

# Mining Relationship between Triggering and Consequential Events in a Short Transaction Database

*Chang-Hung Lee\** , *Philip S. Yu<sup>†</sup>* , and *Ming-Syan Chen<sup>‡</sup>*

## 1 Introduction

Since the earlier work in [2], a broad variety of data mining capabilities has been developed. These studies cover a broad spectrum of topics including: (1) association rule mining [1, 3, 11, 17]; (2) incremental updating [7, 14]; (3) mining of generalized [20], multi-level [8], multi-dimensional rules [22, 23]; (4) constraint-based rule mining [9, 18]; (5) temporal association rule [6, 13, 21]; (6) frequent episodes discovery [15, 16]; and (7) sequential patterns mining [4, 19].

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 [5, 12], 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

---

\*Electrical Engineering Department National Taiwan University, Taipei, Taiwan, ROC, email: chlee@arbor.ee.ntu.edu.tw

<sup>†</sup>IBM T. J. Watson Research P.O.Box 704 Yorktown, NY 10598, email: psyu@us.ibm.com

<sup>‡</sup>Electrical Engineering Department National Taiwan University, Taipei, Taiwan, ROC, email: mschen@cc.ee.ntu.edu.tw

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 1.1:** Consider the database shown in Figure 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 1). Note that since fewer items will be bought at the same time, as shown in Figure 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 [4, 19]. 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 1.2:** Recall the illustrative example in Figure 1. The minimum support (denoted by *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 1). As shown in Figure 1, there is no more frequent 3-items sequential pattern in this example.

Consequently, this paper 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 paper the mining of causality rules from a transaction

Customer	Buying sequences			Item Information	
	CID	TID <sub>1</sub>	TID <sub>2</sub>	TID <sub>3</sub>	Item
S <sub>1</sub>	(B, C)	A	(D, E)	A	TV
S <sub>2</sub>	D	B		B	TV set
S <sub>3</sub>	(A, B)	(C, D)	E	C	Sofa
S <sub>4</sub>	(A, B, E)			D	End table
S <sub>5</sub>	E	D	A	E	Lamp

**Figure 1.** An illustrative example for the mining on a short transaction database

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 events and consequential events. In this paper, 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 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 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. Some interesting causality rules are shown below.

- (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 paper, 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 [4], FP-tree [10], PrefixSpan [19], episode mining algorithms [15, 16], 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.

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 [4, 19], 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 [15, 16] 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.

The rest of this paper is organized as follows. The problem description of mining causality rules is given in Section 2. Section 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 4. This paper concludes with Section 5.

## 2 Problem Description

Specifically, a rule “ $\{c_1\}$  triggers  $\{r_1, \dots, r_k\}$ ”, denoted by  $\{c_1\} \rightsquigarrow \{r_1, \dots, r_k\}$ , 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  $c_1, r_1, \dots$ , and  $r_k$ , 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, “ $\{c_1, \dots, c_m\}$  triggering  $\{r_1, \dots, r_k\}$ ”, denoted by  $\{c_1, \dots, c_m\} \rightsquigarrow \{r_1, \dots, r_k\}$ , may be regarded as a *multi-triggering causality rule*, if there is a sufficient number of sequences containing  $\{c_1, \dots, c_m\}$  followed by the  $r_j$ 's, ( $1 \leq j \leq k$ ) in any order, and furthermore that  $\{c_1, \dots, c_m\}$  and  $\{r_1, \dots, r_k\}$  satisfy some pre-specified constraints. We then have the following definition.

