Expert Systems with Applications 42 (2015) 5754–5778

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

An efficient approach for mining association rules from high utility itemsets

Jayakrushna Sahoo [a], Ashok Kumar Das [b], A. Goswami [a,*]

[a] Department of Mathematics, Indian Institute of Technology, Kharagpur 721 302, India
[b] Center for Security, Theory and Algorithmic Research, International Institute of Information Technology, Hyderabad 500 032, India

ARTICLE INFO

Article history:
Available online 11 March 2015

Keywords:
Data mining
High utility itemset mining
Association rule mining
Condensed representations
Non-redundant association rules

ABSTRACT

Traditional association rule mining based on the support–confidence framework provides the objective measure of the rules that are of interest to users. However, it does not reflect the semantic measure among the items. The semantic measure of an itemset is characterized with utility values that are typically associated with transaction items, where a user will be interested to an itemset only if it satisfies a given utility constraint. In this paper, we first define the problem of finding association rules using utility-confidence framework, which is a generalization of the amount-confidence measure. Using this semantic concept of rules, we then propose a compressed representation for association rules having minimal antecedent and maximal consequent. This representation is generated with the help of high utility closed itemsets (HUCI) and their generators. We propose the algorithms to generate the utility based non-redundant association rules and methods for reconstructing all association rules. Furthermore, we describe the algorithms which generate high utility itemsets (HUI) and high utility closed itemsets with their generators. These proposed algorithms are implemented using both synthetic and real datasets. The results demonstrate better efficiency and effectiveness of the proposed HUCI-Miner algorithm compared to other well-known existing algorithms. In addition, the experimental results show better quality in the compressed representation of the entire rule set under the considered framework.

1. Introduction

An expert system is a computer system, which emulates, or acts in all respects, with the decision-making capabilities of a human expert (Mishra, Das, & Mukhopadhyay, 2014). In general, there are three components associated with an expert system, which are (1) knowledge base, (2) inference engine, and (3) user interface (Huynh-Thi-Le, Le, Vo, & Le, 2015). The central of expert systems is the knowledge base as it has the problem solving knowledge of the particular application (Sadik, 2008). Alonso, Martinez, Perez, and Valente (2012) pointed out the cooperation between expert knowledge and data mining discovered knowledge. They also found that the expert knowledge and discovered knowledge are two powerful tools that can be combined together. Data mining techniques are useful in order to discover efficiently the hidden interesting and useful information from large databases, where the implication of interesting and useful information depends on the problem formulation and the application domain. An important data mining task that has received considerable research attention in recent years is the discovery of association rules from the transactional databases (Agrawal & Srikant, 1994; Han, Pei, & Yin, 2000; Park, Chen, & Yu, 1995; Webb, 2006). The traditional association rules mining (ARM) techniques depend on support confidence framework in which all items are given same importance by considering the presence of an item within a transaction, but not the profit of item in that transaction. The goal of such techniques is to extract all the frequent itemsets, where the itemsets having the given minimum support such that the support is the percentage of transactions containing the itemset, which generate all the valid association rules $A \rightarrow B$ from frequent itemset $A \cup B$ whose confidence has at least the user defined confidence such that the confidence is the percentage of transactions containing itemset B among the set of transactions containing A. In other words, given a subset of the items in an itemset, we need to predict the probability of the purchase of the remaining items in a transactional database. In general, from confidence of a rule generated from an itemset, we can know the percentage of number transactions of the items, which is sold together with remaining items of that itemset. However, we may not know the percentage of its profit obtained. Therefore, if we can know the percentage of the items' profit, we

* Corresponding author. Tel.: +91 3222 283650; fax: +91 3222 255303.
E-mail addresses: jayakrushnas@gmail.com (J. Sahoo), iitkgp.akdas@gmail.com, ashok.das@iiit.ac.in (A.K. Das), goswami@maths.iitkgp.ernet.in (A. Goswami).

http://dx.doi.org/10.1016/j.eswa.2015.02.051
0957-4174/

are in a position to find out a rule, which is more valuable than support and confidence, and as a result, it can allow us to permit with more accurate financial analysis and decisions. Nevertheless, this support–confidence framework does not provide the semantic measure of the rule but only it provides the statistical measure as the relative importance of items is not considered. However, such measure is not an adequate measure to the decision maker as the itemset cannot be measured in terms of stock, cost or profit, called utility. Consider a sales manager who aims to promote itemsets to increase the item selling. The following example is evident that a support–confidence based framework for association rule mining may mislead the manager in the decision making for determining the financial implications of an itemset.

Example 1. Consider the transaction database $\mathcal{D}$ shown in Table 1 that includes nine transactions t_1 through t_9 and eight items A through H. The numbers in the transaction database, which are bracketed, indicate the sales quantity for each item. Table 2 provides the unit profit for each item. The support and utility of the itemset DEF can be calculated using Tables 1 and 2 as 4 and 36, respectively, as the transactions containing DEF are t_2, t_3, t_4 and t_7. Since t_2 includes one D, four Es and five Fs, t_3 includes one D, five Es and one F, t_4 contains one D, two Es and six Fs, and t_7 contains one D, one E and four Fs, a total of four Ds, 12 Es and 16 Fs appear in transactions containing the itemset DEF. Using Table 2, the profit of items D, E and F are respectively 2, 1 and 1. Thus, the profit of the itemset DEF is 36. Using the standard confidence (Agrawal & Srikant, 1994), the confidence of the rule $D \rightarrow EF$ is $4/5 = 80\%$ as only 5 transactions containing in the item D, which are t_2, t_3, t_4, t_7 and t_8. Again, the confidence of the rule $F \rightarrow DE$ is $4/6 = 67\%$. The total utility of items D and F are then 22 and 20, respectively. The contribution of items D and F towards to the total profit of itemset DEF are 8 and 16, respectively. Therefore, if we consider the minimum confidence as 70%, the rule $F \rightarrow DE$ is an invalid rule, but the contribution of F from its utility is more than the contribution of item D towards to the total profit of itemset DEF. This clearly indicates that the selling of itemset DEF contributes a great portion to the total utility of F to 16 out of 20, and hence, the rule, $D \rightarrow EF$, having confidence above the user defined threshold, may mislead to the manager towards the value based decision making.

The support–confidence framework for association rule mining approach explained in Example 1 does not provide any additional knowledge to the manager except the measures that reflects the statistical correlation among items. In addition, it does not reflect their semantic implication towards the mining knowledge. In other words, the support–confidence model may not measure the usefulness of a rule in accordance with a user's objective (for example, profit).

