Isolated items discarding strategy for discovering high utility itemsets

Yu-Chiang Li[a*], Jieh-Shan Yeh[b], and Chin-Chen Chang[c, d]

[a] Department of Computer Science and Information Engineering,
Southern Taiwan University,
No. 1, Nantai St., Yung-Kang City, Tainan 710, Taiwan, R.O.C.
E-mail: lyc002@mail.stut.edu.tw

[b] Department of Computer Science and Information Management,
Providence University,
200 Chung-Chi, Shalu, Taichung 433, Taiwan, R.O.C.
E-mail: jsyeh@pu.edu.tw

[c] Department of Information Engineering and Computer Science,
Feng Chia University,
100 Wenhwa, Seatwen, Taichung 40724, Taiwan, R.O.C.
[d] Department of Computer Science and Information Engineering,
National Chung Cheng University,
168, University Rd., San-Hsing, Min-Hsiung, Chiayi 62102, Taiwan, R.O.C.

Abstract.

Traditional methods of association rule mining consider the appearance of an item in a transaction, whether or not it is purchased, as a binary variable. However, customers may purchase more than one of the same item, and the unit cost may vary among items. Utility mining, a generalized form of the share mining model, attempts to overcome this problem. Since the Apriori pruning strategy cannot identify high utility itemsets, developing an efficient algorithm is crucial for utility mining. This study proposes the Isolated Items Discarding Strategy (IIDS), which can be applied to any existing level-wise utility mining method to reduce candidates and to improve performance. The most efficient known models for share mining are ShFSM and DCG, which also work adequately for utility mining as well. By applying IIDS to ShFSM and DCG, the two methods FUM and DCG+ were implemented, respectively. For both synthetic and real datasets, experimental results reveal that the performance of FUM and DCG+ is more efficient than that of ShFSM and DCG, respectively. Therefore, IIDS is an effective strategy for utility mining.

Keywords**:** Data mining; Association rule; Utility mining

1. Introduction

The development of data mining techniques has focused on efficiently discovering hidden information from large databases that is useful for corporate decision-makers [20]. In recent years, data mining has become an important field of research [13]. Association rule mining [2, 3] is widely used to solve data mining problems in numerous applications, including financial analysis, the retail industry, and business decision-making [13].

In a transaction database where each transaction is a set of items or products, the application of association rules identifies interesting itemsets from the database [2, 3]. Traditionally, an association rule is interesting if its support and confidence values are not less than the minimum support (*minSup*) and minimum confidence (*minConf*) thresholds. An itemset *X* is frequent if the support value of *X* satisfies the *minSup* requirement. Using discovered frequent itemsets can directly generate the corresponding association rules. Accordingly, research on association rule mining usually focuses on establishing efficient methods to identify all frequent itemsets. Numerous efficient methods have been proposed to discover frequent itemsets, such as level-wise algorithms [2, 3, 7, 8, 12, 30] and pattern-growth methods [1, 15, 16, 21, 26].

In many applications, the importance of each item to the user varies. Cai et al. [9] first assigned item weights to overcome this problem. The weight of an item indicates the profitability of the product. Several researchers have proposed weighted association rule schemes [29, 33], but these algorithms still employ support values of itemsets to measure their importance. Support values only consider whether an item is bought in a transaction. The appearance of each item in a transaction is regarded as a binary variable, which does not reflect the quantities or prices of items purchased in each transaction. Table 1 shows a sample transaction database that includes six transactions. The series of numbers in the column “Count” indicates the sale amount for each item in each transaction. Item *B* appears in four transactions; therefore, according to the definition of the support value, item *B* has a support count of four. However, the total sale amount of item *B* is nine (1 + 4 + 1 + 3).

In reality, multiple quantities of a product may be bought in one transaction. An item should be weighted differently for each transaction, even if each transaction has the same length; thus, deriving interesting itemsets from support values may be misleading. Carter et al. [10] propose the share-confidence model to discover useful knowledge about numerical attributes associated with items in a transaction. Several other methods have since been proposed to efficiently discover share-frequent (SH-frequent) itemsets with infrequent subsets [4-6, 17, 18, 22-24]. Yao et al. [34, 35] generalize the share-confidence model [6] to develop the conventional *utility mining* model. This model can be used to measure the *utility* of an itemset in terms of net profit, total cost, or time spent [27, 28, 34, 35].

Applications may have different objectives for various data models; thus, there is no single measure that is suitable for every application. Recently, Yao et al. [36] attempted to build a unified framework for utility-based measures [11, 27, 28, 32, 34-36] that allows the user to select a suitable utility mining tool for a specific application; however, this framework only employs existing tools. Thus, to effectively discover high utility itemsets, the need for efficient algorithms remains urgent.

This study focuses on conventional utility mining. In the conventional utility mining model, an item has both *internal* and *external* utility [35]. The internal utility of an item is the numerical value assigned to it in a transaction, for example, the quantity of an item purchased in a transaction. The external utilities of all items are stored in a *utility table* (i.e. unit profit table or unit cost table). Table 2 provides an example that lists the unit profit for each item. For example, selling one unit of product *A* results in a profit of three dollars. Using the sample database in Table 1 and its associated utility table (Table 2), users can compute the total profit from each itemset. The utility of an itemset is the summation of its item utilities, which are the products of items' internal and external utilities, in each transaction. Consider the transaction database in Table 1 with the external utility values found in Table 2. The utility value of {*A*, *B*} is $(1\times 3+1\times 2)+(3\times 3+1\times 2)=16$, since {*A*, *B*} is only contained in *T01* and *T05*. Based on the sample data, selling products *A* and *B* together will yield a total profit of $16.

Table 1. Example of a transaction database with counting

TID	Transaction	Count
T01	{A, B, C, D, G, H}	{1, 1, 1, 1, 1, 1}
T02	{A, C, E, F}	{4, 3, 1, 2}
T03	{A, C, E}	{4, 3, 3}
T04	{B, C, D, F}	{4, 1, 2, 2}
T05	{A, B, D}	{3, 1, 2}
T06	{B, C, D}	{3, 2, 1}

Table 2. Example of a utility table

Item	A	B	C	D	E	F	G	H
Profit ($)	3	2	1	3	5	2	8	4

The profitability of an itemset or the total cost of stocking an itemset cannot be determined using the support value alone. Thus, in practical terms, utility mining can be more useful than traditional association rule mining.

Example 1.1 Consider the sample database in Table 1 and the unit profit for each item in Table 2. Suppose that the goal of a sales manager is to determine which itemsets can generate a profit greater than the target value (i.e. the threshold). Table 3 lists the 18 most frequent itemsets with the profit of each itemset. Using a traditional association rule mining tool, such as Apriori, with a support threshold of 40% causes {*A*, *E*} and {*A*, *C*, *E*} to be neglected, even though they are the two highest profit itemsets in the utility model. Appearing in transactions *T02* and *T03*, the profits of the itemset {*A*, *C*, *E*} is $(4\times 3+3\times 1+1\times 5)+(4\times 3+3\times 1+3\times 5)=50$. Therefore, utility mining is more beneficial than traditional association rules in such scenarios.

Table 3. Frequent itemsets with support values of at least 33%

Support	Itemset and its profit
83.3%	{C: 10}
66.7%	{A: 36}, {B: 18}, {D: 18}, {B, D: 36}
50.0%	{A, C: 34}, {B, C: 20}, {C, D: 16}, {B, C, D: 32}
33.3%	{E: 20}, {F: 8}, {A, B: 16}, {A, D: 21}, {A, E: 44}, {C, E: 26}, {C, F: 12}, {A, B, D: 25}, {A, C, E: 50}

Given a predefined minimum utility (*minUtil*) threshold, an itemset is considered high utility if its utility value is greater than or equal to the threshold value; otherwise, the itemset has a low utility value. The goal of utility mining is to discover all high utility itemsets in a transaction database using the utility table. The share-confidence model (herein referred to as share mining) is a variant of utility mining. If the internal utility value of each item is multiplied by its external utility value in each transaction, an SH-frequent itemset can be derived and called a high utility itemset. The algorithms for discovering SH-frequent itemsets can easily be modified to find high utility itemsets. Therefore, utility mining methods described in this study will also encompass share mining methods.

A high utility itemset often includes some low utility subsets but may not include any high utility subset. Consequently, the downward closure property of Apriori [2, 3] can not be directly applied to discover high utility itemsets. Intuitively, an exhaustive search method can be applied to identify all high utility itemsets. However, such a method is too time-consuming for a large dataset environment. Several heuristic methods have been proposed to accelerate the discovery of high utility (or SH-frequent) itemsets, such as the MEU (UMining_H) [27, 28, 34, 35], SIP, CAC, and IAB [4, 6] methods. Nevertheless, these predictive methods may not discover some high utility itemsets. Recently, Li et al. first developed some efficient approaches, including the FSM, SuFSM, ShFSM, and DCG methods, to identify all SH-frequent itemsets [22-24]. In the meanwhile, Liu et al. also presented a Two-Phase (TP) method to discover all high utility itemsets [27, 28].

The performances of existing level-wise utility mining methods primarily depend on the number of candidates generated in each pass. The challenge of utility mining is how to effectively reduce the number of candidates. This study proposes the Isolated Items Discarding Strategy (IIDS), which can be applied to each level-wise utility mining method to further reduce the number of redundant candidates. In each pass, a utility mining method with IIDS scans a database that is smaller than the original by skipping isolated items to efficiently improve performance. This study focuses on the task of efficiently discovering all high utility itemsets.

The rest of this paper is organized as follows: Section 2 presents the background and an overview of the current methods for solving the problem of utility mining. The methods of share mining that also work well for utility mining are presented in Section 3. Section 4 explains the proposed isolated items discarding strategy. Section 5 provides experimental results and evaluates the performance of the proposed strategy. Finally, we conclude in Section 6 with a summary of our work.

2. Background and related work

2.1. Support-confidence model

Given a transaction database, the application of association rule mining attempts to discover significant relationships among items. The formal definition is as follows.

