Efficient Mining of Intertransaction Association Rules

Anthony K.H. Tung, *Member*, *IEEE*, Hongjun Lu, *Member*, *IEEE*, Jiawei Han, *Member*, *IEEE*, and Ling Feng, *Member*, *IEEE*

Abstract—Most of the previous studies on mining association rules are on mining *intratransaction associations*, i.e., the associations among items within the *same transaction* where the notion of the transaction could be the items bought by the *same customer*, the events happened on the *same day*, etc. In this study, we break the barrier of transactions and extend the scope of mining association rules from traditional single-dimensional, intratransaction associations to multidimensional, intertransaction associations. An intertransaction association describes the association relationships among *different transactions*. In a database of stock price information, an example of such an association is "*if (company) A's stock goes up on day one, B's stock will go down on day two but go up on day four*." In this case, no matter whether we treat company or day as the unit of transaction, the associated items belong to different transactions. Moreover, such an intertransaction association can be extended to associate multiple properties in the same rule, so that *multidimensional* intertransaction associations can also be defined and discovered. Mining intertransaction associations pose more challenges on efficient processing than mining intratransaction associations because the number of potential association rules becomes extremely large after the boundary of transactions is broken. In this study, we introduce the notion of intertransaction association rule, define its measurements: *support* and *confidence*, and develop an efficient algorithm, FITI (an acronym for "*First Intra Then Inter*"), for mining intertransaction associations, which adopts two major ideas: 1) an intertransaction frequent itemset contains *only* the frequent itemsets of its corresponding intratransaction counterpart; and 2) a special data structure is built among intratransaction frequent itemsets for efficient mining of intertransaction frequent itemsets. We compare FITI with EH-Apriori, the best algorithm in our previous proposal, and demonstrate a substantial performance gain of FITI over EH-Apriori. Further extensions of the method and its implications are also discussed in the paper.

Index Terms—Data mining, association rules, temporal pattern discovery.

1 Introduction

As an important theme on data mining, association rule mining research [1] has progressed in various directions, including efficient, apriori-like mining methods [2], [10], [23], [19], [25], mining generalized, multilevel, or quantitative association rules [20], [8], [21], [7], [17], [14], [9], mining sequential patterns and episodes [2], [16], association rule mining query languages [15], constraint-based rule mining [24], [18], [13], mining correlations, and causal structures [4], [22], etc. However, there is an important form of association rules which are useful but could not be discovered with existing association rule mining framework.

Taking stock market database as an example, association rule mining can be used to analyze the share price movements. Suppose a database registers the price of every stock at the end of each trading day. Association mining may find rules like

- R_1: *When the prices of IBM and SUN go up, at 80 percent of probability the price of Microsoft goes up on the same day*.

While R_1 reflects some relationship among the prices, its role in price prediction is limited; and traders may be more interested in the following type of rules:

- R_2 : *If the prices of IBM and SUN go up, Microsoft's will most likely (80 percent of probability) go up* **the next day** *and then drop* **four days later**.

Unfortunately, current association rule miners cannot discover this type of rule as there is a fundamental difference between the rules like R_2 and those like R_1. A classical association rule like R_1 expresses the associations among items within the *same transaction*. On the other hand, rule R_2 represents some association relationship *among the field values from different transaction records*. To distinguish these two types of transactions, we name the classical association rules as *intratransaction associations* and the latter as *intertransaction associations*.

It is interesting to note that, although there is a temporal component in our example of share price movement, rule R_2 is different from mining sequential patterns in transaction data [3]. For mining sequential patterns, the transactions of each customer along time are treated as one transaction. Rules so discovered are intratransaction in nature because each

- *A.K.H. Tung is with the Department of Computer Science, National University of Singapore, 3 Science Drive 2, Singapore 117543. E-mail: atung@comp.nus.edu.sg.*
- *H. Lu is with the Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon Hong Kong. E-mail: luhj@cs.ust.hk.*
- *J. Han is with the Department of Computer Science, University of Illinois at Urbana-Champaign, Digital Computer Laboratory (DCL), 1304 West Springfield Avenue, Urbana, IL 61801, E-mail: han@cs.uiuc.ca.*
- *L. Feng is with Infolab, Tilburg University, B302 PO Box 90153, 5000 LE Tilburg, Netherlands. E-mail: ling@kub.nl.*

Manuscript received 8 Nov. 1999; revised 15 May 2001; accepted 22 May 2001.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 110908.

sequence is in fact treated as one transaction and the mining process is to find similarities among the sequences.

At the first glance, the work reported by Mannila et al. on mining episodes [16] is similar to what we described in our examples. An episode is a sequence of events and association rules among episodes have the form $P[V] \Rightarrow Q[W]$, where P and Q are event sequences and V and W are time bounds. There is however no clear concept of transaction in a sequence of events. Instead, subsequences are formed using the time bounds and episodes that occurred frequently within these subsequence are discovered using an extension of apriori algorithm [2]. These subsequences, which can contain overlapping event instances are different from transactions which share no common item instances. Even if these subsequences are treated as transactions, the rule so discovered are intratransaction, not intertransaction.

Another area which seems related is the work of Bettini et al. [5], [6] which proposed the use of event structure for the discovery of temporal relationship among events in a time sequence. An event structure consists of a number of variables representing events and temporary constraints among these variables. The goal of the mining is to find temporal patterns in the sequence that could instantiate the event structure. Such work belongs to those of a belief system in which the user tries to verify their belief by providing an event structure to discover a constraint set of rules. Again, there is no clear concept of transactions here.

There are many applications which can benefit from intertransaction association mining. For example, one may discover traffic jam association patterns among different highway segments: "*if highway one east bound is jammed at section two, highway 12 south bound will be jammed at section five 10 minutes later.*" Although it is pretty challenging to find the causal structure for a complex highway system with dynamic traffic flow, intertransaction association mining may help traffic jam pattern prediction. It may also help find that the sale of a particular kind of baby toy will boost the sales of women clothes in the following week, or help discover if there is a drought in region A, there will be a flood in region B two years later from a long term regional weather database.

From the above brief description, we can identify that the first contribution of our work reported in this paper is the formulation of an association mining problem that is more general than what has been discussed in the literature. The new association relationship breaks the barriers of transactions and can be used for certain applications such as prediction. Moreover, from the discussion in the later sections, it will be shown that our formulation can be extended to describe multidimensional intertransaction association rules.

Since intertransaction association breaks the boundaries of transactions, the number of potential itemsets and the number of rules will increase drastically. Another contribution of our work is that we have developed an efficient algorithm for mining intertransaction association rule from large databases. Using the property that "*A frequent intertransaction itemsets must be made up of frequent intratransaction itemsets,*" the mining of intertransaction association rules from large databases is achieved within a reasonable amount of time.

The remainder of the paper is organized as follows: In Section 2, we will define intertransaction association rules formally. In Section 3, we will describe the principles behind our mining methods for intertransaction association rules. We will have a detailed look at the FITI algorithm developed for mining intertransaction association rules in Section 4 followed by performance studies and experiments in Section 5. In Section 6, we will discuss further issues such as the scalability of FITI in the real world situation and suggest some improvements to FITI. In addition to these, we will also discuss how intertransaction association rules can be extended to give multidimensional prediction capability and the implication behind our work. We will conclude with Section 7.

2 Problem Description

In this section, we will describe the problem of mining intertransaction association rules. Before doing so, we will first give some definitions.

Definition 2.1. *Let $\sum = \{e_1, e_2, \ldots, e_u\}$ be a set of literals, called items. Let D be an attribute and $Dom(D)$ be the domain of D. A* **transaction database** *is a database containing transactions in the form of (d, E), where $d \in Dom(D)$ and $E \subseteq \sum$.*

The attribute D in the transaction database is called a *dimensional attribute*. It describes the properties associated with the items, such as time and location. It is assumed that the domain of the dimensional attribute is ordinal and can be divided into equal length intervals. For example, time can be divided into day, week, month, etc., and distance into meter, mile, etc. These intervals can be represented by integers 0, 1, 2, etc., without loss of generality.

When there is an association between items which are m intervals apart, we say that there is an intertransaction association rule which spans across m intervals. While an intertransaction association rule can theoretically span across many intervals, discovering all such rules will take up too much resources and users may not be interested in the rules which span more than a certain number of intervals. To avoid spending unnecessary resources to mine the rules which users are not interested in, a mining parameter called **maxspan** (or $sliding_window_size$) denoted by w is introduced. When mining intertransaction association rules, only the rules which span less than or equal to w intervals will be mined. Users can thus use this mining parameter to avoid mining the rules that span across too many intervals.

Using w, a sliding window in the transaction database is defined as follows:

Definition 2.2. *A* **sliding window** *W in a transaction database T is a block of w continuous intervals along domain D, starting from interval d_0 such that T contains a transaction at interval d_0. Each interval d_j in W is called a subwindow of W denoted as $W[j]$, where $j = d_j - d_0$. We call j the subwindow number of d_j within W.*

We will use Example 2.1 to illustrate these terms.