In order to address the above shortcoming of support confidence framework, several researchers have focused on weighted association rule (Cai, Fu, Cheng, & Kwong, 1998; Ramkumar, Ramkumar, & Shalom, 1998; Tao, Murtagh, & Farid, 2003; Wang, Yang, & Yu, 2000). In such framework, the weights of items (the importance of items to the user) are considered and it also varies differently in application domains. However, this framework has two pitfalls. Firstly, these schemes still consider the support of an itemset to measure their importance and secondly, these models do not employ the quantities or prices of items purchased. Considering both quantities of items in a transaction and weights of items Carter, Hamilton, and Cercone (1997) proposed a share-confidence model to discover association rule among numerical attributes which are associated with items in a transaction. Carter et al.'s share-confidence model deals with the amount-share that is a fraction of total weight but not the utility value, such as the net profit, total cost (Geng & Hamilton, 2006). As a result, this model does not accomplish to conventional utility mining (Lin, Hong, & Lu, 2011; Liu, Liao, & Choudhary, 2005; Yao, Hamilton, & Butz, 2004) in which the requirements of decision makers are used to extract the itemsets with high utility, the utility of itemset is no less than the user specified minimum utility threshold, which are composed of weights and purchased quantities. The weight represents the importance of distinct items known as *external utility*, and the purchased quantity in each transaction is known as *internal utility* of the items. The product of external utility with sum total of internal utility of an item is called *utility* of the item. As utility does not satisfy downward closure property (Liu et al., 2005), most of the methods proposed in the literature are applied to find the candidate high utility itemsets first and then to identify actual high utility itemsets by an additional database scan. Some researchers proposed methods to find high utility itemsets without candidate generations (Fournier-Viger, Wu, Zida, & Tseng, 2014b; Liu & Qu, 2012) to avoid additional database scan. However, the discovering process of high utility itemsets takes more execution time and remains a challenge to formulate more effective algorithms. In this paper, we proposed an effective algorithm which is more than two times faster than the state-of-the-art algorithm for discovering high utility itemsets.

Table 1
An example transaction database $\mathcal{D}$.

T_{id}	Transaction
t_1	$A(4), C(1), E(6), F(2)$
t_2	$D(1), E(4), F(5)$
t_3	$B(4), D(1), E(5), F(1)$
t_4	$D(1), E(2), F(6)$
t_5	$A(3), C(1), E(1)$
t_6	$B(1), F(2), H(1)$
t_7	$D(1), E(1), F(4), G(1), H(1)$
t_8	$D(7), E(3)$
t_9	$G(10)$

Table 2
Utility table.

Item	Utility
A	3
B	4
C	5
D	2
E	1
F	1
G	2
H	1

1.1. Motivating examples of applications for utility-confidence framework

The share-confidence (we call it as the utility-confidence) model can be applied in various applications, including online purchases in e-commerce (Shie, Yu, & Tseng, 2013), retail sales (Barber & Hamilton, 2003; Hilderman, Hamiliton, Carter, & Cercone, 1998), cross-selling (Lee, Park, & Moon, 2013) and profit mining (Chen, Zhao, & Yao, 2007; Wang, Zhou, & Han, 2002). Note that the utility-confidence framework is also applicable to the market share rule (Zhang, Padmanabhan, & Tuzhilin, 2004), where the profit is obtained transaction-wise, but not item-wise. In this paper, we provide the examples, which are from the retail transaction datasets. To increase the profit, the manager decides to reward the customers who purchased more than some value and grant a discount on the purchase, or the manager offering a shipping discount may encourage buyer to buy additional items by which shipping is free

or discounted. For example, if a customer purchases itemset DEF, the shipping cost of F is free or discounted. In other way, if a customer purchases itemset DE, and if he also purchase itemset F, he will get some discount on item F.

In e-commerce retail business, not only the frequencies but also the profits of the products are crucial factors to analyze total profits of each item, decisions on product promotion, offering discount and shipment free decision. Consider a manager who handles a transaction set sketched by Table 1. The manager's main aim is to boost the revenue obtained from an itemset from a value at least equal to 50% to a value at least equal to 60% of the total profits of the product. To further increase the average sales the manager considers to offer some incentives so that customers find the product attractive. Therefore, to boost the sales the manager will give some discount on the product F while purchasing itemset DEF, so as to boost the revenue and as well as sales.

Another better example is suggested as the web recommendation system (Yan & Li, 2006). In this system, recommending web pages is proposed with the consideration of the importance of each page and the number of times the user visited a particular page. In such case, the product of importance of each page and the number of times that page visited gives the utility of the page. So, given a set of visited web pages, the share-confidence model can be useful to determine the dominating web pages in a web transaction records for web recommendation system.

The application of traditional share-confidence model is limited by the number of rules generated that are often not interesting to the user especially when the share threshold is very low and some interesting rules are missed when the share threshold is very high. In the extracted rule set, most of the rules share the same semantic measure or statistical measure with other rules, and they are called the redundant rules. Henceforth, it limits the usefulness of the rule set for the user to validate and take decisions. To represent high utility itemset compactly, Chan, Yang, and Shen (2003) introduced the concept of utility frequent closed patterns, where the notion of high utility itemset is different from that of the conventional utility mining. Later, Shie, Tseng, and Yu (2010) and Shie, Yu, and Tseng (2012) proposed the maximal high utility itemset, and Wu, Fournier-Viger, Yu, and Tseng (2011, 2014) proposed the high utility closed itemset. In these works, there is no algorithm was provided to generate rules. So, if we apply traditional association rule mining, the rules generated from these high utility itemsets would contain redundancy, and also of huge size as they compactly represent high utility itemsets but not the rules.

The above drawbacks and motivating applications motivate us to apply the amount confidence measure (Hilderman et al., 1998) to the utility database. As this measure applies on utility mining, we call it as utility-confidence framework as it allies to amount-confidence framework. The utility-confidence framework is used to mine non-redundant association rules among high utility itemsets, which enable the user to his/her perspectives concerning the importance of rules depending on values and provide useful information. In this framework, at first the high utility itemset with utility value larger than a predefined threshold is mined, and then the rules are generated among the itemsets, whose utility-confidences are larger than the user defined confidence threshold. To discover the non-redundant association rules in support–confidence framework, several approaches have been proposed in the literature (Balcázar, 2013; Sahoo, Das, & Goswami, 2014; Xu, Li, & Shaw, 2011; Yahia, Gasmi, & Nguifo, 2009). In general, the frequent closed itemsets (Pasquier, Bastide, Taouil, & Lakhal, 1999) and frequent generators (Zaki, 2004) are used to discover non-redundant association rules. However, these methods are developed for support–confidence framework. Therefore, in this paper we raise an important question as follows. How can we compress the association rules in utility-confidence framework? We aim to answer this question by integrating the concept of frequent closed itemsets and frequent generator to high utility itemset.

1.2. Our contributions

Our contributions are listed below:

- We first integrate the concept of minimal generators of support–confidence framework to the high utility mining.
- We then introduce the notion of non-redundant association rule in utility-confidence framework to represent association rules compactly and the corresponding algorithm to mine the non-redundant rules. An algorithm named *HUCI-Miner* (high utility closed itemset-Miner) algorithm has been proposed to mine high utility closed itemsets and also their generators simultaneously.
- To mine all high utility itemsets, another algorithm, called *FHIM* (Fast High-utility Itemset Miner), is proposed which is about two times faster than the state-of-the-art existing HUI mining algorithm. The proposed pruning strategies archive efficiency.
- We further propose an inference mechanism to generate all the association rules from the discovered non-redundant rule sets using the high utility closed itemsets.
- Finally, we conduct several experiments to show the efficiency of the proposed algorithms and compactness of the proposed non-redundant rule set.

1.3. The organization of the paper

The rest of the paper is sketched as follows. In Section 2, we briefly review the existing works proposed in the literature. In Section 3, we provide some basic preliminaries and our problem statement by introducing the concept of the utility-confidence measure. Section 4 introduces our proposed FHIM and HUCI-Miner algorithms. We also discuss the procedure for deriving high utility closed itemsets and generators. In Section 5, we provide the method for utility-based non-redundant association rule. In Section 6, we simulate the proposed algorithms method in a real dataset, and then evaluate the performance of our proposed algorithms and compare these with existing related approaches. Finally, the paper ends with the concluding remarks in Section 7.

