The mining of traversal patterns with a single support proceeds as follows. A large reference sequence is a reference sequence that appeared in a sufficient number of times. In a set of maximal forward references, the number of times a reference sequence has to appear in order to be qualified as a large reference sequence is called the minimal support. A large \( k \)-reference is a large reference sequence with \( k \) elements. We denote the set of large \( k \)-references as \( L_k \) and its candidate set as \( C_k \), where \( C_k \), as obtained from \( L_{k-1} \), contains those \( k \)-references that may appear in \( L_k \). Explicitly, \( C_k \) is a superset of \( L_k \). After large reference sequences are determined, maximal reference sequences can then be obtained in a straightforward manner. A maximal reference sequence is a large reference sequence that is not contained in any other maximal reference sequence. For example, suppose that \{\( AB, BE, AD, CG, BG \)\} is the set of large 2-references (i.e., \( L_2 \)) and \{\( ABE, CGH \)\} is the set of large 3-references (i.e., \( L_3 \)). Then, the resulting maximal reference sequences are \( AD, BG, ABE \), and \( CGH \). A maximal reference sequence corresponds to a frequent access pattern in a Web system.

### 6.2.1.2 Dynamic support threshold

As mentioned earlier, dynamic support thresholds are needed for many applications and existing algorithms [23] use only a single user-specified minimum support. In fact, without specific knowledge, users will have difficulties in setting this support threshold to obtain their required results. If the support threshold \( S_{th} \) is set too large, there may be only a small number of results or even no result produced in which case, the user may have to guess a smaller threshold and conduct the mining again, which may or may not give a better result. If the threshold is too small, there may be too many results for the users, implying an extraordinarily long time in the computation. Without loss of generality, this chapter provides the determination of support threshold of Web documents is based on the statistical occurrence frequency of each Web page. Same as in [55, 87] the actual frequencies of the Web pages in the data are used as the basis for support assignments. The formal definition of the dynamic support threshold is given below.

**Definition 6.1:** The dynamic support threshold of the Web page \( p \) is defined as \( min\_sup(p, t) = \beta \times frequency(p, t) \), where the function \( frequency(p, t) \) is the actual occurrence frequency of Web page \( p \) in the data and \( \beta \in [0, 1] \) is a parameter to determine the relationship between the interestingness supports of Web pages and their occurrence frequencies.

**Example 6.2.1:** Consider again the example in Figure 6.1. Suppose the statistical occurrence frequency of Web page \( H \) is one third of that of the entrance Web page \( A \). A proper \( min\_sup(H) \)
value of Web page $H$ for mining frequent interesting patterns should be also about one third of the $\text{min\_sup}(A)$ of Web page $A$ given.

For interest of space, the minimum support threshold $S_{th}$ is employed for pruning some obsolete rules whose Web pages have very low accessing frequencies. Explicitly, once $\beta \ast \text{frequency}(p, t)$ of the Web page $p$ is smaller than $S_{th}$, its corresponding minimum support threshold is given as $S_{th}$. Thus, we have the dynamic support threshold of the newly identified dynamic mining model as the following formulas:

1. $\text{min\_sup}(p, t) = \beta \ast \text{frequency}(p, t)$ if $\beta \ast \text{frequency}(p, t) \geq S_{th}$;
2. $\text{min\_sup}(p, t) = S_{th}$ if $\beta \ast \text{frequency}(p, t) < S_{th}$.

### 6.2.2 Search Space Traversed by Algorithm UTM

As explained, we have to find all maximal traversal patterns that satisfy $\text{min\_sup}(p, t)$ first and then to calculate the occurrences of their corresponding sub-patterns for producing all path traversal patterns hidden in database $D$. Note that the downward closure property, which Apriori-based algorithms are based on to attain good efficiency, does not hold in this problem of mining with different support thresholds. Specifically, even though itemset $X$ is not a frequent pattern itself, one cannot assert that $XY$ is not a frequent pattern since $\text{frequency}(X, t) < \text{min\_sup}(X, t)$ does not imply $\text{frequency}(XY, t) < \text{min\_sup}(XY, t)$ due to the model of different support thresholds.

**Example 6.2.2:** Consider $\text{min\_sup}(x_1, t) = 30$, $\text{min\_sup}(x_2, t) = 20$ and $\text{min\_sup}(x_3, t) = 10$. If we find that pattern $x_1$ is not frequent, then it does not satisfy $\text{min\_sup}$ requirement at level 1. Under
a conventional Apriori-based association rule mining algorithm, this itemset is discarded since it will not be frequent. The potentially frequent itemsets \(x_1x_2\) and \(x_1x_3\) will then not be generated at level 2 for consideration. Clearly, this disposition is incorrect in mining traversal patterns with dynamic support thresholds since \(x_1x_2\) and \(x_1x_3\) still possible to be frequent with \(\text{min\_sup}(x_1x_2,t) = 20\) and \(\text{min\_sup}(x_1x_3,t) = 10\), indicating that the downward property is not valid in dynamic mining path traversal patterns.

Consequently, since the downward level-wise property, which holds for Apriori-like algorithms, is not valid in the Web traversal pattern mining with dynamic thresholds, we next describe the search scenario of \(UTM\) (standing for Uniform Threshold Miner). For practical applications, looking at all subsets of \(I\) is infeasible due to the huge search space. For the special case \(I = \{A^0.1, B^0.2, C^0.3, D^0.4\}\) we visualize the search space that forms a lattice in Figure 6.3 where \(A^0.1\), for example, denotes \(\text{min\_sup}(A, t) = 0.1\). The frequent itemsets are located in the upper part of Figure 6.3 whereas the infrequent ones are located in its lower part. Although not explicitly specifying the support value for each of the itemsets, we assume that the bold border separates the frequent itemsets from the infrequent ones. The existence of such a border is independent of any particular database \(D\) and \(\text{min\_sup}(\cdot)\), and is solely guaranteed by the downward closure property of itemset support. The basic principle of \(UTM\) is to employ this border to efficiently prune the search space. As soon as the border line is found, we are able to restrict ourselves to the determination of the support values of the itemsets above the border and to ignore the itemsets below.

However, it should be noted that a linearly growing number of items still implies an exponential growing number of itemsets to be considered. In fact, as will be validated by experimental results later, the increase of candidates often causes a huge increase of execution time and a drastic performance degradation, meaning that without utilizing the techniques of algorithm \(DTM\) we proposed, a direct extension to prior work is not able to handling the path traversal pattern mining with dynamic support thresholds efficiently.