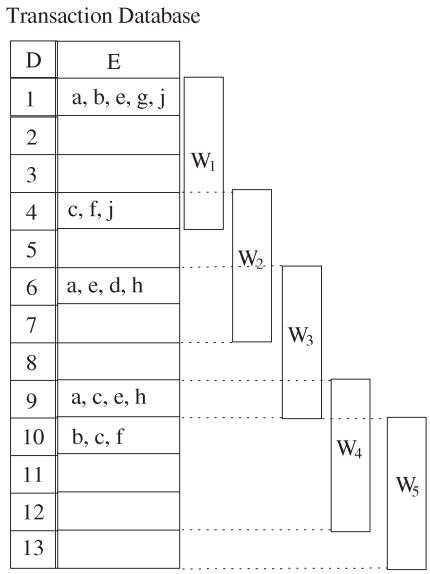


Fig. 1. Transaction database and sliding windows.

Example 2.1. Fig. 1 shows a transaction database T with five transactions located at intervals 1, 4, 6, 9, and 10. Assuming that w has a value of 4, we will have five sliding windows W_1, W_2, W_3, W_4, and W_5, with addresses of 1, 4, 6, 9, and 10, respectively. From the figure, it can be seen that subwindow $W_1[0]$ contains items a, b, e, g, and j while subwindow $W_1[3]$ contains c, f, and j.

Every sliding window in the transaction database forms a *megatransaction* defined below.

Definition 2.3. *Let W be a sliding window with w intervals and u be the number of literals in $\sum$. We define a* **megatransaction** *M contained within W to be*

$$M = \{e_i(j) | e_i \in W[j], 1 \leq i \leq u, 0 \leq j \leq w-1\}.$$

As can be seen from the definition, a megatransaction in a sliding window W is just the set of items in W, each appended with the corresponding subwindow number of the interval that contains the item. For example, in Example 2.1, the megatransaction in W_1 will be $\{a(0), b(0), e(0), g(0), j(0), c(3), f(3), j(3)\}$.

To distinguish the items in a megatransaction from the items in a traditional transaction, the items in a megatransaction are called **extended-items**. We denote the set of all possible extended-items as $\sum'$. Given $\sum$ and w, we will have

$$\sum' = \{e_1(0), \ldots, e_1(w-1), e_2(0), \ldots, e_2(w-1), \ldots, e_u(0), \ldots, e_u(w-1)\}.$$

Next, we will introduce two terms: *intratransaction itemset* and *intertransaction itemset*.

Definition 2.4. *An* **intratransaction itemset** *is a set of items $A \subseteq \sum$. An* ***intertransaction itemset*** *is a set of extended items $B \subseteq \sum'$ such that $\exists e_i(0) \in B, 1 \leq i \leq u$.*

Now, we are ready to define the concept of *intertransaction association rule*.

Definition 2.5. *An* **intertransaction association rule** *is an implication of the form $X \Rightarrow Y$, where*

1. $X \subseteq \sum', Y \subseteq \sum'$.
2. $\exists e_i(0) \in X, 1 \leq i \leq u$.
3. $\exists e_i(j) \in Y, 1 \leq i \leq u, j \neq 0$.
4. $X \cap Y = \emptyset$.

Similar to the studies in mining intratransaction association rules, we introduce two measures of intertransaction association rules: *support* and *confidence*.

Definition 2.6. *Let S be the number of transactions in the transaction database. Let T_{xy} be the set of megatransactions that contain a set of extended-items $X \cup Y$ and T_x be the set of megatransactions that contain X. Then, the support and confidence of an intertransaction association rule $X \Rightarrow Y$ are defined as*[1]

$$support = \frac{|T_{xy}|}{S}, \quad confidence = \frac{|T_{xy}|}{|T_x|}.$$

Given a minimum support level $minsup$ and a minimum confidence level $minconf$, our task is to mine the *complete* set of intertransaction association rules from the transaction database with $support \geq minsup$ and $confidence \geq minconf$. We illustrate the concepts with an example.

Example 2.2. Let $minsup$ and $minconf$ be 40 percent and 60 percent respectively. Then, an example of an intertransaction association rule which will be mined from the database in Fig. 1 will be:

$$a(0), e(0) \Rightarrow c(3).$$

This rule has a support of 40 percent and a confidence of 66 percent.

3 Principles of Intertransaction Association Mining Methods

Like classical association rule mining, the problem of mining intertransaction association rules can be decomposed into two subproblems:

1. Find all intertransaction itemsets with support higher than $minsup$. We call these itemsets **frequent intertransaction itemsets**.
2. For every frequent intertransaction itemset F and for all possible combination of $X \subset F$, output a rule $X \Rightarrow (F - X)$ if all of the following conditions are true:
   a. $\exists e_i(0) \in X, 1 \leq i \leq u$.
   b. $\exists e_i(j) \in (F - X), 1 \leq i \leq u, j \neq 0$.
   c. Confidence of $X \Rightarrow (F - X)$ is higher than $minconf$.

1. Notice that in our definition, we also count the last several transactions in sequence which do not cover the full sliding window, i.e., those not containing the full maxspan of the sliding window.

Of the two subproblems, subproblem one is of major concern as it is the bottleneck of the whole mining process. This is mainly due to the fact that the number of extended-items in $\sum'$ is w times the number of items in $\sum$, as such more candidate intertransaction itemsets are expected to be generated. This will result in higher processing time being spent on computing the support of candidate intertransaction itemsets.

Compared to subproblem one, subproblem two takes up much less processing time and could be solved easily by making minor modifications to a fast algorithm in [2]. As such, we will not discuss subproblem two further but instead focus our discussion on two algorithms, EH-Apriori and FITI, which are derived for solving subproblem one.

In this section, the general concepts of the two algorithmsi will be provided.

3.1 EH-Apriori

With the existence of a megatransaction within each sliding window, a tempting solution for discovering frequent intertransaction itemsets is to extend the Apriori algorithm [2] to mine a database of megatransactions. As the Apriori algorithm assumes the existence of lexicographic order among the extended-items, the extended-items in the megatransaction will be ordered using Definition 3.1.

Definition 3.1. *Let $e_i(d_i)$ and $e_j(d_j)$ be two extended-items in a megatransaction. We say that $e_i(d_i) < e_j(d_j)$ if either of the following two conditions holds:*

1. $d_i < d_j$.
2. $d_i = d_j$ *and* $e_i < e_j$.

Otherwise, we say that $e_j(d_j) < e_i(d_i)$.

Using this definition, we are able to use the Apriori algorithm to discover frequent intertransaction itemsets. To enhance the efficiency further, a hashing technique similar to the one in [19] is used. The general idea of the algorithm is outlined as follows:

When the support of candidate intertransaction one-itemsets is counted by scanning the database, information about candidate intertransaction two-itemsets is collected in advance in such a way that all the possible two-itemsets are hashed to a hash table. Each bucket in the hash table consists of a number to represent how many itemsets have been hashed to this bucket so far. The hash table is then used to reduce the number of candidate intertransaction two-itemsets. This is done by removing a candidate two-itemsets if its corresponding bucket value in the hash table is less than $minsup$. We call this algorithm the *Extended Hash Apriori* or *EH-Apriori* [11].

3.2 FITI Algorithm

Unlike EH-Apriori which is modified from the Apriori algorithm, FITI is an algorithm designed specifically for discovering frequent intertransaction itemsets. FITI makes use of the following property to enhance its efficiency in discovering frequent intertransaction itemsets.

Property 3.1. *Let F be a frequent intertransaction itemset. Let $A_i = \{e_j | 1 \leq j \leq u, e_j(i) \in F\}$, where $0 \leq i \leq (w-1)$. For all i, $0 \leq i \leq (w-1)$, and A_i must be a frequent intratransaction itemset.*

Proof. We will prove this property by contradiction. Let F be a frequent intertransaction itemset. Let

$$A_i = \{e_j | 1 \leq j \leq u, e_j(i) \in F\}, 0 \leq i \leq (w-1).$$

Assume that $\exists A_i$, such that A_i is not a frequent intratransaction itemset. We denote the support of F as $support(F)$ and support of A_i as $support(A_i)$. Since F is a frequent intertransaction itemset, $support(F) \geq minsup$. Also, since A_i is not a frequent intratransaction itemset, then $support(A_i) < minsup$. Hence, we have

$$support(F) > support(A_i).$$

However, we know that for any sliding window W that contains F, A_i will occur in $W[i]$ and each $W[i]$ refers to a different transaction for different W. We can thus conclude that $support(A_i) \geq support(F)$ giving a contradiction and thus proving that Property 3.1 holds. □

Property 3.1 is an important property as it provides a different view of mining frequent intertransaction itemsets. Instead of viewing mining as an attempt to identify frequently occurring patterns formed from the extended-items, we can view it as an attempt to discover frequently occurring patterns formed from frequent intratransaction itemsets. As such in FITI, frequent intratransaction itemsets are first discovered and then frequent intertransaction itemsets are formed from them. This gives rise to the name of FITI which stands for *First Intra Then Inter*.

In FITI, frequent intratransaction itemsets that are discovered in the initial phase are stored in a data structure designed to facilitate the mining of frequent intertransaction itemsets in the later phase. Each frequent intratransaction itemset is given a unique number called an **ID**. By using this ID as an index into the data structure, FITI is able to gather information on intratransaction itemsets quickly. To avoid the need to regenerate frequent intratransaction itemsets during the discovery of frequent intertransaction itemsets, the original database is transformed into another database that stores the IDs of frequent intratransaction itemsets presented in each transaction of the original database. When mining frequent intertransaction itemsets, each intertransaction itemset is represented as a tuple of w IDs. Using this encoding, we formulate two types of joins to generate candidates intertransaction $(k+1)$-itemsets from two existing frequent intertransaction k-itemsets. We will have a close look at FITI in the next section.