2. Related work

In this section, we review the existing methods for association rules, high utility itemset mining, and generation of non-redundant association rules in support–confidence framework. In the support–confidence framework, the non-redundant association rules are generated from the frequent closed itemsets and their generators. We also review existing works on generation of frequent closed itemsets together with their generators as it is essential for generation of non-redundant association rules.

2.1. Association rules mining

Association rules discovery methods find the coincident occurrence of items and build the affinities among them in a transactional database. The methods in the literature are of two types: exhaustive search based algorithms and evolutionary based algorithms. The rule discovery process based on exhaustive search is straightforward, if the supports of the frequent itemsets are known. Hence, the frequent itemsets mining phase of the ARM methods plays a vital role in the rule discovery process. Several methods have been proposed in the literature, which are the variations of the two widely used methods: Apriori (Agrawal & Srikant,

1994) and FP-Growth (Frequent Pattern Growth) (Han et al., 2000), for discovering the frequent itemsets. The Apriori algorithm for discovering frequent itemsets is a level-wise candidate generation approach, which is based on the Apriori property. This property states that if an itemset is not a frequent, no superset of that itemset is frequent. This property reduces the search space, but it takes more database scan in order to calculate the frequency of itemsets and it results in increase in execution time and memory overhead. To overcome multiple database scan, Han et al. (2000) proposed the FP-Growth algorithm, which uses the divide-and-conquer and prefix based depth-first search procedures. In this approach, the database is first converted into a tree structure, named FP-tree, which is a compressed representation of original database. The algorithm then partitions the original database into smaller conditional database for a given itemset and its extensions are mined separately from the conditional database.

The basic evolutionary algorithms for mining association rules are ARMGA (Association Rule Mining Genetic Algorithm) (Yan, Zhang, & Zhang, 2005), QuantMiner (Salleb-Aouissi, Vrain, & Nortet, 2007), GENAR (Genetic Association Rules) (Mata, Alvarez, & Riquelme, 2001), and G3PARM (Grammar-Guided Genetic Programming for Association Rule Mining) (Luna, Romero, & Ventura, 2012). In ARMGA, each association rule is encoded into a single chromosome using the indices of items contained in the rule and another indicator indicates the separation of antecedent from the consequent. That means a chromosome of length k represents an association rule of length $k-1$. The algorithm generates a new chromosome by using the mutation and crossover operators of genetic algorithm. The GENAR (Mata et al., 2001) algorithm discovers the quantitative association rules using the genetic algorithm. Each individual in GENAR represents an association rule and keeps the information about the minimum and maximum values of each numerical attribute. The evolutionary algorithm calculates the fitness of each individual and processes the selection, crossover and mutation operators. QuantMiner (Salleb-Aouissi et al., 2007) is a genetic algorithm based algorithm, which is used to extract quantitative association rules. The algorithm learns the good intervals for numerical attribute and dynamically optimizes the intervals during the mining process and it depends on all the numerical attributes present in the rule. The knowledge from the extracted association rules using G3PARM (Luna et al., 2012) is more expressive and flexible. The algorithm uses the context-free grammar to represent an individual by defining the syntax constraints for both categorical and numerical attributes. The algorithm keeps the best individuals, whose support and confidence are larger than a certain threshold in an auxiliary population of a fixed size. To avoid optimizing the high number of parameters in non-deterministic search procedure, Luna, Romero, Romero, and Ventura (2014) proposed a grammar guided by genetic programming algorithm, which is free from parameters as other existing approaches. Their method discovers the quantitative association rules, which constitutes the small-size gaps.

2.2. High utility itemset mining

The traditional association rule mining methods (Agrawal & Srikant, 1994) are based on support–confidence framework, where all items are considered with the same level of importance. The methods proposed in Agrawal and Srikant (1994), Park et al. (1995), Han et al. (2000) and Pei et al. (2001) to extract association rule follow this classical statistical measurement producing the same result on a given minimum support and minimum confidence. The weighted association rule mining (WARM) generalizes the traditional framework by giving importance to items, where importance is given as weights. Ramkumar et al. (1998) introduced the concept of weighted support of itemsets and weighted association rules on the basis of costs assigned to both items and transactions. Later, considering only the item weights into account, Cai et al. (1998) proposed the weighted support of association rules. However, the weighted support of the association rules does not satisfy the downward closure property, which results in the performance degradation. In order to overcome such problem, by considering transaction weight, Tao et al. (2003) provided the concept of weighted downward closure property. Considering both support and weight of itemsets, Yun (2007) then presented a new strategy, called the weighted interesting pattern mining (WIP). Pears, Koh, Dobbie, and Yeap (2013) further proposed a WARM method that automates the process of weight assignment to the items by formulating a linear model.

In WARM framework, note that the quantities of items in transactions are not considered. Considering items' quantities in transactions and their individual importance, high utility itemset mining (HUIM) received a considerable research attention (Yao et al., 2004; Liu et al., 2005; Yao, Hamilton, & Geng, 2006). Yao et al. (2004, 2006) proposed a mathematical model of utility mining by generalizing the share-confidence model (Barber & Hamilton, 2001). As utility mining does not fulfill the *downward closure property*, Liu et al. (2005) proposed the two-phase algorithm that uses the *transaction-weighted downward closure property* to prune the candidate high utility itemsets in the first phase and then all the complete sets of high utility itemsets are obtained in the second phase. To reduce the number of candidate itemsets in the first phase, Li, Yeh, and Chang (2008) also proposed an isolated items discarding strategy (IIDS) to the level-wise utility mining method.

Ahmed, Tanbeer, Jeong, and Lee (2009) proposed a FP-Growth based algorithm that uses a tree structure, named IHUP-Tree, and efficiently generates the candidate high utility itemsets for incremental and the interactive mining. Yun, Ryang, and Ryu (2014) proposed a faster algorithm than IHUP (Ahmed et al., 2009) named MU-Growth (Maximum Utility Growth) with effective pruning strategies in mining process using the data structure MIQ-Tree (Maximum Item Quantity Tree). To further reduce the number of itemsets in the first phase, Tseng, Wu, Shie, and Yu (2010) and Tseng, Shie, Wu, and Yu (2013) proposed the tree-based methods, named the UP-Growth and UP-Growth$^+$, which use several strategies to decrease the estimated utility value of an itemset, and as a result, they enhance the performance. To avoid the level wise candidate generation and test strategy, Song, Liu, and Li (2014) proposed a concurrent algorithm, called the CHUI-Mine, for mining HUIs from transaction databases using their proposed data structure CHUI-Tree to maintain the information of HUIs. Their proposed algorithm generates the potential high utility itemsets using two concurrent processes: the first process is used for construction and dynamic pruning the tree, and then placing the conditional trees into a buffer, and the second one for reading the conditional pattern list from the buffer and mining HUIs. To speed up the execution and reduce the memory requirement in the mining process, Lan, Hong, and Tseng (2014) proposed an efficient utility mining approach that adopts a projection-based indexing mechanism that directly generates the required itemsets from the transactions database. Ahmed, Tanbeer, Jeong, and Lee (2011) proposed a novel tree-based candidate pruning technique, called the High Utility Candidates Prune (HUC-Prune), for avoiding more database scans and the level-wise candidate generation.

