, , 1–26 ()

An Efficient Approach to Mining Indirect Associations

QIAN WAN qwan@cs.yorku.ca
AIJUN AN aan@cs.yorku.ca
Department of Computer Science, York University, Toronto, ON, Canada M3J 1P3

Editor:

Abstract. Discovering association rules is one of the important tasks in data mining. While most of the existing algorithms are developed for efficient mining of frequent patterns, it has been noted recently that some of the infrequent patterns, such as indirect associations, provide useful insight into the data. In this paper, we propose an efficient algorithm, called *HI-mine*, based on a new data structure, called *HI-struct*, for mining the complete set of indirect associations between items. Our experimental results show that *HI-mine*'s performance is significantly better than that of the previously developed algorithm for mining indirect associations on both synthetic and real world data sets over practical ranges of support specifications.

Keywords: Data mining, association rules, indirect association, algorithm.

1. Introduction

Since it was first introduced by Agrawal et al. [3] in 1993, association rule mining has been studied extensively by many researchers. As a result, many algorithms have been proposed to improve the running time for generating association rules and frequent itemsets. The latest includes *FP-growth* [10, 11], which utilizes a prefix-tree structure for compactly representing and processing pattern information, and *H-mine* [16], which takes advantage of a novel hyper-linked data structure and dynamically adjusts links in the mining process.

While most of the existing algorithms are developed for efficient mining of frequent patterns, it has been noted recently that some of the infrequent patterns may provide useful insight into the data. In [22], a new class of patterns called indirect associations has been proposed and its utilities have been examined in various application domains.

Consider a pair of items, x and y, that are rarely present together in the same transaction. If both items are highly dependent on the presence of another itemsets M, then the pair (x, y) is said to be indirectly associated via M. Figure 1 illustrates a high-level view of an indirect association.

There are many advantages to mining indirect associations in large data sets. For example, an indirect association between a pair of words in text documents can be used to classify query results into categories [22]. For instance, the words coal and data can be indirectly associated via mining. If only the word mining is used in a query, documents in both mining domains are returned. Discovery of the indirect association between coal and data enables us to classify the retrieved documents

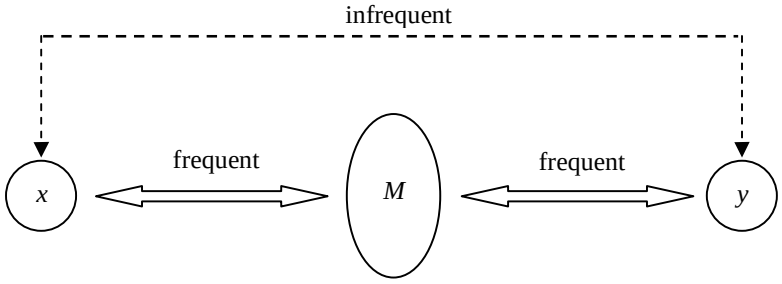

Figure 1. Indirect association between x and y via the mediator M

into coal mining and data mining. There are also potential applications of indirect associations in many other real-world domains, such as competitive product analysis and stock market analysis [22].

An algorithm for mining indirect associations between itempairs, called INDIRECT algorithm, was presented in [22], and will be fully described in the next section of this paper. There are two phases in the algorithm:

1. Extract all frequent itemsets using standard frequent itemset mining algorithms such as *Apriori* [4] or *FP-growth* [10];

2. Discover valid indirect associations by checking all the candidate associations generated from the frequent itemsets.

In this paper, we propose a new data structure, *HI-struct*, and develop a new mining algorithm, *HI-mine*, for finding indirect associations in large databases. We show that they can be used as a formal framework for discovering indirect associations directly, with no need to generate all frequent itemsets as the first step. Empirical evaluations comparing *HI-mine* to two versions of the algorithm described above show that *HI-mine* performs significantly better on both synthetic and real world data sets.

The remaining of the paper is organized as follows. Section 2 reviews related work and briefly exhibits the contribution of the paper. Next, we present the *HI-struct* data structure and the *HI-mine* algorithm in Section 3. Our empirical results are reported in Section 4. Finally, we conclude with a summary of our work and suggestions for future research in Section 5.

2. Related Work

Let $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ be a set of m *items*. A subset $X \subseteq \mathcal{I}$ is called an *itemset*. A k-itemset is an itemset that contains k items.

Table 1. Summary of notations and their meaning

$\mathcal{D}$	a database of transactions
TDB	an example transaction database
$sup(\mathcal{I})$	the support of itemset $\mathcal{I}$
$dep(X, Y)$	dependence between itemsets X and Y
$\langle x, y \mid M \rangle$	an indirect association between x and y via M
t_s	itempair support threshold
t_f	mediator support threshold
t_s	mediator dependence threshold
$IIS(\mathcal{D})$	indirect itempair set of $\mathcal{D}$
$MSS(i)$	mediator support set of item i

Let $\mathcal{D} = \{T_1, T_2, \ldots, T_n\}$ be a set of n transactions, called a *transaction database*, where each transaction T_j ($j \in \{1, 2, \ldots, n\}$) is a set of items such that $T_j \subseteq \mathcal{I}$. Each transaction is associated with a unique identifier, called its *TID*. A transaction T contains an itemset X if and only if $X \subseteq T$.

The *support* of an itemset X is the percentage of transactions in $\mathcal{D}$ containing X: $sup(X) = \|\{t \mid t \in \mathcal{D}, X \subseteq t\}\| \; / \; \|\{t \mid t \in \mathcal{D}\}\|$, where $\|S\|$ is the cardinality of set S.

An itemset X in a transaction database $\mathcal{D}$ is called as a *frequent itemset* if its *support* is equal to, or greater than a user-specified minimum support threshold, *min_sup*. Accordingly, an *infrequent itemset* is an itemset that does not satisfy the user-specified minimum support threshold.

Table 1 summarizes the notations that will be used throughout this paper and their meaning.

2.1. Negative association rules

An association rule is an implication of the form $X \Rightarrow Y$, where $X \subset \mathcal{I}$, $Y \subset \mathcal{I}$, and $X \cap Y = \emptyset$. Here, X is called the *antecedent* and Y is called the *consequent* of the rule. The confidence of an association rule r: $X \Rightarrow Y$ is the conditional probability that a transaction contains Y, given that it contains X. The support of rule r is defined as: $sup(r) = sup(X \cup Y)$.

The importance of extending the current association rule framework to include negative association was first pointed out by Brin *et al.* in [6]. Since then, many techniques for mining negative associations have been developed [18, 22, 26].

In the case of *negative associations* we are interested in finding itemsets that have a very low probability of occurring together. That is, a negative association between two itemsets X and Y, denoted as $X \Rightarrow \overline{Y}$ or $Y \Rightarrow \overline{X}$, means that X and Y appear very rarely in the same transaction.

Mining negative association rules is computational intractable with a naive approach because billions of negative associations may be found in a large database while almost all of them are extremely uninteresting. This problem was addressed in [18] by combining previously discovered positive associations with domain knowl-

edge to constrain the search space such that fewer but more interesting negative rules are mined.

A general framework for mining both positive and negative association rules of interest was presented in [26], in which no domain knowledge was requires, and the negative association rules were given in more concrete expressions to indicate actual relationships between different itemsets. However, although the sets of the positive and negative itemsets of interest in the database were minimized in this framework, the search space for negative itemsets of interest was still huge. Another problem was that it tended to produce too many negative association rules, thus the practical application of this framework remained uncertain.

2.2. Indirect association and INDIRECT algorithm

Indirect association is closely related to negative association, they are both dealing with itemsets that do not have sufficiently high support. Indirect associations provide an effective way to detect interesting negative associations by discovering only "infrequent itempairs that are highly expected to be frequent" without using negative items or domain knowledge.

Definition 1 (Indirect Association). An itempair $\{x, y\}$ is indirectly associated via a mediator M, if the following conditions hold:

1. $sup(\{x, y\}) < t_s$ (*Itempair Support Condition*)
2. There exists a non-empty set M such that:
   (a) $sup(\{x\} \cup M) \geq t_f$, $sup(\{y\} \cup M) \geq t_f$; (*Mediator Support Condition*)
   (b) $dep(\{x\}, M) \geq t_d$, $dep(\{y\}, M) \geq t_d$, where $dep(P, Q)$ is a measure of the dependence between itemsets P and Q. (*Mediator Dependence Condition*)

The thresholds above are called *itempair support threshold* (t_s), *mediator support threshold* (t_f), and *mediator dependence threshold* (t_d), respectively. In practice, it is reasonably to set $t_f \geq t_s$.

