Mining Long High Utility Itemsets in Transaction Databases

GUANGZHU YU, SHIHUANG SHAO and XIANHUI ZENG
Information and Technology College, DongHua University, Shanghai, CHINA
ygz@mail.dhu.edu.cn
shshao@dhu.edu.cn

Abstract: Existing algorithms for utility mining are column enumeration based, adopt an Apriori-like candidate set generation-and-test approach**,** and thus are inadequate on datasets with high dimensions or long patterns. To solve the problem, this paper proposes a hybrid model and a row enumeration based algorithm, i.e., inter-transaction, to discover high utility itemsets from two directions: existing algorithms such as UMining [1] can be used to seek short high utility itemsets from the bottom, while inter-transaction seeks long high utility itemsets from the top. By intersecting relevant transactions, the new algorithm can identify long high utility itemsets directly, without extending short itemsets step by step. In addition, new pruning strategies are used to cut down search space; optimization technique is adopted to improve the performance of the intersection of transactions. Experiments on synthetic data show that our method achieves high performance, especially in large high dimensional datasets.

Keywords: long high utility itemset; hybrid mode; utility; intersection transaction; row enumeration

1 Introduction

The main task of traditional association rule mining (ARM) is to identify frequent itemsets. It treats all the items equally by assuming that the utility of each item is always 1 (item is present) or 0 (item is absent). Obviously, it's unrealistic and will lead to some useful patterns missed. For example, in a transaction database, there are 1000 sale records of milk which occupy 10% of the total transaction number, contributing 1% of the total profit. In the meantime, there are 600 sale records of birthday cake that occupy 6% of the total transaction number, contributing 5% of the total profit. If the support threshold is 8%, according to traditional algorithms for frequent itemset mining, milk will be reported as a frequent itemset and birthday cake will be ignored. But in fact, the market professional must be more interested in birthday cake because it contributes a larger portion to total profit than milk. The example shows that support is not sufficient to reflect user's interest.

According to Expectancy Theory [2], we have the well-known equation "motivation = probability * utility", which says that motivation is determined by the utility of making a decision and the probability of success. In retailing field, users are not only interested in the frequency of occurrence of an itemset (support), but also their utility. So a decision-oriented ARM algorithm should output both the support and the utility of interesting patterns. For this reason, utility-based ARM has been proposed to discover all itemsets in a database with utility values higher than a user specified threshold.

Table 1 is an example of a simplified transaction database where the total utility value is 162. The number in each transaction in table 1 is the sales profit of each item. If s(X) and u(X) represent the support and utility of itemset X respectively, then u(A,B)=43, s(A,B)=5, u(A,B,C) =54, s(A,B,C)=3, u(A,B,C,D)=45, s(A,B,C,D)=2, u(A,B,C,D,E)=57.

Table1: A transaction database

	A	B	C	D	E
T1	0	0	5	0	1
T2	2	3	0	0	0
T3	3	5	15	7	4
T4	0	0	4	7	2
T5	4	5	8	0	0
T6	9	4	0	0	2
T7	6	0	8	3	6
T8	0	0	0	6	3
T9	3	0	0	9	5
T10	3	5	6	1	8

If the support threshold is 3 and the utility threshold is 50, {A,B} is a frequent but not a high utility itemset. On the other hand, {A,B,C} is both a frequent and high utility itemset, {A,B,C,D} is neither a frequent nor a high utility itemset and {A,B,C,D,E} is a high utility but non-frequent itemset.

From the above example, we can draw a conclusion: downward closure property doesn't apply to utility mining. Relevant studies have shown that utility constraint is neither anti-monotone, monotone, succinct, nor convertible [3][4]. Because of this property, most algorithms for frequent pattern mining [5][6][7][8][9][10] can't be used to find high utility itemsets.

Although lots of researches have been conducted to improve the usefulness of traditional ARM [11][12][13][14][15][16][17][18], they are all utility-related, not utility-based. To the best of our knowledge, only UMining [1][19] and Two-phase [20] can be used for utility mining, but both of them are Apriori-like algorithms and are inadequate on datasets with long patterns or high dimensions. To solve the problem, we propose a hybrid top-down/bottom-up search model and a partitioning-based algorithm, inter-transaction, to discover all high utility itemsets from two directions. Under the hybrid model, existing algorithm such as UMining [1] can be used to search the short high utility itemsets by starting from the bottom, while the inter-transaction searches long high utility itemsets by starting from the top, they complement each other.

Inter-transaction is based on the characteristic that there are few common items between or among long transactions, which means that the intersection of multiple long transactions is usually very short. In a high dimensional data environment, the characteristic is especially obvious. This paper emphasizes the introduction of inter-transaction.

2 definitions

Utility of an item is a subjective term dependent on users and applications; it could be measured in terms of profit, cost, risk, aesthetic value or other expressions of user preference. For easy understand, in this paper, we refer to utility of an item as the economic utility such as sales profit, and view all datasets as transaction database, so that we can define the utility of an item as the product of quantity sold and the unit profit of the item.