To avoid the computational cost of candidate generation and utility computation, Liu and Qu (2012) then proposed a data structure, named the utility-list, to store both the utility information about an itemset and the heuristic information for pruning the search space. Using the constructed utility-lists from a mined database, they developed an efficient algorithm, called the HUI-Miner,

which mines high utility itemsets without candidate generation in a depth-first search manner. Their algorithm works in a single phase by directly identifying high utility itemsets in an efficient way and it is also scalable. To reduce the cost of join operation in the calculation of the utility-list of an itemset in HUI-Miner, Fournier-Viger et al. (2014b) improved the HUI-Miner with incorporating the items co-occurrences strategy (named as FHM) which is about six times faster than the HUI-Miner.

2.3. Closed itemsets with their generators and non-redundant association rule mining

To generate both frequent closed itemsets (FCI) and generators, Pasquier et al. (1999) proposed the CLOSE algorithm that is based on level-wise searching approach with the help of Apriori property. Szathmary, Napoli, and Kuznetsov (2007) proposed the ZART algorithm that generates FCIs with their generators in a level-wise manner. They further proposed the Eclat-Z algorithm (Szathmary, Valtchev, Napoli, & Godin, 2008) that mines frequent itemsets in a depth-first way and the FCIs with their generators are identified in level-wise manner. An effective method, named as Touch (Szathmary, Valtchev, Napoli, & Godin, 2009), was developed by combining the FCI method Charm (Zaki, 2004) and the frequent generator (FG) mining algorithm, Talky-G (Szathmary et al., 2009). The FCIs are mined using Charm and FGs are mined using Talky-G and, then Touch associates the generators to their closed itemsets using a suitable hash function.

Wu et al. (2011) introduced the closer concept to high utility itemsets. They called the extracted itemsets as closed$^+$ high utility itemsets. On incorporating closure based on support of itemsets, they proved, first mining the set of high utility itemsets and then applying closed constraint produces the same result while mining all the closed itemsets first and then applying the utility constraint. They proposed an effective method named as CHUD (Closed$^+$ High Utility itemset Discovery) for mining closed$^+$ high utility itemsets. Further, they proposed a method called the DAHU (Derive All High Utility itemsets), to recover all high utility itemsets from the set of closed$^+$ high utility itemsets without further accessing the database. In addition, they proposed AprioriHC and AprioriHC-D algorithms (Wu et al., 2014) and mentioned that CHUD performs better than AprioriHC and AprioriHC-D. Later, Fournier-Viger, Wu, and Tseng (2014c) proposed two concise representations, which are the Generator of High Utility Itemsets (GHUIs) and High Utility Generators (HUGs), by adopting the concept of minimal generators of frequent itemset mining. However, no suitable method for high utility closed itemset with their generator was proposed for high utility itemset mining.

To reduce the number of association rules extracted in support confidence-framework, several methods have been developed in the literature (Balcázar, 2013; Kryszkiewicz, 1998; Sahoo et al., 2014; Xu et al., 2011; Yahia et al., 2009). Kryszkiewicz (1998) proposed the representative association rules (RR) with the help of a cover operator that represents a set of association rules. Zaki (2004) proposed a method to reduce the number of association rules and the extracted rules, called the general rules, which have shortest antecedent and shortest consequent giving an equivalence class of rules of same support and confidence. Pasquier, Taouil, Bastide, Stumme, and Lakhal (2005) defined the min–max rules having minimal antecedent and maximal consequent. Their proposed method eliminates the non-redundant rules as min–max exact and min–max approximate rules from the frequent closed itemsets and their generators. Furthermore, to reduce more rules, Cheng, Ke, and Ng (2008) proposed the concept of δ-tolerance, which is a relaxation on the closure defined on the support of frequent itemset. Yahia et al. (2009) proposed an informative basis to reduce the number of association rules, which is further efficiently compressed by Sahoo et al. (2014). Xu et al. (2011) filtered the min–max rules by defining redundancy and provided the reliable exact basis and reliable approximate basis of the same inference capacity. Balcázar (2013) further obtained a small and crisp set of association rules by the help of confidence boost of a rule, which eliminates the rules with similar confidence.

3. Basic preliminaries and problem statement

In this section, we first discuss some basic preliminaries and then we discuss the concept of the utility-confidence measure in Section 3.2.

Let $I = \{i_1, i_2, i_3, \ldots, i_m\}$ be a finite set of items, where each item $i_\ell, 1 \leqslant \ell \leqslant m$, have an external utility $p_\ell, 1 \leqslant \ell \leqslant m$ in the utility table. A subset $X \subseteq I$ is called an itemset, if X contains k distinct items $\{i_1, i_2, i_3, \ldots, i_k\}$, where $i_\ell \in I, 1 \leqslant \ell \leqslant k$, called a k-itemset. Let $\mathcal{D}$ be the task relevant database composed of utility table and the transaction table $T = \{t_1, t_2, t_3, \ldots, t_n\}$, containing a set of n transactions, where each transaction $t_d \subseteq I, 1 \leqslant d \leqslant n$, in the database be associated with a unique identifier, say t_{id}. In every transaction $t_d, 1 \leqslant d \leqslant n$, each item $i_\ell, 1 \leqslant \ell \leqslant m$ has a non-negative quantity $q(i_\ell, t_d)$, which represents the purchased quantity known as internal utility of the item i_ℓ in the transaction t_d.

3.1. The existing support–confidence measure

Each itemset X has a statistical measure called the support of X, which is defined by the ratio of the number of transactions containing X to the total number of transactions $|\mathcal{D}|$, and denoted by $supp(X)$. In other words, $supp(X) = \frac{|\{t|t \in \mathcal{D}, X \subseteq t\}|}{|\mathcal{D}|}$. Let $\mathcal{F}$ be the set of all the itemsets in $\mathcal{D}$ having positive support and $\mathcal{F} = \{X \mid X \in 2^{\mathcal{I}}, supp(X) > 0\}$. An association rule is an implication of the form $R : X \rightarrow Y$, where $X, Y \subseteq \mathcal{I}, Y \neq \emptyset$, and $X \cap Y = \emptyset$. The itemsets X and Y are called antecedent and consequent of the rule R, respectively. Association rules are associated with two statistical measures, which are support and confidence. The support of the rule R is $supp(X \cup Y)$ and the confidence of the rule R is defined by the ratio of the support of $X \cup Y$ to the support of X, and denoted by $conf(R)$. Hence, it is clear that $conf(R) = \frac{supp(X \cup Y)}{supp(X)}$.

Support and confidence are used as interesting measures for determining the importance of an association rule. The support of a rule precisely provides the statistical useful information of the corresponding itemset from which the rule is derived but it does not indicate the relative importance of itemsets, which limits its financial implications. The confidence measure indicates the statistical conditional measure of the interestingness of the association rule. However, it does not provide any value based decision as indicated in Example 1. Note that there are other interesting measures based on the probability, such as Lift, Conviction and Leverage, which are explained in detail in Geng and Hamilton (2006). However, these interesting measures do not consider the quantity of each item in a transaction and the profit of each item.

3.2. The utility-confidence measure

In this section, we describe the concept of the utility confidence measure, which is useful for our proposed algorithms later in this paper. For this purpose, we use x_i to denote the ith item of an itemset X.

Definition 1. The utility of an item i_ℓ in a transaction t_d is denoted by $u(i_\ell, t_d)$ and defined by the product of internal utility $q(i_\ell, t_d)$ and external utility p_ℓ of i_ℓ, that is, $u(i_\ell, t_d) = p_\ell \times q(i_\ell, t_d)$.

Definition 2. The utility of an itemset X contained in a transaction t_d, denoted by $u(X, t_d)$ and defined by the sum of utility of every items of X in t_d. In other words, $u(X, t_d) = \sum_{i_\ell \in X \wedge X \subseteq t_d} u(i_\ell, t_d)$.