### 6.3 DTM: Dynamic Threshold Miner

In this section, we derive algorithm \(DTM\) (standing for Dynamic Threshold Miner) to determine the frequent traversal patterns, i.e., frequent reference sequences, from the maximal forward references obtained while allowing of different support thresholds for different Web pages. Also, we devise algorithm \(IDTM\) (standing for Incremental Dynamic Threshold Miner) to deal with the incremental mining of frequent Web path traversal patterns. The flowchart of algorithms \(DTM\) and \(IDTM\) is depicted in Figure 6.4. We present algorithm \(DTM\) in Section 6.3.1 and algorithm \(IDTM\) in Section 6.3.2.
6.3.1 Algorithm DTM

As in most previous works, we use \( L_k \) to represent the set of all frequent \( k \)-references sequences and \( C_k \) is a set of candidate \( k \)-reference sequence where \( C_k \) is a superset of \( L_k \). In Section 6.3.1.1, we explore the mining process of algorithm DTM. The functions of Web path candidate generation in DTM, including \( \text{Prepare} \), \( \text{SeqGen}_{C_2} \) and \( \text{SeqGen}_{C_k} \), are devised in Section 6.3.1.2.

6.3.1.1 Web path traversal patterns with dynamic support thresholds

Let \( \text{min}_\sup(p) \) denote the minimum support of page \( p \). The minimum support of a reference sequence \( c \), denoted by \( \text{MinSup}(c) \), is the lowest \( \text{min}_\sup \) value among the pages in the reference sequence, i.e.,
\[
\text{MinSup}(c) = \min_{p \in c} \{ \text{min}_\sup(p) \}.
\]
The algorithm DTM can be outlined as follows.

**Algorithm DTM(\( P \), \( D \))**

1. \( SD = \text{Prepare}(P, D) \);
2. \( L_1 = \{ < s > | s \in SD, s.\text{count} \geq \text{min}_\sup(s) \} \);
3. for \( (k = 2; L_{k-1} \neq \emptyset; k + +) \) do begin
4. if \( (k = 2) \) then \( C_2 = \text{SeqGen}_{C_2}(SD) \);
5. else \( C_k = \text{SeqGen}_{C_k}(L_{k-1}) \);
6. end
7. Scan the database \( D \) and compute the value of \( c.\text{count} \), i.e., the support of each candidate \( c \in C_k \);
8. \( L_k = \{ c \in C_k | c.\text{count} \geq \text{MinSup}(c) \} \);
9. end
(a) Structure of a Web site

(b) Database

(c) min_sup of pages

<table>
<thead>
<tr>
<th>ID</th>
<th>path</th>
<th>min_sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;A, B, C, D&gt;</td>
<td>{A} 5</td>
</tr>
<tr>
<td>2</td>
<td>&lt;C, A, B, D&gt;</td>
<td>{B} 4</td>
</tr>
<tr>
<td>3</td>
<td>&lt;A, B&gt;</td>
<td>{C} 2</td>
</tr>
<tr>
<td>4</td>
<td>&lt;C, A, D&gt;</td>
<td>{D} 1</td>
</tr>
<tr>
<td>5</td>
<td>&lt;A, D&gt;</td>
<td>{E} 1</td>
</tr>
</tbody>
</table>

Figure 6.5: An illustrative example with (a) structure of a Web site, (b) the example database, and (c) minimal supports of pages

10. \( Answer = \cup_k L_k \);

To produce the seeds for generating \( C_2 \), we first have to perform the function of \( Prepare() \) with two arguments \( P \) and \( D \) where \( P \) is the set of pages to be sorted in ascending order of their supports and \( D \) is the database containing all maximal forward references. The function \( Prepare() \) will be described next.

In \( DTM \), frequent Web path traversal patterns are generated by using multiple passes over the database. The large reference sequences \( L_k \) found in the \((k - 1)^{th}\) pass are used to generate the candidate reference sequences \( C_k \) using the \( SeqGen_{C_k} \) function, except when \( k = 2 \) for which the candidate generation function is \( SeqGen_{C_2} \), as stated in step 4 and step 5 of algorithm \( DTM \). Next, the database is scanned and the support of candidates in \( C_k \) is counted (step 7). Finally, we can obtain new large reference sequences by removing those sequences whose support counts are smaller than their corresponding values of \( MinSup(\cdot) \) (step 8). Both \( SeqGen_{C_k} \) and \( SeqGen_{C_2} \) will be described later.

6.3.1.2 Generation of Candidate 2-References

In this section, we describe three major functions, i.e., function \( Prepare \), function \( SeqGen_{C_2} \), and function \( SeqGen_{C_k} \), of Web path candidate generation in algorithm \( DTM \). First, function \( Prepare(P, D) \) is devised to produce not only \( L_1 \) but also the seed for generating \( C_2 \). With the input of two arguments \( P \) and \( D \), function \( Prepare() \) can be outlined below.

**Function Prepare\((P, D)\):** to produce the seeds for the generation of \( C_2 

1. Scan the database \( D \) to record the actual support count of each page \( p \in P \);
2. Sort pages in \( P \) in ascending order of their support counts;
3. Following the sorted order, find the first page \( f \) in \( P \) that the count of \( f \) exceeds \( \text{min_sup}(f) \);
4. Insert page \( f \) into the seed set \( SD \);
5. for each subsequent page \( i \) in \( P \) after \( f \)
### Example 6.3.1:

Consider a Web site as shown in Figure 6.5a. A database of five maximal forward references and the interesting support threshold of each Web page are given in Figure 6.5b and Figure 6.5c, respectively. After one pass of the database scan, we have the following support counts: \( A.\text{count} = 5 \), \( B.\text{count} = 3 \), \( C.\text{count} = 2 \), \( D.\text{count} = 3 \) and \( E.\text{count} = 0 \). As a result, we can obtain set \( SD \) and \( L_1 \) as shown in Figure 6.6b and Figure 6.6c, respectively. It is noted that page \( E \) is not in set \( SD \) since \( E.\text{count} \) is smaller than the value of \( \text{min\_sup}(D) \), and page \( B \) is not in \( L_1 \) because \( B.\text{count} \) is smaller than the value of \( \text{min\_sup}(B) \).

Next, in function \( \text{SeqGen}_{C_2} \), we employ argument \( SD \) to generate \( C_2 \). The algorithm can be outlined as follows.