In the database and probability theories, an indirect association is a well-known property of embedded multi-valued dependency (EMVD) and probability conditional independence, where it is sometimes called an "induced dependence". [25] includes a comprehensive discussion on an independence in a small context becoming a dependence in a larger context in both database and probability settings.

In this paper, the notation $\langle x, y \mid M \rangle$ is used to represent the indirect association between x and y via M. And the *IS* measure [19] is used as the dependence measure for Condition 2(b). Given a pair of itemsets, say X and Y, its *IS* measure can be computed using the following equation:

$$IS(X, Y) = \frac{P(X, Y)}{\sqrt{P(X)P(Y)}} \tag{1}$$

1. Extract frequent itemsets, L_1, L_2,…L_n, using frequent itemsets generation algorithm, where L_i is the set of all frequent i-itemsets.
2. $P = \emptyset$ (set of indirect associations)
3. for k = 2 to n do
4. C_{k+1} = join(L_k, L_k)
5. for each $< x, y , M> \in C_{k+1}$ do
6. if ($sup(\{x, y\}) < t_s$ and $dep(\{x\}, M) \geq t_d$ and $dep(\{y\}, M) \geq t_d$)
7. $P = P \cup \{< x, y , M>\}$
8. end
9. end
10. end

Figure 2. The INDIRECT algorithm

where P denotes the probability that the given itemset appears in a transaction.

An algorithm for mining indirect associations between pairs of items is given in [22] and [20], which is shown in Figure 2. There are two major phases in this algorithm:

1. extract all frequent itemsets using *Apriori*; (step 1)
2. discover all indirect associations by
   (a) candidate generation (step 4);
   (b) candidate pruning (steps 5 - 9).

In the candidate generation step, frequent itemset L_k is used to generate candidate indirect associations for pass k+1, i.e., C_{k+1}. Each candidate in C_{k+1} is a triplet, $\langle x,\ y,\ M\rangle$, where x and y are the items that are indirectly associated via the mediator M. C_{k+1} is generated by joining the frequent itemsets in L_k. A pair of frequent k-itemsets, $\{x_1,\ x_2,\ldots,\ x_k\}$ and $\{y_1,\ y_2,\ldots,\ y_k\}$ are joined together if the two itemsets have exactly k-1 items in common; and thus produce a candidate indirect association $\langle x,\ y,\ M\rangle$, where x and y are the distinct items, one from each k-itemset, and M is the set of common items. For example, two frequent itemsets, $\{a,\ b,\ c,\ d\}$ and $\{a,\ b,\ d,\ e\}$, can be joined together to produce a candidate indirect association, $\langle c,\ e,\ \{a,\ b,\ d\}\rangle$. Since the candidate associations are created by joining two frequent itemsets, they all satisfy the mediator support condition. Therefore, in the steps for candidate pruning, only itempair support condition and mediator dependence condition are checked.

There are two join steps in the INDIRECT algorithm. One is in the first phase for generating all the frequent itemsets with *Apriori*. In *Apriori*, the join operation is used to generate candidate frequent itemsets for pass k+1 based on the frequent

itemsets in L_k. The other join operation is for generating candidate indirect associations, C_{k+1}, from L_k. Both candidate generation steps can be quite expensive, because each of them requires at most $O(\sum_k |L_k| \times |L_k|)$ join operations.

The join operation for generating indirect association candidates is more expensive than that in *Apriori* because the items in an indirect itempair, x and y, do not have to be the last item in each frequent itemset, whereas *Apriori* only combines itemsets that have identical k-1 prefix items, assuming that all the items in an itemset are sorted in lexicographic order.

Moreover, no matter what implementation technique is applied, an *Apriori*-like algorithm may still suffer from nontrivial costs in situations with prolific frequent patterns, long patterns, or quite low minimum support thresholds.

Is there any other way that we may reduce these costs in indirect association mining? Can we avoid generating all the frequent itemsets and a huge set of candidates, and derive indirect association directly using some novel data structure or algorithm?

2.3. Our solution

Our answers to the above questions are based on the following two strategies.

1. Instead of joining two frequent itemsets $\{a, b, c, d\}$ and $\{a, b, d, e\}$ to produce a candidate indirect association $\langle c, e, \{a, b, d\}\rangle$, we first generate the mediator support sets (defined in the next section) of item c and item e. Thus, if itemset $\{a, b, d\}$ can be found in these two sets, then $\langle c, e, \{a, b, d\}\rangle$ must be a valid indirect association.

2. In order to avoid costly candidate generation, we use a divide-and-conquer strategy to build the indirect itempair set (defined in the next section) and mediator support sets by partitioning each set into disjoined subsets and generating each subset in turn.

In the next section, we introduce our solution. The solution is based on the *HI-struct* data structure and the *HI-mine* algorithm, which were inspired by a novel hyper-linked data structure, *H-struct*, and an efficient algorithm, *H-mine*, presented in [16]. *H-struct* and *H-mine* are designed for the purpose of mining frequent patterns. We modify both of them for learning indirect association. With *HI-struct* and *HI-mine*, we do not need to generate all the frequent itemsets before mining indirect associations nor we need to do any join operation for candidate generation. Instead, we generate two new sets: *indirect itempair set* and *mediator support set* by recursively building the *HI-struct* data structures for the database. Then indirect associations are discovered from these two sets directly and efficiently.

3. Mining Indirect Associations Using *HI-mine*

In this section, we first define *Indirect Itempair Set* and *Mediator Support Set*. We then present the *HI-mine* (Hyper-structure Indirect-association Mining) algorithm

and illustrate the mining process of the algorithm using the two sets with an example.

3.1. Indirect Itempair Set and Mediator Support Set

Given a transaction database $\mathcal{D}$ and three thresholds: *itempair support threshold* (t_s), *mediator support threshold* (t_f), and *mediator dependence threshold* (t_d). The *Indirect Itempair Set* and *Mediator Support Set* can be defined as follows.

Definition 2 (Indirect Itempair Set). Let L_1 be the set of 1-itemset of $\mathcal{D}$. We define the indirect itempair set (*IIS*) of $\mathcal{D}$ as:

$$IIS(\mathcal{D}) = \{ < x, y > \big| \{x\} \in L_1 \wedge \{y\} \in L_1 \wedge sup(\{x\}) \geq t_f \wedge sup(\{y\}) \geq t_f \wedge sup(\{x, y\}) < t_s \}$$

Definition 3 (Mediator Support Set). Let L be the set of itemsets of $\mathcal{D}$, and L_1 be the set of 1-itemset of $\mathcal{D}$. The mediator support set (*MSS*) of x ($\{\mathrm{x}\} \in L_1 \wedge sup(\{\mathrm{x}\}) \geq t_f$) is defined as:

$$MSS(x) = \{M \big| M \in L \wedge sup(M \cup \{x\}) \geq t_f \wedge dep(M, \{x\}) \geq t_d\}$$

Based on the definition of indirect association (Definition 1) and the above two definitions (Definition 2 and 3), it's trivial to prove the following two Lemmas:

LEMMA 1 *If $\langle x, y \mid M \rangle$ is an indirect association of $\mathcal{D}$, then: (1) $\langle x, y \rangle \in IIS(\mathcal{D})$; and (2) $M \in MSS(x)$ and $M \in MSS(y)$.*

Proof:
If $\langle x, y \mid M \rangle$ is an indirect association of $\mathcal{D}$, then from Definition 1 we can get that:
(1) $sup(\{x\}) \geq t_f$, $sup(\{y\}) \geq t_f$, and $sup(\{x, y\}) < t_s$. By Definition 2, $\langle x, y \rangle \in IIS(\mathcal{D})$.
(2) $sup(\{x\} \cup M) \geq t_f$, $dep(\{x\} \cup M) \geq t_d$, and $sup(\{y\} \cup M) \geq t_f$, $dep(\{y\} \cup M) \geq t_d$. By definition 3, $M \in MSS(x)$ and $M \in MSS(y)$. ■

LEMMA 2 *Let $\{x\}$, $\{y\}$ and M be three itemsets of $\mathcal{D}$. If: (1) $\langle x, y \rangle \in IIS(\mathcal{D})$; and (2) $M \in MSS(x)$ and $M \in MSS(y)$, then $\langle x, y \mid M \rangle$ must be an indirect association of $\mathcal{D}$.*

Proof:
(1) If $\langle x, y \rangle \in IIS(\mathcal{D})$, then from Definition 2, we can get that $sup(\{x\}) \geq t_f$, $sup(\{y\}) \geq t_f$, and $sup(\{x, y\}) < t_s$.
(2) If $M \in MSS(x)$ and $M \in MSS(y)$, then from Definition 3, we can get that $sup(M \cup \{x\}) \geq t_f$, $dep(M \cup \{x\}) \geq t_d$, and $sup(M \cup \{y\}) \geq t_f$, $dep(M \cup \{y\}) \geq t_d$.

