18

A Search Algorithm for Intertransaction Association Rules

Dan Ungureanu
Politehnica University of Bucharest
Romania

1. Introduction

The problem analysed in this chapter is the discovery of interesting relations between variables in large databases.
The databases are considered to contain transactions that each contain one or more items from a discrete set of items. The relations between the items are expressed in the form of association rules. The problem of searching for association rules has been denoted in the research literature as association rule mining.
The initial association rule mining problem ignored any correlation between the transactions and searched for associations only between items inside a transaction (this case is called case intratransaction association rules mining). To search for associations between items across several transactions ordered on a dimension (usually time or space), intertransaction association rule mining has been used.
We use the stock market database example to differentiate between intra- and inter-transaction analysis. If the database contains the price for each stock at the end of the trading day, an intratransaction association rule might be "If stock prices for companies A and B go up for one day, there is a probability of over c% that the price for company C will also go up the same day". However, analysts might be more interested in rules like "If stock prices for companies A and B go up for one day, there is a probability of over c% that the price for company C will go up two days later." This rule describes a relationship between items from different transactions, and it can be discovered only by using intertransaction analysis.
The main part of association rules mining has been determined to be finding the frequent itemsets.
A search algorithm that finds frequent intertransaction itemsets called InterTraSM will be presented, exemplified and analyzed in this chapter. It was first introduced in (Ungureanu & Boicea, 2008). This algorithm is an extension for the intertransactional case of the SmartMiner algorithm presented in (Zou et al., 2002). InterTraSM focuses on mining maximal frequent itemsets (MFI) – itemsets that are not a subset of any other frequent itemset. Once the MFI have obtained all the frequent itemsets can easily be derived, and they can then be counted for support in a single scan of the database.
The remainder of this chapter is organized as follows.

- Section 2 contains a formal definition for the problem of intertransaction association rules mining.

- Section 3 briefly describes other algorithms used for finding frequent intertransaction itemsets.
- Section 4 introduces the InterTraSM algorithm for searching frequent intertransaction itemsets. It also contains a detailed example of its application, and an analysis of its complexity.
- Section 5 presents an implementation of the InterTraSM algorithm, the experimental results obtained regarding its execution time, and a comparison with the performances of the algorithms from Section 3.
- Section 6 concludes the chapter and proposes future research in this area.

2. Problem description

We consider I to be a set of literals $\{a_1, a_2, \ldots, a_n\}$, also called items.
In the original association rule mining problem a transaction T was defined as a set of items such that $T \subset I$.
A database DB (also called transaction database or transactional database) was defined as a set of transactions with unique transaction IDs $\{T_1, T_2, \ldots, T_n\}$.
An association rule was defined as an implication of the form $X \rightarrow Y$, where X and Y are distinct subsets of I ($X \subset I, Y \subset I, X \cap Y = \emptyset$).
The two main measures used to characterize association rules are support and confidence.
A rule $X \rightarrow Y$ has support s in a database DB if s% of the transactions from DB contain $X \cup Y$.
A rule $X \rightarrow Y$ has confidence c in a database DB if c% of the transactions from DB that contain X also contain Y.
The initial approach to association rule mining only considered rules between items from the same transaction.
In practice though the transactions usually have some additional information attached to them, like the time when they occurred, the customers that purchased the items, or the geographical location.
In order to characterize such contexts of when / where the transactions occurred, dimensional attributes have been introduced. They can be of any kind, as long as they can be associated with the transactions from the database in a meaningful way.
In this paper we consider only the case when there is a single dimensional attribute which has an ordinal domain. Furthermore, the domain of the attribute can be divided into equal-sized intervals, such that at most one transaction from the transaction database is associated with each interval, and that each transaction can be associated with an interval.
Let d_f be the first interval and d_l be the last interval from the domain that have an associated transaction. Then, without loss of generality, we can denote the first interval with 0 and the intervals that follow it with 1, 2, 3 and so on. We consider l to be the number of interval d_l.
We denote with D the domain of the attribute and with Dom(D) the set of intervals $\{0, 1, 2, \ldots, l\}$.
So for the intertransaction case we introduce the following
Definition *Transactional database*. A transactional database is a database that contains transactions of the form $t = (d, T)$, where $d \in dom(D)$ and $T \subset I$.
In practice we are not interested in associations between items from transactions for which the distance between the associated intervals is unlimited. The running time of an algorithm that searches for such associations can be prohibitive, and also the practical use of such associations is limited at best. The goal of our research is used to find previously unknown

correlations between items, but the idea is that there might be an intrinsic relation between the data that would come into light. Such a relation would ensure that future transactions from the same domain would exhibit the same properties. This would mean that we can use the discovered associations to take useful actions related to the domain.

Research has therefore been focused on searching for associations between items from transactions for which the distance between the associated intervals has a maximum value *w* – we will also call this value the maximum span of the intertransaction association rules.