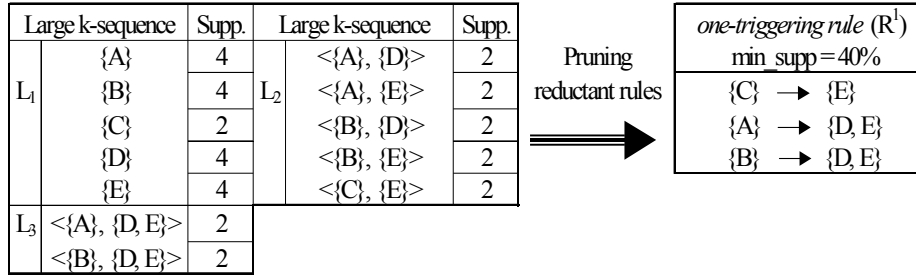
**Definition 1:** A causality rule  $X \rightsquigarrow Y$  is termed to be frequent if and only if its support is larger than the minimum support required, i.e.,  $supp(X \rightsquigarrow Y) > min\_supp$ .

It is noted that in accordance with Definition 1, one may find out both  $X \rightsquigarrow Y$  and  $Y \rightsquigarrow X$  causality rules in the same transaction database. In our opinion, such kinds of causality rules explore the “*Nature Puzzles*”, e.g., both “*Chicken*  $\rightsquigarrow$  *Egg*” and “*Egg*  $\rightsquigarrow$  *Chicken*” are termed as nature causality rules.

### 2.1 First Phase: Discovering One-Triggering Causality Rules

A sequence of  $k$  simple events,  $c_1, r_1, \dots, r_{k-1}$ , 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,  $\{c_1\} \rightsquigarrow \{r_1, \dots, r_{k-1}\}$ , represents a causality rule if the number of sequences containing  $c_1, r_1, \dots, r_{k-1}$ , 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  $\{c_1\} \rightsquigarrow \{r_1, \dots, r_{k-1}\}$ .

Let  $L_k$  represent the large  $k$ -event set. Here, without loss of generality, it is assumed that for each large  $k$ -event rule,  $\{c_1\} \rightsquigarrow \{r_1, \dots, r_{k-1}\}$ , in  $L_k$ , letters in



**Figure 2.** An example of generating large one-triggering event sequences with  $min\_supp = 40\%$

$\{r_1, \dots, r_{k-2}, r_{k-1}\}$  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.

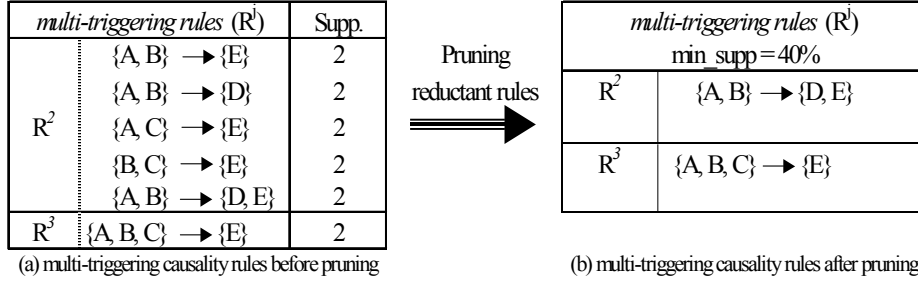
Starting with  $k = 2$ , we can then generate a candidate  $k$ -event set,  $C_k$  from  $L_{k-1}$ . This can be done by joining  $L_{k-1}$  with  $L_{k-1}$  to derive  $C_k$ . Specifically, for  $k = 2$ ,  $C_2$  is the cross-product of  $L_1$  by itself. For  $k > 2$ , any two large  $(k-1)$ -event 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.

## 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  $L_1^1$  is equal to  $L_1$ . Starting with  $j = 2$  and  $i = j + 1$ , we obtain  $L_i^j$  from  $L_{i-1}^{j-1}$ . Specifically, by joining any two rules in  $L_{i-1}^{j-1}$  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 2.1:** Consider two sequences,  $S_1$  and  $S_2$  in  $L_5^2$ . Assuming that  $S_1 = \{A, E\} \rightsquigarrow \{F, H, Y\}$  and  $S_2 = \{E, G\} \rightsquigarrow \{F, H, Y\}$ , the two rules can be joined to form a new candidate rule  $\{A, E, G\} \rightsquigarrow \{F, H, Y\}$  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  $L_i^j$ .