Let I={i_1, i_2, …, i_m} be a set of items, D={T_1, T_2, …, T_n} is a transaction database. Each transaction T_q in database D ($T_q \in D$) is a subset of I, i.e., $T_q \subseteq I$. To simplify a notation, we sometimes write a set { i_1, i_2, …, i_k } as $i_1 i_2 \ldots i_k$. Adapting from the notations described in [1] [20] and [21], we have following definitions:

Definition1. The transaction utility of item x in transaction T_q, denoted $u(x, T_q)$, is the utility brought on by item x when transaction T_q occur. Take the example from table 1, u(A,1)=0, u(A,2)=2.

Definition 2. The transaction utility of itemset X in transaction T_q, denoted $u(X, T_q)$, is the sum of the transaction utility of item x contained in X, i.e.,

$$u(X, T_q) = \sum_{x \in X \wedge X \subseteq T_q} u(x, T_q) \quad (1) \qquad (1)$$

For example, in table 1, u(AB,2)= u(A,2)+ u(B,2)=2+3=5, u(ABC,5)= u(A,5)+u(B,5)+u(C,5) =4+5+8=17.

Definition 3. The partition utility of itemset X in partition P_i , denoted $u(X, P_i)$, is the sum of the transaction utility of itemset X in partition P_i, i.e.,

$$u(X, P_i) = \sum_{T_q \in P_i \wedge X \subseteq T_q} u(X, T_q) \qquad (2)$$

For more details about partitions, refer to [21].

Definition 4. The utility of X in database, denoted $u(X)$, is the sum of the transaction utility of itemset X in database, i.e.,

$$u(X) = \sum_{X \subseteq T_q \wedge T_q \in P_i \wedge P_i \subseteq D} u(X, P_i) = \sum_{T_q \in D \wedge X \subseteq T_q} u(X, T_q) \qquad (3)$$

Examples can be seen in section 1.

Definition 5. The utility of transaction T_q, denoted $u(T_q)$, is the sum of the transaction utility of item x in transaction T_q, i.e.,

$$u(T_q) = \sum_{x \in T_q} u(x, T_q) \qquad (4)$$

Definition 6. Transaction identifier list, denoted tidlist, is a set of transaction ID.

Definition 7. Intersection transaction, denoted T(tidlist), is an itemset obtained from the intersection of transactions listed in tidlist. For example, let T_1=ABDF, T_2=ADFG, T_3=ADFQ, then one of the tidlists is {1,2,3}, and the corresponding intersection transaction T(1,2,3) = $T_1 \cap T_2 \cap T_3$ = ADF. If |tidlist|=k ($1 \le k \le N$, N is the number of transactions), we refer to T(tidlist) as k-intersection transaction.

Although a k-intersection transaction is actually an itemset, there are some differences between them. For example, a k-itemset means an itemset with k items. The term doesn't tell us any information about its support (i.e., the number of transactions containing the itemset); on the contrary, a k-intersection transaction doesn't tell us how many items the itemset has (maybe zero or a positive integer), but it tells us that the itemset stems from the intersection of k transactions, with support no less than k. To emphasize the difference between k and support s, we refer to k as the **current support** of a k-intersection transaction. Obviously, $k \leq s$. Transactions listed in a tidlist can form a partition. If u(T(tidlist)) represents the utility of itemset T(tidlist), u(T(tidlist),tidlist) represents the corresponding partition utility of T(tidlist), according to equation (2) and equation (3), u(T(tidlist),tidlist) should be less than u(T(tidlist)). We also refer to u(T(tidlist),tidlist) as the **current utility** of k-intersection transaction under current support. If k=s, the corresponding tidlist is called **maximal tidlist**, and u(T(tidlist),tidlist)=u(T(tidlist)). In fact, the meaning of the term "maximal tidlist" corresponds to the feature support set in [8].

Definition 8. A long transaction is the transaction that includes more than minlen items. Minlen is a user defined value. Otherwise, called short transaction. Likewise, we can define long/short itemset and long/short pattern, and so forth.

Definition 9. A high utility itemset is the itemset with a utility value higher than a user specified threshold, i.e., minutil. If an itemset is a high utility itemset, we say the itemset is high, otherwise, the itemset is low.

Definition 10. A long high utility itemset is the high utility itemset with length longer than minlen.

Definition 11. A local high utility itemset is an itemset in partition p_i with partition utility value higher than the local utility threshold minutil/n, n is the partition number. We also define local long high utility itemset as the local high utility itemset with more than minlen items. To emphasize the difference betweem local high utility itemset and high utility itemset, we also refer to a (long) high utility itemset as a global (long) high utility itemset. Similar definitions can refer to [21], except they are about frequent itemets.