Definition 3. The utility of an itemset X in $\mathcal{D}$ is denoted by $u(X)$ and defined by the sum of the utilities of X in all the transactions containing X in $\mathcal{D}$, that is, $u(X) = \sum_{X \subseteq t_d \wedge t_d \in \mathcal{D}} u(X, t_d) = \sum_{X \subseteq t_d \wedge t_d \in \mathcal{D}} \sum_{i_\ell \in X} u(i_\ell, t_d)$. The set of transactions containing an itemset X, in database $\mathcal{D}$ is called the projected database of itemset X and it is denoted by $\mathcal{D}_X$.

Definition 4. An itemset X is called a high utility itemset, if the utility of X has at least the user specified minimum utility threshold, *min_util*. Otherwise, it is called a low utility itemset. Let $\mathcal{H}$ be the complete set of high utility itemsets. Then, $\mathcal{H} = \{X \mid X \in \mathcal{F}, u(X) \geqslant min\_util\}$.

Example 2. Consider again the transaction database in Table 1 with the utility table given in Table 2. From Table 2, note that the external utility of item B is 4 and the internal utility of the item B in the transaction t_3 is 4. Thus, the utility of item B in t_3 is $u(B, t_3) = p_B \times q(B, t_3) = 4 \times 4 = 16$. The utility of the itemset BF in the transaction t_6 is $u(BF, t_6) = u(B, t_6) + u(F, t_6) = 4 \times 1 + 1 \times 2 = 6$ and the utility of the itemset BF in $\mathcal{D}$ becomes $u(BF) = u(BF, t_3) + u(BF, t_6) = 23$. Note that, if the minimum utility threshold is 20, the itemset BF is a high utility itemset.

Definition 5. The local utility value of an item x_i in an itemset X, denoted by $luv(x_i, X)$ and defined by the sum of the utility values of the items x_i in all the transactions containing X, that is, $luv(x_i, X) = \sum_{X \subseteq t_d \wedge t_d \in \mathcal{D}} u(x_i, t_d)$.

Definition 6. The local utility value of an itemset X in another itemset Y such that $X \subseteq Y$, denoted by $luv(X, Y)$, is the sum of local utility measure values of each item $x_i \in X$ in itemset Y, which is given by $luv(X, Y) = \sum_{x_i \in X \subseteq Y} luv\ (x_i, Y)$.

To calculate the local utility value of an itemset X in another itemset Y such that $X \subseteq Y$, an *utility unit array* needs to be attached to each high utility itemset, which is defined as follows.

Definition 7 Wu et al., 2011. The *utility unit array* of an itemset $X = \{i_1, i_2, i_3, \ldots, i_k\}$ is denoted by $\mathrm{U}(X) = \{u_1, u_2, u_3, \ldots, u_k\}$, where each u_ℓ is $luv(i_\ell, X), 1 \leqslant \ell \leqslant k$.

Example 3. The *utility unit array*, $\mathrm{U}(X)$ of an itemset X contains the local utility values of the constituent items of X. Consider the itemset ACE which appears in transactions t_1 and t_5 in Table 1. The local utility value of the item A in ACE is $luv(A, ACE) = u(A, t_1) + u(A, t_5) = 21$. The utility unit array of ACE is $\mathrm{U}(ACE) = \{21, 10, 7\}$. Further, the local utility value of itemset AE in ACE is $luv(AE, ACE) = luv(A, ACE) + luv(E, ACE) = 28$.

Property 1 Wu et al., 2011. *For a given itemset X with its utility unit array $\mathrm{U}(X)$, the utility of X is defined as $u(X) = \sum_{x_i \in X} luv(x_i, X)$.*

We follow the definition of amount-confidence measure in association rule mining from Hilderman et al. (1998) and define the following utility-confidence for an association rule.

Definition 8. An association rule is an implication of the form $R : X \rightarrow Y$, where $X, Y \subseteq I, X(\neq \emptyset)$ is known as antecedent of the rule, $Y(\neq \emptyset)$ is known as consequent of the rule, and $X \cap Y = \emptyset$. The utility-confidence of the rule R is denoted by $uconf(R)$ and defined by $uconf(R) = \frac{luv(X, X \cup Y)}{u(X)}$.

Rationale 1. The proposed utility-confidence measure becomes the confidence measure when the internal utility and external utility of the items of an itemset are set to 1. The utility-confidence measure shows the fraction of total utility of the antecedent contributed to the utility of the corresponding itemset. So, the utility-confidence of the rule $R : X \rightarrow Y$ reflects the share of utility of itemset X to the utility of $X \cup Y$. That means that the fraction of utility of X is covered by $X \cup Y$. It is similar to the amount-confidence measure except that it is based on utility value of itemset such as profits in dollars rather than amount-share that is a fraction of total weights.

We say an association rule in utility-confidence framework is valid, if it satisfies the following two conditions:

(i) The antecedent and itemset formed by combination of antecedent and consequent are high utility itemset.
(ii) The utility-confidence is more than or equal to the specified minimum utility-confidence threshold, say *min_uconf*.

Example 4. Consider the transaction table shown in Table 1, and the utility table in Table 2. Let the minimum utility threshold and minimum utility-confidence threshold be 20 and 80%, respectively. Then, the utility of the itemset BDE, whose utility is 23 by Definition 3, is a high utility itemset and contained in the transaction t_3 only. If we consider the rule $B \rightarrow DE$, the utility-confidence of $B \rightarrow DE$ becomes $\frac{luv(B, BDE)}{u(B)} = \frac{u(B, t_6)}{u(B)} = \frac{16}{20} = 80\%$.

Generation of valid utility based association rules from high utility itemsets is relatively straightforward. The rules of the form $R : X \rightarrow Y$ are generated for all high utility itemsets X, and $X \cup Y$, for all $X, Y \neq \phi$, and the rule R provides the utility-confidence of the rule having at least *min_uconf*. Since $X \cup Y$ is a high utility itemset, the generated rule is guaranteed to be high utility. To derive all possible valid rules, we need to examine each high utility itemset and repeat the rule generation process as in Apriori algorithm (Agrawal & Srikant, 1994) with utility constraint. When the minimum utility value is $min\_util = 20$, Table 3 shows all the high utility itemsets. Again, considering the minimum utility-confidence $min\_uconf = 80\%$, Table 4 shows the set of all utility based association rules with confidence above or equal to $min\_uconf = 80\%$.

It can be observed that in Table 4 some rules are redundant to others, for example, $A \rightarrow C, A \rightarrow E$, and $AC \rightarrow E$ are redundant to $A \rightarrow CE$. To eliminate the non-redundant rules in share-confidence framework, we integrate the concept of frequent closed itemsets and frequent generator to high utility itemset mining, and the overall process of all the non-redundant utility based association rules generation is given in Fig. 1. We refer the utility based

Table 3
Extracted HUIs with minimum utility 20.

Itemset	Utility	Itemset	Utility
A	21	*B*	20
D	22	*G*	22
E	22	*F*	20
AC	31	*AE*	28
BE	21	*DF*	24
BF	23	*DE*	37
FE	36	*ACE*	38
AFE	20	*DFE*	36
BDE	23	*BFE*	22
ACFE	25	*BDFE*	24

Table 4
Extracted high utility valid rules with minimum utility-confidence 80%.