Definition *Sliding window.* A sliding window W in a transactional database T represents a set of continuous intervals from the domain D, such that there exists in T a transaction associated to the first interval from W. Each interval is called a subwindow of the sliding window, and they are numbered corresponding to their temporal order d_0, d_1, …, d_w. We also use the notation W[0], W[1], …, W[w] for the intervals of W and we say that w is the size of the sliding window.

Definition *Megatransaction.* Let T be a transactional database, let I be the set of items $\{a_1, a_2, \ldots, a_n\}$ and let W be a sliding window with size w. A *megatransaction* M associated with the sliding window W is the set of all elements denoted with a_i^j, such that a_i belongs to the transaction associated with the interval W[j], for each corresponding value of $1 \le i \le n$, $0 \le j \le w$.

The items from a megatransaction will be called from now on *extended items*.

We denote with E the set of all the possible extended items $\{a_1^0, a_1^1, \ldots, a_1^w, a_2^0, \ldots, a_n^w\}$.

We call an *intratransaction itemset* a set of items $A \subset I$.

We call an *intertransaction itemset* a set of extended items $B \subset E$ such that B contains at least one extended item of the form e_i^0.

Definition *Intertransaction association rule.* An intertransaction association rule has the form $X \rightarrow Y$ where:

i. $X, Y \subset E$

ii. X contains at least one extended item of the form e_i^0, $1 \le i \le n$.

iii. Y contains at least one extended item of the form e_i^j, $1 \le i \le n$, $j > 0$.

iv. $X \cap Y = \varnothing$

Let N be the total number of transactions, T_{XY} be the set of transactions that contain the set $X \cup Y$ and T_X be the set of transactions that contain X.

Then the support of the rule is $s = |T_{XY}| / N$ and the confidence of the rule is $c = |T_{XY}| / |T_X|$.

The research in this domain has been focused on finding association rules whose support and confidence are above some specified minimum thresholds. The support threshold indicates that the two itemsets appear together in the transactional database often enough so we can benefit in practice from finding an association between them, and the confidence threshold indicates a minimum degree of confidence for the association between the two itemsets.

The problem of searching for intertransaction association rules has been divided into two parts:

- finding the intertransaction itemsets that have the support above a specified minimum threshold (this itemsets are said to be frequent)
- finding the association rules

The second problem takes much less computational time than the first one, so it presents little interest for research. A solution has been discussed for example in (Tung et al., 2003). Our work (like most of the research in this area) has therefore focused on developing an algorithm for the first problem.

3. Related work

Several algorithms for finding frequent intertransaction itemsets have been previously introduced, and some of them are presented in this chapter. Many of the algorithms for searching intertransaction association rules (including our algorithm InterTraSM that will be presented in the next chapter) are extensions for the intertransaction case of algorithms developed for searching intratransaction association rules.

3.1 E-Apriori, EH-Apriori

The classic algorithm for searching (intratransaction) association rules is Apriori, introduced in (Agrawal & Srikant, 1994). It uses the following fact: if an itemset that has k elements (also called a k-itemset) is frequent, then all its subsets, (k-1) itemsets, must also be frequent.
The algorithm first determines all the frequent 1 - itemsets (all the frequent items in this case) by counting the support of all the items.
Then, for each k>=2, the algorithms knows that the frequent itemsets must have all their subsets also frequent. The algorithm uses each combination of two itemsets with k-1 elements to see if their reunion is a k-itemset for which all the subsets with k-1 elements are frequent itemsets. For all the k-itemsets that verify this property (these itemsets are called candidate itemsets) their support is counted in a single pass through the database.
The algorithm stops when for a given k no frequent k-itemsets are found.
In order to ensure that the same frequent itemset is not discovered multiple times (starting from different initial items) the algorithm assumes there is a lexical ordering between the items, and at any given steps it tries to add to the current itemset only items that are "greater" (using the lexical ordering) than the last added item.
The E-Apriori (Extended Apriori) algorithm for the intertransaction case, introduced in (Lu et al., 2000), is an extension of the Apriori algorithm so it uses intertransaction itemsets containing extended items. A lexical order is given for the intratransaction items, and for all the extended items from the same interval the lexical order for the corresponding intratransaction items is used. It is also said that all the extended items from interval (i+1) are "greater" than all the extended items from interval i.
It has been observed that the time for obtaining the frequent 2-itemsets has a big impact on the total running time, due to the fact that there are a large number of frequent 1-itemsets, which leads to a large number of candidate 2-itemsets which have to be analized.
The EH-Apriori (Extended Hash Apriori) , also introduced in (Lu et al., 2000), uses a hashing technique to reduce the number of candidate 2-itemsets. As the support of the frequent extended items is computed, each possible 2-extended itemset is mapped to a hash table. Each bucket from the hash table indicates how many extended itemsets has been hashed to it.