**Function \( \text{SeqGen}_{C_2}(SD) \):** to generate \( C_2 \) from the set \( SD 

1. for each pages \( p \) in \( SD \) in the same order do begin
2. \hspace{1em} if \( p.\text{count} \geq \text{min\_sup}(p) \) then
3. \hspace{2em} for each page \( q \) in \( SD \) that is after \( p \) do begin
4. \hspace{3em} \hspace{1em} if \( q.\text{count} \geq \text{min\_sup}(p) \) then
5. \hspace{3em} \hspace{2em} \hspace{1em} insert \( \{pq\} \) and \( \{qp\} \) into \( C_2 \);
6. \hspace{3em} end
7. \hspace{2em} end
8. end
Example 6.3.2: Let us continue with Example 6.3.1. We can get $C_2$ shown in Figure 6.6d. $\{BA\}$ and $\{AB\}$ are not in $C_2$ because the support count of $B$ is smaller than the $\text{min}_\text{sup}(B)$. Hence, $\{BA\}$ and $\{AB\}$ are not frequent.

It is noted that a Web page $p \notin L_1$ does not imply that its corresponding occurrence will not be greater than the $\text{min}_\text{sup}$ of an earlier page in the sorted order. For instance, the page $B$ is in $SD$ but not in $L_1$. If we use $L_1$ to generate $C_2$, some candidate reference sequence such as $\{BD\}$ will be missed. That is the very reason we use the set $SD$ instead of $L_1$ in the function $\text{SeqGen}_{C_2}$.

6.3.1.3 Generation of Candidate k-References

Finally, we introduce the methodology of function $\text{SeqGen}_{C_k}$ with its input argument $L_{k-1}$, i.e., the set of all frequent $(k-1)$-references. Specifically, in the join step, we join $L_{k-1}$ with $L_{k-1}$ first. It is noted that the join step of function $\text{SeqGen}_{C_k}$ is different from the one in [23]. In [23], two distinct sequences from $L_{k-1}$, say $r_1, ..., r_{k-1}$ and $s_1, ..., s_{k-1}$, are joined to form a $k$-reference sequence if either $r_1, ..., r_{k-1}$ contains $s_1, ..., s_{k-2}$ or $s_1, ..., s_{k-1}$ contains $r_1, ..., r_{k-2}$ (i.e., after dropping the first element in one sequence and the element in the other sequence, the remaining two $(k-2)$-reference sequences are identical). Even though being valid in [23], such a join process is not applicable to the mining with dynamic support thresholds.

Example 6.3.3: In [23], $\{AB\}$ and $\{BC\}$ can be joined to form $\{ABC\}$. In the method in [23], if $\{BC\}$ is not a large 2-reference sequence, i.e., $\{BC\} \notin L_2$, then $\{ABC\}$ will not be generated as a candidate 3-reference. This, however, causes a potential large 3-reference $\{ABC\}$ to be incorrectly ignored while considering different support thresholds since we may have $\text{min}_\text{sup}(A) < \text{min}_\text{sup}(B) < \text{min}_\text{sup}(C)$. To address this issue, we devise in this chapter three joinable forms for algorithm $\text{DTM}$, namely head_join, mid_join and tail_join forms.

Definition 6.2: The minimal support page of a reference sequence $r$ is $\text{MSP}(r) = \{p|p \in r, \text{min}_\text{sup}(p) = \text{MinSup}(r)\}$.

For instance, considering $\{ABC\}$ in Example 3.3, we have $\text{MSP}(\{ABC\}) = \{A\}$.

Definition 6.3: Suppose $r$ is a $k$-reference sequence which contains $r_1, ..., r_k$.

(i) If $r_1 \notin \text{MSP}(r)$ and $r_k \notin \text{MSP}(r)$, then $r$ is one reference sequence of mid_join form.

(ii) If $r_1 \in \text{MSP}(r)$, then $r$ is one reference sequence of head_join form.

(iii) If $r_k \in \text{MSP}(r)$, then $r$ is one reference sequence of tail_join form.

Suppose that $\text{min}_\text{sup}(A) < \text{min}_\text{sup}(B) < \text{min}_\text{sup}(C)$. Then $r = \{BAC\}$ is a reference sequence of mid_join form since $r_1 = B \notin \text{MSP}(\{BAC\}) = \{A\}$ and $r_3 = C \notin \text{MSP}(\{BAC\}) = \{A\}$.
\{A\}. In addition, \{ABC\} is a reference sequence of head\_join form and \{BCA\} is a reference sequence of tail\_join form.

The pseudo-code of function \textit{SeqGen}_{C_k} is presented below. In \textit{SeqGen}(\cdot), by using the above mentioned three join forms, we join \(L_{k-1}\) with \(L_{k-1}\) to obtain \(C_k\) in the join step. Then, in the prune step, we delete all sets of references \(c \in C_k\) which are infrequent. Finally, \textit{SeqGen}(\cdot) returns a superset of the set of all frequent \(k\)-references.

\textbf{Function \textit{SeqGen}_{C_k}(L_{k-1})}: to produce \(C_k\) from \(L_{k-1}\)

// First, in the join step, we join \(L_{k-1}\) with \(L_{k-1}\).

1. insert into \(C_k\)
2. select \(p.\,\text{page}_1, p.\,\text{page}_2, \ldots, p.\,\text{page}_{k-1}, q.\,\text{page}_{k-1}\) from \(p, q \in L_{k-1}\) // mid\_join form
   where \(p.\,\text{page}_2 = q.\,\text{page}_1, \ldots, p.\,\text{page}_{k-1} = q.\,\text{page}_{k-2}\)
   and \(p.\,\text{page}_1 \notin \text{MSP}(p)\) and \(q.\,\text{page}_{k-1} \notin \text{MSP}(q)\)
3. Union
4. select \(p.\,\text{page}_1, p.\,\text{page}_2, \ldots, p.\,\text{page}_{k-1}, q.\,\text{page}_{k-1}\) from \(p, q \in L_{k-1}\) // head\_join form
   where \(p.\,\text{page}_1 = q.\,\text{page}_1, \ldots, p.\,\text{page}_{k-2} = q.\,\text{page}_{k-2}\)
   and \(p.\,\text{page}_1 \in \text{MSP}(p)\) and \(p.\,\text{page}_1 \in \text{MSP}(q)\)
5. Union
6. select \(p.\,\text{page}_1, q.\,\text{page}_1, p.\,\text{page}_2, \ldots, p.\,\text{page}_{k-1}\) from \(p, q \in L_{k-1}\) // tail\_join form
   where \(p.\,\text{page}_2 = q.\,\text{page}_2, \ldots, p.\,\text{page}_{k-1} = q.\,\text{page}_{k-1}\)
   and \(p.\,\text{page}_{k-1} \in \text{MSP}(p)\) and \(p.\,\text{page}_{k-1} \in \text{MSP}(q)\)

// Next, in the prune step, we delete all sets of references \(c \in C_k\) which can not be frequent.