Rules	Uconf (%)	Rules	Uconf (%)
$A \rightarrow C$	100	$A \rightarrow E$	100
$B \rightarrow E$	80	$F \rightarrow E$	90
$B \rightarrow F$	100	$D \rightarrow E$	100
$E \rightarrow F$	81	$F \rightarrow D$	80
$A \rightarrow CE$	100	$AC \rightarrow E$	100
$AE \rightarrow C$	100	$B \rightarrow DE$	80
$BE \rightarrow D$	100	$B \rightarrow FE$	80
$BE \rightarrow F$	100	$F \rightarrow DE$	80
$DF \rightarrow E$	100	$AFE \rightarrow C$	100
$B \rightarrow DEF$	80	$BE \rightarrow DF$	100
$BDE \rightarrow F$	100	$BEF \rightarrow D$	100

association rules as high utility association rules (HAR) and utility based non-redundant association rules as non-redundant high utility association rules (NHAR). To identify the high utility closed itemsets with their generators, we propose the HUCI-Miner algorithm. Since the identification of high utility closed itemsets with their generators takes all high utility itemsets as inputs, so another algorithm FHIM is proposed to discover all high utility itemsets efficiently. After identifying the high utility closed itemsets with their generators, the set of NHAR is obtained, which is discussed in Section 5. On demand of a user, the set of all rules is derived from the set of NHAR and high utility closed itemsets.

4. The proposed HUCI-Miner algorithm

In this section, we first discuss some useful definitions and theorems before describing our proposed algorithm.

The Apriori property used to prune the candidate itemset search space cannot be applied directly to mine high utility itemset, since the utility constraint is neither monotone nor anti-monotone. To reduce the size of search space and enhancing the performance of mining task, Liu et al. (2005) proposed the concept of transaction-weighted utility, which satisfies the downward closure property and is based on the following definitions.

Definition 9. The utility of a transaction t_d is denoted by $tu(t_d)$ and defined by $tu(t_d) = u(t_d, t_d)$.

Definition 10. The transaction-weighted utility (TWU) of an itemset X in a database $\mathcal{D}$ is denoted by $twu(X)$ and defined by the sum of the utilities of all the transactions containing X in $\mathcal{D}$, where $twu(X) = \sum_{t_d \in \mathcal{D} \wedge X \subseteq t_d} tu(t_d)$.

Property 2. *The transaction-weighted utility satisfies the downward closure property. That means for a given itemset X, if twu(X) is less than the specified min_util, all supersets of X are not high utility.*

Table 5 shows the transaction-weighted utility of each item. For example, the transaction utility of t_1 is $tu(t_1) = u(A, t_1) + u(C, t_1) + u(E, t_1) + u(F, t_1) = 12 + 5 + 6 + 2 = 25$. Again, consider the itemset *ACE* which is in transaction t_1 and t_5 having the transaction-weighted utility of $ACE, twu(ACE) = tu(t_1) + tu(t_5) = 25 + 15 = 40$. If the *min_util* is set to 45, all supersets of *ACE* are not high utility itemset according to Property 2. For a given itemset, if its transaction-weighted utility has at least *min_util*, we call the itemset as high transaction-weighted utility itemset (HTWUI).

Definition 11. An itemset Y is called the *closure* of an itemset X, denoted by $\gamma(X)$, if there does not exist other large superset of X than Y, with $supp(X) = supp(Y)$. An itemset X is then called the closed itemset, if $X = \gamma(X)$.

Property 3. *For a given itemset* $X, twu(X) = twu(\gamma(X))$. *In other words, the transaction-weighted utility of an itemset is same as its closure.*

Rationale 2. Since the transactions containing an itemset X also contains its closure $\gamma(X)$, their *TWU* values must be same. For example, the itemsets *A* and *CE* have support 2 and belong to the same set of transactions $\{t_1, t_5\}$ in database given in Table 1 and they have the same *TWU* value equals to 40.

Property 4 Wu et al., 2011. The *utility* and *utility unit array* of an itemset X can be calculated from the *utility unit array* of its *closure* itemset $\gamma(X)$.

Property 5. If an itemset X is a high utility itemset, $\gamma(X)$ is also a high utility itemset. However, the converse is not always true.

Table 5
TWU of each item of database given in Table 1.

Item	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*
TWU	40	31	40	72	112	87	30	17

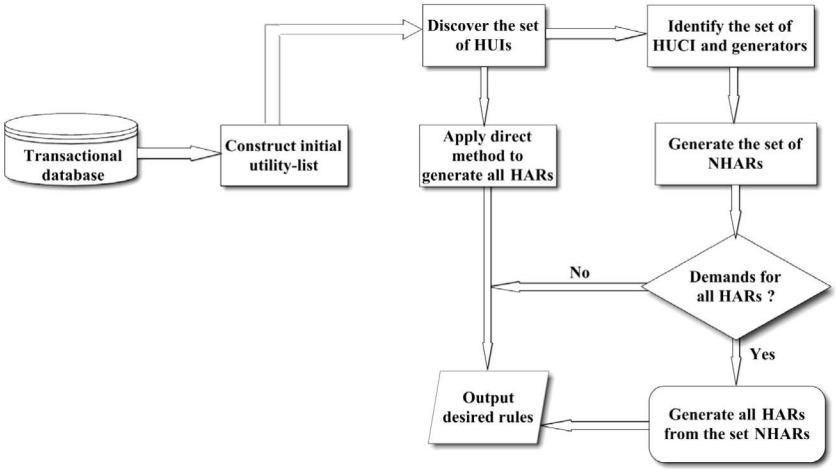


Fig. 1. The overall process of generation of utility based non-redundant association rules.

Rationale 3. The transactions containing an itemset X also contains its closure $\gamma(X)$. Moreover, if X is not closed itemset, $\gamma(X)$ contains more number of items than X as $X \subset \gamma(X)$. So, if X is a high utility itemset, $u(X) \geqslant min\_util$ and so also $u(\gamma(X))$ as it is the sum of utilities of X with some non-negative quantity. For the converse, consider the itemsets *CE* and *ACE*. It can be observed from Table 1 that *ACE* is the closure of *CE*. The utility values of *CE* and *ACE* are 17 and 38, respectively. If we consider the minimum utility threshold as 20, *ACE* is a high utility itemset, but *CE* is not a high utility itemset.

4.1. Integrating the closure property with HUIM

In this subsection, we discuss how to unify the minimal generator concept of the traditional ARM into high utility itemset mining. In general, the itemsets in a transactional database are not completely independent from other itemsets. A group of itemsets are common to the same set of transactions, and hence, they have the same support. Using the closure operator the itemsets can be grouped into equivalent classes. Two itemsets in a class are called equivalent if they belong to the same set of transactions. A maximal element in a class, is called the closed itemset, which is the closure of other itemsets in that class, and the minimal elements (smallest subsets of the maximal element of the class) are called the generators. All other elements in a class can be derived using the closed itemsets, and the generators and all the elements in a class have the same support. The closed itemsets with their generators lead to the fundamental principle behind the effective construction of non-redundant association rules in the support–confidence framework.

A natural question arises that how to incorporate the similar strategy in HUIM, which means the formation of high utility closed itemset together with their generators those are also high utility itemsets. As suggested by Wu et al. (2011), the closure based on utility of itemset does not achieve a high reduction on the number of high utility itemset and they defined the closure on the supports of itemsets. On incorporating closure based on support of itemsets, they showed that the join order of closed constraint and utility constraint is commutative. This means, first mine the set of high utility itemsets and then apply closed constraint which will produce the same result while mining all the closed itemsets first and then applying the utility constraint. Concluding that any equivalence class constructed among high utility itemsets using closure based on support, the maximal element can be found in both ways. However, later we show that, this commutativeness between utility constraint and closure based on support does not hold for generators. As closed high utility itemsets and high utility closed itemsets are same, so Wu et al. (2011) defined the closed$^+$ high utility itemset (CHUI).