4 The FITI Algorithm

In this section, we will first present the FITI algorithm step by step after which we will prove its correctness by induction. In general, FITI consists of the following three phases.

1. Phase I: Mining and Storing Frequent IntraTransaction Itemsets
2. Phase II: Database Transformation
3. Phase III: Mining Frequent InterTransaction Itemsets

The following presents the detailed method phase by phase.

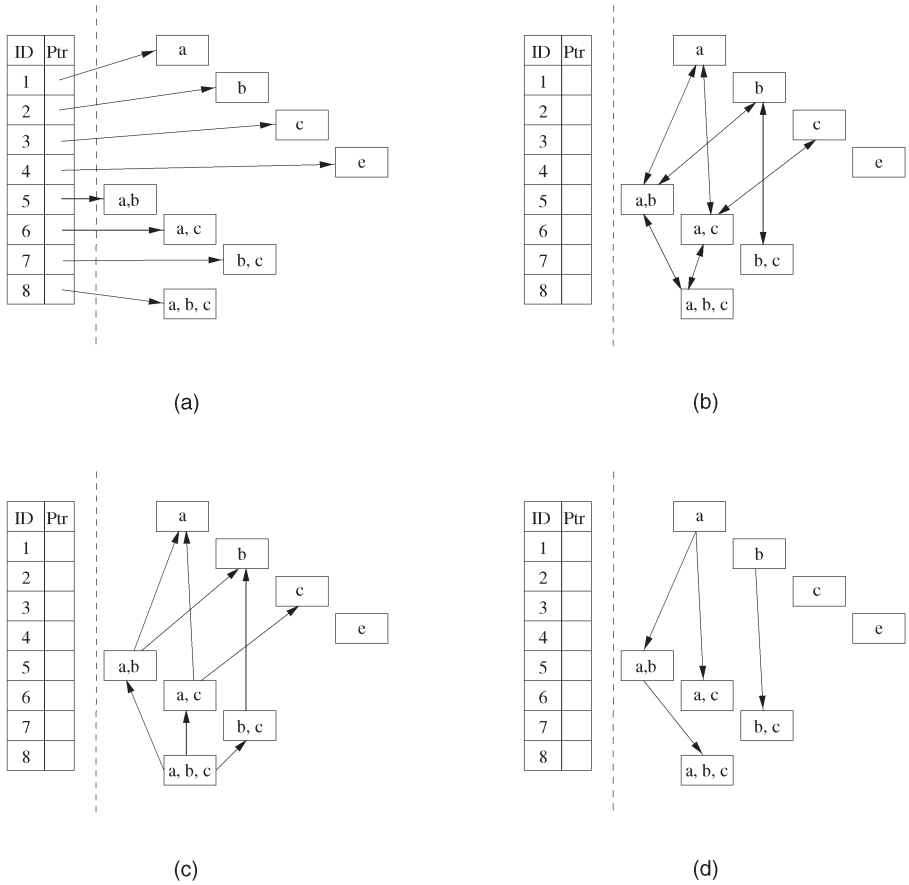


Fig. 2. Data structure for Storing Intratransaction itemsets. (a) Lookup links, (b) generator and extension links, (c) subset links, and (d) descendant links.

4.1 Phase I: Mining and Storing Frequent IntratTransaction Itemsets

In this phase, frequent intratransaction itemsets are first mined using the Apriori algorithm and then stored in a data structure, called **Frequent-Itemsets Linked Table**, or simply **FILT**. Note that there are many fast algorithms developed for discovering frequent intratransaction itemsets after the initial proposal of Apriori [2]. These algorithms can be applied in this step as well.

Since only the frequent intratransaction itemsets are stored in the FILT, the space needed is much smaller than storing all the candidate itemsets. The data structure consists of an *ItemSet Hash Table*, with nodes linked by several kinds of links. To describe clearly these links in FILT, we depict each kind of links separately in Fig. 2.

The following four kinds of links are built in FILT.

1. **Lookup Links**. Each frequent intratransaction itemset is assigned a unique **ID** number that corresponds to a row number in the *ItemSet Hash Table*. Each itemset is stored in a node pointed to by a **lookup link** from the corresponding row in the table, as illustrated in Example 4.1.
   Example 4.1. Let
   $$\{a\}, \{b\}, \{c\}, \{e\}, \{a, b\}, \{a, c\}, \{b, c\},$$
   and $\{a, b, c\}$ be the frequent intratransaction itemsets derived by Apriori. Each is then inserted into FILT with the node pointed to by a corresponding lookup link as shown in Fig. 2a. In the figure, the ID of $\{a\}$ is 1 and the ID of $\{a, b, c\}$ is 8.
2. **Generator and Extension Links**. Given a node N_F that contains an intratransaction k-itemset F, the generator links of N_F point to the two $(k-1)$-itemsets that are combined to form F in the Apriori algorithm. For example, in Fig. 2b, the itemset $\{a, b, c\}$ has its two generator links pointing to $\{a, b\}$ and $\{a, c\}$, respectively. On the other hand, since $\{a, b\}$ and $\{a, c\}$ are combined to form $\{a, b, c\}$, both of them will have an extension link pointing to $\{a, b, c\}$. Because of the nature of generator and extension links, they are depicted in the same diagram and the generator/extension relationship is represented by a bidirectional arrow.
3. **Subset Links**. Given a node N_F that contains an intratransaction k-itemset F, the subset links of N_F point to all subsets of F with size $k-1$. For example, the subset links of $\{a, b, c\}$ point to $\{a, b\}, \{a, c\}$, and $\{b, c\}$ in Fig. 2c.
4. **Descendant Links**. As mentioned earlier, FILT is composed of an array and a hash-tree. Given a node N_F that contains an intratransaction k-itemset F, the

TABLE 1
A Database with Four Transactions

d	E
100	a, b, c
104	a, b, c, d, e
105	a, f, g
109	e, h

TABLE 2
A Set of FIT Tables

F_1		F_2		F_3	
d_i	$IDset_i$	d_i	$IDset_i$	d_i	$IDset_i$
100	1, 2, 3	100	5, 6, 7	100	8
104	1, 2, 3, 4	104	5, 6, 7	104	8
105	1	105		105	
109	4	109		109	

descendant links of N_F point to all of its descendants in the hash-tree. If $F = \{e_1, \ldots, e_k\}$, then its descendants will be a set of $(k+1)$-itemsets of the form $\{e_1, \ldots, e_k, e_{k+1}\}$. For example, in Fig. 2d, the descendants of $\{a\}$ will be $\{a, b\}$ and $\{a, c\}$. Note that, unlike the subset links, the descendant links of a node points to other nodes that share a common suffix with it.

In the following sections, we will see how FILT is used in the database transformation and in the mining of intertransaction itemsets in Phases II and III, respectively.

4.2 Phase II: Database Transformation

After forming the data structure FILT in Fig. 2, the next step of FITI is to transform the database into a set of *encoded Frequent-Itemset Tables*, (called **FIT** tables). We have in total $\mathbf{max_k}$ FIT tables, $\{F_1, \ldots, F_{max_k}\}$, where max_k is the maximum size of the intratransaction itemsets discovered in Phase I. Each table F_k will be of the form $\{d_i, IDset_i\}$, where d_i is the value of the dimensional attribute and $IDset_i$ is the IDs of frequent k-itemsets that are found in the transaction. We will show how a database is transformed in Example 4.2.

Example 4.2. Consider a database with four transactions in Table 1:

Assume a value of 50 percent is set for $minsup$. We will discover the same frequent intratransaction itemsets as in Example 4.1 and max_k will have a value of three because we have a three-itemset $\{a, b, c\}$. This gives rise to three FIT tables, F_1, F_2, and F_3. If itemsets are assigned the same IDs as in Example 4.1, the contents of the three FITs will be as follows:

Algorithm 4.1 describes the step for performing database transformation. This database transformation avoids the reexamination of database T and recomputation of frequent intratransaction itemsets in our future computation. By storing these itemsets in different FIT tables according to their sizes, we can avoid unnecessary I/Os since a table F_k is only read if an intratransaction itemset of size k is found in one of the candidate intertransaction itemsets.

Algorithm 4.1 Database Transformation Algorithm.
Input: A transaction database T and the frequent-itemsets linked table (FILT) H generated from T.
Output: A set of FIT tables: $F_1, \ldots, F_{max_k}$.
Method: The transaction database T is transformed into a set of frequent itemset files, based on the algorithm shown in Table 2.

```
void Tranform() {
  while (!feof(T)) {
    read next transaction T_i;
    write d_i to all F_j;
    Subset(T_i, 1, 1, 0) }
}
void Subset(T_i, index, k, and NodeID) {
  if (k == 1) {
    for each item e_j in T_i {
      search Itemtable for ID of {e_j};
      if (found) {
          let nowID be ID found;
          write nowID to F_1;
          for each item e_m, m > j in T_i {
            search childs of nowID for an itemset I that
            contains e_m;
            if (found) {
                let nextNode be the ID of I;
                Subset(T_i, m + 1, k + 1, nextNode); }
          }
        }
      }
      return;
  }
  else {
    write NodeID to F_k;
    for each item e_m, m >= index in T_i {
      search childs of NodeID for an itemset I that
      contains e_m;
      if (found) {
          let nextNode be the ID of I;
          Subset(T_i, m + 1, k + 1, nextNode); }
    }
    return;
  }
}
```

4.3 Phase III: Mining Frequent Intertransaction Itemsets

After the database transformation, we proceed to Phase III: mining frequent intertransaction itemsets.