Table 2. The transaction database *TDB*

TID	List of itemIDs
T001	A, B, C, D
T002	A, B, E, F
T003	G, H
T004	B, C
T005	A, B, D, E, I
T006	B, C, D
T007	J, K
T008	L, M, N

Therefore, by Definition 1, $\langle x,\ y \mid M \rangle$ must be an indirect association of $\mathcal{D}$. ■

THEOREM 1 *Given IIS($\mathcal{D}$) and all the MSS(x)s where x is a frequent item in $\mathcal{D}$ with respect to t_f, the complete set of indirect associations of $\mathcal{D}$ can be derived by intersecting MSS(x) and MSS(y) for each $\langle x,\ y \rangle$ in IIS($\mathcal{D}$) to generate the set of mediators for itempair $\langle x,\ y \rangle$.*

Proof:

If $IIS(\mathcal{D}) = \emptyset$ or for each $\{x\} \in L_1$, $MSS(x) = \emptyset$, then the complete set of indirect associations of $\mathcal{D}$ is empty.

Suppose $IIS(\mathcal{D}) \neq \emptyset$, and $\langle x,\ y \rangle \in IIS(\mathcal{D})$. Let MS = $MSS(x) \cap MSS(y) \neq \emptyset$, and $M \in$ MS. By Lemma 2, $\langle x,\ y \mid M \rangle$ is an indirect association of $\mathcal{D}$.

We now show that any indirect association $\langle x,\ y \mid M \rangle$ of $\mathcal{D}$ can be obtained by intersecting $MSS(x)$ and $MSS(y)$, where $\langle x,\ y \rangle \in IIS(\mathcal{D})$. Suppose an indirect association $\langle x^{'},\ y^{'} \mid M^{'} \rangle$ of $\mathcal{D}$ cannot be generated in this way. Then, there are two possibilities:

(1) $\langle x^{'},\ y^{'} \rangle$ cannot be found in $IIS(\mathcal{D})$. That is, $\langle x^{'},\ y^{'} \rangle \notin IIS(\mathcal{D})$. From Lemma 1, we have $\langle x^{'},\ y^{'} \rangle \in IIS(\mathcal{D})$. This is a contradiction.

(2) $M^{'}$ cannot be generated by intersecting $MSS(x^{'})$ and $MSS(y^{'})$. That is $M^{'} \notin MSS(x^{'}) \cap MSS(y^{'})$. From Lemma 1, we have $M^{'} \in MSS(x^{'})$ and $M^{'} \in MSS(y^{'})$. This is a contradiction.

Therefore, all indirect associations of $\mathcal{D}$ can be derived from $IIS(\mathcal{D})$ and all the $MSS(x)$s, where $\{x\} \in L_1$. ■

3.2. HI-struct: Design and Construction

The design and construction of *HI-struct* for efficient indirect association mining are illustrated in the following example. The example transaction database *TDB* is shown in Table 2. The *HI-struct* of *TDB* is a dynamic data structure that changes during the process of recursively generating $IIS(\mathcal{D})$ and *MSS*s.

The initial *HI-struct* is constructed in the following steps.

1. Scan the transaction database *TDB* once. Collect the set of frequent items F (with respect to t_f) and their supports. Sort F in support descending order as

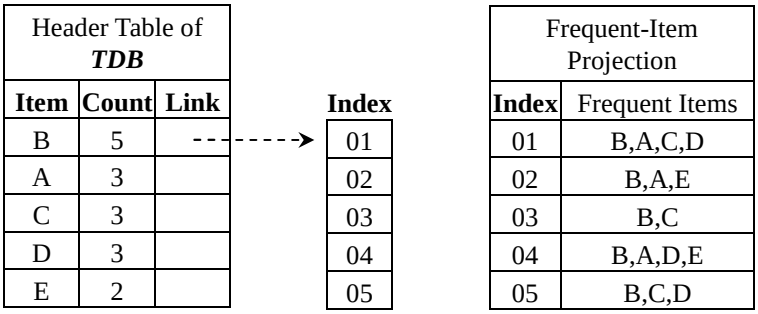

Figure 3. The initial *HI-struct* of *TDB*

L, the list of sorted frequent items. For the example database, L is {B, A, C, D, E}. Then a header table H is created, where each frequent item has an entry with three fields: an item-id, a support count, and a pointer to a queue.

2. For each transaction $\mathcal{T}$ in *TDB*, select and sort the frequent items in $\mathcal{T}$ according to the order of L. Let the sorted frequent item list in $\mathcal{T}$ be $[t|T]$, where t is the first element and T is the remaining list. $[t|T]$ is called the frequent-item projection of $\mathcal{T}$. Add $[t|T]$ to a frequent-item projection array, and append $[t|T]$'s index of the array to t's queue. Thus, all indexes of the frequent-item projections with the same first item (in the order of L) are linked together as a queue, and the entries in the header table H act as the heads of the queues.

The initial *HI-struct* of the example database is shown in Figure 3. Since all frequent item projections in our example database start with B, the queues for other items than B are empty at the moment[1]. After the initial *HI-struct* is constructed, the remaining mining process is performed on the *HI-struct* only, without referencing any information in the original database.

The subsequent mining process involves building $IIS(TDB)$ and MSS of each frequent item. We use a divide-and-conquer strategy to build these sets by partitioning each set into disjoined subsets and generating each subset in turn (see Figure 4). Following the support descending order of frequent items: B, A, C, D, E, the complete $IIS(TDB)$ and MSSs of all the frequent items in our example database can be partitioned into 5 subsets as follows:

(1) those containing item B;

(2) those containing item A but no item B;

(3) those containing item C, but no item B nor A;

(4) those containing item D, but no item B nor A nor C;

(5) those containing only item E.

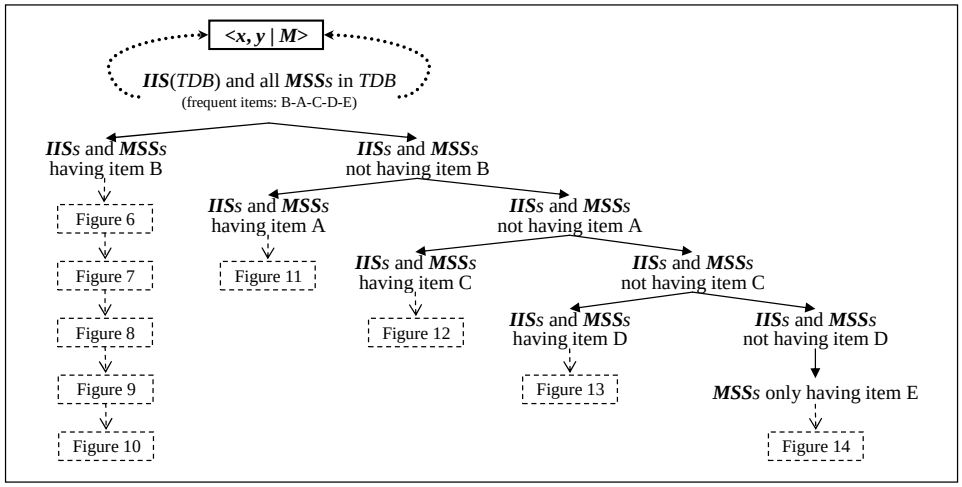

Figure 4. Divide-and-conquer strategy to build $IIS(TDB)$ and all MSSs

Header Table of *TDB*		
Item	**Count**	**Link**
B	5	
A	3	
C	3	
D	3	
E	2	

Index
01
02
04
03
05

Frequent-Item Projection	
Index	Frequent Items
01	B,A,C,D
02	B,A,E
03	B,C
04	B,A,D,E
05	B,C,D

Figure 5. HI-struct of TDB after mining B-projected database

Clearly, all the frequent-item projections containing item B, referred to as the B-projected database, are already linked in the B-queue in the initial header table, which can be traversed efficiently.

In the next section, we will show that, by mining the B-projected database recursively, *HI-mine* can find $IIS(TDB)$ and MSSs that contain item B. And then, each index in B-queue is moved into the queue for the next item in header table H following item B in the order of L to mine $IIS(TDB)$ and MSSs containing item A but not B.