3.2 FITI

Another algorithm for searching intertransaction association rules, initially introduced in (Tung et al., 2003) is called FITI (First Intra Then Inter).
FITI uses the property that given a frequent intertransaction itemset, any subset for which the same interval is associated to all the extended items must correspond to a frequent intratransaction itemset. Using this property the algorithm first determines all the frequent intratransaction itemsets, and using them determines next all the frequent intertransaction itemsets – hence the name of the algorithm.
We briefly describe the three phases of the FITI algorithm:

1. Find the frequent intratransaction itemsets (using any algorithm for intratransaction association rule mining, like Apriori). The itemsets will be stored in a special data structure called FILT (Frequent-Itemsets Linked Table). This consists of a hash table of itemsets where the nodes are linked with multiple types of connections. Each frequent itemset is associated a unique ID.
2. The database is transformed into a set of encoded frequent-itemsets tables (FIT tables). The number of FIT tables is the maximum size of the frequent intratransaction itemsets discovered in phase 1, and each table corresponds to a particular size (from 1 to the maximum value). For any FIT table, corresponding to a size denoted with k, the records have the form $\{d_i, IDset_i\}$ where d_i represents a value of the dimensional attribute (an interval), and $IDset_i$ represents the set of IDs (defined in phase 1) of frequent intratransaction itemsets with size k that are found in the transaction associated with d_i.
3. The frequent intertransaction itemsets are discovered using a level-wise process similar to the one used in Apriori: all the candidates with k elements are determined, their support is computed during one pass through the database and then using the frequent itemsets with k elements we go on to compute all the candidates with (k+1) elements.

For the discovery of frequent itemsets with two items FITI uses a hashing approach similar to the one used in EH-Apriori.

For k>=2, the algorithm defines two kinds of combining two itemsets with k elements in order to generate a candidate itemset with (k+1) elements:

a. an intratransaction join is performed between two itemsets who have (k-1) of their items identical, and the items that are different belong to the same interval of the sliding window
b. a cross-transaction join is performed between two itemsets who have (k-1) of their items identical, and the items that are different belong to different intervals I1 and I2 of the sliding window. In order not to generate the same itemsets through both intratransaction and cross-transaction joins, the following restriction are placed for cross-transaction joins:

- the first itemset must not have any items associated to interval I2 or any items associated to any interval after I1
- the second itemset must not have any items associated to interval I1 or any items associated to any interval after I2
- any interval must have associated at most one extended item in both itemsets

3.3 ITP-Miner

The ITP-Miner algorithm, initially introduced in (Lee & Wang, 2007), tries to avoid the breadth first search approach of Apriori-like algorithms. In order to realize only one pass through the database the algorithm uses a special data structure called dimensional attribute list (dat-list) to store the dimensional attributes (the corresponding intervals in fact) associated to a frequent itemset. For a given frequent itemset A, if the megatransactions in which A appears start at the intervals $t_1, t_2, \ldots, t_n$ then the associated dat-list is $A(t_1, t_2, \ldots, t_n)$. The algorithm also assumes a lexical ordering of the intratransaction items, which leads to an ordering of the extended items in the way described for the E-Apriori and EH-Apriori algorithms. A convention is used that the extended items in a frequent itemset from a dat-list are stored in increasing order (using the ordering previously described).

The algorithm uses a structure called ITP-Tree that is a search tree that has dat-lists as nodes. The parent node of a dat-list $a(t_1, t_2, \ldots, t_n)$ is the dat list $b(u_1, u_2, \ldots, u_m)$ where b is

obtained from a by removing the last extended item and m>=n. The root of an ITP-Tree is the empty itemset $\varnothing$ and the nodes on the same depth level are ordered in increasing order depending on the extended items from the itemset.
The algorithm starts with determining the frequent items and their dat-lists, and for determining the frequent itemsets with more than one element it doesn't read the database again, but it works with the already determined dat-lists.
A join between two frequent itemsets $\{u_0(0), u_1(i_1), \ldots, u_{k-1}(i_{k-1})\}$ and $\{v_0(0), v_1(j_1), \ldots, v_{k-1}(j_{k-1})\}$ is deemed possible if the first (k-1) extended items are identical and $u_{k-1}(i_{k-1}) < v_{k-1}(j_{k-1})$ using the given order relation. The unified itemset is then $\{u_0(0), u_1(i_1), \ldots, u_{k-1}(i_{k-1}), v_{k-1}(j_{k-1})\}$- so it is the same principle of generating candidate itemsets as the one used for Apriori-like algorithms, only a DFS search is used instead of a BFS-search.
The dat-list for a candidate itemset has for the set of dimensional attributes the intersection of the sets of dimensional attributes from the two itemsets used to generate it. The size of this set determines the support of the candidate itemset, and in this way it is determined if the itemset is frequent.
The following pruning strategies are used by ITP-Miner to reduce the size of the search space:

- pruning of infrequent itemsets with 2 elements: a hash table is used to check if a pair of frequent items can generate a frequent 2-itemset before the join is performed. A hash table H2 is created while finding the frequent items and their dat-lists
- pruning of infrequent itemsets with k elements: before performing a join of 2 frequent itemsets with (k-1) elements, the hash table H2 is used to check if the two extended items that differ might represent a frequent itemsets with 2 elements (once again the property that any subset of a frequent itemset must also be frequent is used here)

3.4 EFP-Growth

We will now summarize the EFP-Growth algorithm, initially presented in (Luhr et al., 2007). The algorithm uses the pattern-growth property and it is an adaptation for the intertransaction case of the FP-Growth algorithm described in (Han et al., 2000). The algorithm uses an EFP-Tree (Extended Frequent Pattern Tree) which is an adaptation for the intertransaction case of the FP-Tree used in the FP-Growth algorithm.
The EFP-Tree is composed of intraitem nodes ordered by descending frequency, and each node can have zero or one interitem subtree which is a FP-Tree where the frequency ordering of the interitems is conditioned on the intraitem parent node.
The construction of the EFP-Tree is done in three passes over the database.
The frequency of the extended items is computed in the first pass and the frequent intraitems and interitems are found.
In the second pass the intraitem tree is built and for each transaction the infrequent intraitems are removed and the frequent items are sorted in decreasing order of frequency. Also the conditional frequencies for the interitems are computed (the frequencies only in the transactions in which the intraitems appear).
In the third pass the interitem sub-trees are build.
The trees are build by passing through each transaction of the database, and for each item encountered by going one step lower in the tree (or creating a new descendent node with the new item if it doesn't exist) and increasing the support with 1. When the first interitem is reached the processing passes to the interitem subtree corresponding to the current intraitem node.

After the EFP-Tree is built the algorithm computes the association rules using the pattern growth property. A divide and conquer strategy is used to construct trees conditioned on known frequent base rules and to take the dot product of the frequent items in the conditional tree and the conditional base itemset to produce new rules, which in turn will become the conditional base for the new set of rules to be mined.
The algorithm uses two types of conditional trees:

- a conditional EFP-Tree that is used to find the intraitems and interitems that can be used to extend the present intraitem rule suffix
- a FP-Tree of the interitems inherited by the conditional base

4. The InterTraSM algorithm

4.1 Theoretical presentation

We will present in this chapter the InterTraSM algorithm for mining intertransaction association rules, which was first introduced in (Ungureanu & Boicea, 2008). The algorithm is an adaptation for the intertransaction case of the SmartMiner algorithm for finding (intratransaction) association rules, which was introduced in (Zou et al., 2002).
The InterTraSM algorithm (like the SmartMiner algorithm which it extends) searches for maximal frequent itemsets (itemset which are frequent, but for which no superset of them is also frequent) using a depth first search. Then all the frequent itemsets are derived from the maximal frequent itemsets (MFI).
The algorithms consists of a series of steps, each of which is applied to a node of a search tree. We will describe next what data is available for each node and how that data is processed.
We identify a node with X:Y, where X (the head) is the current set of extended items that has been discovered to form a frequent itemset, and Y (the tail) is the set of extended items that still has to be explored to find all the maximal frequent itemsets that contain X is a subset.
The starting node of the algorithm is identified by $\varnothing$:E (the empty set and the set of all possible extended items).
As in the SmartMiner algorithm, we also use some additional data attached to each node to help our algorithm:

- we define the transaction set T(X) to represent all the transactions from the database that contain the itemset X. For the starting node (where X is the empty set) the transaction set is the entire database, and we will show how T(X) is obtained for each subsequent node
- if we denote with M the known frequent itemsets (the itemsets determined to be frequent before we start processing of the current node) and with N = X : Y the current node, then the tail information of M to N is the parts from Y (the tail of the node) that can be inferred from M. For the processing of a given node the algorithm we use the tail information for the frequent itemsets discovered previously.

The entry data for a node consists then of:

- the transaction set T(X)
- the tail Y
- the tail information for the node that has been obtained so far (also called global tail information, or Ginf). This information is passed from the parent node, and it contains the itemsets that have been previously discovered to be frequent in T(X)

The exit data for a node consists of:

- the updated global tail information Ginf
- the local maximal frequent itemsets discovered are the node and its descendants

The data processing at the level of a node from the search tree is described below:

1. count the support for each item from Y in the transaction set T(X)
2. remove the infrequent items from Y
3. while Y has at least one element
   4. if there is an item in Ginf with the size equal to the length of the tail, the itemset containing all the items from the tail has already found to be frequent in previous processing steps, so we can skip to step 12
   5. select an item a_i from Y
   6. the head of the next state S_{i+1} will be $X_{i+1} = X \cup \{a_i\}$
   7. the tail of the next state S_{i+1} will be $Y_{i+1} = Y \setminus \{a_i\}$
   8. the global tail information for S_{i+1} is computed by projecting on Y_{i+1} the itemsets that contain a_i
   9. recursively call the algorithm for the node $N_{i+1} = X_{i+1} : Y_{i+1}$
      The returned values will be Mfi_i the local maximal frequent itemsets found at the node N_{i+1}. The global tail information Ginf is updated to include Mfi_i and then it is projected on the remaining tail. The members subsumed by Mfi_i are marked as deleted
   10. $Y = Y_{i+1}$
11. end while
12. $Mfi = \cup (a_i Mfi_i)$
13. return Mfi and the current updated value of Ginf

The InterTraSM algorithm uses extended items instead of the intratransaction items used by SmartMiner. Also for the first level nodes we select while starting from the root node we choose only extended items which have 0 as the associated interval – because each intertransaction association rule must contain at least one extended item with interval 0 in the head of the rule.

We next analyse the complexity of the InterTraSM algorithm.

We consider the following variables that affect the complexity of the algorithm:

- $|D|$ - the number of intervals for the domain D. This is equal to the number of intertransactions from the database
- W – the size of the sliding window
- $|L1|$ - the number of frequent intratransaction items

The processing for a node from the search tree is divided in the following parts:

1. Determining the support in T(X) for each element from Y
   The number of megatransaction from T(X) is limited to $|D|$ and the number of elements from Y is limited to $|L1| \times W$, so the number of operations for this step is $O(|D| \times |L1| \times W)$.
2. Preparing the entry data for each sub-node, calling each sub-node recursively (without taking into account the processing inside the sub-nodes) and computing the exit data using the data returned by the subnode – the complexity is $O(|L1| \times W)$.

We conclude that the maximum complexity for a node is $O(|D| \times |L1| \times W)$, but the maximum average complexity on all nodes will be lower since both the number of transactions in T(X) and the number of items in Y decrease while we descend into the search tree.

4.2 An example

We will next present a detailed example of the execution steps of the algorithm.

We consider first a set of items {a, b, c, d}.

We consider next the following transactional database: with 5 transactions:
T0 a,b
T1 a,c
T2 c
T3 a,b
T4 c,d
We consider the maximum span of the intertransaction association rules we want to discover to be 1 (so the size of the sliding window will be w=1).
We will work then with the following 5 megatransactions:
M0: a^0, b^0, a^1, c^1
M1: a^0, c^0, c^1
M2: c^0, a^1, b^1
M3: a^0, b^0, c^1, d^1
M4: c^0, d^0
The minimum support threshold for the rules we want to discover is s=0.4 (40%).
We have E = {a^0, b^0, c^0, d^0, a^1, b^1, c^1, d^1}
The entry node N0 of the algorithm has X = , Y = E, Ginf = $\varnothing$ and
T(X) = {M0, M1, M2, M3, M4}.
We start by computing the support for each item from Y in T(X) (step 1 for node N0)
We find that s(a^0)=3, s(b^0)=2, s(c^0)=3, s(d^0)=1, s(a^1)=2, s(b^1)=1, s(c^1)=3, s(d^1)=1.
The frequent extended items are the ones who have the support value at least 2 (40% of the total transactions).
When we remove the infrequent items (step 2 for node N0) we remain with {a^0, b^0, c^0, a^1, c^1}.
We select a^0 to be X1 - the head of the next node N1, and the tail Y1 will be {b^0, c^0, a^1, c^1}. Ginf will be $\varnothing$ and T(X1) = {M0, M1, M3}.
We call the algorithm recursively for node N1.
We compute the support for the items from Y1 in T(X1) (step 1 for node N1)
We find that s(b^0)=2, s(c^0)=1, s(a^1)=1, s(c^1)=3.
When we remove the infrequent items (step 2 for node N1) we remain with { b^0, c^1}
We next select b^0 to be X2 - the head of the next node N2, the tail Y2 will be { c^1}, Ginf = $\varnothing$ and T(X2) = {M0, M3}. We call the algorithm recursively for node N2.
We compute the support for the items from Y2 in T(X2) (step 1 for node N2)
We find that s(c^1)=2. Since there is only one element in the tail and it is frequent we can only select c^1 as the head for the next node N3 and the tail will be empty, so N3 will return $\varnothing$ as MFI. No more items remain in the tail to be processed, so the node N2 will return MFI = {c^1}
After N2 returns, the Ginf for N1 will be updated to be {c^1}.
Since the remaining value of the tail is {c^1}, according to step 4 for N1 we can skip to step 12.
The node N1 will return MFI= {b^0c^1}
After node N1 returns, the Ginf for node N0 is updated to {b^0c^1}
We next return to step 3 and we select another item from the remaining tail {b^0, c^0, a^1, c^1}
We select item b^0 to be X4, Y4 will be {c^0, a^1, c^1}, Ginf for node N4 will be the projection of Ginf on Y4 - {c^1}, T(X4) = {M0, M3}.
We next call the algorithm recursively for node N4.
We compute the support for the items from Y4 in T(X4) (step 1 for node N4)
We find that s(c^0)=0, s(a^1)=1, s(c^1)=2.
When we remove the infrequent items (step 2 for node N4) we remain with {c^1}.
According to step 4 we can skip to step 12.