Let $I = \{i_1, i_2, \ldots, i_n\}$ be the set of items. Let $DB = \{T_1, T_2, ..., T_z\}$ be the transaction database, where T_q is a transaction in DB and is a subset of I. That is, $\forall\, T_q \in DB$, $T_q \subseteq I$, $1 \le q \le z$. Let X be a set of items, called an itemset. If X is a subset of a transaction T_q, T_q is said to support X. The notation $X \Rightarrow Y$ expresses the form of an association rule, where $X \subseteq I$, $Y \subseteq I$ and $X \cap Y = \phi$. The two attributes *support* and *confidence* of each rule must satisfy the two user pre-defined *minSup* and *minConf* thresholds, respectively. If s% of transactions in DB contain $X \cup Y$, the support value of $X \Rightarrow Y$ is s%. The itemset $X \cup Y$ with length k is called a frequent k-itemset if its support value is not less than *minSup*. The rule $X \Rightarrow Y$ has the confidence value c% if the transactions containing X in DB in which c% of them also contain Y.

Apriori is a multiple passes algorithm [2, 3], is the best-known method for discovering frequent itemsets. The Apriori principle states that each subset of a frequent itemset must be frequent; otherwise the itemset is infrequent. This property is also called the downward closure property or the anti-monotone property. In each pass, Apriori scans a database once and employs the downward closure property to filter out many useless candidates.

2.2. Formal description of utility mining

Share mining has been proposed to overcome the shortcomings of traditional data mining, which overlooks the variance in sale quantity and price/profitability among items in a transaction [6]. Utility mining, a generalized form of the share mining model, is based on measuring internal and external utilities [35]. Given a transaction database, a minimum utility threshold, and a utility table, the goal of utility mining is to discover all high-utility itemsets. According to the problem statement and the definitions in [35], the notations and definitions of utility mining, with some modifications for consistency, are described as follows:

Definition 2.1. A *k*-itemset *X* has an associated set of transactions in *DB,* denoted as DB_X, where $DB_X = \{T_q \in DB \mid X \subseteq T_q \subseteq I\}$. For example, in Table 1, $DB_{\{C,D\}}$ = {*T01*, *T04*, *T06*}.

Definition 2.2. The internal utility value of item i_p in transaction T_q, denoted as $iu(i_p, T_q)$, is the value of i_p in T_q. For example, in Table 1, $iu(C, T02) = 3$.

Definition 2.3. The external utility of item i_p in a transaction database, denoted as $eu(i_p)$, is the value of i_p in the utility table of the database. For example, in Table 2, $eu(C) = 1$ and $eu(D) = 3$.

Definition 2.4. The utility value of item i_p in transaction T_q, denoted as $util(i_p, T_q)$, is the product of $iu(i_p, T_q)$ and $eu(i_p)$. That is, $util(i_p, T_q) = iu(i_p, T_q) \times eu(i_p)$, where $i_p \in T_q$. For example, in Tables 1 and 2, $util(B, T04) = 4 \times 2 = 8$. Intuitively, this can be viewed as when a dealer sells four *B*s and yields a profit of eight dollars in the transaction *T04*. The utility value of itemset *X* in transaction T_q, denoted as $util(X, T_q)$, is the sum of the utility value of each item of *X* in T_q, where $util(X, T_q) = \sum_{i_p \in X \subseteq T_q} util(i_p, T_q)$. For example, in Tables 1 and 2, $util(\{C, E, F\}, T02) = util(C, T02) + util(E, T02) + util(F, T02) = 3\times1 + 1\times5 + 2\times2 = 12$.

In particular, $util(T_q, T_q)$ is called the *transaction utility value* of T_q. Table 4 lists the transaction utility values of the sample database in Table 1.

Table 4. Transaction utility values of the sample database in Table 1

Transaction	T01	T02	T03	T04	T05	T06	Tutil(DB)
Transaction utility	21	24	30	19	17	11	122

Definition 2.5. The local utility value of an itemset *X* in *DB*, denoted as *Lutil*(*X*), is the sum of the itemset utility values of *X* in *DB*$_X$. That is, $Lutil(X) = \sum_{T_q \in DB_x} util(X, T_q)$ $= \sum_{T_q \in DB_x} \sum_{i_p \in X \subseteq T_q} util(i_p, T_q)$. For example, in Table 1, *Lutil*({*C*,*D*}) = *util*({*C*, *D*}, *T01*) + *util*({*C*, *D*}, *T04*) + *util*({*C*, *D*}, *T06*) = 4 + 7 + 5 = 16.

Definition 2.6. The total utility value of *DB*, denoted as *Tutil*(*DB*), is the sum of all transaction utility values in *DB*. That is, *Tutil*(*DB*) = $\sum_{T_q \in DB} util(T_q, T_q)$. For example, *Tutil*(*DB*) = 122 as shown in Table 4.

Definition 2.7. The utility value of itemset *X* in *DB*, denoted as *UTIL*(*X*), is the ratio of the local utility value of *X* to the total utility value in *DB*. That is, *UTIL*(*X*) = $\frac{Lutil(X)}{Tutil(DB)}$. In other words, *UTIL*(*X*) indicates the percentage of the utility value that itemset *X* contributed in *DB*. For example, in Table 1, $UTIL(\{C, D\}) = 16/122 = 13.1\%$.

Henceforth, in this study, the utility value of an itemset *X* refers to *UTIL*(*X*), except where indicates otherwise.

Definition 2.8. Given a *minUtil* value, if $UTIL(X) \geq minUtil$, the itemset *X* is a high utility itemset; otherwise *X* is a low utility itemset. The local utility value of the threshold is called the *minimum local utility value*, denoted as *minLutil*. Clearly, $minLutil = minUtil \times Tutil(DB)$.

Example 2.1. Consider the transaction database presented in Table 1 and *minUtil* = 30%. Table 5 lists the local utility value and the utility value of each 1-itemset, where

Tutil(*DB*) = 122. Let *X* = {*A*, *C*, *E*}, *Lutil*(*X*) = *util*(*X*, *T02*) + *util*(*X*, *T03*) = 20 + 30 = 50. Therefore, *UTIL*(*X*) = $\frac{Lutil(X)}{Tutil(DB)}$ = 50/122 = 41.0% ≥ 30%. The itemset *X* is a high utility itemset. Table 6 lists all high utility itemsets.

Although there are two high utility itemsets in the sample database as listed in Table 6, Table 5 shows that there are no high utility 1-itemsets (the utility values of 1-itemsets are all less than 30%). Thus, in Example 2.1, applying the downward closure property to the utility mining model will reveal no high utility itemsets.

Table 5. All local utility values of 1-itemsets of Table 1

1-itemset *X*	{A}	{B}	{C}	{D}	{E}	{F}	{G}	{H}	Total
Lutil(*X*)	36	18	10	18	20	8	8	4	122
UTIL(*X*)	29.5%	14.8%	8.2%	14.8%	16.4%	6.6%	6.6%	3.3%	100%

Table 6. All high utility itemsets

High utility itemset *X*	{A, E}	{A, C, E}
Lutil(*X*)	44	50
UTIL(*X*)	36.1%	41.0%

2.3. Existing algorithms

Exhaustive search methods, such as ZP and ZSP [4, 6], can discover all high utility itemsets in a database but may be excessively time-consuming for real-world applications. On the other hand, predictive approaches generally cannot ensure that the mining result contains the complete set of high utility itemsets [5, 6, 34, 35]. To address this urgent problem, Li et al. proposed the FSM algorithm, a non-exhaustive search method, to discover all SH-frequent itemsets [22]. Liu et al. presented a Two-Phase (TP) algorithm for the same purpose [27, 28]. Li et al. also suggested efficient algorithms such as ShFSM and DCG [23, 24].

TP and ShFSM methods employ similar properties to speed up the mining process. The greatest difference between the two methods is that TP has two phases, a level-wise process with multiple passes in the first phase (Phase I) and an extra *DB* scanning pass in the second phase (Phase II). ShFSM does not require that the additional second phase.

ShFSM relies on the critical function value of each candidate to determine which candidates are useless. Let X be a candidate k-itemset, where $k > 0$. If the critical function value of X, $CF(X)$, is less than the minimum threshold, then no superset of X can be SH-frequent [23]. Therefore, in the k-th pass, ShFSM scans the database to calculate the share value of each itemset. Then, ShFSM removes all useless candidate k-itemsets and employs the remaining candidates to generate the candidate (k+1)-itemsets for the next pass.

However, ShFSM does require the join and prune steps of candidate generation in each pass. Therefore, Li et al. proposed the Direct Candidates Generation (DCG) algorithm to improve the performance of the mining process [24]. DCG is a level-wise method that DCG maintains an array for each candidate during each pass. The array of each candidate k-itemset stores the critical function values of its (k+1)-supersets. Thus, after the k-th pass, DCG discovers all SH-frequent k-itemsets and directly generates all candidate (k+1)-itemsets for the next pass without join and prune steps.

3. Utility mining using share mining methods

Given a transaction database with a utility table, if the value of i_p in each transaction is replaced by $iu(i_p, T_q) \times eu(i_p)$, then a utility mining task simply becomes a share mining process. Therefore, each approach for discovering share-frequent itemsets also works well in utility mining.

Definition 3.1. Let X be a k-itemset. A superset of X with length $k + i$ contained in a transaction T_q is denoted as X^{k+i}, where $X \subset X^{k+i} \subseteq T_q \in DB$ and $i > 0$. For example, in Table 1, let $X = \{C, E\}$. $X^{k+1} = \{A, C, E\}$ or $\{C, E, F\}$, since $\{A, C, E\}$ and $\{C, E, F\}$ both contain X, have three elements, and appear in at least one transaction in DB.

Definition 3.2. Let X^{k+i} be an arbitrary (k+i)-superset of k-itemset X, where $i > 0$. Function $CF(X)$ is a critical function of X if $Lutil(X^{k+i}) \leq CF(X)$ for all X^{k+i}. That is, $CF(X)$ is the upper bound of the utility value of X's (k+i)-supersets.

Theorem 3.1. Let X^{k+i} be an arbitrary (k+i)-superset of k-itemset X, where $i > 0$. Assume that there exists a critical function $CF(X) \geq Lutil(X^{k+i})$ for all X^{k+i}. If $CF(X) <$ *minLutil*, then no superset of X has high utility.