3 Inter-transaction algorithm

Any high utility itemset must be in a closed itemset (pattern). This means if we can firstly identify all closed itemsets, then mine each closed itemset separately to find all high utility subsets that the closed itemset contains, all high utility itemsets can be identified. Like CARPENTER and TD-CLose [9], inter-transaction is based on row enumeration. Since each closed itemset can be expressed as an intersection transaction [8], mining all intersection transaction has the same power as mining all closed itemsets. In order to avoid the costly process of pattern matching and the complicated data structure such as X-conditional transposed table (which are used in CARPENTER), inter-transaction enumerates every intersection transaction and then computes the current utility values of all subsets that the intersection transaction contains. For details, refer to subroutine Gen-LHU-itemsets.

3.1. Partition method

If N is the number of transactions (rows), there will be 2^N combinations of transactions at the worst situation. As the number of transactions grows, the explosive growth of the combination of rows causes the performance of row-enumeration methods decrease dramatically. In a real database, the number of transactions can easily reach to several millions, and enumerating all the 2^N intersection transactions is not feasible. To solve the problem, the inter-transaction adopts a partition method to divide a database into multiple partitions, with each partition containing a fitting amount of transactions. In the first scan of a database, inter-transaction finds all local long high utility itemsets from every partition, and then these local long high utility itemsets are merged to generate a set of potential long high utility itemsets. In the second scan of the database, the actual utility and support for these itemsets are computed and global high utility itemsets are identified. The whole process is just like the one described in [21]. The correctness of the partition method is guaranteed by theorem 1:

Theorem 1 suppose D is a transaction database, $P=P_1, P_2, \ldots, P_j$ is a set of partitions of D ($\sum_{i=1}^{j} P_i = D, P_i \cap P_j = \Phi, i \neq j$). If $X \subseteq I$ is a high utility itemset, it will appear as a local high utility itemset in at least one of the partitions.

Proof. Let X be a high utility itemset, then $u(X) \geqslant$ minutil. Divide D into n partitions, then X may fall into m partitions ($1 \leqslant m \leqslant n$). Assume $B=\mathrm{Max}(u(X,P_i))$ denote the biggest partition utility value of X in all partitions, By definition 4, we have

$$u(X) = \sum_{X \subseteq T_q \wedge T_q \in P_i \wedge P_i \subseteq D} u(X, P_i) \leq mB$$

$$\text{If } B < \frac{\text{minutil}}{n}, \text{ then}$$

$$u(X) < \frac{m}{n}\text{minutil} \leq \text{minutil}$$

But this is a contradiction.

Let u be total utility value, and coefficient a be the minimum acceptable ratio of the utility value of an itemset to the total utility value in the database. Suppose we divide the database D into n partitions, the local utility threshold ($\frac{\text{minutil}}{\text{n}} = \frac{a*u}{\text{n}}$) should be far larger than the average transaction utility ($\frac{\text{u}}{\text{N}}$), denoted as $\frac{\alpha * \text{u}}{\text{n}} >> \frac{\text{u}}{\text{N}}$. Otherwise, a large amount of local high utility itemsets would be generated. Let S be partition size, we have:

$$S = \frac{N}{n} >> \frac{1}{a} \qquad (5)$$

Inequation (5) contradicts the goal of the partition method (reducing the amount of transactions in a partition). Experiments show that it's applicable for S to be between $5/a$ and $10/a$ in the context of our datasets.

3.2. Task Decomposing

If the partition number is n, the size of partitions will be N/n, and the total number of potential intersection transactions becomes $n2^{N/n}$. When N is too large, enumerating $n2^{N/n}$ intersection transactions is still not feasible. The partition method is insufficient in reducing the search space.

Generally, given a proper length threshold minlen, most of the patterns in a database are short. Although the number of long patterns is usually much smaller than that of short patterns, it's usually true for these long patterns to cost most of the resources in finding all high utility (or frequent) itemsets when a down-top method is adopted, since a long pattern always means a lot of short patterns have to be handled ahead. On the other hand, there are few common items between or among long transactions, especially in sparse high dimensional data, which means the intersection of multiple long transactions, i.e., intersection transaction is usually very short. The two characteristics are very useful for our algorithm because it can obtain long itemsets directly by intersecting relevant transactions, without extending a short itemset step by step to obtain a long itemset. On the contrary, short itemsets are relatively dense; the overhead of enumerating all intersection transactions (including short intersection transactions) will be too high. Based on the different features, it's reasonable for us to decompose the mining task into two subproblems (discovering long high utility itemsets and short high utility itemsets), so that we can choose proper algorithms to solve the subproblems separately.

If we aim to find long high utility patterns directly by intersecting relevant transactions, a new pruning strategy can be used to narrow the search space: filter out all short (intersection) transactions. The rationale behind the pruning strategy is that short transactions have no effect on the support or utility of long patterns/itemsets, and the intersection of a short transaction with another transaction must be short. Now that the intersection of two long transactions is usually very short, large amounts of intersection transactions can be pruned out in time.