Intertransactions itemsets are represented by their *ID encoding* defined below.

Definition 4.1. *Let F be an intertransaction itemset. Let $A_i = \{e_j | 1 \le j \le u, e_j(i) \in F\}$, where $0 \le i \le (w-1)$. The* ***ID encoding*** *of F is of the form $I = \{I_0, \ldots, I_{w-1}\}$, where I_i is the ID of A_i if $|A_i| > 0$, and zero otherwise.*

Example 4.3. Let $F = \{a(0)\ a(1)\ b(1)\ c(3)\}$ be an intertransaction itemset and let the maxspan be four. Then, the ID encoding of F will be $I = \{1, 5, 0, 3\}$ according to Fig. 2.

The mining algorithm follows the Apriori principle [2] to perform level-wise mining, i.e., using intertransaction frequent k-itemsets (for $k \geq 2$) to form candidate frequent $(k+1)$-itemsets. Moreover, if a candidate $(k+1)$-itemset does not have its corresponding intratransaction itemset in the FIT table F_j, it will be removed automatically. The method is presented in Algorithm 4.2. Notice that the algorithm must run on top of an *input layer* which provides a "sliding window" view to the mining algorithm.

Algorithm 4.2 Level-Wise Mining of Frequent InterTransaction Itemsets.
Input: A set of FIT tables: $F_1, \ldots, F_{max_k}$, and the minimum support threshold: $minsup$.
Output: The complete set of frequent intertransaction itemsets.
Method: The method follows the Apriori principle and performs level-wise mining of frequent intertransaction itemsets in the FIT tables, using the slide window size w. The outline is presented below, whereas the details are explained in the following subsections.

```
Generate frequent intertransaction 2-itemsets, L_2;
k=3;
while (L_{k-1} ≠ ∅) {
    Generate candidate intertransaction k-itemsets, C_k;
    Scan transformed database to update the count for C_k;
    Let L_k = {c ∈ C_k | support(c) ≥ minsup};
    k++;
}
```

The discovery of frequent intertransaction itemsets is the performance bottleneck of the whole mining process. We will look at how we tackle this bottleneck in more detail. In Section 4.3.1, the input layer will first be described after which we will see how frequent intertransaction two-itemsets are mined in Section 4.3.2. The generation of candidate intertransaction k-itemsets (for $k > 2$) is discussed in Section 4.3.3, and the counting function is explained in Section 4.3.4.

4.3.1 The Input Layer

The input layer provides a "sliding window" view to the mining algorithm and helps optimize the mining in two ways.

First, it limits its input to a set of essential F_k's instead of reading the whole set of FITs. For example, when we are counting support for candidate intertransaction two-itemsets, only F_1 need to be accessed as no frequent intratransaction itemsets with size more than one will be found in any intertransaction two-itemset. Similarly, as the size of the candidate intertransaction itemsets grows, it may come to a stage where no frequent intratransaction one-itemsets are involved in any candidate intertransaction itemsets. In such a case, the input layer will not access F_1 when forming the sliding window.

Second, as the sliding window is moved along the dimension D, new transactions that move into the sliding window will be filtered to remove the IDs of frequent intratransaction itemsets that are not present in any candidate intertransaction itemsets. This will help to reduce the time required to process a sliding window when counting support for candidate intertransaction itemsets.

4.3.2 Generating Frequent Intertransaction Two-Itemsets

Like EH-Apriori, FITI makes use of the hashing approach introduced in [19] to prune off candidate intertransaction itemsets. FITI, however, has one major difference from the approach in [19]. It updates the bucket values only after mining frequent intratransaction itemsets. Because of this, FITI will already have knowledge of what are the frequent one-itemset. As such, it generates fewer candidate itemsets because there are fewer false count contributed by infrequent candidate two-itemsets.

The hash function that we used in FITI is as below.

Function 4.1. *Given a candidate itemset $I = \{I_0, I_1, \ldots, I_{w-1}\}$, its bucket number B will be computed using the following formula:*

$$B = ((I_0 \times w + m) \times N_1 + I_m)\ mod\ Size,$$

where N_1 is the number of frequent one-itemsets discovered, $Size$ is the size of the hash-table, $m \neq 0$, and $I_m \neq 0$.

The idea behind such a hash function is simple. Each intertransaction two-itemsets can be represented by a three-tuple, $\langle I_0, I_m, m\rangle$. If we use a three-dimensional array $M[1..N_1][1..N_1][1..w]$ to store these tuples, then each cell in M will contain only one tuple if we use $M[I_0][I_m][m]$ to store the tuple $\langle I_0, I_m, m\rangle$. However, because we are using a one-dimensional hash-table, we need to map a cell $M[I_0][I_m][m]$ from the three-dimensional array into the hash-table by computing the corresponding row number of the hash table as $(I_0 \times w + m) \times N_1 + I_m$. To cater for the possibility that the size of the hash table might be smaller than $w \times N_1^2$, we took the modulus on $Size$ to wrap the mapping to the top of the table.

Because of this, Function 4.1 has the desirable property that only one candidate itemset will be hashed to one hash basket if $Size \geq w \times N_1^2$. If $Size$ is less than $(w-1) \times N_1^2$, the maximum number of candidate two-itemsets that will hash to the same basket will increase proportionally with respect to $\frac{(w-1)\times N_1^2}{Size}$.

4.3.3 Generation of Candidate Frequent Intertransaction k-Itemsets for $k > 2$

Now, we examine how to generate candidate frequent intertransaction k-itemsets for $k > 2$. This process is similar to its intratransaction mining counterpart, i.e., the generation of candidate frequent intratransaction k-itemsets for $k > 2$ in the Apriori algorithm [2]. First, the frequent $(k-1)$-itemsets are joined to form a set of k-itemsets. Then, we check whether all the subsets of every so generated k-itemset are frequent. Those which can pass the test form the set of candidate frequent intertransaction k-itemsets. Finally, their counts can be collected by scanning the relevant FIT tables once.

To join two frequent intratransaction itemsets, two kinds of joins can be performed to create new candidate

itemsets: *intratransactions join* and *cross-transactions join*. We will examine them in detail giving examples to illustrate our points. In these examples, we will assume that IDs are assigned to frequent intertransaction itemsets as in Example 4.1.

Definition 4.2. *Let* $I = \{I_0, I_1, \ldots, I_{w-1}\}$ *and*

$$J = \{J_0, J_1, \ldots, I_{w-1}\}$$

be two frequent intratransaction itemsets. **Intratransactions join** *can be performed on* I *and* J *if and only if both of the following conditions are true.*

1. *There exists a* p *such that* I_p *and* J_p *are the generators of a frequent intratransaction itemset* K_p *(we can make use of hash-tree in Fig. 1 to check for this condition).*
2. *For all* q, $q \neq p$, $I_q = J_q$.

If I *and* J *can be intratransaction joined, the resulting candidate itemset will be* $K = \{K_0, \ldots, K_p, \ldots, K_{w-1}\}$, *where* $K_q = I_q = J_q$ *for* $p \neq q$.

Example 4.4 shows how two intertransaction itemsets are intratransaction joined.

Example 4.4. Consider two itemsets $I = \{1, 0, 5, 0, 2\}$ and $J = \{1, 0, 6, 0, 2\}$. Condition 1 of the above definition will be satisfied because I_2 and J_2 are the generators of a frequent intratransaction itemset, with ID = 8 according to Example 4.1 with the itemsets shown in Fig. 2. Condition 2 in the definition is also satisfied because all other subwindows of I and J are equivalent. As such, I and J can be intratransaction joined to produce $\{1, 0, 8, 0, 2\}$.

Definition 4.3. *Let* $I = \{I_0, I_1, \ldots, I_{w-1}\}$ *and*

$$J = \{J_0, J_1, \ldots, J_{w-1}\}$$

be two frequent intratransaction itemsets. **Cross-transaction join** *can be performed for* I *and* J *if and only if all of the following conditions are true:*

1. *There exists a* p *such that* $I_p \neq 0$ *and* $J_p = 0$.
2. *There exists a* q, $q \neq p$ *such that* $I_q = 0$ *and* $J_q \neq 0$.
3. *For all* r, $r \neq p$, $r \neq q$, $I_r = J_r$.
4. I_p *and* J_q *are the last nonzero subwindows in* I *and* J, *respectively.*
5. *All subwindows of* I *and* J *either contain no itemset or only frequent intratransaction one-itemsets.*

If the above conditions are satisfied, the resulting candidate itemsets will be $K = \{K_0, \ldots, K_p, \ldots, K_q, \ldots, K_{w-1}\}$, *where* $K_p = I_p$, $K_q = J_q$, *and for all* r, $r \neq p$, $r \neq q$, *and* $K_r = I_r = J_r$.