Definition 12. An itemset X is called high utility closed itemset, if $X = \gamma(X)$ and $u(X) \geqslant min\_util$.

4.1.1. Generators with utility constraints

Following the definition of generators of a closed itemset in traditional ARM, the problem arises how we can incorporate it to high utility itemset mining. This can be done in two ways. First mine all generators based on support constraint and then apply utility constraint. Secondly, first find all high utility itemset and then mine all generator applying support constraint. We will analyze this joining order between utility constraint and support constraint to extract the generators and conclude that the result of both ordering are different. In first case, we miss the minimal high utility itemsets of an equivalence class and in the later case we do not lose the minimal high utility itemsets. As a consequence, we suggest finding the minimal generators among high utility itemsets. Thus, the utility constraint should be considered first and then support constraint.

For first composition method, we mine generators based on support and then prune itemsets which do not satisfy the utility constraints. Based on composition order, we can define *generator with high utility* as follows.

Definition 13 (*Generator with high utility*). An itemset X is a *generator with high utility*, if there is no proper subset Z of X such that $supp(Z) = supp(X)$. Moreover, X must satisfy the utility constraints.

Definition 13 simply states that in order to mine *generator with high utility*, using support constraint, the itemsets are first verified whether they are generators or not. Then the itemsets are tested for high utility itemset with utility constraints. So, if an itemset is not a generator, the itemset is not a *generator with high utility*, without verifying with utility constraints. For example, the itemset $AC(2, 31)$ (itemset(support,utility)) in the transactional database given in Table 1 with minimum utility constraint 20, is not a *generator with high utility* without testing with utility constraint as it is not a generator because there is a subset $A(2, 21)$ whose support is same as support of $AC(2, 31)$. Again, the itemset $AF(1, 14)$ is a generator but not a *generator with high utility* as it does not satisfy the utility constraint of minimum utility 20.

Since the above composition has no restriction on the utility of the subset of generators, so after applying utility constraint some generators are pruned. As all the itemsets of an equivalence class can be generated from both closed itemsets and their generators, if some generators are pruned, some subsets of that itemset may not be generated. For example, assume that the minimum utility constraint is 20 in the transactional database shown in Table 1. Consider the equivalence class of the itemset $ACFE(1, 25)$. The generators of this closed itemset are $AF(1, 14)$ and $CF(1, 7)$. After applying the utility constraint, note that there is no generator of $ACFE(1, 25)$ because both $AF(1, 14)$ and $CF(1, 7)$ are pruned as their utility is less than 20. However, from Table 3 we observe that the itemset $AFE(1, 20)$ is a high utility itemset belonging to the equivalence class of $ACFE$ $(1, 25)$. But in the generation process of generators, the itemset $AFE(1, 20)$ is pruned as it is the superset of $AF(1, 14)$ with same support. In other words, before $AF(1, 14)$ is pruned by the utility constraint, the itemset $AFE(1, 20)$ is pruned by itemset $AF(1, 14)$ as both have same support and later one is the subset of the former. As a result, the generator itemset of $ACFE(1, 25)$ is empty. So, while applying traditional itemset generation procedure from generators of an equivalence class, the itemset $AFE(1, 20)$ will not generate. This problem arises because of considering the closure property first and the utility constraint later.

The other way to join the utility constraints and the closure property is that we need to first mine all the itemsets with utility constraint and then apply closure property to compute the generators. From this ordering, we can define *high utility generators* as follows.

Definition 14 (*High utility generators*). An itemset X is a *high utility generator*, if it is a high utility itemset and there exists no proper high utility subset Z such that $supp(Z) = supp(X)$.

In this approach after extracting all high utility itemsets, the closure property is applied to compute the generators. Again, from these high utility closed itemsets and their generators, all other high utility itemsets of that equivalent class can be generated using traditional method by combining the itemsets. From the analysis, the results of the joining order between the two constraints are different.

Rationale 4. The basic difference between Definitions 13 and 14 is based on the constraint applied to the subset Z. Even though the generator takes the utility constraint into account in both definitions, but these definitions differ the way utility constraint is applied. In Definition 13, initially there is no restriction on the

utility constraint of Z, whereas in Definition 14 it is for high utility itemset. On the other hand, in Definition 13, initially the support is applied and later the utility constraint, which is opposite to that of Definition 14.

4.1.2. Winding up the discussions

It is well-defined from the analysis, we loose some minimal high utility itemsets, if we first find generators based on support and then apply utility constraint. Nevertheless, in Definition 14, we generate the actual high utility generators. As a consequence, the second approach generates high utility minimal generators of an equivalence class in high utility itemset mining. Throughout this paper we apply the second approach and now onwards we mean the generator means the *high utility generator*.

Assume there is a pre-determined total ordering Ω among the items I in the database $\mathcal{D}$. Accordingly, if item i is occurred before item j in the ordering, we denote this by $i \prec j$. For $\forall j \in Y$, if $i \prec j$, we say $i \prec Y$, where Y is an itemset. Similarly, for itemsets X and Y if $x_i \prec Y_i$ in accordance to the order relation Ω, for $1 \leqslant i \leqslant m, m = min(|X|, |Y|)$, we say $X \prec Y$. This ordering can be used to enumerate all the itemsets without duplication. Hereinafter, we always consider an itemset as an *ordered set*, in particular, it is a sequence of distinct and increasingly sorted items with respect to the TWU values of items. If the TWU values of two items are equal, they are sorted according to the lexicographic order.

Let f be a function that assign to each itemset $X \in \mathcal{H}$ to the set of all transactions that contain X, that is, $f(X) = \{t_d \in T \mid X \subseteq t_d, X \in \mathcal{H}\}$. Clearly, for $X \subset Y, f(Y) \subseteq f(X)$. Two itemsets $X, Y \in \mathcal{H}$ are said to be equivalent, denoted by $X \cong Y$, iff $f(X) = f(Y)$. The set of itemsets that are equivalent to an itemset X is denoted by $[X]$ and given by $[X] = \{Y \in \mathcal{H} \mid X \cong Y\}$.

Theorem 1. *Let $X \in \mathcal{H}$. If $luv(X, Y) = u(X)$, then $Y \in [\gamma(X)]$.*

Proof. Since $X \subset Y, luv(X, Y) = u(X)$ and $X \in \mathcal{H}$. We then have $Y \in \mathcal{H}$. From Definitions 3, 5 and 6, both X and Y are contained in same transaction. Thus, $f(X) = f(Y) = f(\gamma(X))$ and $Y \cong \gamma(X)$. Hence, $Y \in [\gamma(X)]$. □

Corollary 1. *If $supp(X) = supp(Y)$ and $X \subseteq Y$, then $luv(X, Y) = u(X)$.*

Corollary 2. *For a given itemset X, if there does not exist any itemset $Y \supset X$ such that $luv(X, Y) = u(X)$, then X is a closed itemset.*

Definition 15. An itemset $X \in [X]$ is called a generator, if X has no proper subset in $[X]$. In other words, it has no proper subset with the same support and it is a high utility itemset.