The *HI-struct* after this adjustment is shown in Figure 5. For instance, the first item after item B of index 01 is item A, so index 01 is moved into A-queue, while index 03 is moved into C-queue since item C is the first item after item B of index 03. After the subsets containing item A but not B are mined, other subsets of $IIS(TDB)$ and MSSs are mined similarly.

3.3. HI-mine Algorithm

There are two phases in the algorithm. In the first phase, we construct *HI-struct* and generate itempair support set of the database and mediator support set of each frequent item. In the second phase, we generate the complete set of indirect associations based on the itempair support set and the mediator support sets. The algorithm is described as follows.

Algorithm: ***HI-mine***. (Mine indirect associations using an *HI-struct*)

Input:

A transaction database ($\mathcal{D}$), itempair support threshold (t_s), mediator support threshold (t_f) and mediator dependence threshold (t_d).

Output:

The complete set of indirect associations between itempairs.

Method:

1: build the initial *HI-struct* for $\mathcal{D}$ which includes a header table H and the frequent-item projection array.
2: **for** each i such that item i is in the header table of *HI-struct* **do**
3: create header table H_i by scanning i-projected database in the same way as building header table H except that item i is not considered (see Figures 6, 11 - 14)
4: call ***hi_mine***(H_i)
5: append all the indexes in i-queue to the proper queues[2] in H (see Figure 5)
6: **end for**
7: **if** $IIS(\mathcal{D}) \neq \emptyset$ **then**
8: **for all** itempair $\langle x, y\rangle$ in $IIS(\mathcal{D})$ **do**
9: $MS \leftarrow MSS(x) \cap MSS(y)$
10: **if** $MS \neq \emptyset$ **then**
11: **for all** mediator M in MS **do**
12: output $\langle x, y \mid M\rangle$
13: **end for**
14: **end if**
15: **end for**
16: **else**
17: output "Indirect associations do not exit in this database"
18: **end if**

procedure ***hi_mine***(H_m)
(Recursively mine the header table of itemset m and update $IIS(\mathcal{D})$ and $MSS(j)$, $j \notin m$)

1: **for** each item j in the header table H_m **do**
2: **if** the size of m is 1 and j's count $<$ *minimum itempair support count* **then**

3: add $\langle m, j \rangle$ to $IIS(\mathcal{D})$
4: **else if** j's count > *minimum mediator support count* **then**
5: **if** $IS(j,m) > t_d$ **then**
6: add m to $MSS(j)$
7: **end if**
8: create header table H_{mj} by scanning j-queue in H_m (i.e., mj-projected database) in the same way as building H except that item j and items in m are not considered (see Figure 8)
9: call ***hi_mine***(H_{mj})
10: **end if**
11: append all the indexes in j-queue to the proper queues[3] in H_m (see Figure 7)
12: **end for**

Let L = "x_1, x_2, ..., x_n" be the list of frequent 1-itemsets of $\mathcal{D}$ (with respect to t_f) in frequency-descending order. After build the initial *HI-struct* for $\mathcal{D}$ in step 1, the *HI-mine* algorithm first constructs the header table of item x_1 in step 3, using the x_1-queue in *HI-struct*, and calls procedure *hi_mine*(x_1) in step 4 to find $IIS(\mathcal{D})$ and *MSS*s that contain item x_1.

In procedure *hi_mine*(x_1), it tries to find those *indirect itempairs* that contain item x_1, and check whether $\{x_1\}$ can be added to the mediator support sets. Moreover, it further partitions the subsets of *MSS*s and does recursive mining when item x_1 is locally frequent with respect to t_f.

After $IIS(\mathcal{D})$ and *MSS*s that contain item x_1 are found, all the indexes in x_1-queue are moved to the proper queues in header table H to mine all the $IIS(\mathcal{D})$ and *MSS*s that contain item x_2 but not x_1.

Similarly, we can find the complete sets of $IIS(\mathcal{D})$ and $MSS(x)$, where $x \in L$. Following Theorem 1, we can generate all the indirect associations of $\mathcal{D}$ from these sets. This task is carried out from step 7 to step 18 in the algorithm.

3.4. Example

Figure 6 to Figure 14 show the execution of the algorithm on the transaction database *TDB* given in Table 2. The itempair support threshold t_s and mediator support threshold t_f are set to be 25% (minimum support count and minimum mediator support count are both 2)[4], and the minimum dependence threshold t_d is 0.5.

First, to find $IIS(TDB)$ and *MSS*s that contain item B, a B-header table H_B (shown in Figure 6) is created by traversing the B-queue in the header table H (shown in Figure 3) once. In H_B, every frequent item, except for B itself, has an entry with the same fields as H, i.e., item-id, support count and a pointer to a queue.

The support count in H_B records the support of the corresponding item in the B-queue. For example, since item A appears 3 times in the frequent-item projections of B-queue, the support count in the entry for A in H_B is 3. And all indexes of the frequent-item projection with the same first two items are linked together as a queue, and the entries in the header table H_B act as the heads of the queues.

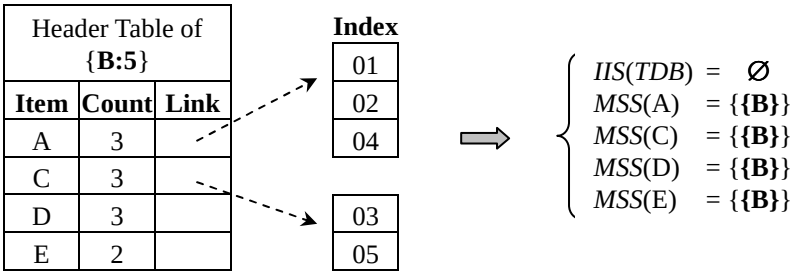

Figure 6. Header table H_B and mining result

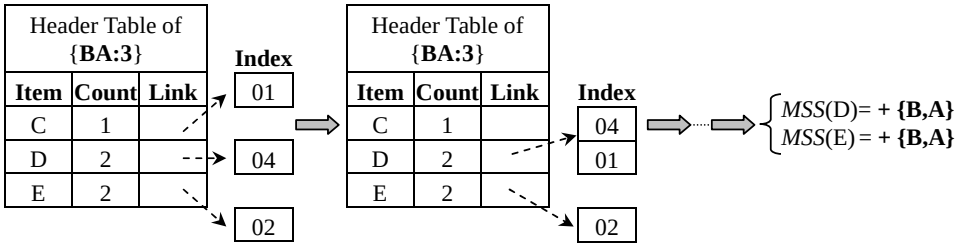

Figure 7. Header table H_{BA} and mining result

For instance, the C-queue in H_B stores all indexes of the frequent-item projections with the same first two items BC.

Since all the items in H_B are locally frequent, there is no indirect itempair contains item B, and $IIS(TDB)$ is empty after this scan. Then we compute the *IS* measure between B and each item in H_B:

$$IS(B, A) = 3/\sqrt{3 \times 5} = 0.77 \tag{2}$$

$$IS(B, C) = 3/\sqrt{3 \times 5} = 0.77 \tag{3}$$

$$IS(B, D) = 3/\sqrt{3 \times 5} = 0.77 \tag{4}$$

$$IS(B, E) = 2/\sqrt{2 \times 5} = 0.63 \tag{5}$$

They all pass the minimum dependence threshold 0.5. Therefore, {B} should be in the *MSS* of each of these items. The result is shown in Figure 6.

After {B} is appended into *MSS*s, a header table H_{BA} is created by examining A-queue in H_B in the same manner as in generating H_B from the B-queue in *H*. The header table H_{BA} is shown in the most left part of Figure 7. Then, the algorithm recursively exams the BA-projected database to determine whether {B,A} belongs to the *MSS*s of items C, D and E.