The proof of Theorem 3.1 is provided in Appendix A.

According to Theorem 3.1, if $CF(X) <$ *minLutil*, then no superset of X has high utility. Thus, X can be removed from the candidate set after all high utility itemsets having a length less than or equal to $|X|$ are obtained, and the inequality also holds.

The goal of the existing algorithms for utility mining is to efficiently eliminate useless candidates in each pass. Reducing the critical function value of each itemset increases the performance of the utility mining process. Therefore, the calculation of $CF(X)$ plays an important role in utility mining. In the next section, this study introduces the strategy of isolated items discarding, which can be applied to existing utility mining methods to further reduce the number of candidates and to improve performance. This study applies the two best share mining methods, ShFSM and DCG, to the utility mining model. ShFSM is an efficient and typical method for share mining. The ShFSM method, modified [23] for utility mining is provided as follows:

Definition 3.3. Let X be a k-itemset. The set which contains all X^{k+i} in DB is denoted as $S(X^{k+i})$, that is, $X^{k+i} \in S(X^{k+i})$. For example, in Table 1, Let $X = \{C, E\}$, $S(X^{k+1}) = \{\{A, C, E\}, \{C, E, F\}\}$.

Definition 3.4. Given a transaction database and a k-itemset X, $db_{S(X^{k+i})}$ is the set of transactions in which each transaction contains at least one X^{k+i} in DB, where $i > 0$. For example, in Table 1, if $X = \{C, E\}$ and $i = 1$, then $db_{S(X^{k+1})} = \{T02, T03\}$.

Lemma 3.1. $db_{S(X^{k+i})} = db_{S(X^{k+1})}$, for any k-itemset X and any $i > 0$.

Theorem 3.2. Given a transaction database DB, let X^{k+i} be an arbitrary $(k+i)$-superset of k-itemset X, where $i > 0$, then $Lutil(X^{k+i}) \leq Tutil(db_{S(X^{k+1})})$.

Their proofs are detailed in Appendix A.

ShFSM utilizes $Tutil(db_{S(X^{k+1})})$ as the critical function value of X. According to Definition 3.2 and Theorem 3.1, if $Tutil(db_{S(X^{k+1})}) < minLutil$, then all supersets of the k-itemset X are low utilities. The value of $Tutil(db_{S(X^{k+1})})$ can easily be obtained by scanning DB. An arbitrary candidate X can be pruned if $CF(X) < minLutil$. Therefore, ShFSM avoids producing too many unnecessary candidates. ShFSM is a multiple-pass algorithm. In the k-th pass, ShFSM generates candidate k-itemsets, then scans the database once to determine high utility itemsets and compute the critical function value of each candidate. After scanning the database, the remaining set of candidate k-itemsets is called RC_k, in which the critical function value of each candidate is above $minLutil$. Applying the Apriori join and prune steps to RC_k, ShFSM generates candidate

(k+1)-itemsets for the next pass.

The DCG method is efficient for discovering SH-frequent itemsets [24]. To discover high utility itemsets, like ShFSM, we simply replace the share value of each item with its utility value in the dataset and properly set up the minimum threshold. DCG can be easily utilized for mining high utility itemsets.

4. Proposed Strategy and Algorithms

As described in the previous section, a well-designed critical function not only greatly reduces the number of candidate itemsets, but also significantly increases the performance of the mining process. This study proposes the Isolated Items Discarding Strategy (IIDS) as an efficient way of designing a critical function. IIDS can be applied to any existing level-wise utility mining method (including ShFSM, DCG, and TP) that uses a critical function to decrease the number of candidates. Although, the TP method does not employ the concept of a critical function, the transaction-weighted downward closure property [27, 28] can be regarded as a variant of the critical function.

4.1 Isolated items discarding strategy (IIDS)

For level-wise utility mining methods, some definitions for IIDS are as follows:

Definition 4.1. Given *minUtil*, C_k is the set of candidate k-itemset in the k-th pass of the utility mining process. After the k-th pass, the process generates RC_k. RC_k is a subset of C_k, in which each k-itemset X satisfies the inequality $CF(X) \geq minLutil$.

Definition 4.2. For a level-wise utility mining method, after the k-th pass, RC_k can be generated, where $k > 0$. An item $i_p \in I$ is *isolated* if $i_p \notin X$ for all $X \in RC_k$. Let $ISet_{k+1}$ be the set of isolated items, which is generated after the k-th pass. That is, $ISet_{k+1} = I - \{\forall i_p \mid i_p \in \bigcup_{X_j \in RC_k} X_j\}$. For example, consider the database in Table 1 and the utility table in Table 2 using ShFSM to discover high utility itemsets. Let $minUtil$ = 30%, $minLutil$ = $122 \times 30\% = 33.6$. $C_1 = \{A, B, C, D, E, F, G, H\}$. After the first scan DB, we obtain $CF(A)$ = 92, $CF(B)$ = 68, $CF(C)$ = 105, $CF(D)$ = 68, $CF(E)$ = 54, $CF(F)$ = 43, $CF(G)$ = 21, and $CF(H)$ = 21. Therefore, $RC_1 = \{A, B, C, D, E, F\}$. According to the definition, $ISet_2$ =

$\{G, H\}$.

For each $X \in C_k$, if $CF(X) \geq minLutil$, then $X \in RC_k$. Therefore, an isolated item cannot appear in any itemset whose critical function value is above the minimum threshold.

Definition 4.3. In the k-th pass, where $k > 0$, given a transaction T_q, let NT_q^k denote a transaction that contains all items of T_q, exclusive of all items in $ISet_k$. That is, $NT_q^k = T_q - ISet_k$. Similarly, NDB^k denotes the set which consists of NT_q^k for all T_q in DB. NDB_X^k is the set which consists of NT_q^k for all T_q in DB_X. In the first pass, $ISet_1$ is set to $\varnothing$. Since for each $NT_q^k \subseteq T_q$, $NDB^k \subseteq DB$.

Definition 4.4. Given $minUtil$, let $HUI(DB)$ denote the set of all high utility itemsets and let HI_r denote a high utility itemset. Therefore, $HI_r \in HUI(DB)$, where $1 \leq r \leq |HUI(DB)|$. In addition, $HUI_k(DB)$ is a subset of $HUI(DB)$, where the length of each high utility itemset in $HUI_k(DB)$ is k and $k > 0$. Similarly, $HUI_{\geq k}(DB)$ is a subset of $HUI(DB)$, in which the length of each high utility itemset in $HUI_{\geq k}(DB)$ is larger than or equal to k.

Lemma 4.1. Given $minUtil$ and a utility table, for all $HI_r \in HUI_{\geq k}(DB)$, if $HI_r \subseteq T_q$, then $HI_r \subseteq NT_q^k$.

Corollary 4.1. Given $minUtil$ and a utility table, for all $HI_r \in HUI_{\geq k}(DB)$, if $HI_r \subseteq T_q$, then $util(HI_r, T_q) = util(HI_r, NT_q^k)$, where $k > 0$.

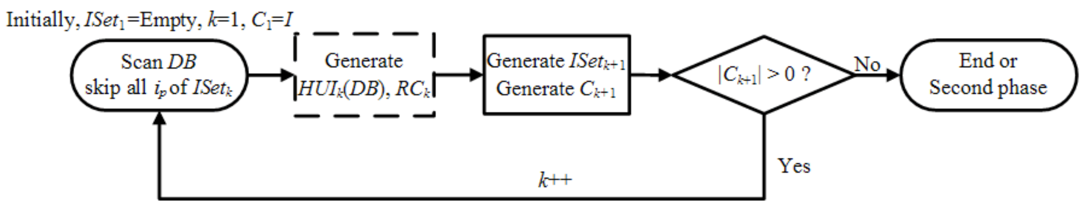
Theorem 4.1. Given $minUtil$ and a utility table, the two high utility sets, $HUI_{\geq k}(DB)$ and $HUI_{\geq k}(NDB^k)$, of the utility mining on DB and NDB^k, respectively, are identical, where $k > 0$.

Theorem 4.2. Given *minUtil* and a utility table, the two high utility sets, $HUI_k(DB)$ and $HUI_k(NDB^k)$, of the utility mining on DB and NDB^k, respectively, are identical, where $k > 0$.

Their proofs are detailed in Appendix B.

Since $HUI_k(DB) = HUI_k(NDB^k)$ in each pass, deleting each isolated item of $ISet_k$ from DB does not change the result of utility mining. To avoid generating extra NDB^k and rapidly increasing I/O costs in each pass, when scanning DB, the proposed IIDS skips isolated items of $ISet_k$ in each transaction. IIDS can be applied to any efficient and level-wise utility mining method to improve performance by replacing DB with NDB^k.

Fig. 1 shows the utility mining process with IIDS. If the utility mining method is a two-phase algorithm, such as TP, $HUI_k(DB)$ is not generated in the dashed line box, and the method requires an extra second phase to determine $HUI(DB)$. This study proposes that IIDS be applied to the two best share mining methods, ShFSM and DCG, and renames them Fast Utility Mining (FUM) and DCG+, respectively.



Initially, $ISet_1$=Empty, k=1, C_1=I
Scan DB skip all i_p of $ISet_k$
Generate $HUI_k(DB)$, RC_k
Generate $ISet_{k+1}$ Generate C_{k+1}
$|C_{k+1}| > 0$?
No
End or Second phase
Yes
k++

Fig. 1. Process of utility mining with IIDS

4.2 FUM and DCG+ methods

Let $Ndb^k_{S(X^{k+1})}$ be the transaction set of NT_q^k in which each NT_q^k contains at least one X^{k+1}. In the k-th pass, FUM replaces the critical function $Tutil(db_{S(X^{k+1})})$ of each candidate of ShFSM with $Tutil(Ndb_{S(X^{k+1})})$. If $k = 1$, then the set of isolated items is empty and $Tutil(db_{S(X^{k+1})}) = Tutil(Ndb_{S(X^{k+1})})$. Scanning NDB^k can easily obtain the value of $Tutil(Ndb_{S(X^{k+1})})$. For each itemset X, $Tutil(Ndb_{S(X^{k+1})}) \leq Tutil(db_{S(X^{k+1})})$. Therefore, FUM has a lower critical function value of each k-itemset X than ShFSM does for $k > 1$. If each itemset in RC_k is sorted in alphabetical order, the join step can efficiently skip joining useless itemsets. Thus, instead of RC_k joining RC_1 in the join step, FUM uses the sorted itemsets in RC_k to join with each other. The following example shows the difference in C_k and RC_k sizes between ShFSM and FUM.