7. for each set of reference \(c \in C_k\) do begin
8. for each \(k-1\) subsets \(s\) of \(c\) do begin
9. if \(|\text{MSP}(c)| \geq 2\) or \(\text{MinSup}(s) = \text{MinSup}(c)\) then
10. if \((s \notin L_{k-1})\) then
11. delete \(c\) from \(C_k\);
12. end
13. end

Definition 6.3 leads directly to Lemma 6.1 below.

\textbf{Lemma 6.1}: Any reference sequence belongs to one kind of joinable form as mentioned in Definition 6.3.

By omitting its straightforward proof, Theorem 1 can be obtained from Lemma 1 and function \textit{SeqGen}_{C_k}.

\textbf{Theorem 6.1}: The join step of \textit{SeqGen}_{C_k} will produce all possible frequent \(k\)-reference sequences.

\textbf{Example 6.3.4}: Consider \(L_3\) shown in Figure 6.7a and \(\text{min\_sup}(A) < \text{min\_sup}(B) < \text{min\_sup}(C) < \text{min\_sup}(D) < \text{min\_sup}(E)\). In the join step, \{CBDE\} is generated from \{CBD\} and \{BDE\} as one form of mid\_join. \{ADEC\} and \{ADCE\} are generated from \{ADE\} and \{ADC\} as one form
of head _join. The potential candidates (C_4^*), which are generated after the join step, are shown in Figure 6.7b. Then, in the prune step, we delete sequence \{ADCE\} because sequence \{ACE\} is not in L_3. The sequence \{CBDE\} is not deleted although sequence \{CDE\} is not in L_3 since the MinSup (\{CDE\}) = min_sup(C) > min_sup(B). Although not satisfying min_sup(C), the support of \{CDE\} may or may not satisfy min_sup(B) since min_sup(B) < min_sup(C).

6.3.2 Algorithm Incremental DTM (IDTM)

The mining process with the incremental mining capability can be decomposed into two procedures below.

1. **Preprocessing procedure** deals with the mining on the original database \(D\).

2. **Incremental procedure** employs for the incremental update of the mining on an ongoing time-variant database \(D' = D + D^+\).

In accordance with the dynamic threshold mining techniques of algorithm DTM in the preprocessing procedure, algorithm Incremental DTM (abbreviated as IDTM) is devised to maintain frequent reference sequences in the incremental procedure. Recall the time-variant database as shown in Figure 6.2 again. Let \(t_j\) be the last time point of database \(D\) and \(t_j\) be the last time point of database \(D'\) where \(D^+\) represents the dataset between \(t_i\) and \(t_j\). With properly employing the cumulative information discovered from the preprocessing procedure by algorithm DTM, as will become clear later, algorithm IDTM can minimize the I/O cost of the incremental update on frequent patterns to only one scan of database \(D'\). Let \(L_k^D\) be the set of frequent \(k\)-reference sequences generated by database \(D\). \(C_k^D\) represents the set of candidate \(k\)-reference sequences generated by algorithm DTM from the preprocessing procedure. The meanings of various symbols used are given in Table 6.1.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D$</td>
<td>The original portion of an ongoing database</td>
</tr>
<tr>
<td>$D^+$</td>
<td>The added portion of an ongoing database</td>
</tr>
<tr>
<td>$D' = D + D^+$</td>
<td>The whole ongoing database</td>
</tr>
<tr>
<td>$\text{min}_\text{sup}(p, t)$</td>
<td>The minimum support threshold of page $p$ at time $t$</td>
</tr>
<tr>
<td>$\text{MinSup}(c, t)$</td>
<td>The minimum support threshold of a reference sequence $c$ at time $t$</td>
</tr>
<tr>
<td>$L^D_k$</td>
<td>The set of frequent $k$-reference sequences generated by the database $D$</td>
</tr>
<tr>
<td>$C^D_k$</td>
<td>The set of candidate $k$-reference sequences generated by the database $D$</td>
</tr>
<tr>
<td>$c.\text{count}$</td>
<td>The count of reference sequence $c$ in the database $D$</td>
</tr>
</tbody>
</table>

Table 6.1: Meanings of symbols used in the Web mining with dynamic thresholds

To deal with the incremental procedure of database $D'$, algorithm IDTM can be outlined below.

**IDTM: Incremental Dynamic Traversal Miner($N, D'$)**

1. $SD' = \text{Prepare}(P, D')$;
2. $L^D'_1 = \{ <s > | s \in SD', s.\text{count} \geq \text{min}_\text{sup}(s, t_j) \}$;
3. $C^D_1 = SD'$;
4. for $(k = 2; C^D_{k-1} \neq \emptyset; k++)$
   5. if $k = 2$ then $C^D_2 = \text{SeqGen}_{C_2}(SD')$;
   6. else $C^D_k = \text{SeqGen}_{C_k}(C^D_{k-1} \cup L^D_{k-1})$;
7. Scan database $D'$ to update the counts of reference sequences in $C^D_k$;
8. for each reference sequence $c \in C^D_k$
   9. if $c \in L^D_k$ then
      10. if $c.\text{count}^{D^+} + c.\text{count}^{D} \geq \text{MinSup}(c, t_j)$ then insert $c$ into $L^D_k$;
      11. delete $c$ from $C^D_k$;
      12. else if $c \notin L^D_k$ then
          13. if $c.\text{count}^{D^+} \leq \text{MinSup}(c, t_j) - \text{MinSup}(c, t_i)$ then delete $c$ from $C^D_k$;
9. end
15. end
16. Scan database $D$ to update the counts of reference sequences in $\cup_k C^D_k$;
17. for each reference sequence $c \in \cup_k C^D_k$
18. if $c.\text{count}^{D^+} + c.\text{count}^{D} \geq \text{MinSup}(c, t_j)$ then insert $c$ into $L^D_k$;
19. end
20. Answer = $\cup_k L^D_k$;