The following example illustrates a cross-transaction join.

Example 4.5. Consider two itemsets $I = \{1, 0, 2, 1, 0\}$ and $J = \{1, 0, 2, 0, 4\}$. Condition 1 is satisfied because we have $I_3 \neq 0$ and $J_3 = 0$. The fact that $I_4 = 0$ and $J_4 \neq 0$ satisfies Condition 2. As all other subwindows are equivalent, Condition 3 is satisfied. Condition 4 is met because I_3 and J_4 are the last nonzero subwindows in I and J, respectively. Since each subwindow either contains one or no items, Condition 5 is also satisfied and, thus, we can cross join I and J to produce $\{1, 0, 2, 1, 4\}$.

Condition 4 for cross-transaction join ensures that the same candidate itemset will not be generated twice by two different intertransaction joins while Condition 5 for cross-transaction join ensures that the same candidate set will not be generated twice by both intratransaction and cross-transaction join. We will illustrate the need for the two conditions in Example 4.6.

Example 4.6. Let $I = \{1, 0, 4, 0, 1\}$ and

$$J = \{1, 0, 0, 2, 1\}.$$

If we had no Condition 4, then I and J could be cross-transaction joined to formed $\{1, 0, 4, 2, 1\}$. However, the intertransaction itemset $\{1, 0, 4, 2, 1\}$ can also be obtained by cross-transaction joining $\{1, 0, 4, 0, 1\}$ and $\{1, 0, 4, 2, 0\}$.

Similarly, if we let $I = \{1, 0, 5, 0, 1\}$ and

$$J = \{1, 0, 5, 2, 0\},$$

then I and J can be cross-transaction joined to form $\{1, 0, 5, 2, 1\}$ if we have no Condition 5. However, the intertransaction itemset $\{1, 0, 5, 2, 1\}$ can also be obtained by joining $\{1, 0, 1, 0, 1\}$ and $\{1, 0, 2, 0, 1\}$. To avoid this redundancy, we introduce Conditions 4 and 5.

To implement the defined joins, we first observe the following property:

Property 4.1. *If two itemsets* I *and* J *can be intratransaction or cross-transaction joined, then they will become an equivalent itemset if each of them has a relevant subwindow set to zero.*

We will use Example 4.7 to illustrate Property 4.1.

Example 4.7. We go back to Example 4.5 where $I = \{1, 0, 5, 0, 1\}$ and $J = \{1, 0, 6, 0, 1\}$ can be intratransaction joined. If we set the third subwindow of I and third subwindow of J to zero, then both I and J will be equivalent to $\{1, 0, 0, 0, 1\}$. As another example, we consider Example 4.6 in which $I = \{1, 0, 2, 1, 0\}$ and $J = \{1, 0, 2, 0, 4\}$ can be cross-transaction joined. If we set the fourth subwindow of I to zero and the fifth subwindow of J to zero, then both I and J will be equivalent to $\{1, 0, 2, 0, 0\}$.

With this observation, we came up with the hash-table method for generating candidate $(k+1)$-itemsets from frequent intertransaction k-itemsets. Each hash bucket in the table points to a linked list of pointers and each of the pointers points to a frequent intertransaction k-itemset that is hashed to the hash bucket. A k-itemsets will be hashed multiple times, each time with one of its subwindow being set to zero. To avoid unnecessary processing, subwindows that originally have a value of zero will not be hashed. This is because no new candidate will be generated by going through the linked list in such a case.

Function 4.2. *Given a frequent intertransaction itemset* $I = \{I_0, I_1, \ldots, I_{w-1}\}$ *(one subwindow is already set to zero), our hash function* $h(I)$ *is computed as*

$$h(I) = \sum_{p=0}^{w-1} I_p \times TotalItem^{w-p} \; mod \; Hsize,$$

where $TotalItem$ is the total number of frequent intraitemset mined, and $Hsize$ is the size of the hash-table.

This hash function essentially treats I as a number in base $TotalItem$, converts it into its decimal form, and then finds the remainder when it is divided by $Hsize$.

When a k-itemset is hashed to a particular linked list, FITI will go through the linked list to check for valid conditions for intratransaction join or cross-transaction join. If the conditions are valid, a join will take place to produce a new candidate itemset. When the end of the list is reached, a pointer to the itemset that is being hashed will be inserted. To enhance efficiency, we do not check for cross-transaction join for $k > w$. This is because Condition 4 for cross-transaction join will definitely not be satisfied for $k > w$.

To further prune off candidate itemsets, we delete all the candidate $(k+1)$-itemsets which have some k-subsets that are not frequent k-itemsets. The generations of the k-subsets are done using the data structure shown in Fig. 1, by using the following algorithm.

Algorithm 4.3 *Generation of all the* (k-itemset) *subsets of an intertransaction* $(k+1)$-itemset.
Input: *An intertransaction* $(k+1)$-itemset $I = \{I_0, \ldots, I_{w-1}\}$.
Output: *All of the* (k-itemset) *subsets of* I.
Method: *Notice that the FILT structure is used in the generation although the general idea is similar to* [2].

```
let S be the set of k-subsets of I;
S = {};
for (p = 0; p < w; p++) {
    if (I_p != 0) {
        if (I_p is an intratransaction one-itemsets) {
            if (p!=0)
                add {I_0, ..., I_{p-1}, 0, ..I_{w-1}} to S
            else
                add {I_1, .., I_{w-1}, 0} to S
        } else
        { let I_p be an intratransaction h-itemsets, h > 1
          // Note: Hash-tree in Fig. 1 is used here
            for each (h-1)-subset of I_p
            { let t be the ID of the (h-1)-subset
              add {I_0, ..., I_{p-1}, t, ..I_{w-1}} to S
            }
        }
        }
}
return S;
```

4.3.4 The Counting Function

Having generated the candidate itemsets, the next step is to access the FIT tables to determine the number of occurrences of each candidate itemset in the database. To facilitate efficient counting, candidate itemsets are first inserted into a hash-tree. A node of the hash-tree is either an *interior* node or a *leaf* node. The internal node consists of branches that point to other nodes, whereas the leaf nodes consist of pointers to candidate intertransaction itemsets. The hash-tree has a depth of exactly w levels and the root node is defined to be at level 1. When an intertransaction itemset $I = \{I_0, \ldots, I_{w-1}\}$ is hashed, we start from the root and go down the tree until the leaf is reached. At an interior node at depth d, we will use binary search to look for a branch for I_{d+1}. If it is not found, a branch will be created and inserted at the appropriate position such that the branches in the node are kept in sorted order of the IDs. When a leaf node is reached, a pointer to the itemset will be inserted while keeping the pointers in sorted order of I_{w-1}.

Starting from the root node, the counting function increases the count for all the candidates contained in a sliding window W as follows: If we are at an interior node of depth d and there exists a branch corresponding to ID = 0, move down that branch. For the remaining branches, we move down the branch only if the corresponding ID of a branch can be found in $W[d]$ of the sliding window. This procedure is done recursively to the nodes which the branches point to. When we reach a leaf node, we will check for pointers such that their corresponding IDs are found in $W[w-1]$. The count of candidate itemsets which are point to by these pointers will then be increased.

After the FIT tables are scanned, candidate itemsets that have a lower support level than $minsup$ will be pruned off. The remaining candidates are frequent itemsets and they will be used to generate the next batch of candidate itemsets. This process repeats until all the candidate itemsets are pruned off at the end of the counting function.

4.4 Correctness of FITI

To prove the correctness of FITI, we need to prove the following lemma.

Lemma 4.1. *Let $minsup$ and w denote the minimum support level and maxspan for mining frequent intertransaction itemsets in a transaction database T, respectively. If L_n is the set of frequent intertransaction k-itemsets that exists in T and L'_n is the set of intertransaction k-itemsets discovered by FITI, we claim that $L'_n = L_n$ for $n \geq 1$.*

Proof. We will prove this claim by induction. Let P_n be the statement that "$L'_n = L_n$ for all $n \geq 1$." We know that P_1 is true since L'_1 is generated by the Apriori algorithm which counts the support of all items in T. We also know that P_2 is true as L'_2 is generated from L'_1 by generating all the combinations of intertransaction 2-itemsets from items in L'_1 and counting support for these combinations.

Let us assume that P_k is true. When $n = k+1$, let C'_{k+1} be the set of candidate intertransaction itemsets generated from L_k by either cross-transaction join or intertransaction join. We first prove $L_{k+1} \subseteq C'_{k+1}$ by contradiction.

Let us first assume that $L_{k+1} \not\subseteq C_{k+1}$. This will imply that $\exists K \in L_{k+1}, where K \notin C_{k+1}$. Let the ID encoding of K be $\{K_0, \ldots, K_{w-1}\}$. We consider the following two cases:

Case 1: There exists a K_i in K which corresponds to an frequent intratransaction itemset of size greater than two. Let I_i and I_j be the two generators of K_i and let the two intertransaction itemsets I and J be $\{K_0, ..., I_i, .., I_{w-1}\}$ and $\{K_0, ..., J_j, .., K_{w-1}\}$, respectively.