Example 4.1. Consider the example database in Table 1 and the utility table in Table 2 with *minUtil* of 30%. Therefore, $minLutil = 122 \times 30\% = 36.6$. In Figs. 2 and 3, the number in the middle of each box is the critical function value calculated by ShFSM and FUM, respectively. The number in the bottom of each box is the local utility value. The two colored boxes denote the two high utility itemsets as shown in Figs. 2 and 3. For ShFSM in Fig. 2, $CF(\{A, B\}) = Tutil(db_{S(X^{k+1})}) = util(T01, T01) + util(T05, T05) = 21 + 17 = 38$. Contrast this to FUM in Fig. 3, where $RC_1 = \{A, B, C, D, E, F\}$ and $ISet_2 = \{G, H\}$ after the first pass. For the second pass, FUM scans DB, skipping isolated items. The database NDB^2 and the transaction utility of each T_q^2 are listed in Table 7. None of the transactions in NDB^2 include an isolated item. Therefore, $CF(\{A, B\}) = Tutil(Ndb^k_{S(X^{k+1})}) = util(T01\text{-}\{G, H\}, T01\text{-}\{G, H\}) + util(T05, T05) = 9 + 17 = 26$. Since $CF(\{A, B\}) < 36.6$, no superset of $\{A, B\}$ has a high utility value.

In Fig 2, in the three levels of the lattice, ShFSM generates 8, 15, and 5 candidates, respectively. The candidate itemsets in the dotted line boxes do not appear in RC_k. As shown in the solid line boxes, ShFSM maintains 6 candidates in RC_1, 9 candidates in RC_2, and one candidate in RC_3. The critical function values of them are at least 36.6.

The difference between ShFSM and FUM arises in the second and third passes. In the second pass, FUM produces 15 candidates, finds a high utility itemset $\{A, E\}$, and then keeps seven itemsets in RC_2 as shown in the solid line boxes in the second level of Fig. 3. Next, FUM produces two candidate 3-itemsets, $\{A, C, E\}$ and $\{B, C, D\}$, from

RC$_2$. In the third pass, the high utility itemset {*A*, *C*, *E*} is discovered. No critical function value of candidate 3-itemsets reaches 36.6, |*RC*$_3$| = 0. The FUM process terminates. In the second and third passes, FUM has smaller C_k and RC_k sets than ShFSM.

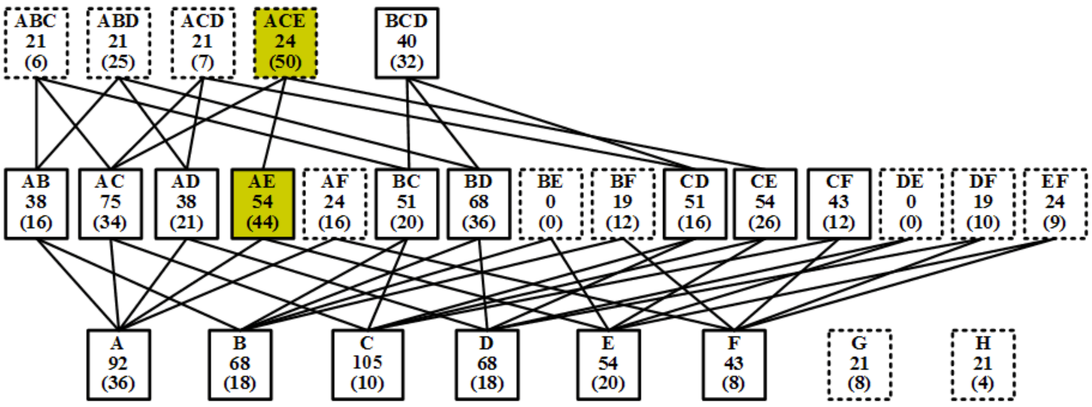


Fig. 2. An example of candidate generation by ShFSM with *minLutil* = 36.6

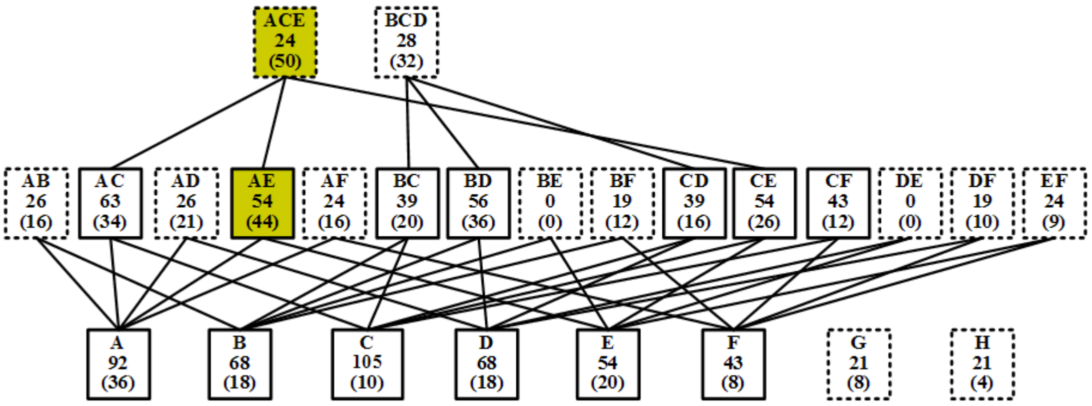


Fig. 3. An example of candidate generation by FUM with *minLutil* = 36.6

Table 7. *NDB*^{2}: Scan *DB* in the second pass of FUM by skipping isolated items

TID	Transaction (NT_q^2)	Count	Isolated item	Transaction utility
T01	{A, B, C, D}	{1, 1, 1, 1}	G, H	9
T02	{A, C, E, F}	{4, 3, 1, 2}		24
T03	{A, C, E}	{4, 3, 3}		30
T04	{B, C, D, F}	{4, 1, 2, 2}		19
T05	{A, B, D}	{3, 1, 2}		17
T06	{B, C, D}	{3, 2, 1}		11

The pseudo-code of the FUM algorithm is as follows:

Algorithm FUM

Input: (1) *DB*: a transaction database with counts, (2) *UT*: utility table, and (3) *minUtil*: minimum utility threshold

Output: All high utility itemsets

Procedure:

```
// $minLutil = minUtil \times Tutil(DB)$
1.  $k$:=1; $HUI_1(DB)$:=$\phi$; $ISet_1$:=$\phi$; $C_1$:=$I$;
2.  foreach $T \in DB$ // scan $DB$
3.      accumulate $\forall Lutil(i_p)$, and accumulate $\forall CF(i_p)$ // $\forall i_p \in T$ && $\forall i_p \in C_k$
4.  foreach $i_p \in C_k$  // check all candidates
5.      if $Lutil(i_p) \geq minLutil$ // high utility
6.          $HUI_k(DB)$:=$HUI_k(DB)+i_p$;
7.      if $CF(i_p)<minLutil$
8.          $C_k$:=$C_k-i_p$;  // delete useless item
9.          $ISet_{k+1}$:=$ISet_{k+1}+i_p$  // add isolated item
10. $RC_k$:=$C_k$;
11. while $|RC_k|>0$  // next pass
12.     $k$++; $HUI_k(DB)$:=$\phi$; $ISet_{k+1}$:=$\phi$;
13.     foreach $X_p$, $X_q \in RC_{k-1}$  // use $RC_{k-1}$ to generate $C_k$
14.         $C_k$:=Apriori-gen($X_p$, $X_q$); // candidate generation
15.     foreach $T \in DB$ // scan $DB$
16.         accumulate $\forall Lutil(X)$, and accumulate $\forall CF(X)$ by skipping $\forall i_p \in ISet_k$; // $\forall X \subseteq T$ && $\forall X \in C_k$
17.     foreach $X \in C_k$  // check all candidates
18.         if $Lutil(X) \geq minLutil$ // high utility
19.             $HUI_k(DB)$:=$HUI_k(DB)+X$;
20.         if $CF(X)<minLutil$
21.             $C_k$:=$C_k-X$; // delete useless itemset
22.     $RC_k$:=$C_k$;
23.     determinate $ISet_{k+1}$ in which no item appears in $RC_k$;
24. return $HUI(DB)=\bigcup_k HUI_k(DB)$;
```

Lines 9 and 23 generate a set of isolated items, $ISet_{k+1}$, in each pass. In lines 15 and 16, the process skips all isolated items to compute the critical function value of each candidate when scanning *DB*, so that the critical function value of each candidate can be reduced.

Similarly, DCG+ is a fast utility mining DCG algorithm to be used with the proposed IIDS. The following briefly describes how DCG is extended to become DCG+ by implementing IIDS.

Definition 4.5. Let the candidate *k*-itemset *X* be $\{i_1, i_2, \ldots, i_k\}$ in the order of literals, and let $i_q \in I$ be an item. If $i_k < i_q$ then the (k+1)-superset of X $\{i_1, i_2, \ldots, i_k, i_q\}$ is defined as the *monotone (k+1)-superset* of *X* and is denoted as $X_{i_q}^{k+1}$. For the example in Table 1, let $X = \{B, C, D\}$, $X_F^{k+1} = \{B, C, D, F\}$.

In the *k*-th pass, DCG scans the database once to determine which candidates are high utility. Meanwhile, DCG calculates the critical function values of each candidate's all monotone (k+1)-supersets. Let *X* be a candidate with length *k*, DCG calculates the critical function value of each monotone (k+1)-superset of *X*, $X_{i_q}^{k+1}$, as $Tutil(DB_{X_{i_q}^{k+1}})$. The critical function value of $X_{i_q}^{k+1}$ is the upper bound of its utility value. If $Tutil(DB_{X_{i_q}^{k+1}}) \geq minLutil$, then DCG adds $X_{i_q}^{k+1}$ to C_{k+1}. Thus, DCG directly generates C_{k+1} without the join and prune steps.