Consider again the database with the customer-sequences shown in Figure 1.



**Figure 3.**  $R^j$  sets of multi-triggering causality rules on mining the database in Figure 1 where  $j \geq 2$

The minimum support is assumed to be 40% (i.e., two customer sequences). 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,  $\langle \{A, B\}, \{D, E\} \rangle$  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 2 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 2. 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 2.

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  $C_i^j$  are counted. As a result, the total causality rules from the database in Figure 1 are shown in Figure 3. 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 3a, though having minimum supports, do not appear in the answer in Figure 3b because they are redundant.

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.

### 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 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 3.1, these three algorithms present good efficiency and scalability 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.

Basically, algorithm  $HM_C$  demonstrates the matching processing driven by candidate sets in Section 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 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 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 1.

$S_i$	the $i^{th}$ sequence in an event sequence database $\mathcal{D}$
$a_{ij}$	a potential triggering event in the event sequence $S_i$
$P_{ij}$	the sub-sequence of $S_i$ that contains all of the following events of $a_{ij}$
$\Phi_{ij}$	the sorted sub-sequence of $P_{ij}$ that contains all basic events in $P_{ij}$
$C_i^j$	candidate $i$ event rule with $j$ triggering events
$L_i^j$	large $i$ event rule with $j$ triggering events

Table 1: Meanings of symbols used

#### 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 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.

### 3.2 Algorithm $HM_C$ : 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  $HM_C$  together with the candidate-sets-based hierarchical matching methodology. The algorithm of  $HM_C$  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  $HM_C$  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  $HM_C$ :** Candidate-sets based hierarchical matching

1. begin;
2. obtain  $L_1$  by counting occurrences;
3. set  $k$  to 2;
4. set  $C_k$  to  $L_{k-1} * L_{k-1}$ ;
5. set  $i$  to 1;
6. if  $(i \leq N)$  {
7.     if  $(C_k \neq \phi)$  {
8.         denote the  $i^{th}$  sequence as  $S_i = \{a_{i1}, a_{i2}, \dots, a_{ig_i}\}$ ;
9.         set  $j$  to 1;
10.        if  $(j \leq g_i - k + 1)$  {
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. if  $(m \leq |D_y^C|)$  {
17. denote the  $m^{th}$  candidate rule in  $D_y^C$  as  $yw_1...w_{k-1}$ ;
18. if  $(w_1...w_{k-1}$  is a sub-sequence of  $\Phi_{ij}$ ) {
19. increment count of  $yw_1...w_{k-1}$  by 1;
20. } else {set  $m$  to  $m + 1$ };
21. } else {set  $j$  to  $j + 1$ };
22. } else {set  $i$  to  $i + 1$ };
23. } else {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  $S_i = \langle a_{i1}, a_{i2}, \dots, a_{ig_i} \rangle$ . 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  $w_1w_2\dots w_{k-1}$  is a sub-sequence in  $P_{ij}$ , the occurrence count of  $yw_1w_2\dots w_{k-1}$  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  $L_2, \dots, L_{k-1}$ , 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 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 1<sup>st</sup> 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.

### 3.3 Algorithm $HM_D$ : Data-Sets-Based Hierarchical Matching

On the other hand, the procedure  $HM_D$  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  $HM_C$ , algorithm  $HM_D$  will have  $w_1\dots w_{k-1}$  be generated from  $\Phi_{ij}$  and then determine if  $yw_1\dots w_{k-1}$  is in  $C_k$ .*

**Example 3.3:** Recall the sequence  $S_1 = \langle (A, D), B, D, C \rangle$  in Example 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  $HM_C$ . Part of algorithm of  $HM_D$  is outlined below.