Case 2: The subwindows of K either contain intratransaction one-itemsets or no items.

TABLE 3
Parameters

Parameter	Meaning
$\|D\|$	number of transactions
$\|T\|$	average size of the transactions
$\|L\|$	number of potentially frequent inter-transaction itemsets
$max(\|L\|)$	maximum size of the transactions
$\|I\|$	average size of the potentially frequent inter-transaction itemsets
$max(\|I\|)$	maximum size of the potentially frequent inter-transaction itemsets
$\|\sum\|$	number of items
R	maximum interval of itemsets

Let K_i and K_j be the last two nonzero subwindows in K and let the two intertransaction k-itemsets I and J be $\{K_0, ..., K_i, 0, 0, ..\}$ and $\{K_0, ..., K_j, 0, 0...\}$, respectively.

For both cases, I and J are subsets of K and are of size k. As such, it must be that $I \in L_k$ and $J \in L_k$. Hence, we can conclude that $I \in L'_k$ and $J \in L'_k$ since we assume P_k is true. For Case 1, $K \in C'_{k+1}$ because we can perform an intratransaction join on I and J. By performing a cross-transaction join on I and J in Case 2, we will also have $K \in C'_{k+1}$. We thus have a contradiction to the original assumption that $L_{k+1} \not\subseteq C'_{k+1}$ and thus conclude that $L_{k+1} \subseteq C'_{k+1}$.

Since $L_{k+1} \subseteq C'_{k+1}$ and $L'_{k=1}$ is obtained by pruning off members of C'_{k+1} with support less than $minsup$. We can conclude that P_{k+1} is true. Since P_1, P_2 are true and $P_k \Rightarrow P_{k+1}$ is true, P_n is true by induction. □

Since $L'_n = L_n$ for all n and FITI outputs $\cup L'_n$ for all n as the set of frequent intertransaction itemsets, we can conclude that FITI outputs the correct set of frequent intertransaction itemsets in a transaction database.

5 Performance Study

To assess the performance of the proposed algorithms, experiments were conducted using synthetic data.

5.1 Generation of Synthetic Data

The method used by this study to generate synthetic transactions is similar to the one used in [2] with some modifications noted below. Table 3 summarizes the parameters used.

Transaction sizes are typically clustered around a mean, with a small portion of transactions having many items. Typical sizes of intertransaction itemsets are also clustered around a mean, with a small portion of frequent intertransaction itemsets having a large number of items across different transactions.

We first generate a set L of the potentially frequent intertransaction itemsets, which may span across different transactions, e.g., $\{a(0), b(1), c(2)\}$, and then assign a frequent intertransaction itemset from L to corresponding transactions.

The number of potentially frequent intertransaction itemsets is set to $|L|$. A potentially frequent intertransaction itemset is generated by first picking the size of the itemset from a Poisson distribution with mean equal to $|I|$. The maximum size of potentially frequent intertransaction itemsets is $|max(|I|)|$. Items and their intervals in the first frequent intertransaction itemset are chosen randomly, where item is picked up from one to $|\sum|$, and its interval is picked up from zero to $R-1$. To model the phenomenon that frequent intertransaction itemsets often have common items and intervals, some fraction of items and their intervals in subsequent itemsets are chosen from the previous itemsets generated. We use an exponentially distributed random variable with mean equal to the *correlation level* to decide this fraction for each itemset. The remaining items and their intervals are picked at random. In the data sets used in the experiments, the correlation level is set to 0.5. After generating all the items and intervals for a frequent intertransaction itemset, we revise each of its intervals by subtracting the minimum interval value of this frequent itemset. In this way, the minimum interval of each potentially frequent intertransaction itemset is always 0.

After generating the set L of potentially frequent intertransaction itemsets, we generate transactions in the database. Each transaction is assigned a series of potentially frequent intertransaction itemsets. However, upon the generation of one transaction, we actually need to consider a list of consecutive ones starting from this transaction, as items in a frequent itemset may span across different transactions. For example, after selecting the frequent intertransaction itemset $\{a(0), b(1), c(2)\}$ for current transaction T_c, we should assign item a to T_c, item b to its next transaction T_{c+1}, and item c to T_{c+2}.

Before assigning items to a list of consecutive transactions, we should also determine the sizes of these transactions first. The size of each transaction is picked from a Poisson distribution with mean equal to $|T|$. The maximum size of transactions is $max(|T|)$. Each potentially frequent intertransaction itemset has a weight associated with it, which corresponds to the probability that this itemset will be picked. This weight is picked from an exponential distribution with unit mean, and is then normalized so that the sum of the weights for all the itemsets in L is one. The next itemset to be put in the transaction is chosen from L by tossing an $|L|$-sided weighted coin, where the weight for a side is the probability of picking the associated itemset.

If the frequent intertransaction itemset picked on hand does not fit in the current or any one of its successive transactions, this itemset is put in these transactions anyway in half the cases, and the itemset enters an *unfit* queue for the next transaction in the rest of the cases. Each

TABLE 4
Parameter Settings

Parameter	Data Set 1	data Set 2
$\|D\|$	10000	1000
$\|T\|$	5	100
$max(\|T\|)$	10	200
$\|L\|$	1000	1000
$\|I\|$	5	6
$max(\|I\|)$	10	10
$\|\sum\|$	500	500
R	4	6

time, we pick itemsets from this queue first, according to the first-in-first-out principle. Only when the queue is empty do we perform random selection from the set L. Similar to [2], we also use a corruption level during the transaction generation to model the phenomenon that all the items in a frequent itemset do not always occur together. This corruption level for an itemset is fixed and is obtained from a normal distribution with mean 0.5 and variance 0.1.

Using the method above, we generated two sets of synthetic data using the parameters shown in Table 4. The first data set has more transactions but less items in each transaction. Data set two, on the other hand, has only 1,000 transactions but each transaction contains 100 items on average. Later in this section, we will study the performance of our algorithms by varying one parameter of these data sets while keeping other parameters at the default values shown in the table.

5.2 Relative Performance

To compare the performance of EH-Apriori and FITI, we vary the support level and run the two algorithms on data sets one and two with maxspan equal to four and six intervals, respectively. In general, Phase I and II of the FITI algorithm account for five percent to 10 percent of the running time for FITI. The results are shown in Table 5. We plot the running time of EH-Apriori and FITI against the support level and the graphs are shown in Fig. 3a and Fig. 3b for data set 1 and 2, respectively.

As clearly shown in the graphs, FITI outperforms EH-Apriori by a large margin for both data sets. This is not surprising because FITI avoids a large amount of recomputation by making use of FILT and the FIT tables.

We also observed that the running time of both the algorithm is much higher for data set two. This is due to the large number of items in each transaction of data set two and this causes a large number of combinations to be generated when we try to mine patterns across a few transactions.

As for the memory requirement of the algorithms, FITI takes up a memory size of 20K and 8K for data set one and two, respectively, when the lowest support levels of 0.1 percent and 11 percent are used. Note that we use the lowest support levels here since they result in the maximum memory requirement. For the same support levels, EH-Apriori require 17K and 7K on data set one and two. EH-Apriori has a lower memory requirement because it does not stored a FILT structure which take up 8K and 3K in FITI. Since FITI is substantially faster than EH-Apriori as a result a small increase in memory usage, we feel that FITI is still the better algorithm for mining intertransaction association rules. The remaining experiments are performed only on FITI due to the high running time of EH-Apriori.

5.3 Effect of Increasing Maxspan

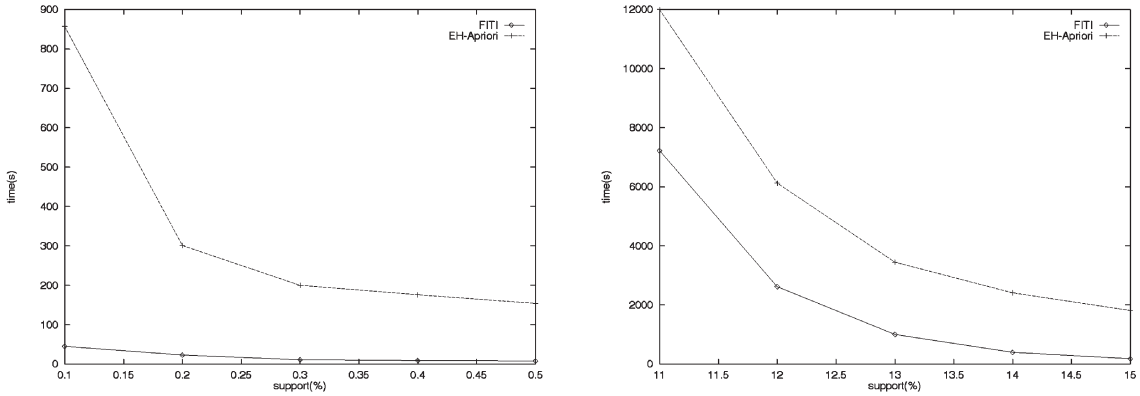
To investigate the effect of increasing maxspan, we vary maxspan from two to eight for both data sets. The minimum support level for data set one is 0.1 percent, whereas the minimum support level for data set two is 15 percent. The results are shown in Fig. 4. Both the graphs can be separated into two regions, $\text{maxspan} \leq R$, and $\text{maxspan} > R$. For $\text{maxspan} \leq R$, the graphs curve upwards because the number of frequent intertransaction itemsets increases exponentially as maxspan increases. There are, however, no frequent intertransaction itemsets that span across more than R intervals. As such when maxspan is greater than R, the graph grows linearly reflecting the fact that additional time is needed to process the extra subwindows.