According to Theorem 4.2, $HUI_k(DB) = HUI_k(NDB^k)$. In each pass, instead of scanning *DB*, DCG+ scans NDB^k, which is a smaller database than *DB*, to obtain the identical set of high utility *k*-itemsets. DCG+ calculates the critical function value of each monotone (k+1)-superset of *X*, $X_{i_q}^{k+1}$, as $Tutil(NDB^k_{X_{i_q}^{k+1}})$. According to Definition 4.3, $NDB^k_{X_{i_q}^{k+1}} \subseteq DB_{X_{i_q}^{k+1}}$. Therefore, $Tutil(NDB^k_{X_{i_q}^{k+1}}) \leq Tutil(DB_{X_{i_q}^{k+1}})$. IIDS can reduce the critical function values, so that DCG+ can delete more useless candidates than DCG. Without the redundancy, this study omits the detailed algorithm of DCG+.

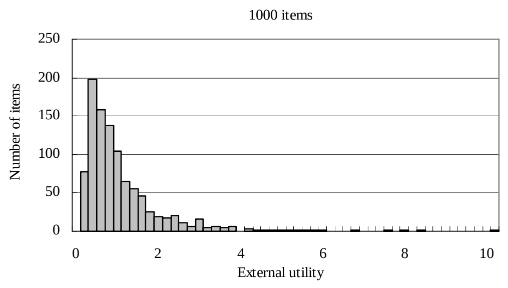
A utility mining method with IIDS scans a database that is smaller than the original by skipping isolated items to reduce the critical function values of candidates. A low critical function value indicates the low upper bound of the candidate's utility value. Thus, a utility mining method with IIDS generates fewer candidates than the utility mining method without IIDS to improve performance. In addition to utility mining, IIDS can be employed for traditional frequent itemset mining. According to the Apriori property [2, 3], if an itemset is infrequent, then all supersets of the itemset are infrequent. If the utility value of each itemset is replaced with its support value and RC_k is the set of frequent k-itemsets in each pass, then the utility mining with IIDS can be utilized for mining frequent itemsets. Therefore, IIDS also works well to discover traditional frequent itemsets.

5. Experimental results

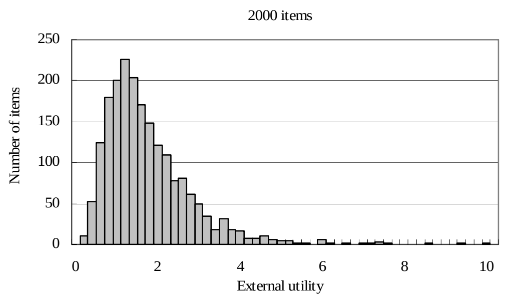
The performance of two methods FUM and DCG+, in which IIDS was implemented, was compared with that of TP, ShFSM, and DCG. The experiments were done on an AMD Barton ES 2900+ (2000 MHz) PC with 3 GB of main memory, running the Windows XP Professional operating system. All algorithms were implemented in Visual C++ 6.0 and applied to several synthetic and real datasets. To reduce the effect of disk writing, all discovered high utility itemsets were stored in the main memory. All experimental synthetic datasets and a real dataset were adopted from NU-MineBench 2.0, a powerful benchmark suite consisting of multiple data mining applications and databases [31].

5.1. Synthetic datasets

An IBM synthetic data generator [19] was used for this study. The parameters of the generator are introduced in [3] and modified in [27]. The generated datasets are classified into two groups: (1) T10.I6, with a mean transaction size of 10 and mean size of the maximal potentially frequent itemsets of six; and (2) T20.I6, with a mean transaction size of 20 and mean maximal potentially frequent itemsets size of six. In each itemset of the synthetic datasets, internal utilities between one and four were randomly generated. Observed from real world databases, most items are in the low profit range [27, 28]. Therefore, the external utility of each item was heuristically chosen between 0.01 and 10 and randomly generated with a log-normal distribution. Fig. 4 shows the external utility distribution with 1000 and 2000 distinct items. The most items have a low external utility value.



(a) 1000 distinct items

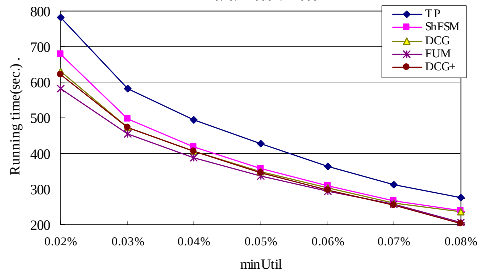


(b) 2000 distinct items

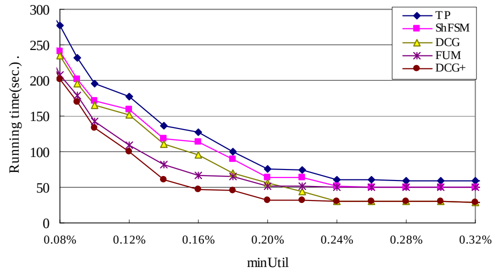
Fig. 4. External utility distribution with distinct items

Figs. 5 through 7 show the performance curves associated with these algorithms performed on T10.I6.D1000k.N1000, T10.I6.D100k.N2000, and T20.I6.D1000k.N1000, respectively. FUM and DCG+ had better performance than ShFSM and DCG, respectively. The experimental results demonstrate that employing IIDS can improve the performance of ShFSM and DCG. The running time of ShFSM was less than that of TP for scanning the database once. In a low *minUtil* value, FUM performed the best, followed by DCG+ and DCG (see Figs. 5a, 6a, and 7a). In a high *minUtil* value, DCG+ and DCG performed the best (see Figs. 5b, 6b, and 7b).

As shown in Fig. 5, DCG+ was the most efficient method in a *minUtil* range of 0.07% to 0.2%, followed by FUM, DCG, ShFSM, and TP, respectively. In a *minUtil* range of 0.08% to 0.28%, FUM outperformed DCG (Fig. 7). With a *minUtil* of 0.20%, the execution times of DCG+, FUM, DCG, and ShFSM were 41.9%, 61.2%, 80.2%, and 93% of the TP, respectively (Fig. 7a).

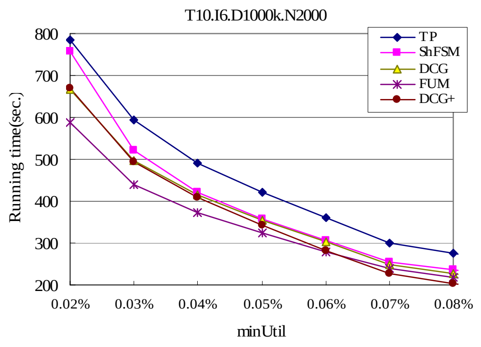


(a) between 0.02% and 0.08%

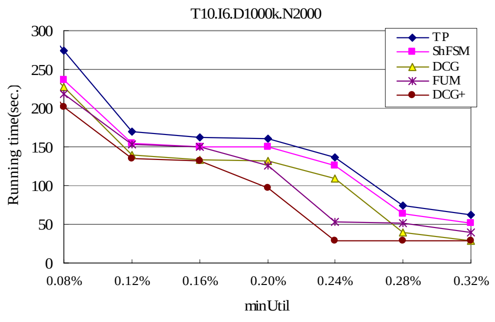


(b) between 0.08% and 0.32%

Fig. 5. Comparison of running time using T10.I6.D1000k.N1000 with various *minUtil*

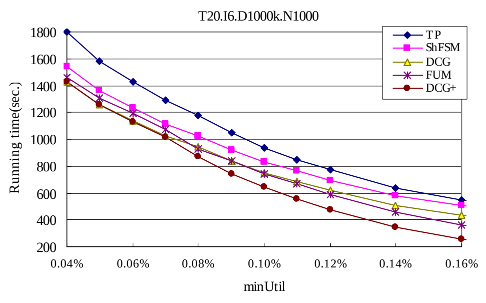


(a) between 0.02% and 0.08%,

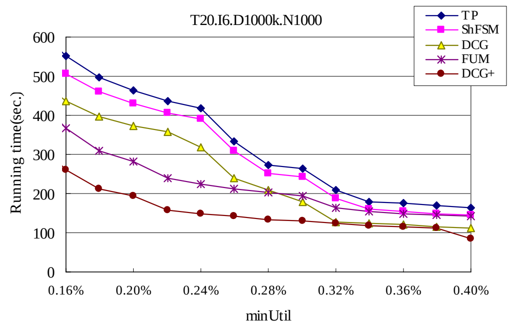


(b) between 0.08% and 0.32%

Fig. 6. Comparison of running time using T10.I6.D1000k.N2000 with various *minUtil*



(a) between 0.04% and 0.16%



(b) between 0.16% and 0.4%

Fig. 7. Comparison of running time using T20.I6.D1000k.N1000 with various *minUtil*

Table 8 lists the candidate numbers of C_k and RC_k among the five algorithms in each pass using T10.I6.D1000k.N1000 with *minUtil* of 0.12%. Except for the first and the second passes, FUM and DCG+ generated a smaller candidate set than ShFSM and DCG, respectively. The running time of Phase I of TP was 3.9 seconds less than the total running time of ShFSM. However, TP required Phase II to determine *HUI*(*DB*). The total running time of TP was 177.1 seconds, while the time for ShFSM was 159.2 seconds. Although no high utility itemset has a length between five and six, these algorithms did discover the high utility 7-itemset (see the last column of Table 8). FUM and DCG+ scanned the database 7 times; ShFSM and DCG 11 times; and TP 12 times.

Table 8 demonstrates that IIDS can help to reduce the size of candidate set and reduce the number of passes to scan the database.