**Code Segment of Algorithm  $HM_D$ :** Data-sets-based hierarchical matching

- 14a. if ( $a_{ij}$  is a triggering event in  $C_k$ )
- 15a.     generate  $D_y^D$  from  $\Phi_{ij}$  and set  $m$  to 1;
- 16a.     if ( $m \leq |D_y^D|$ )
- 17a.         denote the  $m^{th}$  candidate rule in  $D_y^D$  as  $yw_1...w_{k-1}$ ;
- 18a.         if ( $yw_1...w_{k-1}$  is a member of  $C_k$ )
- 19a.             increment count of  $yw_1...w_{k-1}$  by 1;
- 20a.         else {set  $m$  to  $m + 1$ };

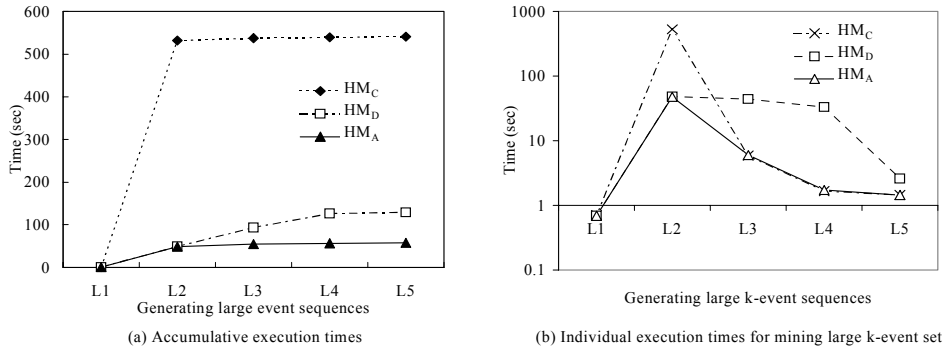
Consequently, the complexity of algorithm  $HM_D$  is approximately  $O(\frac{n!}{(n-k)!})$  for generating a large  $k$ -event set. Recall the corresponding complexity, i.e.,  $O(|C_k| \times n \times k)$ , of algorithm  $HM_C$  as mentioned in Section 3.2. Note that the efficiency of algorithms  $HM_C$  and  $HM_D$  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  $HM_D$  will perform better in the case of low order candidate  $k$ -event sets, where  $|C_k|$  is very huge. On the other hand, algorithm  $HM_C$  is powerful for generating higher order of large  $k$ -event sets, where  $|C_k|$  is very small. By combining the advantages of  $HM_C$  and  $HM_D$ , we next present the adaptive algorithm  $HM_A$  in Section 3.4 for the overall performance improvement.

### 3.4 Algorithm $HM_A$ : Adaptive Hierarchical Matching

Since fewer candidate rules usually incur shorter execution time needed, with combining two algorithms  $HM_C$  and  $HM_D$ , algorithm  $HM_A$  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.  $(|C_k| \times k) > \frac{(n-1)!}{(n-k)!}$ , the adaptive algorithm  $HM_A$  is similar to algorithm  $HM_D$  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  $HM_C$ -like procedure by setting  $D_y^A = D_y^C$  in the matching phase will greatly improve the program execution.

## 4 Experimental Studies

Explicitly, we generated several different transaction databases from a set of potentially frequent itemsets to evaluate the performance of the three proposed algorithms, i.e.,  $HM_C$ ,  $HM_D$  and  $HM_A$ . 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 4.1. The relative performance of algorithms is presented in Section 4.2. We conduct the scale-up performance evaluation in Section 4.3.



**Figure 4.** The execution times of three hierarchical matching algorithms on  $C10 - T2.5 - S4 - I1.25$  with  $min\_supp = 0.5\%$

## 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 [4, 19]. 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.

## 4.2 Relative Performance

Figure 4 shows the relative execution times for the three algorithms with the database  $C10 - T2.5 - S4 - I1.25$  as the minimum support is 0.5%. As shown in Figure 4b, the experimental results show that  $HM_D$  outperforms  $HM_C$  when generating the large 2-event set and  $HM_C$  has better performance in generating  $L_k$  as  $k > 2$ . As a result of combining the advantages of both  $HM_C$  and  $HM_D$ , the adaptive algorithm  $HM_A$  outperforms algorithms  $HM_C$  and  $HM_D$  for generating any order of large event sets. As shown in Figure 4a, algorithm  $HM_A$  significantly reduces the total execution time for mining of  $L_k$ .