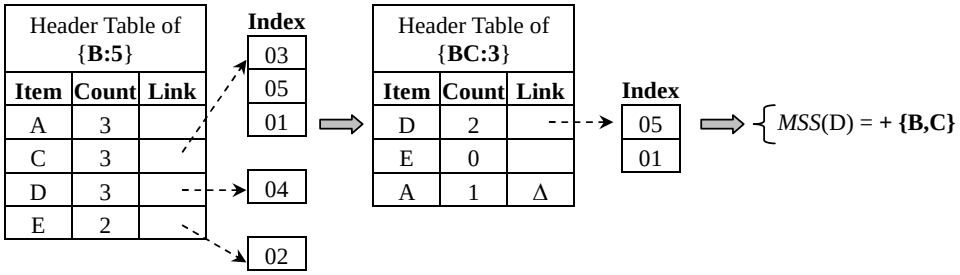

Figure 8. Adjusted header table H_B, header table H_{BC} and mining result

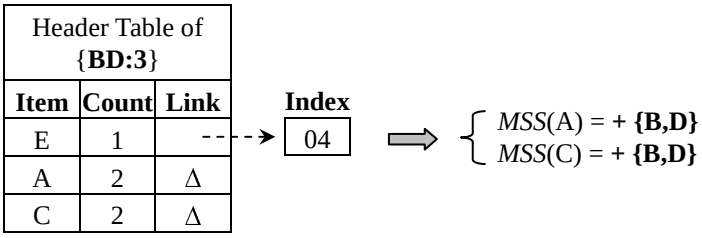

Figure 9. Header table H_{BD} and mining result

Since the local support count of C is less than 2, {B,A} is not added to MSS(C) and the search along path BAC completes. Thus the index in the C-queue of H_{BA} is moved into the D-queue of H_{BA} because D follows C in the projection corresponding to the index, which is the first projection B, A, C, D. The resulting header table after this adjustment is shown in the middle of Figure 7.

Since D is locally frequent and passes the dependence threshold, {B,A} is added to MSS(D). Then a header table H_{BAD}(not shown here) is created, which contains no local frequent items, and thus search along path BAD completes.

Similarly, {B,A} is added to MSS(E) and the process of mining the header table H_{BA} finishes. Then each index in the A-queue in table H_B is moved to proper queues as shown in the most left part of Figure 8.

After the above adjustment, the C-queue in H_B(also referred to as BC-queue) collects the complete set of frequent-item projections containing items B and C. Thus, by further creating a header table H_{BC}(shown in the middle of Figure 8), MSSs containing item B and C but not A can be mined recursively.

Please note that item A appears in H_{BC} because it does not belong to B,C and it appears in the frequent-item projections of BC-queue. However, its queue is always empty, that is, we will not append any index to its queue after D-queue or E-queue in H_{BC} has been mined since it has been considered in the mining of the BA-queue. Thus, the A-queue in H_{BC} is marked with "△". We need an entry for A here because we need to output the correct support mediators in MSS(A) if the local count of A is above the minimum mediator support count. The result is

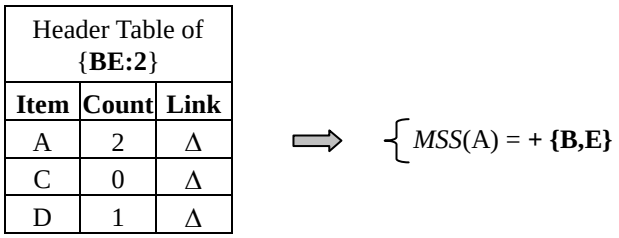

Figure 10. Header table H_{BE} and mining result

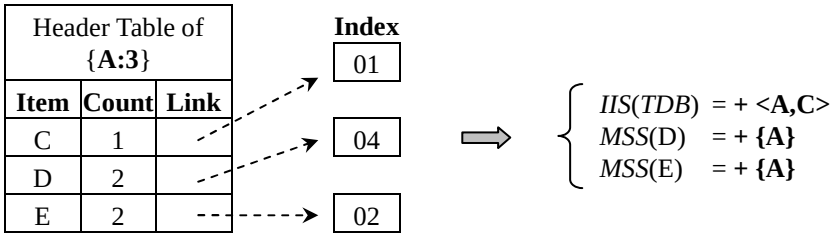

Figure 11. Header table H_A and mining result

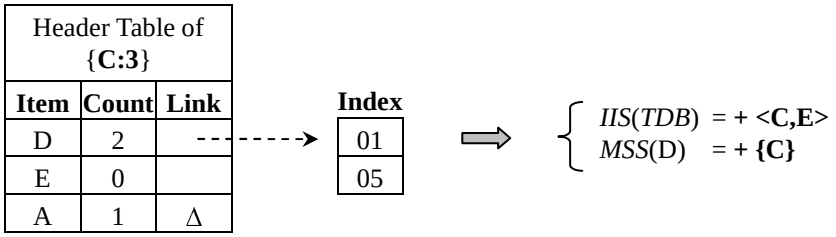

Figure 12. Header table H_C and mining result

shown in Figure 8. And the header table H_{BD} and H_{BE}, and their corresponding mining results are shown in Figure 9 and Figure 10 respectively.

In the next step, all the indexes in B-queue are moved into the proper queues in H to mine $IIS(TDB)$ and MSSs that contain item A but not B, and other subsets of them. The header table H after this adjustment is shown in Figure 5.

Header table H_A and the mining result are shown in Figure 11. Since C is locally infrequent with respect to A, pair $\langle$A,C$\rangle$ is added to $IIS(TDB)$.

Similarly, the mining process continues to discover $IIS(TDB)$ and MSSs that contain item C, item D and item E, as shown from Figure 12 to Figure 14. It is easy to see that the above mining process finds the complete $IIS(TDB)$ and MSSs because we partition the sets into disjoined subsets and mine each subset

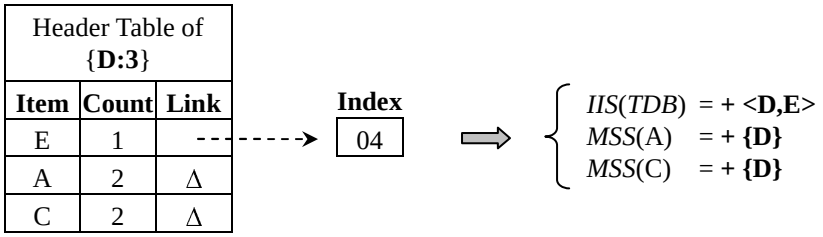

Figure 13. Header table H_D and mining result

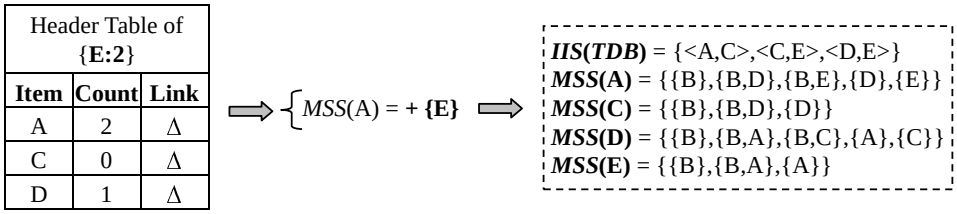

Figure 14. Header table H_E and mining result

by further partitioning it recursively. The complete $IIS(TDB)$ and MSSs for our example database TDB are shown in Figure 14.

The second phase of the *HI-mine* algorithm is to compute the set of mediators for each *indirect itempair* in $IIS(TDB)$ (see steps 7-18 in *HI-mine* algorithm). For example, the set of mediators for itempair $\langle$A, C$\rangle$ in $IIS(TDB)$ is computed by intersecting MSS(A) and MSS(C), which results in {{B}, {D}, {B, D}}. Therefore, three indirect associations are discovered for itempair $\langle$A,C$\rangle$:

$$\langle \text{A, C} \mid \{\text{B}\}\rangle, \langle \text{A, C} \mid \{\text{D}\}\rangle, \langle \text{A, C} \mid \{\text{B, D}\}\rangle$$

Similarly, the following indirect associations are discovered for itempairs $\langle$C,E$\rangle$ and $\langle$D,E$\rangle$:

$$\langle \text{C, E} \mid \{\text{B}\}\rangle$$
$$\langle \text{D, E} \mid \{\text{A}\}\rangle, \langle \text{D, E} \mid \{\text{B}\}\rangle, \langle \text{D, E} \mid \{\text{A, B}\}\rangle$$

From Theorem 1, we can guarantee that the above mining process finds the complete set of indirect associations without duplication.

3.5. Complexity Analysis

Suppose there are N transactions in $\mathcal{D}$. Let f be the number of frequent items of size 1, m be the maximal length of a single frequent itemset. We will now briefly discuss the computational complexity of our algorithm from the following three aspects: database scan, space usage and time complexity.

3.5.1. Database scan The two most important performance factors of the association rules mining are the number of passes made over the database and the efficiency of those passes [6]. The *HI-mine* algorithm described in the previous section shows that if the frequent-item projections of $\mathcal{D}$ plus a set of header tables can fit in main memory then two and only two scans of $\mathcal{D}$ are needed to build the initial *HI-struct*. First, it scans $\mathcal{D}$ once to get the set of frequent items. Then it scans $\mathcal{D}$ again to construct an *HI-struct*. After the initial *HI-struct* is constructed, the remaining mining process is performed on the *HI-struct* only, without referencing any information in the original database.