Table 8. Candidate number comparison in each pass using T10.I6.D1000k.N1000 with *minUtil* = 0.12%

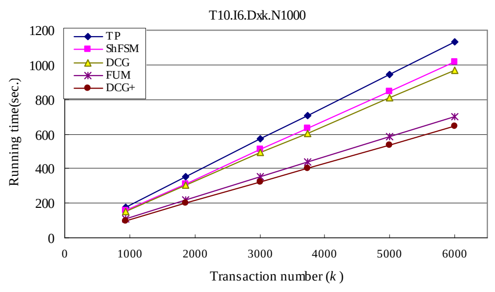
Method / Pass (k)	TP (Phase I)		ShFSM		DCG		FUM		DCG+		$\|HUI_k\|$
	$\|C_k\|$	$\|RC_k\|$	$\|C_k\|$	$\|RC_k\|$	$\|C_k\|$	$\|RC_k\|$	$\|C_k\|$	$\|RC_k\|$	$\|C_k\|$	$\|RC_k\|$	
k=1	1000	893	1000	893	1000	NA	1000	893	1000	NA	238
k=2	398 278	9472	398 278	9472	9472	NA	398 278	9338	9472	NA	18
k=3	95 038	1357	95 038	1356	1357	NA	93 431	1116	1120	NA	1
k=4	1635	1421	1635	1421	1421	NA	1355	491	491	NA	1
k=5	1440	1386	1440	1385	1386	NA	549	56	56	NA	0
k=6	1117	1103	1117	1103	1103	NA	28	28	28	NA	0
k=7	700	684	700	680	684	NA	8	0	8	NA	1
k=8	332	332	332	330	332	NA	0	0	0	NA	0
k=9	110	110	110	110	110	NA	0	0	0	NA	0
k=10	22	22	22	22	22	NA	0	0	0	NA	0
k=11	2	2	2	0	2	NA	0	0	0	NA	0
Total	499 674	16 782	499 674	16 772	16 889	NA	494 649	11 922	12 175	NA	259
Time(sec)	177.1 (155.3)		159.2		151.3		109.1		100.2		

Table 9 lists the candidate numbers generated from the four algorithms with a length greater than two. In the first pass of database scan, the set of isolated items was empty, so the IIDS took no action. Therefore, Table 9 only compares the difference of the numbers of candidates, with at least length three, generated by algorithms with and without implementing IIDS. The percentages in the column “Reducing rate 1” indicate that FUM saves percentage of generated candidates of the ShFSM did. The percentage numbers in the column “Reducing rate 2” indicate that DCG+ saves percentage of generated candidates of DCG did. In Table 9, the algorithm with IIDS always generates fewer candidates than the corresponding algorithm without IIDS. For T10.I6.D1000k.N1000 and T20.I6.D1000k.N1000, utility mining with a high *minUtil* value can obtain a significant improvement. For the dataset T10.I6.D1000k.N2000, the improvement is significant in two *minUtil* ranges (0.06%-0.14% and 0.20%-0.24%).

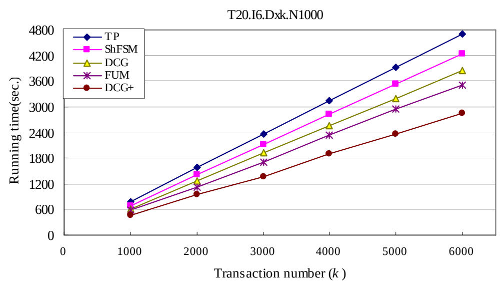
Table 9. Candidate number comparison with length greater than two

Dataset	minUtil	ShFSM (A)	FUM (with IIDS) (B)	Reducing rate 1 ($\frac{A-B}{A}\times 100\%$)	DCG (C)	DCG+ (with IIDS) (D)	Reducing rate 2 ($\frac{C-D}{C}\times 100\%$)
T10.I6. D1000k. N1000	0.04%	2 604 920	2 600 651	0.16%	415 654	412 914	0.66%
	0.06%	1 065 356	1 056 684	0.81%	185 127	176 963	4.41%
	0.08%	454 015	443 752	2.26%	61 871	51 632	16.55%
	0.10%	201 351	194 599	3.35%	17 351	10 551	39.19%
	0.12%	100 396	95 371	5.01%	6417	1703	73.46%
	0.14%	51 042	48 385	5.21%	2217	451	79.66%
	0.16%	27 356	26 634	2.64%	370	106	71.35%
	0.18%	14 852	14 235	4.15%	174	61	64.94%
	0.20%	8067	7601	5.78%	21	0	100.00%
T10.I6. D1000k. N2000	0.04%	674 897	669 019	0.87%	413 768	406654	1.72%
	0.06%	248 961	229 023	8.01%	187 048	159 362	14.80%
	0.08%	87 622	66 936	23.61%	70 603	43 641	38.19%
	0.10%	38 080	14 156	62.83%	31 496	5222	83.42%
	0.12%	6048	4702	22.42%	4039	2316	42.66%
	0.14%	2949	2678	9.19%	2313	1985	14.18%
	0.16%	2313	1985	14.18%	2216	2186	1.35%
	0.18%	2045	2032	0.64%	1988	1981	0.33%
	0.20%	2002	1905	4.85%	1986	1776	11.08%
	0.22%	1990	501	74.82%	1981	182	90.81%
	0.24%	1956	166	91.51%	1902	0	100.00%
T20.I6. D1000k. N1000	0.06%	10 764 231	10 752 539	0.11%	483 418	477 647	1.19%
	0.10%	4 357 889	4 338 928	0.44%	137 096	120 669	11.98%
	0.14%	2 042 690	2 027 660	0.74%	34 450	21 020	38.98%
	0.18%	1 054 549	1 041 902	1.20%	10 643	4037	62.07%
	0.22%	574 654	563 818	1.89%	3385	759	77.58%
	0.26%	329 566	321 728	2.38%	1906	314	83.53%
	0.30%	192 540	186 190	3.30%	165	16	90.3%

Fig. 8 shows the scalability of these algorithms by increasing the number of transactions on T10.I6.Dxk.N1000 and T20.I6.Dxk.N1000, respectively, with *minUtil* of 0.12%. The number of transactions varies from 1000k to 6000k. The running time of each algorithm approximately increases linearly with the growth of *DB*. In Fig. 8a, the running time of FUM and DCG+ is less than that of FSM and DCG, achieving 31% and 35%, respectively. For a longer mean transaction size (Fig. 8b), FUM and DCG+ have better performance than ShFSM and DCG with running times of 20% and 30%, respectively. Thus, the utility mining methods with IIDS significantly reduced the running time while offering linear scalability.



(a) T10.I6.DxK.N1000



(b) T20.I6.DxK.N1000

Fig. 8. Scalability with the number of transactions while *minUtil* = 0.12%

5.2. Real dataset

This study also evaluated these algorithms using a real dataset. The Chain-store dataset was taken from a major grocery store chain in California and contained 1 112 949 transactions and 46 086 distinct items. The utility table stored the profit for each item. The total profit of the dataset is $26 388 499.80.

Fig. 9 presents the performance with several *minUtil* values from 0.05% to 0.36%. Because DCG and DCG+ maintained an extra array for each candidate, the main memory could not keep all candidates in each pass; for this reason, DCG and DCG+ are not illustrated in Fig. 9. Fig. 9 shows that FUM outperformed ShFSM and TP; the running time of FUM was only 64.6% and 76.7% of the times for ShFSM and TP, respectively, with *minUitl* = 0.12%.

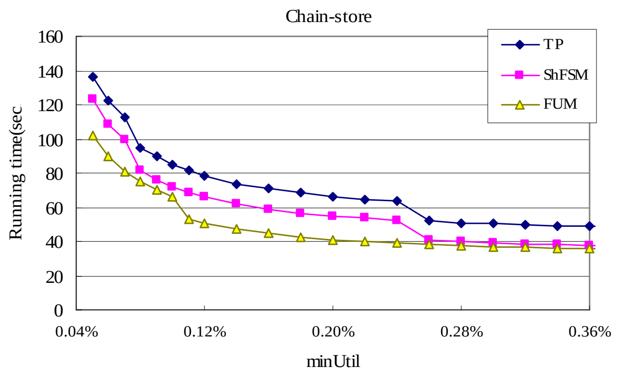


Fig. 9. Comparison of running time using the real dataset with *minUtil* between 0.05% and 0.36%

Table 10 lists the candidate numbers of C_k and RC_k among the three algorithms in each pass using the real dataset with *minUtil* of 0.06%. The total running time of TP was 122.5 seconds, which was 13.9 seconds slower than ShFSM's 108.6 seconds. FUM scanned the database four times; ShFSM, five times; and TP, six times. Table 10 demonstrates that IIDS can help to significantly reduce the size of the candidate set with length greater than two and can reduce the number of passes required to scan the real dataset. For example, in the third pass, ShFSM generated 78 238 candidates, while FUM only generated 45 795 candidates.

Table 10. Candidate number comparison in each pass using the real dataset with *minUtil* = 0.06%

Method / Pass (k)	TP (Phase I)		ShFSM		FUM (with IIDS)		$\|HUI_k\|$
	$\|C_k\|$	$\|RC_k\|$	$\|C_k\|$	$\|RC_k\|$	$\|C_k\|$	$\|RC_k\|$	
k=1	46 086	6344	46 086	6335	46 086	6335	154
k=2	20 119 996	7873	20 062 945	7869	20 062 945	5454	39
k=3	78 278	1083	78 238	1081	45 795	170	3
k=4	596	43	596	43	44	0	0
k=5	1	0	1	0	0	0	0
Total	20 244 957	15 343	20 187 866	15 328	20 154 870	11 959	196
Time(sec)	122.5 (103.9)		108.6		89.1		

To analyze the difference between the high utility itemsets and the support-based frequent itemsets, this experiment employed the FP-growth algorithm [16] to generate all frequent itemsets. For the Chain-store dataset with *minSup* = 0.0073%, FP-growth generated 14 352, 33 371, 6569, 441, and 14 frequent itemsets with length from one to five, respectively.

The three high utility itemsets with length three, which were discovered from Chain-store using FUM with *minUtil* = 0.06%, their local utility values and their support values as shown in Table 11. Numbers in an itemset indicate the *ID*s of products. Sales managers are interested in finding out which itemsets can generate high profits, but the traditional frequent itemset mining method may not satisfy this goal. For example, a utility mining approach discovered the three high utility 3-itemsets with *minUtil* of 0.06% (Table 11). Selling the combination of products, {39182, 39206, 39695}, earned a profit of $25 484.30. FP-growth discovered six frequent 3-itemsets, omitting the first two highest utility itemsets, {39182, 39206, 39695} and {39681, 39690, 39692} when the minimum support threshold was 0.1%. To obtain the two highest utility 3-itemsets, the *minSup* threshold must be set less than 0.0074%. Nevertheless, the low threshold value, 0.0073%, resulted in 6569 3-itemsets being generated. FP-growth generated too many useless frequent 3-itemsets which interfered in selecting of high profit itemsets. Even when the interesting itemsets were discovered, the real profits were still unknown.