3.3 algorithms

Based on the above discussion, inter-transaction can be described as follows:
Input: A database D, minutil, minlen
Output: All long high utility itemsets.

```
1) P=partition-database(D)        //divide D into multiple partitions
2) n=number of partitions
3) **for** i=1 to n **do begin**
4)     read-in-partition ($P_i \in P$)
5)     $H^i$=gen-LHU-itemsets($P_i$)
6) **end**
7) **for** (i=minlen; $H_i^j \neq \varphi, j = 1,2,...n$ ; i++) **do begin**
8)     $C_i^G = \bigcup_{j=1,2,...n} H_i^j$
9) **end**
10) **for** i=1 to n **do begin**
11)     read_in_partition ($P_i \in P$)
12)     **for all** candidates $c \in C^G$ compute utility c.utility in terms of equation (3), along with support c.count
13) **end**
14) $H^G$={ $c \in C^G$ |c.utility ≧ minutil}
15) Answer= $H^G$
```

Notations used in inter-transaction are shown in table 2. More details can refer to definition 11 in this paper and [21]. The algorithm is very similar to the partition algorithm [21], but there are two important differences between them. One is that in the partition algorithm, the size of partitions is chosen in terms of the main memory size, such that at least those itemsets and other information that is used for generating candidates can fit in the main memory, whereas inter-transaction seeks balance between keeping a higher local utility threshold and reducing the amount of transactions in a partition. The other difference is that in the partition algorithm a certain algorithm such as Apriori is used to generate local frequent itemsets of all length, whereas the inter-transaction discovers only local long high utility itemsets via enumerating intersection transactions.

Table 2: Notations used in algorithm

Notation	Meaning
H_k^p	Set of local high utility k-itemsets in partition p
H^p	Set of local high utility itemsets. $H^p = \bigcup_{k>minlen} H_k^p$
C_k^G	Set of global candidate k-itemsets
C^G	Set of global candidate itemsets. $C^G = \bigcup_{k>minlen} C_k^G$
H_k^G	Set of global high utility k-itemsets
H^G	Set of global high utility itemsets. $H^G = \bigcup_{k>minlen} H_k^G$

Gen-LHU-itemsets is responsible for generating local long high utility itemsets in a partition. In the subroutine, a tidlist is used to record which transactions are involved in an intersection transaction. If s(T(tidlist),tidlist) and u(T(tidlist),tidlist) represent the current support and current utility of a T(tidlist) respectively, T(tidlist).tidlist represents the transaction identifier list associated with T(tidlist), let tidlist= tidlist1 ∪ tidlist2 (tidlist1≠tidlist2), we have:

$$\begin{aligned}\mathrm{T(tidlist)} &= \mathrm{T(tidlist1 \cup tidlist2)} \\ &= \mathrm{T(tidlist1)} \cap \mathrm{T(tidlist2)}\end{aligned} \tag{6}$$

$$\begin{aligned}\mathrm{T(tidlist).tidlist} &= tidlist \\ &= \mathrm{tidlist1 \cup tidlist2}\end{aligned} \tag{7}$$

$$\begin{aligned}\mathrm{s(T(tidlist), tidlist)} &= |\mathrm{T(tidlist).tidlist}| \\ &= |\mathrm{tidlist1 \cup tidlist2}|\end{aligned} \tag{8}$$

$$\mathrm{u(T(tidlist), tidlist)} = \sum_{Tq \in tidlist} u(T(tidlist), T_q) \tag{9}$$

If X is a sub-itemset of T(tidlist) (X∈T(tidlist)), we can use equation (10) to compute the current utility of X under current support:

$$\mathrm{u(X, tidlist)} = \sum_{Tq \in tidlist \wedge X \in T(tidlist)} u(X, T_q) \tag{10}$$

Both equation (9) and equation (10) stem from equation (2). Subroutine gen-LHU-itemsets is described as follows:

Input: A partition P_i, minutil, minlen

Output: All local long high utility itemsets in P_i

1) Take a partition P_i and calculate the utility of each transaction ($1 \leqslant k \leqslant N$) T_q independently according to equation (4) for individual transaction or equation (9) for intersection transaction. If $u(T_q)$ or $u(T_q,tidlist)$ is more than minutil/n, put T_q into the set of local high utility k-itemsets: $H_k^p = H_k^p \cup T_q$ ($|T_q|$=k), then call procedure mine_single_trans;
2) Perform all the intersections of any two long transactions;
3) If there are no long transactions, the subroutine ends;
4) Filter out all the short intersection transactions;
5) Check all the long intersection transactions. If T(tidlist1)=T(tidlist2), merge the repetitious intersection transactions into a single one, i.e., T(tidlist), such that T(tidlist) = T(tidlist1) = T(tidlist2), tidlist = tidlist1 ∪ tidlist2. All the long intersection transactions can form a new partition, go to step 1.

Since the intersection of two transactions is not longer than any of the two transactions, the method has a good convergence.