The node N4 will return MFI = ∅
We next return to step 3 and we select another item from the remaining tail $\{c^0, a^1, c^1\}$.
We select item c^0 to be X5, Y5 will be $\{a^1, c^1\}$. Ginf for node N5 will be the projection of Ginf on Y5 - $\{c^1\}$. T(X5) = {M1, M2,M4}.
When call the algorithm recursively for node N5, no frequent items are found in the tail.
We next return to step 3 and we select another item from the remaining tail $\{a^1, c^1\}$.
We select item a^1 to be X6, Y6 will be $\{c^1\}$. Ginf for node N6 will be the projection of Ginf on Y6 - $\{c^1\}$. T(X6) = {M1, M3}.
When call the algorithm recursively for node N6, no frequent items are found in the tail.
We then return to step 3, but since the condition in step 4 is verified we go to step 12.
Node N0 returns MFI = $\{a^0b^0c^1, c^0, a^1\}$.
So the exit data of the algorithm, the maximal frequent itemsets are MFI = $\{a^0b^0c^1, c^0, a^1\}$.

5. Experiments and results

We have developed an implementation of the algorithm in C, using a structure similar to that from the SmartMiner algorithm. We used C in order to better control the memory usage and execution speed of the algorithm.
The traditional form of input data for algorithms that search for frequent itemsets is a series of transactions containing items and having associated domain intervals. We have instead used an orthogonal view of the input data. We have first determined the frequent items, and then for each frequent item we have used a bitset containing one bit for each megatransaction from the database. The bit is 1 if the item appears in the megatransaction and 0 if it does not appear.
The program was benchmarked under a Windows XP operating system, on a PC with Intel Pentium 4 processor with a speed of 3GHz and memory of 1GB. The code has been written and compiled using Visual Studio 2003.
We have used both real and artificial data to test the performance of our algorithm.
The real data consists of two datasets, WINNER and LOSER, similar to those described in (Lee & Wang, 2007). They have been obtained from the values of 10 stock exchange indices for the trading days between January 1, 1991 to December 31, 2005: ASX All Ordinaries Index (ASX), CAC40 Index (CAC), DAX Index (DAX), Dow Jones Index (DOW), FTSE 100 INDEX (FTS), Hang Seng Index (HSI), NASDAQ Index (NDQ), Nikkei 225 Index (NKY), Swiss Market Index (SMI) and Singapore ST Index (STI). In the WINNER set a transaction for a trading day contains the stock indices whose value rises for the day, while in the LOSER set a transaction for a trading day contains the stock indices whose value falls for the day.
For both the WINNER and LOSER datasets we have used the sliding window size to be 4 (the maximum span of the intertransaction association rules). We have varied the minimum support threshold value from 4% to 12% and the results are presented in Fig. 1 and Fig. 2.
The same data sets (obtained from the same stock indices for the same period) were also used to evaluate the ITP-Miner algorithm in (Lee & Wang, 2007). The same sliding window size was used and the minimum support varied between the same values, while the program was run on a processor with similar properties to the one we used using Microsoft Visual C++ 6.0. We have observed a difference of an order of magnitude between the execution times, especially larger when the minimum support threshold decreases (and there are more frequent itemsets to be found). For example for the LOSER data set when the minimum support threshold is set at 4% InterTraSM takes less than 6 seconds, while the authors reported that ITP-Miner takes about 100 seconds.