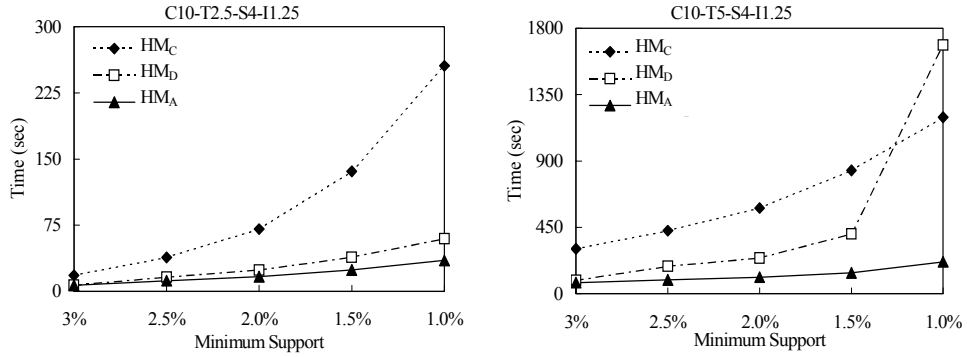


Figure 5. Relative performance of execution time

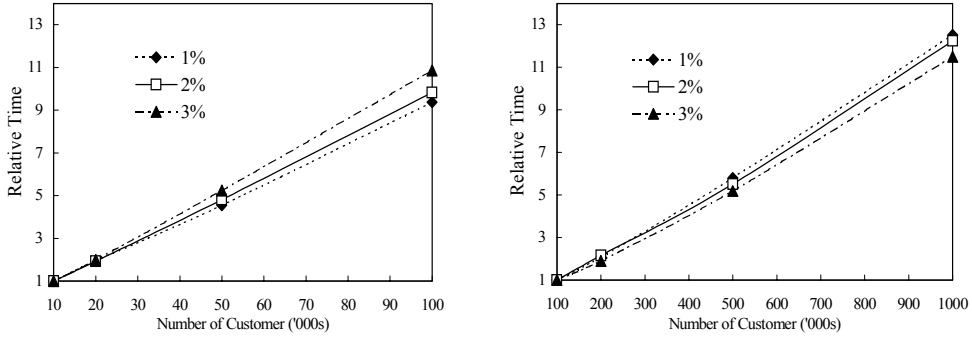


Figure 6. Scale-up experiments for the number of customers

In addition, Figure 5 shows the relative execution times for the three algorithms as the minimum support is decreased from 3% to 1%. 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,  $HM_D$  performs worse than the other two algorithms because it generates and counts a much larger number of potential candidate,  $D_y^D$ , from datasets to match the candidates in candidate-sets. On the other hand,  $HM_C$  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,  $HM_C$  does not perform as well as the other two algorithms. With aggregating the advantages of  $HM_D$  and  $HM_C$ ,  $HM_A$  outperforms in any value of minimum support. These experimental results conform with the complexity analysis of the proposed algorithms.

### 4.3 Scale-up

We next present in this subsection the results of scale-up experiments for the algorithm  $HM_A$ . 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 6 shows how algorithm  $HM_A$  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 6 the execution times of algorithm  $HM_A$  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.

## 5 Conclusions

As explored in this paper, 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,  $HM_A$ , was shown to outperform the other two algorithms,  $HM_C$  and  $HM_D$ , which are based on candidate-sets-based and datasets-based hierarchical matchings, respectively. Scale-up experiments showed that all three algorithms scale linearly with the number of customer transactions.