Subroutine mine_single_trans tries to discover all local long high utility itemsets that an intersection transaction contains. It can be described as follows:

Input: T_q=t(tidlist), minutil, minlen //T_q can be an individual transaction or intersection transaction

Output: All local long high utility itemsets in T_q

Method:

```
1)  Sort the transaction $T_q$ decreasingly by its utility value: $T_q= t_0\ t_1\ t_2 \dots t_{k-1}\ t_k \dots t_{L-1}$, such that $u(t_i,T_q) \ge u(t_j,T_q) (i \le j)$;
2)  Let k=L-1;
3)  p=0;
4)  Let X=$t_p\ t_{p+1} \dots t_{p+k-1}$. Compute u(X, tidlist) according equation (10). if u(X, tidlist)>=minutil/n, add X into $H_k^p$, go to step 5, otherwise, the subroutine ends;
5)  For j=1 to k do begin
6)      Count=0;
7)      For i=p to L-k-1 do begin
8)          Replace $t_{k-j}$ in itemset X with $t_{k+i}$, obtaining a new itemset X'. If u(X',tidlist)>=minutil/n, add X' into candidate set $H_k^p$, count increases by one; if u(X',tidlist)<minutil/n, break (exit loop);
9)      End;
10)     If count=0, break;
11) End;
12) If there isn't a high utility k-itemset, the
```

subroutine ends; if all the k-itemsets verified in step 8 are high utility itemsets，p increases by one, go to step 4 until all the k-itemsets are verified;

13) Let k=k-1, go to step 3, until k=minlen;

The subroutine chooses a sorting algorithm to sort the items in descending order by utility value so that we can build and check only potential high utility itemsets, pruning out low utility itemsets as early as possible. Suppose there exist two itemsets $Y_i=X\cup t_i$, $Y_j=X\cup t_j$, if i<j, then $u(Y_i) > u(Y_j)$. If Y_i is high, we have to build and test Y_j to decide whether Y_j is high; if Y_i is low, Y_j must be low. In this way, we can prune out large amounts of itemsets which can be expressed as $X\cup t_k$, if only $X\cup t_i$ is low, and k>i. Another pruning strategy is, if all the k-itemsets are low, all (k-l)-itemsets must be low ($1\le l\le k-1$). Theorem 2 can guarantee the correctness of the pruning strategies.

Definition 13 Given two closed patterns $I'\subset I''\subseteq I$, if there exist no closed pattern I''', such that $I'\subset I'''\subset I''$, I' and I'' are adjoining each other.

Theorem 2 Given two adjoining closed patterns I' *and* I'', satisfying $I'\subset I''\subseteq I$. if I'' is not a high utility itemset, its subset I''' ($I'\subset I'''\subseteq I''$) must be low.

Proof. Let closed itemsets $I'=\{i_1,i_2,...i_m\}$, $I''=\{i_1,i_2,...i_m,...,i_n\}$, $I'''=I''-X\,(X\subseteq I''-I')$, By definition 13, I''' must be non-closed pattern, and $s(I''')=s(I'')$. (Shown in figure 3)

By definition 3, we have:

$$u(I'')=\sum_{T_q\in D\wedge I''\subseteq T_q} u(I'',T_q)=\sum_{T_q\in D\wedge (I'''+X)\subseteq T_q} u((I'''+X),T_q)$$

$$\ge \sum_{T_q\in D\wedge (I'''+X)\subseteq T_q} u((I'''),T_q)$$

$$=\sum_{T_q\in D\wedge I'''\subseteq T_q} u((I'''),T_q)=u(I''')$$

If $u(I'')\le\frac{1}{n}\min util$, then $u(I''')\le\frac{1}{n}\min util$. End.

The following example can show how the subroutine works. Suppose there are three transactions T_1, T_3, T_7, their intersection is equal to ABCDEF, i.e., T(1,3,7)=ABCDEF, and the corresponding utility values can be seen in table 3.

Table 3:An intersection transaction and item utility values

	A	B	C	D	E	F
T_1	2	1	1	0.3	1	0.5
T_3	3	1	2	0.3	1	0.5
T_7	1	2	2	0.4	1	1
Partition utility of items	6	4	5	1	3	2

Let T_q= T(1,3,7), corresponding item utility values are 6, 4, 5, 1, 3 and 2 respectively. After sorting, T_q can be expressed as ACBEFD, with item utility values decreasing gradually. Here the length of T_q is 6, i.e., L=6. If minutil=18, minlen=3, itemsets will be examined in the order shown in table 4.

Table 4: The process of calculating utility

	Itemset Utility	comments
1	u(ACBEF)= 20	add ACBEF to H_i
2	u(ACBED)= 19	add ACBED to H_i
3	u(ACBFD)= 18	add ACBFD to H_i
4	u(ACEFD)= 17	stop finding 5-itemsets
5	u(ACBE) =18	add ACBE to H_i
6	u(ACBF) =17	stop finding 4-itemsets
7	u(ACB) =15	Algorithm end

Although sorting an (intersection) transaction is costly, according to the step 1) of subroutine gen-LHU-itemsets, only a small amount of (intersection) transactions need to call the subroutine. In this way, mining an individual transaction doesn't cause high computational cost.