Operations in line 1 are the same as those in $DTM$ for producing the seeds for generating $C^D_2$. For each subsequent pass, say pass $k$, we make use of $\text{SeqGen}_{C_k}$ to generate $C^D_k$ except when $k = 2$, we use $\text{SeqGen}_{C_2}$ to generate $C^D_2$. After one scan of the increment $D^+$, we obtain $c.\text{count}^{D^+}$ of each reference sequence $c$ in $C^D_k$. Moreover, we can know $c.\text{count}^{D^+}$ of the reference sequences which exist both in $C^D_k$ and $L^D_k$. Therefore, for each reference sequence $c$ which exists both in $C^D_k$ and $L^D_k$, if $c.\text{count}^{D^+} \geq \text{MinSup}(c, t_j)$, we can delete $c$ from $C^D_k$ and insert it into $L^D_k$; else we only need to delete $c$ from $C^D_k$. The reference sequences which are in $C^D_k$ but not in $L^D_k$ are deleted from $C^D_k$ if $c.\text{count}^{D^+} \leq \text{MinSup}(c, t_j) - \text{MinSup}(c, t_i)$. In the next pass, $C^D_{k+1}$ is generated by $\text{SeqGen}_{C_k}()$ which
takes $C_2^{D'} \cup L_2^{D'}$ as its argument. Finally, we need to scan the original database $D$ once to update the count of candidate reference sequence in $\cup_k C_k^{D'}$. It can be seen that every reference sequence $c$ in $\cup_k C_k^{D'}$ will be inserted into the corresponding $L_k^{D'}$ if it is frequent with respect to $\text{MinSup}(c, t_j)$.

**Example 6.3.5**: Consider the database in Figure 6.8b with the corresponding $\text{min} \_ \text{sup}$ of each page in Figure 6.8a. Assume that frequent $k$-reference sequences in the original database $D$ have been determined by algorithm $DTM$ as shown in Figure 6.8c. Further, $SD'$ and $L_1^{D'}$ are given in Figure 6.9. Similarly to the process in algorithm $DTM$, $C_2^{D'}$ is generated by function $\text{SeqGen}_{C_2}(\cdot)$. Next, after the scan of $D^+$, some candidate reference sequences in $C_2^{D'}$ can be pruned. It is noted that candidate reference sequences in $C_2^{D'}$ may be frequent if they are also in $L_2^{D'}$. As a result, $\{CD, BD, AD, CA\}$ are removed from $C_2^{D'}$, while $\{BD\}$ is not inserted into $L_2^{D'}$ since $BD.\text{count}^{D'} + BD.\text{count}^D = 2 < \text{MinSup}(BD, t_j) = 3$. Candidate reference sequences in $C_2^{D'}$ which are not in $L_2^{D'}$ may remain in $C_2^{D'}$ if they satisfy certain conditions as states in step 13 of algorithm $IDTM$. For instance, if $DE.\text{count}^{D'} = 1 \leq \text{MinSup}(DE, t_j) - \text{MinSup}(DE, t_i)$, $\{DE\}$ could not be frequent in $D'$ and should be removed from $C_2^{D'}$. In the next pass, $C_3^{D'}$ is generated by $\text{SeqGen}_{C_3}(\cdot)$. It is noted that the argument of $\text{SeqGen}_{C_3}(\cdot)$ is $C_2^{D'} \cup L_2^{D'}$, rather than only $L_2^{D'}$. The subsequent processes are presented in Figure 6.9. Finally, the scan of original database $D$ is employed to update the count of candidate reference sequence in $\cup_k C_k^{D'}$. Then, the reference sequence $c$ in $\cup_k C_k^{D'}$ will be inserted into the corresponding $L_k^{D'}$ if it is frequent under $\text{MinSup}(c, t_j)$. For the example candidate reference sequence $\{ACE\}$ in $C_3^{D'}$, we will insert it into $L_3^{D'}$ since $ACE.\text{count}^{D'} = 2 \geq \text{MinSup}(ACE, t_j) = 2$.
Several enhanced techniques are employed to improve the execution performance of DTM and IDTM. First, DTM can generate $C'_3$ from $C_2 \times C_2$ instead of from $L_2 \times L_2$, and then we can find $L_2$ and $L_3$ together when next database scan is performed. By using this technique of scan reduction, we can save one round of database scan. The total number of database scans can be further reduced to two, if $C'_k$ for $k \geq 3$ is generated from $C'_{k-1}$ and all $C'_k$ for $k \geq 2$ can be stored in the memory. As such, once $C_2$ has been determined, only one additional scan of database is needed in the mining process.

In addition, the hash filtering technique is very efficient for the generation of candidate reference sequences. In essence, the hash filtering technique employs a hash table, which is built in the previous pass to test the eligibility of a $k$-itemset [68]. The hash filtering technique adds a candidate into $C_k$ only if that $k$-itemset is hashed into a hash entry whose value is larger than the corresponding MinSup($\cdot$). With the characteristic of a uniform support threshold, most previous works, e.g., DHP [68], employ a single hash table to effectively prune out a huge amount of infrequent itemsets from $C_k$ candidates, especially for $C_2$. In accordance with the concept of DHP, an itemset is viewed infrequent if the total counts of all $k$-reference sequences in one memory bucket are below the lowest MinSup value of $k$-reference sequences. However, to deal with different support thresholds, since each $k$-reference sequence has its MinSup value, this value has to be stored in each bucket of the hash table. It is noted that using

---

**Figure 6.9:** Frequent reference sequences generation for the dynamic mining with IDTM

6.3.2.1 Use of Scan Reduction and Multiple Hash Tables
a single hash table, the efficiency of hashing is controlled by the lowest MinSup of k-reference sequences of each bucket. If the variation in MinSup of k-reference sequences in each bucket is significant, the advantage of hashing will diminish. Consequently, the use of multiple hash table is devised in this chapter to deal with the very problem of mining with different thresholds. That is, we would like to map k-reference sequences with n different values of MinSup into m hash tables where $1 \leq m \leq n$.

The notion of using multiple hash tables can be illustrated by the example below.

**Example 6.3.6:** Consider a database of three maximal forward references in Figure 6.10a and the support thresholds of Web pages are shown in Figure 6.10b. For each maximal forward reference, after the occurrences of all 1-reference sequences are counted, all 2-reference sequence of this maximal forward reference are generated as shown in Figure 6.10c. Next, each 2-reference sequence is hashed into the hash table according to its MinSup value. For instance, the MinSup value of \{AC\} is 3, then \{AC\} is hashed into hash table 1 in Figure 6.10d.