Table 11. High utility 3-itemsets discovered from Chain-store using *minUtil* = 0.06%

High utility 3-itemset *X*	*Lutil*(*X*)	Support	Support ranking in all 3-itemsets
{39182, 39206, 39695}	25 484.3	0.0074%	6400
{39681, 39690, 39692}	19 520.8	0.0093%	3710
{21283, 21308, 22900}	19 064.9	0.1002%	6

Fig. 10 shows the corresponding support values of the 196 high utility itemsets with *minUtil* of 0.06%. The *x*-axis represents the ranked high utility itemsets decreasing ordered by their utility values, and the *y*-axis represents their support values. The itemset ranked 31st had the maximum support value 5.734%. The support value (3.149%) of the itemset ranked 169th was over 400 times the support value (0.007%) of the itemset ranked 87th. In this case, an itemset with a higher profit had a lower support value. Traditional frequent itemset mining using a support threshold cannot effectively discover high utility itemsets. Therefore, utility mining is more useful for a profit-oriented business environment than the traditional association rule mining that is currently used in practice.

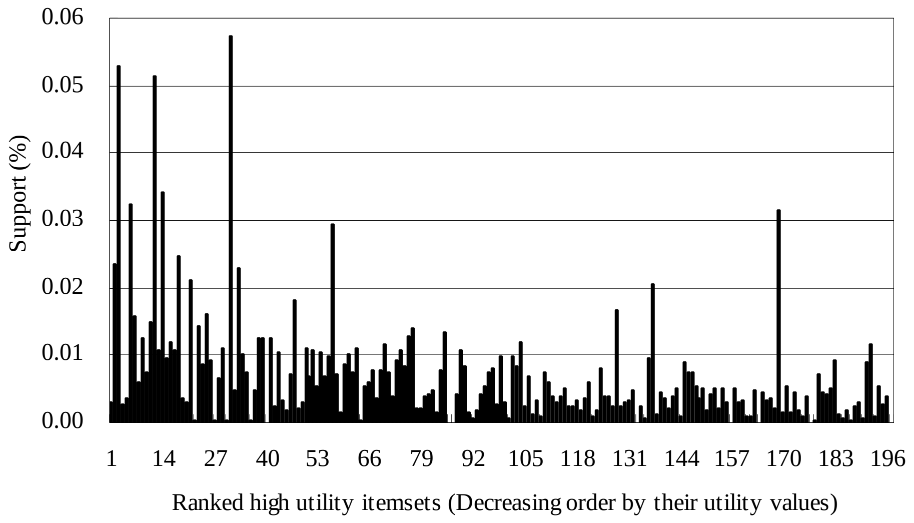


Fig. 10. Support values of 196 high utility itemsets, ranked by utility value

6. Conclusions

Increasing the profit of a corporation is one of the most important goals of data mining. Traditional association rules methods only consider whether an item is bought in a transaction. However, customers can buy more than one of the same item in a transaction, and the unit profit for each item may vary. Utility mining, a generalized form of share mining, has been proposed to overcome the drawback of traditional association rule mining. However, the Apriori principle cannot be directly applied to efficiently discover high utility itemsets as this becomes time-consuming. The ability to efficiently identify high utility itemsets is crucial for utility mining. Therefore, this study proposes the Isolated Items Discarding Strategy (IIDS) to identify isolated items from transactions and ignore them in the process of candidate itemset generation. The contributions of this study are as follows:

1. Propose IIDS to reduce the critical function values of itemsets.
2. The experimental results using synthetic and real datasets reveal that the performances of FUM and DCG+ were better than that of ShFSM and DCG, respectively. IIDS can further decrease the number of candidates and efficiently increase the performance of these utility mining methods.
3. Theoretical proofs and experimental results indicate that the IIDS is a promising strategy for utility mining.

IIDS can also be applied to Apriori-like traditional mining. In the future, the authors will extend the application scope of IIDS to some classification models. Classification is an important problem in data mining; several researchers have integrated classification and association rule mining [14, 25]. Thus, the connection between utility mining and associative classification should be further investigated.

Acknowledgements

The authors would like to acknowledge the helpful comments made by the anonymous reviews of this paper. The second author was supported by the National Science Council, Taiwan and Providence University, Taiwan.

Appendix A. Proofs for the theorems and lemma given in Section 3

Proof of Theorem 3.1.

For each X^{k+i}, since $Lutil(X^{k+i}) \leq CF(X) < minLutil$, X^{k+i} is a low utility itemset. X^{k+i} is an arbitrary superset of X. Therefore, if $CF(X) < minLutil$, then no superset of X has high utility.

Q.E.D

Proof of Lemma 3.1.

(1) $db_{S(X^{k+i})} \subseteq db_{S(X^{k+1})}$: Let T be an arbitrary transaction and $T \in db_{S(X^{k+i})}$. That is, there exists a $(k+i)$-itemset X^{k+i}, for some $i > 0$, such that $X^{k+i} \subseteq T$. Clearly, any $(k+1)$-subset of X^{k+i} containing X is also contained in T. Therefore, $T \in db_{S(X^{k+1})}$.

(2) $db_{S(X^{k+i})} \supseteq db_{S(X^{k+1})}$: Let T be an arbitrary transaction and $T \in db_{S(X^{k+1})}$. That is, there exists a $(k+1)$-itemset X^{k+1}, such that $X^{k+1} \subseteq T$. Clearly $X^{k+1} \in S(X^{k+i})$, according to Definition 3.4, $T \in db_{S(X^{k+i})}$.

Hence, we obtain $db_{S(X^{k+i})} = db_{S(X^{k+1})}$.

Q.E.D

Proof of Theorem 3.2.

For an arbitrary $(k+i)$-superset X^{k+i} of X, since $X^{k+i} \in S(X^{k+i})$, $DB_{X^{k+i}} \subseteq db_{S(X^{k+i})}$. By Lemma 3.1, $DB_{X^{k+i}} \subseteq db_{S(X^{k+1})}$. According to Definitions 2.5 and 2.6, $Lutil(X') \leq$ $Tutil(db_{S(X^{k+1})})$

Q.E.D

Appendix B. Proofs for the lemma, corollary, and theorems given in Section 4

Proof of Lemma 4.1.

Since for all $HI_r \in HUI_{\geq k}(DB)$, the length of HI_r, $|HI_r|$, is at least k. According to Definition 4.2, let $T_q = NT_q^k \bigcup t_q$, where $NT_q^k \bigcap t_q = \phi$ and $t_q \subseteq ISet_k$. Each item in t_q has no high utility superset with length at least k in $HUI_{\geq k}(DB)$. So that for all $i_p \in t_q$, $i_p \notin HI_r$. $HI_r \bigcap t_q = \phi$. If $HI_r \subseteq T_q$ then $HI_r \subseteq T_q$ - t_q. Therefore, $HI_r \subseteq NT_q^k$.

Q.E.D

Proof of Corollary 4.1.

According to Lemma 4.1, if $HI_r \subseteq T_q$, $HI_r \subseteq NT_q^k$. For any $i_p \in HI_r$, $iu(i_p, T_q) = iu(i_p, NT_q^k)$ since NT_q^k is a subset of T_q. Moreover, $util(i_p, T_q) = iu(i_p, T_q) \times eu(i_p) = iu(i_p, NT_q^k) \times eu(i_p) = util(i_p, NT_q^k)$. Therefore, $util(HI_r, T_q) = \sum_{i_p \in HI_r \subseteq T_q} util(i_p, T_q) = \sum_{i_p \in HI_r \subseteq NT_q^k} util(i_p, T_q) = \sum_{i_p \in HI_r \subseteq NT_q^k} util(i_p, NT_q^k) = util(HI_r, NT_q^k)$.

Q.E.D

Proof of Theorem 4.1.

(1) (1) $HUI_{\geq k}(NDB^k) \subseteq HUI_{\geq k}(DB)$: According to Definition 4.3, NDB^k is the database that consists of NT_q^k for all T_q in DB. The mapping function of transactions is "one-to-one and onto" between NDB^k and DB. For an arbitrary itemset NT_q^k in NDB^k and the corresponding T_q in DB, we have $NT_q^k \subseteq T_q$. So that for all T_q in DB, $X \subseteq T_q$, we have $util(X,T_q) \geq util(X,NT_q^k)$. $\sum_{T_q \in DB} util(X, T_q) \geq \sum_{NT_q^k \in NDB^k} util(X, T_q)$. That is,

$HUI(NDB^k) \subseteq HUI(DB)$. Therefore, $HUI_{\geq k}(NDB^k) \subseteq HUI_{\geq k}(DB)$.

(2) $HUI_{\geq k}(DB) \subseteq HUI_{\geq k}(NDB^k)$: Assume there exists $HI_r \in HUI_{\geq k}(DB)$, but $HI_r \notin HUI_{\geq k}(NDB^k)$. Therefore, there exists an transaction T_q in DB such that $util(HI_r, T_q) > util(HI_r, NT_q{}^k)$. However, according to Corollary 4.1, $util(HI_r, T_q) = util(HI_r, NT_q{}^k)$. This contradicts the assumption. Therefore, $HUI_{\geq k}(DB) \subseteq HUI_{\geq k}(NDB^k)$.

According to the above two cases, we have $HUI_{\geq k}(DB) = HUI_{\geq k}(NDB^k)$.

Q.E.D.

Proof of Theorem 4.2.