# Bibliography

- [1] R. Agarwal, C. Aggarwal, and V.V.V. Prasad. A Tree Projection Algorithm for Generation of Frequent Itemsets. *Journal of Parallel and Distributed Computing (Special Issue on High Performance Data Mining)*, 2000.
- [2] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. *Proc. of ACM SIGMOD*, pages 207–216, May 1993.
- [3] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. *Proc. of the 20th International Conference on Very Large Data Bases*, pages 478–499, September 1994.
- [4] R. Agrawal and R. Srikant. Mining Sequential Patterns. *Proc. of the 11th International Conference on Data Engineering*, pages 3–14, March 1995.
- [5] M.-S. Chen, J. Han, and P. S.Yu. Data Mining: An Overview from Database Perspective. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):866–883, December 1996.
- [6] X. Chen and I. Petr. Discovering Temporal Association Rules: Algorithms, Language and System. *Proc. of 2000 Int. Conf. on Data Engineering*, 2000.
- [7] D. Cheung, J. Han, V. Ng, and C.Y. Wong. Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique. *Proc. of 1996 Int'l Conf. on Data Engineering*, pages 106–114, February 1996.
- [8] J. Han and Y. Fu. Discovery of Multiple-Level Association Rules from Large Databases. *Proc. of the 21th International Conference on Very Large Data Bases*, pages 420–431, September 1995.
- [9] J. Han, L. V. S. Lakshmanan, and R. T. Ng. Constraint-Based, Multidimensional Data Mining. *COMPUTER (special issues on Data Mining)*, pages 46–50, 1999.
- [10] 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.

- [11] 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.
- [12] J. Hipp, U. Güntzer, and G. Nakhaeizadeh. Algorithms for association rule mining – a general survey and comparison. *SIGKDD Explorations*, 2(1):58–64, July 2000.
- [13] C.-H. Lee, C.-R. Lin, and M.-S. Chen. On Mining General Temporal Association Rules in a Publication Database. *Proc. of 2001 IEEE International Conference on Data Mining*, November 2001.
- [14] C.-H. Lee, C.-R. Lin, and M.-S. Chen. Sliding-Window Filtering: An Efficient Algorithm for Incremental Mining. *Proc. of the Tenth ACM Intern'l Conf. on Information and Knowledge Management*, November 2001.
- [15] H. Mannila and D. Rusakov. Decomposition of event sequences into independent components. *Proc. of the First SIAM Conference on Data Mining*, 2001.
- [16] H. Mannila and J. Sepponen. Finding similar situations in sequences of events via random projections. *Proc. of the First SIAM Conference on Data Mining*, 2001.
- [17] J.-S. Park, M.-S. Chen, and P. S. Yu. Using a Hash-Based Method with Transaction Trimming for Mining Association Rules. *IEEE Transactions on Knowledge and Data Engineering*, 9(5):813–825, October 1997.
- [18] J. Pei and J. Han. Can We Push More Constraints into Frequent Pattern Mining? *Proc. of 2000 Int. Conf. on Knowledge Discovery and Data Mining*, August 2000.
- [19] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. *Proc. of 2001 Int. Conf. on Data Engineering*, 2001.
- [20] R. Srikant and R. Agrawal. Mining Generalized Association Rules. *Proc. of the 21th International Conference on Very Large Data Bases*, pages 407–419, September 1995.
- [21] R.R. Muntz W. Wang, J. Yang. TAR: Temporal Association Rules on Evolving Numerical Attributes. *Proc. of 2000 Int. Conf. on Data Engineering*, 2001.
- [22] K. Wang, S.Q. Zhou, and S.C. Liew. Building Hierarchical Classifiers Using Class Proximity. *Proc. of 1999 Int. Conf. on Very Large Data Bases*, pages 363–374, 1999.
- [23] C. Yang, U. Fayyad, and P. Bradley. Efficient discovery of error-tolerant frequent itemsets in high dimensions. *Proc. of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2001.