### 6.4 Performance Studies

To assess the performance of algorithms proposed, we perform several experiments on a computer with a CPU clock rate of 450 MHz and 128 MB of main memory. The methods used to generate synthetic data are described in Section 6.4.1. Performance comparison of algorithms DTM and UTM is conducted in Section 6.4.2. Results on scaleup experiments of IDTM are presented in Section 6.4.3.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>(D)</td>
<td>Number of transactions in the original database</td>
</tr>
<tr>
<td>(d)</td>
<td>Number of incremental transactions</td>
</tr>
<tr>
<td>(N)</td>
<td>Number of vertices</td>
</tr>
<tr>
<td>(F)</td>
<td>Maximum fanout</td>
</tr>
<tr>
<td>(L)</td>
<td>Average pattern length</td>
</tr>
<tr>
<td>(P)</td>
<td>Total number of patterns</td>
</tr>
<tr>
<td>(C)</td>
<td>Corruption level</td>
</tr>
</tbody>
</table>

Table 6.2: Meaning of parameters for the synthetic data in the Web mining with dynamic thresholds

### 6.4.1 Generation of synthetic workload

This section contains the experimental results on the evaluation of DTM. We will show that the new model allows us of identifying frequent reference sequence with very low supports whereas avoiding generating a huge number of meaningless frequent reference sequence. In our experiment, the browsing scenario in a World Wide Web environment is considered. To generate a synthetic workload, we employ the same method as the one adopted in [63]. Also, to determine the values of parameters, we referenced some logged traces collected from a gateway in our working location.

The site structure is first generated with two significant parameters, i.e., the total number \(N\) of Web documents and the maximum number \(F\) of hypertext links inside a document. Each Web document has a size (in KB) which follows a mixed distribution, i.e., Lognormal, for sizes less than 133 KB and Pareto for larger ones. 93% of the documents are HTML documents and the remaining ones are large binary objects (i.e., images, sounds, etc.). For each document, its fanout, i.e. the number of its hypertext links, is determined by a uniform distribution between 1 and \(F\).

Large paths are chosen from a collection of \(P\) paths which represent the patterns. The length of each path pattern is determined by a Poisson distribution with the average length \(L\). Each user traversal, is uniformly selected from the \(P\) path patterns. The corruption of patterns is represented with the corruption level \(C\) which is the number of vertices from the path pattern to be substituted. This number follows a Poisson distribution with average value \(C\). Table 6.2 lists all the symbols used. Datasets are characterized by the values for parameters \(N\), \(L\), \(F\), \(D\) and \(d\). In the following, we use the notation \(Nx - Ly - Fz - Dm - dn\) to represent a database in which \(N = x\) thousands, \(D = m\) thousands, \(d = n\) thousands, \(|L| = y\), and \(|F| = z\). More details for the synthetic data generator can be found in [63].

123
Figure 6.11: Number of candidate and frequent reference sequence found
6.4.2 Performance Comparison between DTM and UTM

In our experiments, we employ the actual frequencies of the pages in the data set as the basis for the min_sup assignments. Specifically, we use the formulas in Definition 1, i.e., \( \text{min}_\text{sup}(p, t) = \beta \times \text{frequency}(p, t) \) if \( \beta \times \text{min}_\text{sup}(p, t) \geq S_{th} \), and \( \text{min}_\text{sup}(p, t) = S_{th} \), otherwise. The function \( \text{frequency}(p, t) \) is the actual occurrence frequency of Web page \( p \) in the data. \( S_{th} \) is the user-specified lowest minimum node support allowed. \( \beta \) is set between 0 and 1, so as to reflect the relationship between the supports of the Web pages and their frequencies. If \( \beta = 0 \), we have only one minimum support, \( S_{th} \), which is the same as the traditional path traversal pattern mining. If \( \beta = 1 \) and \( \text{frequency}(p, t) \geq S_{th} \), \( \text{frequency}(p, t) \) is the min_sup value for page \( p \).

We generated a number of datasets to conduct our experiments to ensure the reliability of the results obtained. However, only the results on the dataset, \( N1 - L12 - F20 - D100 \), are presented and those on other datasets are omitted for interest of space. For our experiment, we use three values of \( S_{th} \), 100 (i.e., 100,000 * 0.001), 300 (i.e., 100,000 * 0.003) and 500 (i.e., 100,000 * 0.005).

6.4.2.1 Experimental Results on the Number of Reference Sequence

Figure 6.11 shows the number of candidate and frequent reference sequences found. The horizontal lines give the number of candidate and frequent reference sequences found using algorithm UTM with uniform minimal supports at 100, 300 and 500. To show the impact of the value of \( \beta \) to the numbers of candidate and frequent reference sequences found by DTM, six values of \( \beta \), i.e., 1, 0.8, 0.6, 0.4, 0.2 and 0.1, are considered.

As depicted in Figure 6.11, the numbers of candidate and frequent reference sequences are significantly reduced by DTM especially when \( \beta \) is large. When \( \beta \) becomes smaller, the numbers of candidate and frequent reference sequences found by DTM approach to those found by UTM. This agrees with our intuition since when \( \beta \) becomes smaller, the number of pages whose min_sup values approach \( S_{th} \) will increase.

6.4.2.2 Experimental results on the Execution Time

Figure 6.12 shows the comparison on execution time between UTM and DTM. The horizontal lines indicate the execution times of algorithm UTM of a uniform minimal at 100, 300, and 500. It can be seen that DTM also reduces the execution time significantly. Similarly, the numbers of candidate reference sequences and frequent reference sequence become smaller when the value of \( \beta \) increases. When \( \beta \) decreases, the difference of execution time of DTM and UTM decreases.
Figure 6.12: The relative execution time of algorithms $DTM$ and $UTM$
Scaleup performance of IDTM in various of $|D|$:

- $S_{th} = 0.001\times|D|$
- $S_{th} = 0.003\times|D|$
- $S_{th} = 0.005\times|D|$

Relative time vs. $|D|$, the original database size:

Scaleup performance of IDTM in various of $|d|$:

- $S_{th} = 0.001\times|D|$
- $S_{th} = 0.003\times|D|$
- $S_{th} = 0.005\times|D|$

Relative time vs. $|d|$, incremental transaction number:

The ratio of IDTM over DTM in various of $|D|$:

- $S_{th} = 0.001\times|D|$
- $S_{th} = 0.003\times|D|$
- $S_{th} = 0.005\times|D|$