3.4 **Data layout alternatives for inter-transaction**

Conceptually, a database is a two-dimensional matrix and can usually be implemented in four different ways (HIV, HIL, VTV, and VTL) [22][23]. If we express each transaction in horizontal item-vector format (HIV), an intersection transaction can be obtained from the intersection of bit-vectors. Although the bitwise logical (And) operation is well supported by computer hardware and very efficient, the overall performance of the intersection of two transactions decreases dramatically with the increase of the number of items. For example, if the number of items is 8k, we have to use 1k bytes (8k bits) to express each transaction. In order to perform the intersection of two transactions, 8k bit operations are needed and this is intolerable. If two transactions are represented only in HIL format, the benefit of bitwise logical operations couldn't be shared. When the length of data in HIL format becomes long, the performance decreases dramatically.

Some optimization techniques such as run-length encoding (RLE), VIPER [22] and DIFFSET [23] have been proposed to enhance the performance of the intersection of two bit-vectors in vertical mining algorithms. But they have limited improvement on the speed of the intersection of long transactions, the reason may be that most of them aim at reducing the memory requirement of algorithms, and thus adopt various compressed formats to store databases. Because transaction intersection can't be performed directly in these compressed formats, extra format converting can't be avoided. So in our algorithm, we not only refrain from compressing each row in main memory, but also use redundant information to reduce the amount of bitwise logical operations.

Besides the HIV format, we also store each transaction in HIL format. Although this method will waste lots of memory, the cost is affordable because the partition method can save lots of memory. HIV format is used to perform the intersection of bit-vectors, while HIL format is used to store redundant information, which can guide us to choose only necessary bits in a bit-vector to perform bitwise logical (And) operation. Let S1 be the length of transaction T_1 in HIL format, S2 be the length of transaction T_2, if S1>S2, we choose T2 (the shorter transaction in HIL format) as the benchmark to determine on what bits the bitwise logical operation should be performed: if the k-th bit in T_2 is 1, bitwise And operation should be performed on the bit, all other bits should be set 0 in the result of intersection operation. For example, there are 3 transactions, the corresponding data can be seen in table 5:

Table 5: The process of computing T(1,2,3)

	HIV format	HIL format
T_1	1011,1100,1100,0110,001	1,3,4,5,6,9,10,14,15,19
T_2	***1***000,0000,0000,0000,0***11***	1,18,19
T(1,2)	***1***000,0000,0000,0000,00***1***	1,19
T_3	1011,1100,1101,0110,001	1,3,4,5,6,9,10,12,14,15,19
T(1,2,3)	1000,0000,0000,0000,001	1,19

If we want to get T(1,2,3), we can first get T(1,2) by intersecting transactions T_1 with T_2, then we get T(1,2,3) by intersecting T(1,2) with T_3. In order to get T(1,2), we choose the shorter transaction T_2 as the benchmark and decide bitwise And operations should be performed only on the first bit, the eighteenth bit and the nineteenth bit (written in bold Italic in table 5). As an intermediate result, we get T(1,2)={1,19}(in HIL format). In the subsequent process of intersecting T(1,2) with T_3, we choose T(1,2) as the benchmark and decide that bitwise And operations should be performed only on the first bit and the last bit. We get the final result T(1,2,3)= {1,19}. As shown in table 5, only five bit operations are needed for the whole process.

In this way, the amount of bit operations linearly depends only on the length of the short transaction in HIL format. The experiment shows this method outperforms VIPER and DIFFSET.

4 Experimental results

All the experiments were performed on a 2GHz Legend server with 4GB of memory, running windows 2003. The program was coded in Delphi 7.

Seven datasets were used in our experiments; all were generated by IBM quest data generator [24]. Six of them are T40.I30.D8000K with 0.5k, 1k, 2k, 4k, 8k and 16K items respectively, the seventh is T20.I6.D8000K with 4k items, where T# stands for the average length of transactions, I# for the average length of maximal potentially large itemsets and D# for the number of transactions. Because the generator only generates the quantity of 0 or 1 for each item in a transaction, we use Delphi function "RandG" to generate random numbers with Gaussian distribution, which mimic the quantity sold of an item in each transaction. The unit profit of each item is defined as item ID%100, where % is a modulus operator.

Figure 1 presents the scalability of inter-transaction by increasing the number of transactions from 0.25M to 8M. Experimental results show that our algorithm scales linearly with the number of transactions. We also modified our program to find long frequent itemsets, and a similar trend is obtained just as shown in figure 1. Obviously, mining frequent patterns takes fewer times than mining high utility itemsets.

Figure 2 shows the performance when varying the number of items. Different from other algorithms, the performance of inter-transaction improves as the number of items increases. The reason is that the number of items has a direct relationship with the sparseness of a dataset. The more items there are, the sparser the dataset, and the shorter the intersection of two transactions. That means inter-transaction can enumerate all long intersection transactions easily in a sparse dataset. From figure 2 we can observe that inter-transaction is suitable for those datasets with more than 1k items.

In figure 3, minlen is the minimum length of itemsets that the inter-transaction can discover within a reasonable time. It actually decides the task assigned to inter-transaction. Figure 3 shows that minlen decreases as the number of items increases, which means inter-transaction can complete more mining tasks in a sparse database. The reason is just the same as mentioned above.