5.4 Effect of Increasing the Number of Transactions

We shall now investigate how FITI performs when the number of transactions in the database increases. We vary the number of transactions in dataset one from 10k to 1,000k with support = 0.1 percent and maxspan = 4. The result Fig. 5a shows that the running time of FITI increases linearly with the number of transactions in the database. We again confirm this in Fig. 5b where we increase the number of transactions in data set two from 0.5k to 8k and mine it with a support level of 15 percent and a maxspan of six.

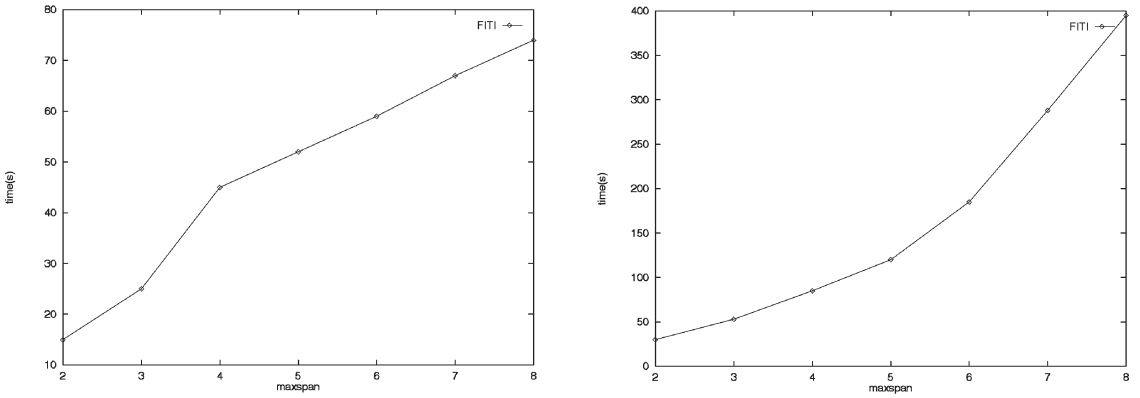
TABLE 5
Comparison of EH-Apriori and FITI

Data Set 1			Data Set 2		
Support	EH-Apriori	FITI	Support	EH-Apriori	FITI
0.1%	857s	45s	11.0%	12000s	7230s
0.2%	301s	23s	12.0%	6128s	2620s
0.3%	200s	11s	13.0%	3449s	1004s
0.4%	176s	9s	14.0%	2411s	397s
0.5%	154s	8s	15.0%	1815s	183s



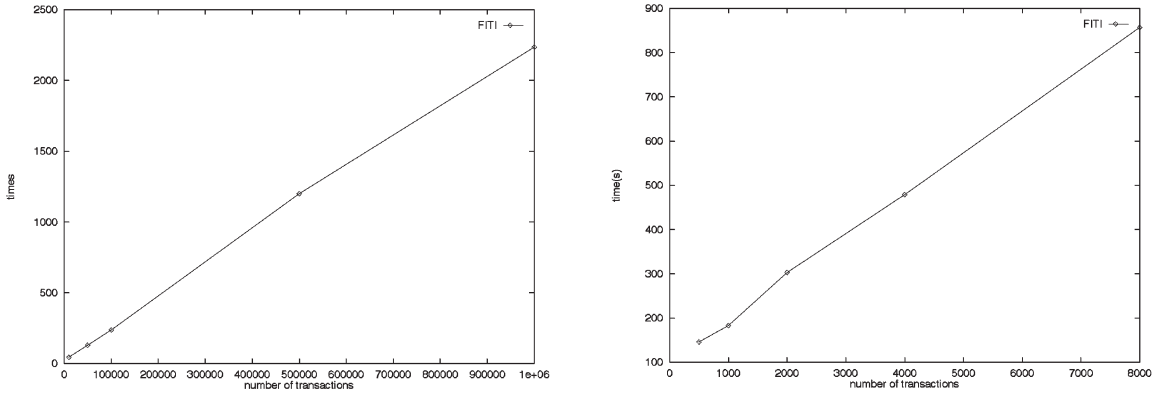
(a) (b)

Fig. 3. Comparing performance of EH-Apriori and FITI. (a) Data set 1 with maxspan = 4 and (b) data set 2 with maxspan = 6.



(a) (b)

Fig. 4. Effect of increasing maxspan. (a) Data set 1 with support = 0.1 percent and (b) data set 2 with support = 15 percent.



(a) (b)

Fig. 5. Effect of increasing the number of transactions. (a) Data set 1: support = 0.1 percent, maxspan = 4 and (b) Data set 2: support = 15 percent, maxspan = 6.

5.5 Effect of Increasing the Average Transaction Size

We will next examine the effect of increasing the average transaction size on the running time of FITI. We increase the average transaction size of data set one from five to 20 and mine the data set with a support level of 0.5 percent and a maxspan of four intervals. We leave out data set two in this experiment because the running time will be too long if we increase the transaction size of data set two further. The result of the experiment is shown in Fig. 6. As can be seen

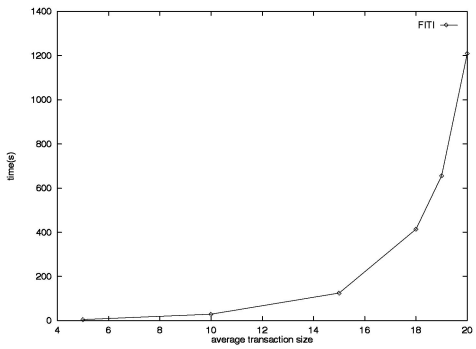


Fig. 6. Effect of increasing the size of transactions. Data set 1: support = 0.5 percent, maxspan = 4.

from the graph, the running time of FITI increases drastically when the average transaction size increases. This is due to the large number of frequent intertransaction itemsets which are generated in each pass when the average transaction size increases. This large number of frequent intertransaction itemsets results in a large number of candidate itemsets to be counted. For example, when the average size of the transaction is 20, we have one million candidate intertransaction three-itemsets which require 949 seconds for counting.

5.6 Experiment on Real-Life Data

To assess the usefulness of intertransaction association rules, we apply FITI to a data set comprising of 10 stock exchange indices for 98 trading days from 3 July 1998 to 17 November 1998. These 10 stock exchange indices are ASX All Ordinaries Index (ASX), CAC40 Index (CAC), DAX Index (DAX), Dow Jones Index (DOW), FT-SE 100 Index (FTS), Hang Seng Index (HSI), NASDAQ Index (NDQ), Nikkei 225 Index (NKY), Swiss Market Index (SMI), and Singapore ST Index (STI). This data set is separated into two portions. The first portion, called WINNER, consists of stock indices which rises for the day, while the other portion called LOSER consists of those which fall. By mining the WINNER portion with $maxspan = 5$, we obtained some rules. One of them that involved STI is "NDQ(0), CAC(3) $\Rightarrow$ STI(4)" with $support$ = 18 percent and $confidence$ = 81 percent. This rule mean the following: "If NDQ rise on the first day follow by CAC on the fourth day, then STI rise at the fifth day with $support$ = 18 percent and $confidence$ = 80 percent." The same is done to the LOSER portion and a rule that involved STI is "NKY(0), CAC(1) $\Rightarrow$ STI(2)" with $support$ = 25 percent and $confidence$ = 80 percent.

Using thses two rules, we adopted a simple stragety of buying when the precedent of the first rule occurred and selling when the precedant of the second rule occurred. This stragety is applied on the same data set to assess the profitability of adopting it. The profit at the time of selling is computed by subtracting the STI index at the time of buying from the STI index at the time of selling. This profit is accumulated and it is found that the accumulated profit is always postive at any one time. From this result, we can see that intertransaction association rules are in fact quite useful for the purpose of stock prediction. Note that, in general, stock prediction is a form of classification task and association rules have been found to be useful in such a task [12].

6 Conclusion

We introduced a new problem of mining intertransaction association rules and provided an extensive study of it. Two algorithms, EH-Apriori and FITI were implemented for mining intertransaction association rules. FITI proves to be much faster than EH-Apriori and its performance was found to be acceptable for real life application. To improve FITI further, we suggest various ways of reducing its execution time and memory requirement. While we believe that the present form of intertransaction association rules will prove to be useful in providing prediction capability along a single dimension, we feel that this usefulness can be further enhanced if prediction along multiple dimensions is possible. As such, we extend our definition of intertransaction association rules to n-dimensional intertransaction association rules. The success of FITI provides us with the necessary ground work for mining this new form of association rules.

Acknowledgments

The work of Anthony K.H. Tung and Hongjun Lu is partially supported by a grant from the National 973 project of China (No. G19908030414). In addition, Anthony K.H. Tung was also supported in part by the Natural Sciences and Engineering Council of Canada and the Networks of Centres of Excellence/Geomatics for Informed Decisions research grant together with Jiawei Han.