If the *HI-struct* cannot be held in main memory, then in the second scan of database, we partition the set of frequent-item projections into f parts, where each part collects the frequent-item projections of one frequent item. At the same time, the first partition (i.e., the set of frequent-item projections start with item B in our example) is held in the main memory to build a partial *HI-struct* which contain enough information for mining $IIS(\mathcal{D})$ and *MMS*s that contain the first frequent item. After that, the frequent-item projections in the first partition are moved into proper partitions (in the same way as we adjust the indexes of frequent-item projections in main memory), and the second partition is loaded in the main memory to build and mine the partial *HI-struct* of the second frequent item (item A in our example). This process continues until $IIS(\mathcal{D})$ and *MMS*s that contain the last frequent item (item E in our example) are generated.

One can easily see that a partial *HI-struct* is usually orders of magnitude smaller than the global *HI-struct*. Therefore, there is a good chance for it to be fit in the main memory. But if not, we can further partition the set of frequent-item projections of each frequent item, and the process can go on recursively until the partial *HI-struct* fits in the main memory.

3.5.2. Space usage For each transaction in $\mathcal{D}$, the initial *HI-struct* stores its frequent-item projection, and the space requirement is: $\sum_{i=1}^{N}|pro(T_i)|$, where $|pro(T_i)|$ is the length of a frequent-item projection of a transaction T_i. Besides frequent-item projections, *HI-struct* also stores a header table H. The maximal number of entries in the table is at most the number of frequent items, so the space requirement of head table H is at most $3f$ and the total size of all the queues is less than N.

To mine the *HI-struct*, the only space overhead is a set of local header tables. If the maximal length of a single frequent itemset is m, then the corresponding frequent itemset tree[5] would contain a maximum of $f{\cdot}2^m$ nodes. Therefore, in the worst case, our algorithm will generate at most $f{\cdot}2^m$ header tables. However, there are only a very limited number of header tables exist simultaneously in the algorithm. For the previous example, to compute the frequent itemset ACDE, only the header table for the prefixes of ACDE, i.e., H_A, H_{AC}, H_{ACD} and H_{ACDE} are needed. All the other header tables either are already used and can be released, or have not been generated yet. The header tables for frequent itemsets having item B have already been used and can be released since all the frequent itemsets having item B have been computed before frequent itemset ACDE. On the other hand, all the other header tables will be used later and need not be generated at

this moment. Therefore, the number of header tables is no more than the maximal length of a single frequent itemset that can be found. Thus the total space usage of *HI-mine* algorithm in the worst case is:

$$m(3f+N)+\sum_{i=1}^{N}|pro(T_i)|$$

3.5.3. Time complexity After building the initial *HI-struct*, there are two phases in the algorithm: first, generate $IIS(\mathcal{D})$ and all the *MSS*s; and then generate all the indirect associations. The performance of the first phase dominates the performance of the overall algorithm, since the size of transaction data is usually much more larger than the size of $IIS(\mathcal{D})$ and *MSS*s.

It's easy to find that the complexity of the first phase depends on the total number of the header tables used in the mining process, which is at most $f{\cdot}2^m$ as described in the space usage section. In the worst case (e.g. header table H), the time complexity of constructing and mining each header table is $O(p{\cdot}N)$, where p is the maximal length of the frequent itemset projections. Therefore, the time complexity of the first phase is $O(2^m{\cdot}f{\cdot}p{\cdot}N)$ in the worst case.

Suppose there are u itempairs that can be found in $IIS(\mathcal{D})$ and the maximal number of mediators in a single mediator support set is v, then the second phase will take at most $O(u{\cdot}v^2)$ time to discover the complete set of indirect associations. Hence, the overall worst-case complexity of this algorithm is $O(2^m{\cdot}f{\cdot}p{\cdot}N + u{\cdot}v^2)$. Please note that N is far more greater than m, f, p, u and v in real world applications, thus the total time is $O(N)$.

4. Experimental Evaluation and Performance Study

In this section, we report our experimental results on the performance of *HI-mine* in comparison with two versions of the INDIRECT algorithm, INDIRECT-A and INDIRECT-F, which extract frequent itemsets using *Apriori* and *FP-growth* in the first step, respectively.

All the experiments are performed on a 533-MHz Pentium PC machine with 128M main memory, running on Microsoft Window 2000 Professional. All the programs are written in Sun Java 1.3.1. The algorithms are tested on two types of data sets: synthetic data, which mimic market basket data, and anonymous web data, which belong to the domain of web log databases. To evaluate the performance of the algorithms over a large range of data characteristics, we have tested the programs on various data sets and only the results on some typical data sets are reported here. Moreover, these algorithms generate exactly the same set of indirect associations for the same input parameters.

Please note that run time used here means the total execution time, i.e., the period between input and output. Also, in all reports, the run time of *HI-mine* include the time of constructing *HI-struct*, and the run time of INDIRECT-F include the time of constructing *FP-tree* from the original database as well.

Table 3. Parameters used in the synthetic data generation

$\|D\|$	Total number of transactions
$\|T\|$	Average size of transactions
$\|I\|$	Average size of maximal potentially frequent itemsets
$\|L\|$	Number of maximal potentially frequent itemsets
N	Total number of items

Table 4. Parameters settings of synthetic data sets

Name	$\|T\|$	$\|I\|$	$\|D\|$	Size in Megabytes
T5.I3.D20k	5	3	20k	0.6
T10.I5.D20k	10	5	20k	1.0
T5.I3.D50k	5	3	50k	1.3
T10.I5.D50k	10	5	50k	2.2
T5.I3.D100k	5	3	100k	2.5
T10.I5.D100k	10	5	100k	4.4

4.1. Experiments with Synthetic Data

The synthetic data sets which we used for our experiments were generated using the procedure described in [4]. These transactions mimic the actual transactions in a retail environment. The transaction generator takes the parameters shown in Table 3.

Each synthetic data set is named after these parameters. For example, the data set T10.I5.D20K uses the parameters $|T| = 10$, $|I| = 5$, and $|D| = 20000$. For all the experiments, we generate data sets by setting $N = 1000$ and $|L| = 2000$ since these are the standard parameters used in [4]. We chose 2 values for $|T|$: 5 and 10. We also chose 2 values for $|I|$: 3 and 5. And the number of transactions are set to 20000, 50000 and 100000. Table 4 summarizes the data set parameter settings. For the same $|T|$ and $|D|$ values, the size of data sets in megabytes are roughly equal for the different value of $|I|$.

In our experiments, the itempair support threshold is set to be the same as the mediator support threshold, and the mediator dependence threshold is set to be 0.1. Figures 15 and 16 show the execution times for the six synthetic data sets given in Table 4 for decreasing values of mediator support threshold. As the minimum mediator support threshold decreases, the execution times of all the algorithms increase because of increases in the total number of frequent itemsets.

4.2. Experiments with Real-World Data

We test the algorithms on two real-world data sets. One of them was obtained from http://kdd.ics.uci.edu/databases/msweb/msweb.html. It was created by sampling and processing the www.microsoft.com logs. The data records the use of www.microsoft.com by 38000 anonymous, randomly-selected users. For each user,