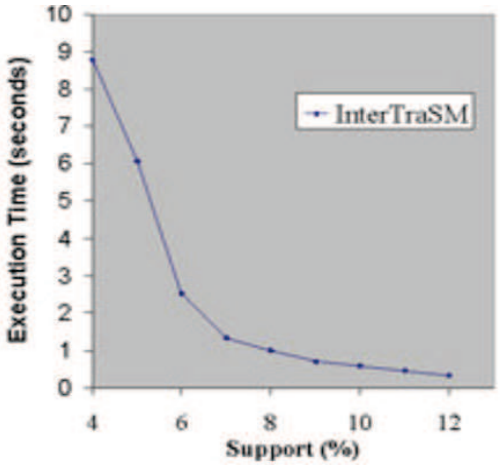


Fig. 1. Execution time vs minimum support, WINNER data set, w=4

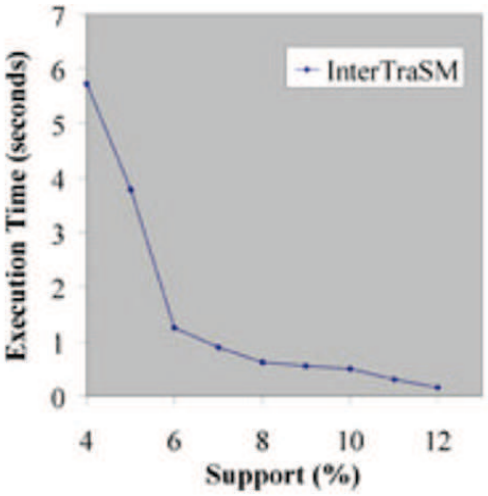


Fig. 2. Execution time vs minimum support, LOSER data set, w=4

The synthetic data was created using the generator described in (Luhr et al., 2007), gracefully provided to us by its authors.
The generation of data is a process with two steps:

- first a set of candidate frequent intertransaction itemsets is created
- then this set is used to populate with items the transactions from the dataset

We have generated two artificial datasets, representing sparse and dense data. This was the same method used in (Luhr et al., 2007) to evaluate the performances of the FITI and EFP-Tree algorithms.
The characteristics of the artificial data created are influenced by some parameters that guide the generation process. These parameters have the following values for the two artificial datasets we produced:

Parameter name	Sparse dataset	Dense dataset
Number of intratransactions	500	200
Size of the intertransaction pool	50	200
Average length of intratransactions	5	25
Maximum length of intratransactions	10	50
Average length of intertransactions	5	8
Maximum length of intertransactions	10	20
Maximum number of unique items	500	100
Maximum interval span of intertransactions	4	6

Table 1. Values of parameters used in the generation of the synthetic data sets

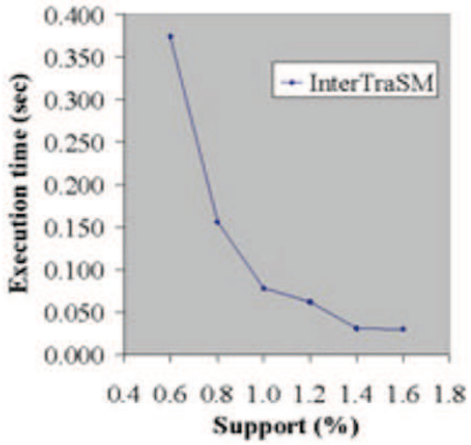


Fig. 3. Execution time vs minimum support, sparse data set, w=4

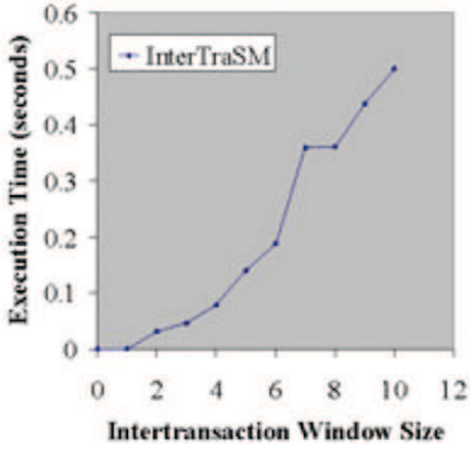


Fig. 4. Execution time vs sliding window size, sparse data set, minsup = 1%

We have compared the performance of InterTraSM and the performance of EFP-Growth for synthetic data sets created with the same parameters by the same generator (probably not

identical data sets since the actual generation is random, but the data sets have the same characteristics).

For the synthetic sparse data set:

- we have varied the minimum support threshold from 1.6% to 0.6%, with the sliding window size set to 4 – the results are in Fig. 3.
- we have varied the sliding window size from w=0 to w=10 while we have kept a fixed minimum support threshold of 1% - the results are in Fig. 4.

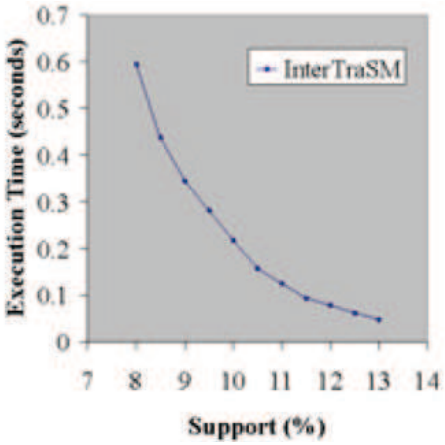


Fig. 5. Execution time vs minimum support, dense data set, w=6

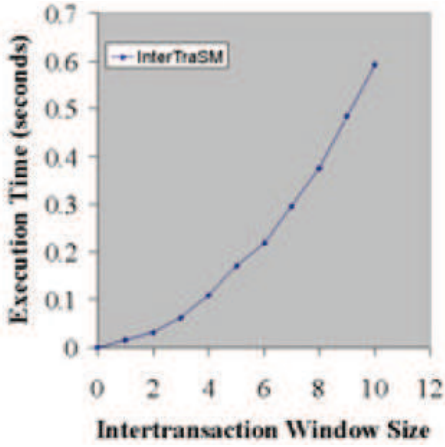


Fig. 6. Execution time vs sliding window size, dense data set, minsup = 10%

For the synthetic dense data set:

- we have varied the minimum support threshold from 13% to 8%, with the sliding window size set to 6 – the results are in Fig. 5.
- we have varied the sliding window size from w=0 to w=10 while we have kept a fixed minimum support threshold of 10% - the results are in Fig. 6.