References

[1] R. Agrawal, T. Imielinski, and A. Swami, "Mining Association Rules Between Sets of Items in Large Databases," *Proc. 1993 ACM-SIGMOD Int'l Conf. Management of Data*, pp. 207-216, May 1993.

[2] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," *Proc. 1994 Int'l Conf. Very Large Data Bases*, pp. 487-499, Sept. 1994.

[3] R. Agrawal and R. Srikant, "Mining Sequential Patterns," *Proc. 1995 Int'l Conf. Data Eng.*, pp. 3-14, Mar. 1995.

[4] S. Brin, R. Motwani, and C. Silverstein, "Beyond Market Basket: Generalizing Association Rules to Correlations," *Proc. 1997 ACM-SIGMOD Int'l Conf. Management of Data*, pp. 265-276, May 1997.

[5] C. Bettini, X.S. Wang, and S. Jajodia, "Testing Complex Temporal Relationships Involving Multiple Granularities and Its Application to Data Mining," *Proc. 15th ACM Symp. Principles of Database Systems*, June 1996.

[6] C. Bettini, X. Sean Wang, and S. Jajodia, "Mining Temporal Relationships with Multiple Granularities in Time Sequences," *Data Eng. Bull.* vol. 21, pp. 32-38, 1998.

[7] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama, "Data Mining Using Two-Dimensional Optimized Association Rules: Scheme, Algorithms, and Visualization," *Proc. 1996 ACM-SIGMOD Int'l Conf. Management of Data*, pp. 13-23, June 1996.

[8] J. Han and Y. Fu, "Discovery of Multiple-Level Association Rules from Large Databases," *Proc. 1995 Int'l Conf. Very Large Data Bases*, pp. 420-431, Sept. 1995.

[9] M. Kamber, J. Han, and J.Y. Chiang, "Metarule-Guided Mining of Multi-Dimensional Association Rules Using Data Cubes," *Proc. Third Int'l Conf. Knowledge Discovery and Data Mining*, pp. 207-210, Aug. 1997.

[10] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A.I. Verkamo, "Finding Interesting Rules from Large Sets of Discovered Association Rules," *Proc. Third Int'l Conf. Information and Knowledge Management*, pp. 401-408, Nov. 1994.

[11] H. Lu, J. Han, and L. Feng, "Stock Movement and n-Dimensional Intertransaction Association Rules," *Proc. 1998 SIGMOD Workshop Research Issues on Data Mining and Knowledge Discovery*, vol. 12, pp. 1-7, June 1998.

[12] B. Liu, W. Hsu, and Y. Ma, "Integrating Classification and Association Rule Mining," *Proc. 1998 Int'l Conf. Knowledge Discovery and Data Mining,* pp. 80-86, Aug. 1998.
[13] L.V.S. Lakshmanan, R. Ng, J. Han, and A. Pang, "Optimization of Constrained Frequent Set Queries With 2-Variable Constraints," *Proc. 1999 ACM-SIGMOD Int'l Conf. Management of Data,* pp. 157-168, June 1999.
[14] B. Lent, A. Swami, and J. Widom, "Clustering Association Rules," *Proc. 1997 Int'l Conf. Data Eng.,* pp. 220-231, Apr. 1997.
[15] R. Meo, G. Psaila, and S. Ceri, "A New SQL-Like Operator for Mining Association Rules," *Proc. 1996 Int'l Conf. Very Large Data Bases,* pp. 122-133, Sept. 1996.
[16] H. Mannila, H. Toivonen, and A.I. Verkamo, "Discovery of Frequent Episodes in Event Sequences," *Data Mining and Knowledge Discovery,* vol. 1, pp. 259-289, 1997.
[17] R.J. Miller and Y. Yang, "Association Rules Over Interval Data," *Proc. 1997 ACM-SIGMOD Int'l Conf. Management of Data,* pp. 452-461, May 1997.
[18] R. Ng, L.V.S. Lakshmanan, J. Han, and A. Pang, "Exploratory Mining and Pruning Optimizations of Constrained Associations Rules," *Proc. 1998 ACM-SIGMOD Int'l Conf. Management of Data,* pp. 13-24, June 1998.
[19] J.S. Park, M.S. Chen, and P.S. Yu, "An Effective Hash-Based Algorithm for Mining Association Rules," *Proc. 1995 ACM-SIGMOD Int'l Conf. Management of Data,* pp. 175-186, May 1995.
[20] R. Srikant and R. Agrawal, "Mining Generalized Association Rules," *Proc. 1995 Int'l Conf. Very Large Data Bases,* pp. 407-419, Sept. 1995.
[21] R. Srikant and R. Agrawal, "Mining Quantitative Association Rules in Large Relational Tables," *Proc. 1996 ACM-SIGMOD Int'l Conf. Management of Data,* pp. 1-12, June 1996.
[22] C. Silverstein, S. Brin, R. Motwani, and J. Ullman, "Scalable Techniques for Mining Causal Structures," *Proc. 1998 Int'l Conf. Very Large Data Bases,* pp. 594-605, Aug. 1998.
[23] A. Savasere, E. Omiecinski, and S. Navathe, "An Efficient Algorithm for Mining Association Rules in Large Databases," *Proc. 1995 Int'l Conf. Very Large Data Bases,* pp. 432-443, Sept. 1995.
[24] R. Srikant, Q. Vu, and R. Agrawal, "Mining Association Rules with Item Constraints," *Proc. Third Int'l Conf. Knowledge Discovery and Data Mining (KDD '97),* pp. 67-73, Aug. 1997.
[25] H. Toivonen, "Sampling Large Databases for Association Rules," *Proc. 1996 Int'l Conf. Very Large Data Bases,* pp. 134-145, Sept. 1996.

Anthony K.H. Tung received the BSc and MSc degrees in computer science from the National University of Singapore in 1997 and 1998, respectively. In 2001, he received the PhD degree in computer science from Simon Fraser University. Dr. Tung is currently an assistant professor in the Department of Computer Science, National University of Singapore. His research interests involve various aspects of databases and data mining including buffer management, frequent pattern discovery, spatial clustering, outlier detection, and classificationi analysis. He is a member of the IEEE.

Hongjun Lu received the BSc degree from Tsinghua University, China, and the MSc and PhD degrees from the Department of Computer Science, University of Wisconsin-Madison. Before joining the Hong Kong University of Science and Technology, he was an engineer in the Chinese Academy of Space Technology (1968-1979), a principal research scientist in the Computer Science Center of Honeywell Inc., Minnesota, (1985-1987), and a professor at the School of Computing of the National University of Singapore. His research interests are in data/knowledge base management systems with emphasis on query processing and optimization, physical database design, and database performance. His recent research work includes data quality, data warehousing, and data mining. He is also interested in the development of Internet-based database applications and electronic business systems. He has been published extensively in important international database conferences and journals such as SIGMOD, VLDB, ICDE, EDBT, TKDE, and VLDB Journal. He is actively involved in international professional activities. He is currently a trustee of the VLDB Endowment and a member of the ACM SIGMOD advisory board. He also serves as a member of the ACM SIGKDD International Liaisons. He is the chair of the steering committee of the International Conference on Web-Age Information Management (WAIM), and the cochair of the steering committee of Pacific-Asia Conference of Knowledge Discovery and Data Mining (PAKDD). He is a member of the IEEE.

Jiawei Han received the PhD degree in computer science from the University of Wisconsin at Madison in 1985. He is currently a professor at the Department of Computer Science, University of Illinois at Urbana-Champaign. He was previously the director of the Intelligent Database Systems Research Laboratory, and endowed University Professor, School of Computing Science, Simon Fraser University, Canada. He has conducted research in the areas of data mining and knowledge discovery, data warehousing, spatial databases, deductive and object-oriented databases, multimedia databases, and Web technology, with more than 150 journal and conference publications. He has served or is currently serving in the program committees of more than 100 international conferences and workshops, including SIGMOD 2002, ICDE 2002 (vice chairman), SIAM-DM 2002 (PC cochairman), SIGKDD 2001 (Best paper award chairman), SIAM-DM 2001 (PC cochairman), VLDB 2000, SIGMOD 2000 (demo chair), EDBT 2000, etc. He has been serving on the executive committee of ACM-SIGKDD. He served on the editorial board for *IEEE Transactions on Knowledge and Data Engineering* (1996-2000), and has been serving as an editor for Data Mining and Knowledge Discovery and the *Journal of Intelligent Information Systems*. He is a member of the IEEE.

Ling Feng received the BSc and PhD degrees in computer science from Huazhong University of Science and Technology, China, in 1990 and 1995, respectively. She was a postdoctoral fellow in the Department of Information Systems and Computer Science at the National University of Singapore from 1995 to 1997, and a lecturer in the Department of Computing at Hong Kong Polytechnic University from 1997 to 1999. She is currently an assistant professor in the Infolab in the Department of Information Management at Tilburg University, the Netherlands. Her research interests include data warehouses, data mining and its applications, distributed object-oriented database management systems, knowledge-based information systems, and Web-based information technologies. She has been engaged in 12 different international and national projects as principal investigator, project leader, chief designer, and developer, and has published more than 50 research papers in international and national conferences and journals. She is a member of the IEEE.

▷ **For more information on this or any computing topic, please visit our Digital Library at** http://computer.org/publications/dlib.