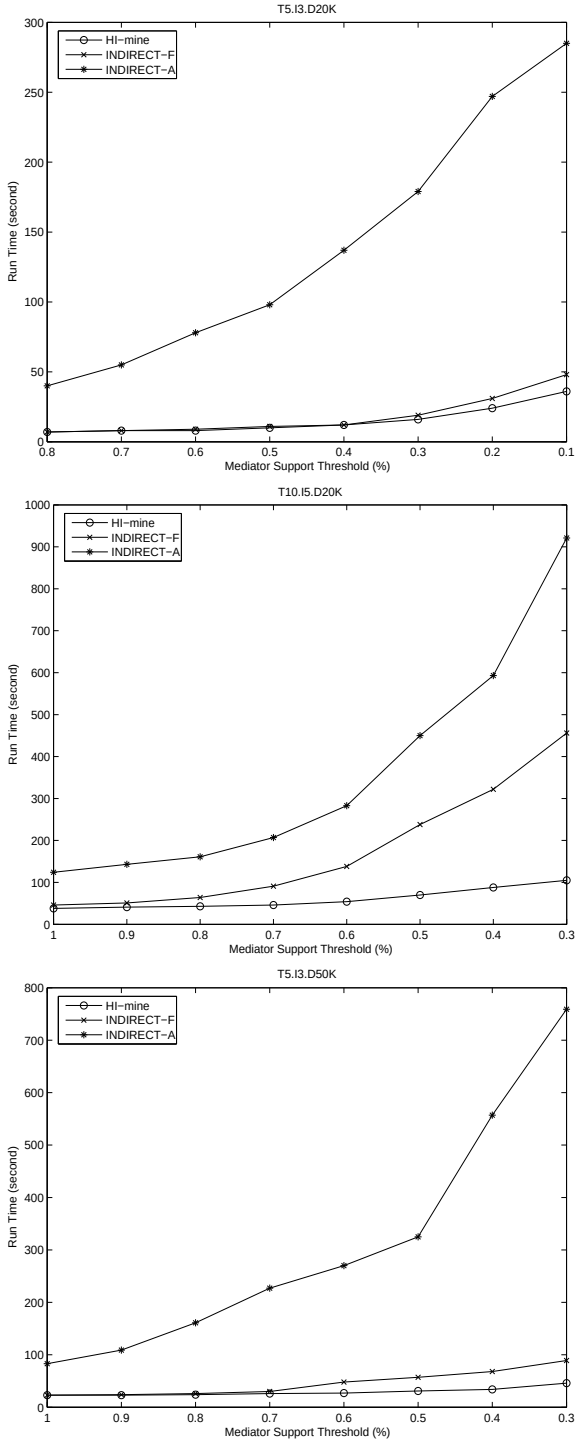

Figure 15. Run time comparison on synthetic data set (1)

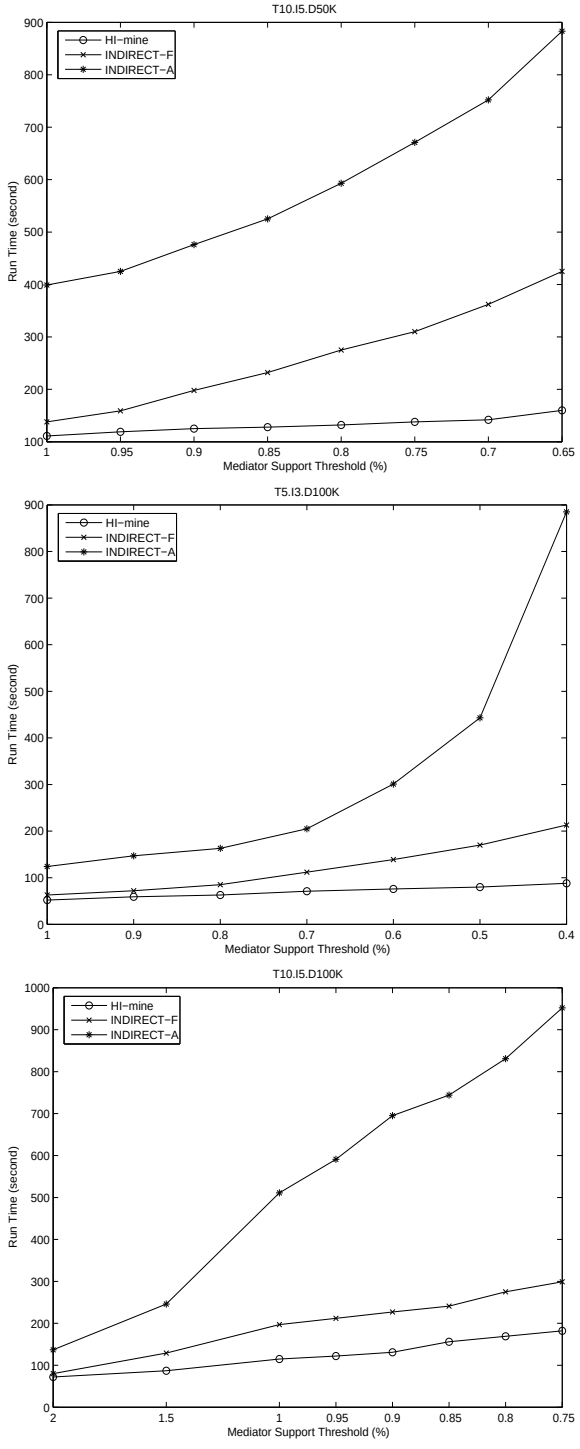

Figure 16. Run time comparison on synthetic data set (2)

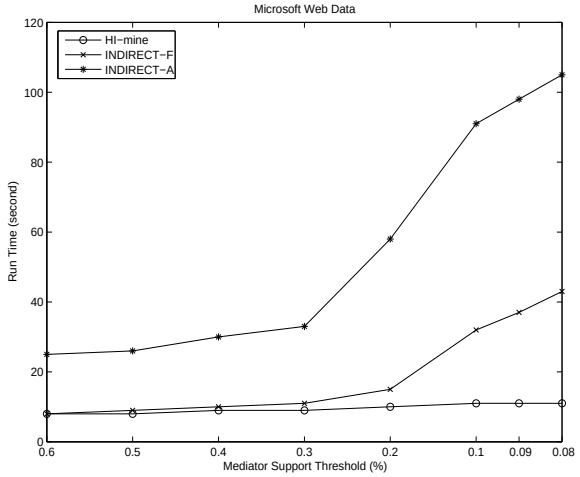

Figure 17. Run time comparison on Microsoft web log data

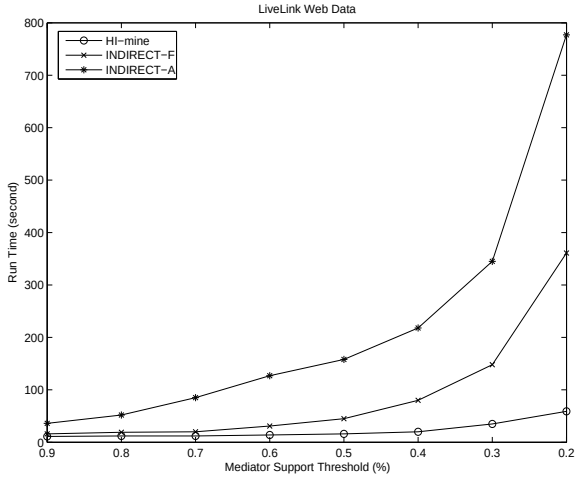

Figure 18. Run time comparison on Livelink web log data

the data lists all the areas of the web site that user visited in a one week time frame. The data set contains 32711 instances (transactions) with 294 attributes (items); each attribute is an area of the www.microsoft.com web site. The corresponding performance curves are illustrated in Figure 17.

The other data set was first used in [9] to discovery interesting association rules from Livelink [6] web log data. This data set is not publicly available for proprietary

reasons. The log files contain Livelink access data for a period of two months (April and May 2002). The size of the raw data is 7GB. The data describe more than 3,000,000 requests made to a Livelink server from around 5,000 users. Each request corresponds to an entry in the log files. The detail of data preprocessing, which transformed the raw log data into the data that can be used for learning association rules, was described in [8].

The resulting session file used in our experiment was derived from the 10-minute time-out session identification method. The total number of sessions (transactions) in the data set is 30,586 and the total number of objects [7] (items) is 38,679. The corresponding performance chart is shown in Figure 18.

4.3. Performance Study

As we can see from the figures, the *HI-mine* algorithm outperforms the other two algorithms on all data sets. At high support threshold values, *HI-mine* and INDIRECT-F have similar performance and they both outperform INDIRECT-A. However, as the support threshold goes lower, the gap between INDIRECT-F and *HI-mine* and the gap between *HI-mine* and INDIRECT-A become larger. It is interesting to observe that the lines for *HI-mine* in the figures are quite flat, which means that the run time of *HI-mine* does not increase much as the support threshold goes lower.

For synthetic data sets T10.I5.D20k and T5.I3.D50k, INDIRECT-F has almost the same execution time as *HI-mine* when the mediator support threshold is over 0.7%. When the mediator support threshold decreases under 0.7%, the performance gap becomes outstanding. At the reasonable low support threshold of 0.5% in T10.I5.D20k, for example, *HI-mine* requires 70 seconds, whereas INDIRECT-F requires 238 seconds and INDIRECT-A requires 450 seconds. At the even lower support threshold of 0.3% in T5.I3.D50k, *HI-mine* requires 46 seconds, while INDIRECT-F requires 89 seconds and INDIRECT-A requires 759 seconds.