Figure 4 shows that the size of partitions is very important for inter-transaction. Just as we have mentioned in section 3.1, a partition that is too small or too large will degrade the performance of the algorithm.

To test the total performance of the hybrid model, we choose Two-phase to mine short high utility itemsets and then compare the performance of the hybrid method (here hybrid method means inter-transaction + Two-phase) and Two-phase. That's to say, the total running time of the hybrid method is equal to the running time of inter-transaction (used to find long high utility itemsets) plus the running time of Two-phase (used to find short high utility itemsets). Experiments were performed on T20I6D8000K and T40I30D8000K, minlen is set 3 for T20I6D8000K and 5 for T40I30D8000K, the number of items is 4k. Corresponding performance curves are illustrated in figures 5 and 6.

From the two figures we can observe the hybrid model is not suited for dataset with only short patterns, this is because inter-transaction can't take obvious effect on these datasets. As for those datasets with lots of long patterns, Two-phase has to extend short itemsets step by step to obtain long itemsets, while inter-transaction obtains long itemsets directly by intersecting relevant transactions. In this situation, the hybrid method has great advantages over the Two-phase algorithm.

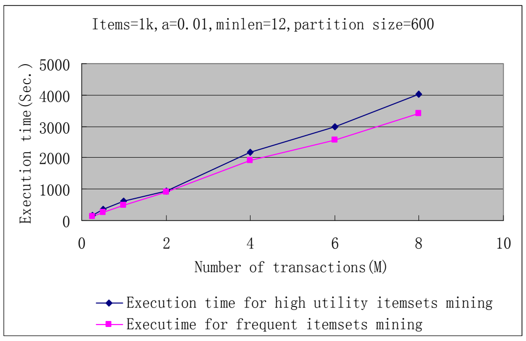

Fig.1. Scalability with the number of transactions

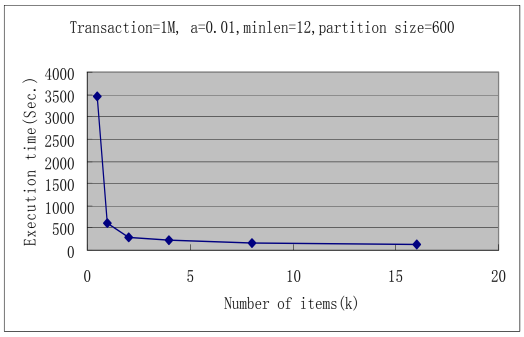

Fig.2. Scalability with the number of items

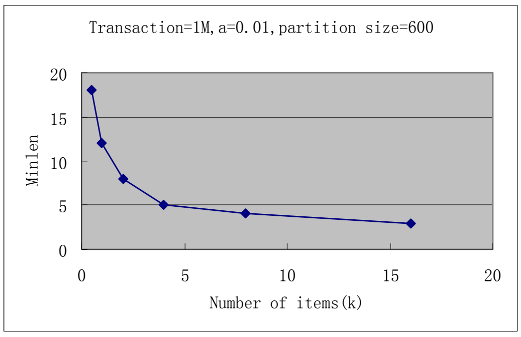

Fig.3. The effect of the number of items on minlen

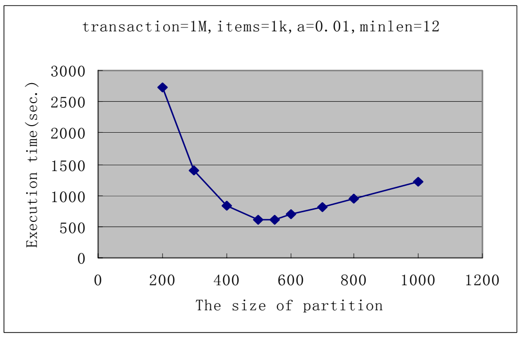

Fig.4. The effect of size of partitions on performance

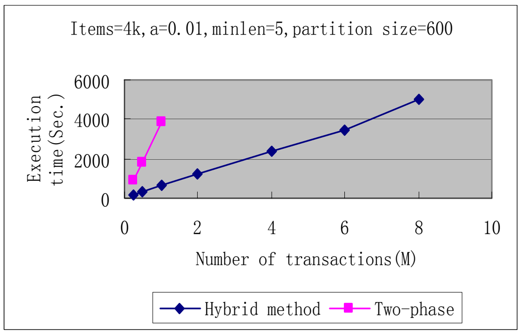

Fig.5. The execution time for T40I30D8000

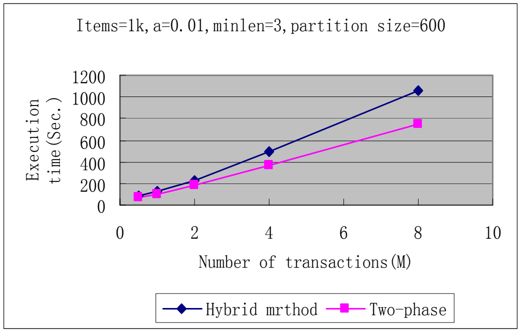

Fig. 6. The execution time for T20I6D8000

5 conclusions

The paper proposes a hybrid model to discover high utility itemsets from two directions. The intention of the hybrid model is to decompose a complex problem into two easy sub-problems, then use proper methods to solve them separately. Inter-transaction integrates the advantages of the partition algorithm and row enumeration algorithms. For example, it scans a database at most twice, is ideally suited for parallelization and large high dimensional databases, and so on. In the mean time, its performance is affected by the characteristics of the database, including data skew and the number of items. Parameters such as minimum length threshold minlen and the size of partitions also affect its performance.