Since the execution times observed here are all under 1 second and the execution times reported in (Luhr et al, 2007) on a similar processor have values of tens or evens hundreds of seconds, even considering for the different implementation environments we can conclude that InterTraSM generally performs at least an order of magnitude better than EFP-Growth.

6. Conclusion

We have approached in this paper the issue of searching for intertransaction association rules. We have presented the theoretical description of the problem and we have discussed the differences from the classical (intratransaction) association rule mining. We have presented some previous algorithms and then we have focused on InterTraSM - which uses a depth first search strategy and unlike the previous algorithms introduced it searches for maximal frequent itemsets. This reduces the number of support counting operations that need to be performed. Experiments performed with similar data and on similar processors with the ones used in previous algorithms show a difference of at least an order of magnitude in favour of InterTraSM.
In the future the algorithm should be applied on more real data sets and the performance should be measured. Also some interestingness measures for intertransaction association rules should be applied and the results should be analysed.

7. References

Agrawal, R. & Srikant, R. (1994). Fast algorithms for mining association rules, *Proceedings of the 20th International Conference on Very Large Databases (VLDB)*, pp. 487-499, Santiago, Chile, September 1994, Morgan Kaufmann

Han, J.; Pei, J. & Yin, Y. (2000). Mining frequent patterns without candidate generation, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pp. 1-12, Dallas, United States, May 2000, ACM

Lee, A.J.T. & Wang, C-S. (2007). An efficient algorithm for mining frequent inter-transaction patterns. *Information Sciences: an International Journal,* Vol. 177, Issue 17, (September 2007), pp. 3453-3476

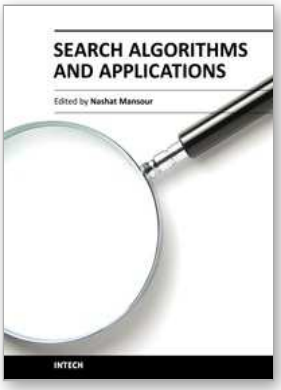
Lu, H.; Feng, L. & Han, J. (2000). Beyond intra-transaction association analysis: mining multi-dimensional inter-transaction association rules. *ACM Transactions on Information Systems,* Vol. 18, Issue 4, (October 2000), pp. 423-454

Luhr, S.; West, G. & Venkatesh, S. (2007). Recognition of emergent human behaviour in a smart home: A data mining approach. *Pervasive and Mobile Computing,* Vol. 3, Issue 2, (March 2007), pp. 95-116

Tung, A.K.H.; Lu, H.; Feng, L. & Han, J. (2003). Efficient mining of intertransaction association rules. *IEEE Transactions on Knowledge and Data Engineering,* Vol. 15, Issue 1, (January 2003), pp. 43-56

Ungureanu, D. & Boicea, A. (2008). Intertrasm - a depth first search algorithm for mining intertransaction association rules, *ICSOFT 2008 - Proceedings of the Third International Conference on Software and Data Technologies*, pp. 148-153, Porto, Portugal, July 2008, INSTICC Press 2008

Zou, Q.; Chu, W. & Lu, B. (2002). SmartMiner: a depth first algorithm guided by tail information for mining maximal frequent itemsets, *Proceedings of the 2002 IEEE International Conference on Data Mining*, pp. 570-577, Maebashi City, Japan, December 2002, IEEE Computer Society



Search Algorithms and Applications

Edited by Prof. Nashat Mansour

ISBN 978-953-307-156-5
Hard cover, 494 pages
Publisher InTech
Published online 26, April, 2011
Published in print edition April, 2011

Search algorithms aim to find solutions or objects with specified properties and constraints in a large solution search space or among a collection of objects. A solution can be a set of value assignments to variables that will satisfy the constraints or a sub-structure of a given discrete structure. In addition, there are search algorithms, mostly probabilistic, that are designed for the prospective quantum computer. This book demonstrates the wide applicability of search algorithms for the purpose of developing useful and practical solutions to problems that arise in a variety of problem domains. Although it is targeted to a wide group of readers: researchers, graduate students, and practitioners, it does not offer an exhaustive coverage of search algorithms and applications. The chapters are organized into three parts: Population-based and quantum search algorithms, Search algorithms for image and video processing, and Search algorithms for engineering applications.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821