On Microsoft and LiveLink web data sets, when the support threshold is large, such as 0.4% for Microsoft data set and 0.7% for LiveLink data set, INDIRECT-F behaves the same as *HI-mine*. However, the performance gap becomes significant when the support threshold decreases to lower value. For example, *HI-mine* finishes in 35 seconds while INDIRECT-F runs over 148 seconds and INDIRECT-A requires 345 seconds for the support level of 0.3% in LiveLink data set. When the support threshold decreases to an even lower level, improvements of *HI-mine* are more striking.

The reason for which INDIRECT-F is better than INDIRECT-A is that *FP-growth* does not generate candidates when it generates frequent patterns and the generation of frequent patterns is based on a compressed tree structure (*FP-tree*), which is usually much smaller than the original database. However, INDIRECT-F generates candidates for indirect associations using a join operation. *HI-mine* does not perform any candidate generation. It discovers indirect associations directly based on the *HI-struct* data structure.

The reason that the run time of *HI-mine* does not change much with the support threshold is that, when the support threshold decreases, the number of frequent itemsets increases, but the number of indirect associations may not increase because there are fewer indirect itempairs. For example, at mediator support 0.2% in Figure 18, the number of indirect associations in Livelink web data is only 555 while the number of frequent itemsets is 29,685. On the other hand, the run time of INDIRECT depends primarily on the number of frequent itemsets generated by *Apriori* or *FP-growth*. Therefore, avoiding generating all the frequent itemsets in *HI-mine* makes it the most efficient on both synthetic and real-world data sets at all levels of support threshold.

5. Conclusion

In this paper, we have proposed an efficient algorithm, *HI-mine*, which uses a new data structure, *HI-struct*, to discover all indirect associations between items. The salient features of *HI-mine* include that it avoids generating all the frequent itemsets before generating indirect associations and that it generates indirect associations directly without candidate generation. We have compared this algorithm to the previously known algorithm, the INDIRECT algorithm, using both synthetic and real-world data. As shown in our performance study, the proposed algorithm significantly outperforms the INDIRECT algorithm, which uses a standard frequent itemset generation algorithm such as *Apriori* and *FP-growth* to extract the frequent itemsets before mining indirect associations.

For future research, there are several unresolved issues we need to address.

1. *Scalability issue.* The current implementation of *HI-mine* algorithm compresses the database into frequent-item projections. If the projected database can be stored in the main memory, there is no extra disk I/O in the subsequent mining process. Otherwise, multiple scans of (part of) the projected database (usually much smaller than the original database if the database is sparse) are needed in the process of learning $IIS(\mathcal{D})$ and *MSS*s. We will work on the issue of how to further reduce disk I/Os when the database is huge, e.g., with millions of transactions.

2. *Measure selection.* In addition to the *IS* measure used in our thesis, various interestingness measures have been proposed to identify the statistical significance of association rules. In the next step of our study, a proper evaluation of these measures should be carried out before deciding what is the right measure to use for finding interesting indirect associations. Threshold selection is another issues that needs further investigation.

3. *Framework extension.* It is possible to extend our work to discover indirectly associated itemsets rather than between a pair of items. In addition, we have ideas for how to develop a broader framework to directly generate both positive and negative association rules, and to improve the efficiency for interesting rule generation as well.

Acknowledgments

This research is supported by research grants from the Natural Sciences and Engineering Research Council (NSERC) of Canada and Communications and Information Technology Ontario (CITO). We would like to thank Mr. Miao Wen for his help in implementing the INDIRECT algorithm.

Notes

1. The initial header table of a database may contain more than one queue. We use a simple example for the convenience of explanation.
2. The proper queue is the queue of the item right after item i in the corresponding frequent-item projection.
3. The proper queue is the queue of the item right after item j in the corresponding frequent-item projection.
4. The two thresholds are of the same value here just for the convenience of explanation. They can be different.
5. See the example frequent itemset tree shown in [12] for more detail
6. Livelink is a product of Open Text Corporation (http://www.opentext.com).
7. An object could be a document (such as a PDF file), a project description, a task description, a news group message, a picture and so on [8].

References

1. R. Agarwal, C. Aggarwal, and V. V. V. Prasad. Depth first generation of long patterns. In *Proceedings of ACM-SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2000.
2. R. Agarwal, C. Aggarwal, and V. V. V. Prasad. A tree projection algorithm for generation of frequent itemsets. In *Journal of Parallel and Distributed Computing (Special Issue on High Performance Data Mining)*, 2000.
3. R. Agarwal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C., USA, May 1993.
4. R. Agarwal and R. Strikant. Fast algorithms for mining association rules. In *Proceedings of 20th International Conference on Very Large Data Bases*, pages 487–499, Santiago, Chile, September 1994.
5. R. J. Bayardo. Efficiently mining long patterns from databases. In *Proceedings of the International ACM SIGMOD Conference*, pages 85–93, May 1998.
6. S. Brin, R. Motwani, J. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proceedings of the International ACM SIGMOD Conference*, pages 255–264, Tucson, Arizona, USA, May 1997.
7. U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery: an overview. In AAAI Press/MIT Press, editor, *Advances in Knowledge Discovery and Data Mining*, pages 1–36, 1996.
8. X. Huang, A. An, and N. Cercone. Evaluation of interestingness measures in a real world application. In *submitted for Journal publication*, 2002.
9. X. Huang, A. An, N. Cercone, and G. Promhouse. Discovery of interesting association rules from livelink web log data. In *Proceedings of IEEE International Conference on Data Mining*, Maebashi City, Japan, 2002.
10. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 1–12, Dallas, TX, May 2000.

11. J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A Frequent-Pattern Tree Approach. In *Journal of Data Mining and Knowledge Discovery*, 8. 53–87, 2004.
12. J. Liu, Y. Pan, K. Wang, and J. Han. Mining frequent itemsets by opportunistic projection. In *Proceedings of ACM-SIGKDD International Conference on Knowledge Discovery and Data Mining*, Edmonton, Canada, July 2002.
13. H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. In *AAAI Workshop on Knowledge Discovery in Databases*, pages 181–192, July 1994.
14. J. S. Park, M. S. Chen, and P. S. Yu. An effective hash-based algorithm for mining association rules. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, San Jose, CA, May 1995.
15. J. Pei. *Pattern-Growth Methods for Frequent Pattern Mining*. PhD thesis, Simon Fraser University, 2002.
16. J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang. H-mine: hyper-structure mining of frequent patterns in large database. In *Proceedings of the IEEE International Conference on Data Mining*, San Jose, CA, November 2001.
17. A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *Proceedings of the 21st International Conference on Very Large Data Bases*, Zurich, Switzerland, September 1995.
18. A. Savasere, E. Omiecinski, and S. Navathe. Mining for strong negative associations in a large database of customer transactions. In *Proceedings of the 14th International Conference on Data Engineering*, pages 494–502, Orlando, Florida, February 1998.
19. P. Tan and V. Kumar. Interestingness measures for association patterns: A perspective. In *KDD 2000 Workshop on Postprocessing in Machine Learning and Data Mining*, Boston, MA, August 2000.
20. P. Tan and V. Kumar. Mining indirect associations in web data. In *Proc of WebKDD2001: Mining Log Data Across All Customer TouchPoints*, August 2001.
21. P. Tan, V. Kumar, and H. Kuno. Using sas for mining indirect associations in data. In *Proc of the Western Users of SAS Software Conference*, 2001.
22. P. Tan, V. Kumar, and J. Srivastava. Indirect association: mining higher order dependencies in data. In *Proceedings of the 4th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 632–637, Lyon, France, 2000.
23. P. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *Proceedings of the 8th International Conference on Knowledge Discovery and Data Mining*, Edmonto, CA, July 2002.
24. Q. Wan and A. An. Efficient mining of indirect associations using hi-mine. In *Proceedings of 16th Conference of the Canadian Society for Computational Studies of Intelligence, AI 2003*, Halifax, Canada, June 2003.
25. Wong and C. J. Butz. Constructing the dependency structure of a multi-agent probability network. *IEEE Transactions on Knowledge and Data Engineering*, 13(3):395-415, 2001.
26. X. Wu, C. Zhang, and S. Zhang. Mining both positive and negative association rules. In *Proceedings of the 19th International Conference on Machine Learning (ICML-2002)*, pages 658–665, Sydney, Australia, July 2002.
27. M. Zaki and M. Orihara. Theoretical foundations of association rules. In *Proceedings of the 3rd ACM-SIGMOD Workshop on Research Issues in Data Mining and Kownledge Discovery*, Seattle, WA, June 1998.
28. Z. Zheng, R. Kohavi, and L. Mason. Real world performance of association rule algorithms. In *Proceedings of International Conference on Kownledge Discovery in Databases*, August.