References

References
1. Yao H. and Hamilton, H.J., Mining itemset utilities from transaction databases, Data & Knowledge Engineering, 59 , 2006, pp. 603 – 626
2. V. H. Vroom, Work and Motivation, John Wiley, 1964
3. Shen Y. D., Zhang Z. and Yang Q, Objective-oriented utility-based association mining, Proceedings of the 2002 IEEE International Conference on Data Mining, 2002, pp. 426-433
4. Yao H. and Hamilton H.J., A Unified Framework for Utility Based Measures for Mining Itemsets, Proceedings of the 2006 International Workshop on Utility-Based Data Mining, Philadelphia, PA, 2006, pp. 28-37
5. Han J.W., Pei J. , Yin Y. and Mao R, Mining Frequent Patterns without Candidate Generation: A Frequent pattern Tree Approach, Data Mining and Knowledge Discovery, 8, 2004, pp. 53-87
6. Ramesh C. Agarwal, Charu C. Aggarwal and V.V.V. Prasad, A tree projection algorithm for generation of frequent itemsets, Journal of Parallel and Distributed Computing, 61(3), 2000, pp. 350-371
7. Bayardo R, Efficiently mining long patterns from databases. Proceedings of the ACM SIGMOD International Conference on Management of Data, 1998, pp. 85-93
8. Feng P., Gao. C.and et al.. CARPENTER: Finding closed patterns in long biological database. Proc. of SIGKDD, Washington, 2003, pp. 413-419
9. Hongyan Liu, Jiawei Han and et al. Mining frequent Patterns from Very High Dimensional Data: A Top-down Row Enumeration Approach. Proceedings of the Sixth SIAM International Conference on Data Mining, Bethesda, Maryland, 2006, pp. 20-22
10. Feng P., Gao C., and et al. COBBLER: combining column and row enumeration for closed pattern discovery**.** Proceedings of the 16th International Conference on Scientific and Statistical Database Management, 2004, pp: 21-30
11. K. Wang, S. Zhou, J. Han, Profit mining: from patterns to action, Proceedings of International Conference on Extending Database Technology, 2002, pp. 70-87
12. Lin TY, Yao YY, Louie E. Mining Value Added Association rules, Proceedings of PAKDD, 2002, pp. 328-333
13. Aumann Y., Lindell Y. , A Statistical Theory for Quantitative Association Rules, Journal of Intelligent Information Systems, 20, 2003, pp. 255–283
14. Geoffrey I. Webb, Discovering associations with numeric variables, Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, 2001, pp. 383-388
15. C.H. Cai, Ada W.C. Fu, C.H. Cheng and W.W. Kong, Mining Association Rules with Weighted Items, Proceedings of the International Database Engineering and Applications Symposium, 1998, pp. 68-77
16. Lu S.F., Hu H.P. and Li F, Mining weighted association rules, Intelligent Data Analysis, 5, 2001, pp. 211-225
17. Chan R., Yang, Q., and Shen, Y.D., Mining high utility itemsets, Proceedings of the 3rd IEEE International Conference on Data Mining, 2003, pp.19-26
18. Barber, B., and Hamilton, H. J., Extracting Share Frequent Itemsets with Infrequent Subsets, Data Mining and Knowledge Discovery, 7, 2003, pp. 153-185
19. Yao H., Hamilton, H.J. and Butz, C.J. A Foundational Approach to Mining Itemset Utilities from Databases. Proceedings 2004 SIAM International Conference on Data Mining, 2004, pp. 482-486
20. Liu, Y., Liao, W.-K., and Choudhary, A fast high utility itemsets mining algorithm, Proceedings of the First International Workshop on Utiliy-based Data Mining, 2005, pp. 90-99
21. Savasere A, Omiecinsky E and Navathe S (1995). An efficient algorithm for mining association rules in large databases. 21st Int'l Conf. on Very Large Databases, Zurich, Switzerland, 1995, 432-444
22. Shenoy P, Haritsa J. R. and et al. Turbo-charging Vertical Mining of Large Databases. Proceedings of ACM SIGMOD International Conference on Management of Data, Dallas, Texas, 2000, pp. 22-33
23. Zaki M J, Gouda K. Fast vertical mining using diffsets. In Proc. of ACM SIGKDD, Washington DC, 2003, PP. 326-335
24. http://www.almaden.ibm.com/cs/projects/iis/hdb/Projects/data_mining/datasets/syndata.html