Relative Time Ratio (IDTM/DTM) vs. $|D|$, updated transaction number:

The ratio of IDTM over DTM in various of $|d|$:

- $S_{th} = 0.001\times|D|$
- $S_{th} = 0.003\times|D|$
- $S_{th} = 0.005\times|D|$

Relative Time Ratio (IDTM/DTM) vs. $|d|$, incremental transaction number:

Figure 6.13: Scale-up experiments on algorithm $IDTM$
6.4.3 Scale-Up Experimental Results on IDTM

Next, we present the scale-up experiments of algorithm IDTM. The scale-up experimental results for various datasets are obtained. As shown in Figure 6.13, the experimental results are evaluated with the dataset \( N1 - L12 - F20 - Dm - d10 \) where the number of transactions increases from 100,000 to 800,000. The value of \( \beta \) is set to 0.4. With three distinct values of \( S_{th} \) taken into consideration, Figure 6.13a and Figure 6.13b show the scale-up performance of algorithm IDTM as \( |D| \) and \( |d| \) increase. Note that the execution times are normalized with respect to the times for the 100,000 transactions dataset in the Figure 6.13a for better readability. The second scaleup experiment with the dataset \( N1 - L12 - F20 - D1000 - dn \) shows the results of DTM when the number of transactions in the incremented dataset varies from 100 thousands to 800 thousands. The execution times are normalized with respect to the times for the 100,000 incremented transaction dataset in the Figure 6.13b. It can be seen from Figure 6.13b that the execution time increases linearly with the growth of the incremental size, showing good scalability of DTM.

To further understand the impact of \( |D| \) and \( |d| \) to the relative performance of algorithms DTM and IDTM, we conduct the scaleup experiments and show the results on Figure 6.13c and Figure 6.13d where the value in y-axis corresponds to the ratio of the execution time of IDTM to that of DTM. Hence, the smaller this ratio, the further algorithm IDTM outperforms DTM. Figure 6.13c shows the referenced ratio obtained from an updated database over datasets of \( N1 - L12 - F20 - Dm - d_{10} \). With the value \( \frac{|D|}{|d|} = 10 \), the ratio of the execution time of IDTM to that of DTM decreases when the amount of updated database \( |D| \) grows, meaning that the advantage of IDTM over DTM increases as the database size increases. Figure 6.13d shows the execution time ratio for different values of \( |d| \). It can be seen that the execution time ratio becomes larger with the growth of the incremental transaction number \( |d| \), meaning that the advantage of IDTM over DTM becomes even more prominent as the incremental portion is finer.

6.5 Summary

This chapter broadened the horizon of frequent path traversal pattern mining by introducing a flexible model of mining Web traversal patterns with dynamic thresholds. Specifically, we explored a new data mining capability which involves mining path traversal patterns with the concept of dynamic thresholds in a time-variant Web environment. By properly employing some effective techniques devised for joining reference sequences, the proposed algorithm DTM not only possessed the capability of mining with dynamic thresholds, but also significantly improved the execution efficiency as well as contributed to the incremental mining of Web traversal patterns. In addition, an innovative hash method with multiple
hash tables was designed as an efficient technique for the mining with dynamic support thresholds. Performance of algorithm $DTM$ and $UTM$ was comparatively analyzed. It has been shown that the option of algorithm $DTM$ is very advantageous and leads to prominent performance improvement. Scale-up experiments on $IDTM$ have also been conducted.
Chapter 7

Conclusions

The discovery of association relationship among a huge database has been known to be useful in selective marketing, decision analysis, and business management [22, 40]. A popular area of applications is the market basket analysis, which studies the buying behaviors of customers by searching for sets of items that are frequently purchased either together or in sequence.

Let $I = \{x_1, x_2, ..., x_m\}$ be a set of items. A set $X \subseteq I$ with $k = |X|$ is called a $k$-itemset or simply an itemset. Let a database $D$ be a set of transactions, where each transaction $T$ is a set of items such that $T \subseteq I$. A transaction $T$ is said to support $X$ if and only if $X \subseteq T$. Conventionally, an association rule is an implication of the form $X \implies Y$, meaning that the presence of the set $X$ implies the presence of another set $Y$ where $X \subset I$, $Y \subset I$ and $X \cap Y = \phi$. The rule $X \implies Y$ holds in the transaction set $D$ with confidence $c$ if $c\%$ of transactions in $D$ that contain $X$ also contain $Y$. The rule $X \implies Y$ has support $s$ in the transaction set $D$ if $s\%$ of transactions in $D$ contain $X \cup Y$.

For a given pair of confidence and support thresholds, the problem of mining association rules is to identify all association rules that have confidence and support greater than the corresponding minimum support threshold (denoted as min_supp) and minimum confidence threshold (denoted as min_conf). Association rule mining algorithms [5] work in two steps: (1) generate all frequent itemsets that satisfy min_supp; (2) generate all association rules that satisfy min_conf using the frequent itemsets. This problem can be reduced to the problem of finding all frequent itemsets for the same support threshold. Since the early work in [5], several efficient algorithms to mine association rules have been developed in recent years.

While these are important results toward enabling the integration of association mining and fast searching algorithms, e.g., BFS and DFS which are classified in [40], we note that these mining methods cannot effectively be applied to the mining of a large incremental temporal database which is of increasing popularity recently. Specifically, some phenomena are observed when we take the issues of Incremental Updates, Weighted Transactions, Publication-like Items, Short Transactions, and Dynamic Thresholds into consideration.
To remedy this, we explored in Chapter 2 an effective sliding-window filtering (abbreviatedly as SWF) algorithm for incremental mining of association rules. In essence, by partitioning a transaction database into several partitions, algorithm SWF employed a filtering threshold in each partition to deal with the candidate itemset generation. Under SWF, the cumulative information of mining previous partitions was selectively carried over toward the generation of candidate itemsets for the subsequent partitions. Algorithm SWF not only significantly reduced I/O and CPU cost by the concepts of cumulative filtering and scan reduction techniques but also effectively controlled memory utilization by the technique of sliding-window partition. More importantly, algorithm SWF was particularly powerful for efficient incremental mining for an ongoing time-variant transaction database. By utilizing proper scan reduction techniques, only one scan of the incremented dataset was needed by algorithm SWF. The I/O cost of SWF was, in orders of magnitude, smaller than those required by prior methods, thus resolving the performance bottleneck. Extensive experimental studies were performed to evaluate performance of algorithm SWF. Sensitivity analysis of various parameters was conducted to provide many insights into algorithm SWF. It is noted that the improvement achieved by algorithm SWF is even more prominent as the incremented portion of the dataset increases and also as the size of the database increases.