By Definition 4.4, clearly, $HUI_{\geq k}(DB)$ is the disjoint union of $HUI_k(DB)$ and $HUI_{\geq k+1}(DB)$, and $HUI_{\geq k}(NDB^k)$ is the disjoint union of $HUI_k(NDB^k)$ and $HUI_{\geq k+1}(NDB^k)$. According to Theorem 4.1, we have $HUI_{\geq k}(DB) = HUI_{\geq k}(NDB^k)$, for all $k > 0$. That is, $HUI_{\geq k}(DB) = HUI_{\geq k}(NDB^k)$ and $HUI_{\geq k+1}(DB) = HUI_{\geq k+1}(NDB^k)$ both hold, therefore, $HUI_k(DB) = HUI_{\geq k}(DB) - HUI_{\geq k+1}(DB) = HUI_{\geq k}(NDB^k) - HUI_{\geq k+1}(NDB^k) = HUI_k(NDB^k)$.

Q.E.D.

References

[1] R.C. Agarwal, C.C. Aggarwal, and V.V.V. Prasad, “A tree projection algorithm for generation of frequent itemsets,” *Journal of Parallel and Distributed Computing*, 61 (2001) 350-371.

[2] R. Agrawal, T. Imielinski, and A. Swami, “Mining association rules between sets of items in large databases,” in *Proceedings 1993 ACM SIGMOD Intl. Conf. on Management of Data*, Washington, D.C., May 1993, pp. 207-216.

[3] R. Agrawal and R. Srikant, “Fast algorithms for mining association rules,” in *Proceedings 20th Intl. Conf. on Very Large Data Bases*, Santiago, Chile, September 1994, pp. 487-499.

[4] B. Barber and H.J. Hamilton, “Algorithms for mining share frequent itemsets containing infrequent subsets,” in D.A. Zighed, H.J. Komorowski, and J.M. Zytkow (Eds.), *Lecture Notes in Computer Sciences 1910 --- 4th European Conf. on Principles of Data Mining and Knowledge Discovery* (*PKDD 2000*), Springer-Verlag, Berlin, 2000, pp. 316-324.

[5] B. Barber and H.J. Hamilton, “Parametric algorithm for mining share frequent itemsets,” *Journal of Intelligent Information Systems*, 16 (2001) 277-293.

[6] B. Barber and H.J. Hamilton, “Extracting share frequent itemsets with infrequent subsets,” *Data Mining and Knowledge Discovery*, 7 (2003) 153-185.

[7] F. Berzal, J.-C. Cubero, N. Marín, and J.-M. Serrano, “TBAR: an efficient method for association rule mining in relational databases,” *Data & Knowledge Engineering*, 37 (2001) 47-64.

[8] S. Brin, R. Motwani, J.D. Ullman, and S. Tsur, “Dynamic itemset counting and implication rules for market basket data,” in *Proceedings 1997 ACM SIGMOD Intl. Conf. on Management of Data*, Tucson, AZ, May 1997, pp. 255-264.

[9] C.H. Cai, A.W.C. Fu, C.H. Cheng, and W.W. Kwong, “Mining association rules with weighted items,” in *Proceedings 1998 Intl. Database Engineering and Application Symposium*, Cardiff, Wales, U.K., July 1998, pp. 68-77.

[10] C.L. Carter, H.J. Hamilton and N. Cercone, “Share based measures for itemsets,” in H.J. Komorowski and J.M. Zytkow (Eds.), *Lecture Notes in Computer Science* 1263 --- *1st European Symposium on Principles of Data Mining and Knowledge Discovery* (*PKDD 1997*), Springer-Verlag, Berlin, 1997, pp. 14-24.

[11] R. Chan, Q. Yang, and Y.-D. Shen, “Mining high utility itemsets,” in *Proceedings 3rd IEEE Intl. Conf. on Data Mining*, Melbourne, FL, December 2003, pp. 19-26.

[12] C.-C. Chang and C.-Y. Lin, “Perfect hashing schemes for mining association Rules,” *The Computer Journal*, 48 (2005) 168-179.

[13] M.-S. Chen, J. Han, and P.S. Yu, “Data mining: an overview from a database perspective,” *IEEE Trans. Knowledge Data Engineering*, 8 (1996) 866-883.

[14] G. Dong, X. Zhang, L. Wong, and J. Li, “CAEP: classification by aggregating emerging patterns,” in S. Arikawa and K. Furukawa (Eds.), *Lecture Notes in Computer Sciences 1721 --- 2nd Intl. Conf. on Discovery Science* (*DS 1999*), Springer-Verlag, Berlin, 1999, pp. 30-42.

[15] G. Grahne and J. Zhu, “Fast algorithms for frequent itemset mining using FP-trees,” *IEEE Trans. Knowledge and Data Engineering*, 17 (2005) 1347-1362.

[16] J. Han, J. Pei, Y. Yin and R. Mao, “Mining frequent pattern without candidate generation: a frequent pattern tree approach,” *Data Mining and Knowledge Discovery*, 8 (2004) 53-87.

[17] R.J. Hilderman, "Predicting itemset sales profiles with share measures and repeat-buying theory," in J. Liu, Y.-M. Cheung, and H. Yin (Eds.), *Lecture Notes in Computer Science 2690 --- 4th Intl. Conf. on Intelligent Data Engineering and Automated Learning* (*IDEAL 2003*), Springer-Verlag, Berlin, 2003, pp. 789-795.

[18] R.J. Hilderman, C.L. Carter, H.J. Hamilton, and N. Cercone, "Mining association rules from market basket data using share measures and characterized itemsets," *Intl. Journal of Artificial Intelligence Tools*, 7 (1998) 189-220.

[19] IBM Almaden Research Center, "Synthetic data generation code for associations and sequential patterns," Available at: http://www.almaden.ibm.com/software/projects/hdb/resources.shtml (accessed 15 November 2005).

[20] M. Kantardzic, "Data Mining: Concepts, Models, Methods, and Algorithms," *John Wiley & Sons, Inc.*, New York, 2002.

[21] Y.-C. Li and C.-C. Chang, "A new FP-tree algorithm for mining frequent itemsets," in C.-H. Chi and K.-Y. Lam (Eds.), *Lecture Notes in Computer Science 3309 --- Advanced Workshop on Content Computing* (*AWCC* 2004), Springer-Verlag, Berlin, 2004, pp. 266-277.

[22] Y.-C. Li, J.-S. Yeh, and C.-C. Chang, "A fast algorithm for mining share-frequent itemsets," in Y. Zhang, K. Tanaka, J.X. Yu, S. Wang, and M. Li (Eds.), *Lecture Notes in Computer Science 3399 --- 7th Asia-Pacific Web Conference on Web Technologies Research and Development* (*APWeb 2005*), Springer-Verlag, Berlin, 2005, pp. 417-428.

[23] Y.-C. Li, J.-S. Yeh, and C.-C. Chang, "Efficient algorithms for mining share-frequent itemsets," in *Proceedings 11th World Congress of Intl. Fuzzy Systems Association*, Beijing, China, July 2005, pp. 534-539.

[24] Y.-C. Li, J.-S. Yeh, and C.-C. Chang, "Direct candidates generation: a novel algorithm for discovering complete share-frequent itemsets," in L. Wang and Y. Jin

(Eds.), *Lecture Notes in Artificial Intelligence 3614 --- 2nd Intl. Conf. on Fuzzy Systems and Knowledge Discovery* (*FSKD 2005*) , Springer-Verlag, Berlin, 2005, pp. 551-560.

[25] B. Liu, W. Hsu, and Y. Ma, “Integrating classification and association rule mining,” in *Proceedings 4th Intl. Conf. on Knowledge Discovery and Data Mining*, New York, August 1998, pp. 80-86.

[26] J. Liu, Y. Pan, K. Wang, and J. Han, “Mining frequent item sets by opportunistic projection,” in *Proceedings 8th ACM-SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, Alberta, Canada, July 2002, pp. 229-238.

[27] Y. Liu, W.-K. Liao, and A. Choudhary, “A two-phase algorithm for fast discovery of high utility itemsets,” in T.B. Ho, D. Cheung, and H. Liu (Eds.), *Lecture Notes in Computer Science 3518 --- 9th Pacific-Asia Conf. on Advances in Knowledge Discovery and Data Mining* (*PAKDD 2005*), Springer-Verlag, Berlin, 2005, pp. 689-695.

[28] Y. Liu, W.-K. Liao, and A. Choudhary, “A fast high utility itemsets mining algorithm,” in *Proceedings 1st Intl. Conf. on Utility-Based Data Mining*, Chicago, IL, August 2005, pp. 90-99.

[29] S. Lu, H. Hu, and F. Li, “Mining weighted association rules,” *Intelligent Data Analysis,* 5 (2001) 211-225.

[30] J.S. Park, M.-S. Chen, and P.S. Yu, “An effective hash-based algorithm for mining association rules,” in *Proceedings 1995 ACM-SIGMOD Intl. Conf. on Management of Data*, San Jose, CA, May 1995, pp. 175-186.

[31] J. Pisharath, Y. Liu, J. Parhi, W.-K. Liao, A. Choudhary, and G. Memik, “NU-MineBench version 2.0 source code and datasets,” Available at: http://cucis.ece.northwestern.edu/projects/DMS/MineBench.html (accessed 16 January 2006).

[32] Y.-D. Shen and Z. Zhang, "Objective-oriented utility-based association mining," in *Proceedings 2002 IEEE Intl. Conf. on Data Mining*, Maebashi City, Japan, December 2002, pp. 426-433.

[33] F. Tao, F. Murtagh, and M. Farid, "Weighted association rule mining using weighted supports and significance framework," in *Proceedings 9th ACM-SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, Washington, DC, August 2003, pp. 661-666.

[34] H. Yao and H.J. Hamilton, "Mining itemset utilities from transaction databases," *Data & Knowledge Engineering*, 59 (2006), 603-626.

[35] H. Yao, H.J. Hamilton, and C.J. Butz, "A foundational approach to mining itemset utilities from databases," in *Proceedings 4th SIAM Intl. Conf. on Data Mining*, Lake Buena Vista, FL, April 2004, pp. 482-486.

[36] H. Yao, H.J. Hamilton, and L. Geng, "A unified framework for utility based measures for mining itemsets," in *Proceedings ACM SIGKDD 2nd Workshop on Utility-Based Data Mining*, Philadelphia, Pennsylvania, August 2006, pp. 28-37.