Furthermore, without fully considering the time-variant characteristics of items and transactions, it is noted that some discovered rules may be expired from users' interest. In other words, some discovered knowledge may be obsolete and of little use, especially when we perform the mining schemes on a transaction database of short life cycle products. This aspect is, however, rarely addressed in prior studies. In view of this, we broadened in Chapter 3 the horizon of frequent pattern mining by introducing a weighted model of transaction-weighted association rules in a time-variant database. Specifically, we proposed an efficient Progressive Weighted Miner (abbreviated as PWM) algorithm to perform the mining for this problem as well as conducted the corresponding performance studies. In algorithm PWM, the importance of each transaction period was first reflected by a proper weight assigned by the user. Then, PWM partitioned the time-variant database in light of weighted periods of transactions and performs weighted mining. Algorithm PWM was designed to progressively accumulate the itemset counts based on the intrinsic partitioning characteristics and employed a filtering threshold in each partition to early prune out those cumulatively infrequent 2-itemsets. With this design, algorithm PWM was able to efficiently produce weighted association rules for applications where different time periods were assigned with different weights and lead to results of more interest. The correctness of PWM was proved and some of its theoretical properties were derived. Extensive experimental studies were conducted to evaluate the performance of PWM. Explicitly, the execution time of PWM was, in orders of magnitude, smaller than those required by other competitive schemes which were directly extended from existing methods. Sensitivity analysis of various parameters was conducted to provide
many insights into algorithm *PWM*.

In addition, in Chapter 4, we explored a new problem of mining *general temporal association rules* in *publication databases*. In essence, a publication database was a set of transactions where each transaction $T$ was a set of items of which each item contains an individual exhibition period. The current model of association rule mining was not able to handle the publication database due to the following fundamental problems, i.e., (1) lack of consideration of the *exhibition period* of each individual item; (2) lack of an equitable support counting basis for each item. To remedy this, we proposed an innovative algorithm *Progressive-Partition-Miner* (abbreviated as *PPM*) to discover general temporal association rules in a publication database. The basic idea of *PPM* was to first partition the publication database in light of exhibition periods of items and then progressively accumulate the occurrence count of each candidate 2-itemset based on the intrinsic partitioning characteristics. Algorithm *PPM* was also designed to employ a filtering threshold in each partition to early prune out those cumulatively infrequent 2-itemsets. The feature that the number of candidate 2-itemsets generated by *PPM* was very close to the number of frequent 2-itemsets allows us to employ the scan reduction technique to effectively reduce the number of database scan. Explicitly, the execution time of *PPM* was, in orders of magnitude, smaller than those required by other competitive schemes which were directly extended from existing methods. The correctness of *PPM* was proved and some of its theoretical properties were derived. Sensitivity analysis of various parameters was conducted to provide many insights into algorithm *PPM*.

On the other hand, it is noted that the existing models of rule mining might not be able to discover user preferred frequent patterns efficiently due to the following two fundamental problems: (1) The puzzles for mining association rules on a short transaction database; (2) lack of long patterns for sequential pattern mining. To remedy this, Chapter 5 explored the mining of causality rules. The causality rule explored in this dissertation consists of a sequence of triggering events and a set of consequential events, and was designed with the capability of mining non-sequential, inter-transaction information across multiple categories. Hence, the causality rule mining provided a very general framework for rule derivation. Note, however, that to count the occurrence in the sequence database of each candidate $k$-event rule for causality rule mining, it was necessary to scan through the sequence database to do a sub-sequence matching. This procedure was very costly particularly in the presence of a huge number of candidate sets and a large event database, and in our opinion, cannot be dealt with by direct extensions from existing rule mining methods. In view of this, we devised three candidate-matching algorithms, denoted by $HM_C$, $HM_D$, and $HM_A$, to minimize the computing cost needed by the first phase of discovering all *one-triggering* causality rules. In accordance with the concept of *hierarchical sub-sequence matching*, these three algorithms presented good efficiency and salability in the mining of causality rules. The proposed adaptive algorithm, $HM_A$, was shown to outperform the other two algorithms, $HM_C$ and
$H_{MD}$, which were based on candidate-sets-based and data-sets-based hierarchical matchings, respectively. Scale-up experiments showed that all three algorithms scale linearly with the number of customer transactions, the number of transactions per customer, and also the number of items in a transaction.

Moreover, in Chapter 6, we broadened the horizon of frequent path traversal pattern mining by introducing a flexible model of mining Web traversal patterns with dynamic thresholds. Specifically, we explored a new data mining capability which involves mining path traversal patterns with the concept of dynamic thresholds in a time-variant Web environment. By properly employing some effective techniques devised for joining reference sequences, the proposed algorithm $DTM$ (standing for Dynamic Threshold Miner) not only possessed the capability of mining with dynamic thresholds, but also significantly improved the execution efficiency as well as contributes to the incremental mining of Web traversal patterns. In addition, an innovative hash method with multiple hash tables was designed as an efficient technique for the mining with dynamic support thresholds. Performance of algorithm $DTM$ and the extension of existing methods was comparatively analyzed. It was shown that the option of algorithm $DTM$ was very advantageous and lead to prominent performance improvement. Sensitivity analysis on various parameters was also conducted.

Great strides have been made in knowledge discovery and data mining during the 1990s. It is expected to continue to flourish into the new millennium. While there have been many reported “success” stories surrounding data mining (e.g., most direct marketing companies and many biotech companies use some form of data mining, and there are thousands of such applications worldwide), many data mining projects remain in the realm of research. Continued success in data mining depends on continued research.
Bibliography


[36] J. Han and J. Pei. Mining Frequent Patterns by Pattern-Growth: Methodology and Implications. *ACM SIGKDD Explorations* (Special Issue on Scalable Data Mining Algorithms), December 2000.


[38] J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. *Proc. of 2000 ACM-SIGMOD Int. Conf. on Management of Data*, pages 486–493, May 2000.

[39] J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. *Proc. of 2000 ACM-SIGMOD Int. Conf. on Management of Data*, May 2000.


137


[85] A. K. H. Tung, J. Han, L. V. S. Lakshmanan, and R. T. Ng. Constraint-Based Clustering in Large Databases. *Proc. of 2001 Int. Conf. on Database Theory*, January 2001.


138