Theorem 2. *Let X be a high utility itemset and $x \in X$. If $X \setminus x$ is a high utility itemset, $X \in [X \setminus x]$ iff $supp(X) = supp(X \setminus x)$.*

Proof. Let $x \in X$ and $X \setminus x$ be a high utility itemset. Let $X \in [X \setminus x]$. Then $f(X) = f(X \setminus x)$ means that $supp(X) = supp(X \setminus x)$. The converse follows trivially. □

Theorem 3. *Let X be a high utility itemset. Then, X is a generator iff $supp(X) \neq min\{supp(X \setminus x) : x \in X, (X \setminus x) \in \mathcal{H}\}$.*

Proof. Let X be a generator. Let g be a high utility itemset of length $k-1$ with minimum support and a subset of X. Then, $g \subset X \Rightarrow f(g) \supseteq f(X)$. If $f(g) = f(X), supp(X) = supp(g)$ and X is not a generator. Moreover, it is not the element with the smallest support, whose closure is $\gamma(X)$. This concludes that $f(g) \supset f(X)$ and hence, $supp(X) \neq min\{supp(X \setminus x) : x \in X, (X \setminus x) \in \mathcal{H}\}$. On the other hand, if $supp(X) \neq min\{supp(X \setminus x) : x \in X, (X \setminus x) \in \mathcal{H}\}, X$ is the smallest element of the closure $\gamma(X)$. Hence, X is a generator. □

Corollary 3. *Let X be a high utility itemset. If X is not a generator, then $supp(X) = min\{supp(X \setminus x) : x \in X, (X \setminus x) \in \mathcal{H}\}$.*

Property 6. *For a high utility closed itemset X, if g be a generator, then $u(g, X) < u(g', X)$, where $g' \in [X]$ and $g \subset g'$.*

Rationale 5. Since $g' \in [X]$, both g and g' belong to the same set of transactions. Furthermore, as $g \subset g'$, Property 6 holds as a consequence of Property 5.

Theorem 4. *Let $X \in \mathcal{H}$. The statement "If X is a generator, then $\forall Y \in \mathcal{H}, Y \subset X, Y$ is a generator" is false in this context.*

Proof. This can be proved by giving a counter example. Consider the itemset *AFE* with support 1 and utility value 20. If the minimum utility is set to 20, this is a generator in the context. However, the subsets *AF* and *AE* are not generators. Note that *AF* is not a high utility itemset, whereas *AE* becomes a high utility itemset. □

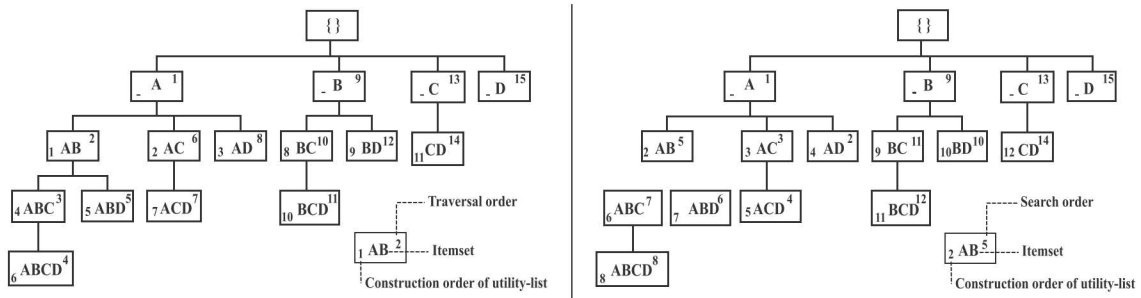
Rationale 6. Theorem 4 states that the subsets of a high utility generator may or may not be a generator. However, in the support–confidence framework the subsets of a generator are generators, and hence, if an itemset is not a generator, its superset is not also a generator. This property is used to prune the itemsets space to obtain the generators in support–confidence framework. Since this property does not satisfy in the high utility context, this pruning strategy cannot be employed. *AF* is pruned as it is not a high utility itemset, and *AE* is pruned as it is not a generator. Thus, by the traditional procedure of a support–confidence framework, the itemset *AFE* will not be generated. However, it is a minimal itemset in [*AFE*].

4.2. Mining high utility itemsets

It is observed from the analysis in Section 4.1 that in order to extract all high utility generators, we need to discover the entire space of high utility itemsets. It is thus necessary to provide an effective procedure to generate all high utility itemsets. In this section, using the concept of vertical mining of HUI (Liu & Qu, 2012), FHM (Fournier-Viger et al., 2014b), and the proposed pruning strategies, an efficient method is proposed to extract all high utility itemsets.

4.2.1. Vertical high utility itemset mining

To avoid the computational cost of candidate generation and multiple database scans for utility computation, a vertical mining method named as HUI-Miner, proposed by Liu and Qu (2012). The HUI-Miner algorithm generates the high utility itemsets in a single phase using a vertical representation of the database. To represent the database vertically, it uses a novel data structure, named *utility-list* to each itemset. To generate all the high utility itemsets, it performs a depth-first search in a pre-order way, from left-to-right as shown in Fig. 2. The utility and *utility-list* of a k-itemset is constructed from the *utility-list* of $(k-1)$-itemsets. The symbol '–' in Fig. 2 represents that the *utility-list* of the items are generated simultaneously from the mined database. Note that



(a) Pre-order traversal in (Fournier-Viger et al., 2014b; Liu and Qu, 2012)

(b) Reverse pre-order traversal in a sub-tree of the root in FHIM

Fig. 2. The search space traversal.

there is no tree used in the algorithms, Fig. 2 only shows the generation of itemsets in a recursive call.

Definition 16 Liu and Qu, 2012. Let Ω be a pre-determined total ordering among the items I in a database $\mathcal{D}$. Given an itemset X and a transaction $t, X \subseteq t$, after sorting the transaction t and X according to the order Ω, the set of items after X in t is denoted by t/X. For a given itemset X in $\mathcal{D}$, the *utility-list* of X is a collection of tuples $(tid, iutil, rutil)$ for each transaction t_{tid} containing X. The utility of X in t_{tid}, denoted by *iutil*, is the sum of the utilities of all items in t_{tid}/X, that is, $rutil = \sum_{i \in t_{tid}/X} u(i, t_{tid})$. We denote the *utility-list* of X as $UL(X)$.

Example 5. Consider the transaction t_1 in Table 1. After sorting it according to the order Ω, we have $t_1 = \{A, C, F, E\}$. For the item C in t_1, we have $rutil = u(F, t_1) + u(E, t_1) = 8$. The *utility-list* element of C for transaction t_1 is $(1, 5, 8)$. In the second scan of the database $\mathcal{D}$, the procedure generates the *utility-list* for each promising item as shown in Table 6. The construction of the *utility-list* of a k-itemset X, where $k \geqslant 2$, can be obtained using the *construct* algorithm (Liu & Qu, 2012).

Property 7 Liu and Qu, 2012. For a given itemset X with its *utility-list*, if the sum of *iutil* values of the *utility-list* of X is less than *min_util*, X is not a high-utility itemset. Otherwise, it is a high utility itemset.

Property 8 Liu and Qu, 2012. Let X be an itemset with its *utility-list*. Given an ordering Ω, an itemset Y is an extension of X, if $Y = X \cup Z$, for some Z with $X \prec Z$. Given a minimum utility threshold *min_util*, if the sum of all the *iutil*'s and *rutil*'s in *utility-list* is less than *min_util*, there does not exist any extension Y of X, which is a high utility itemset. We call the sum of all the *iutil*'s and *rutil*'s in *utility-list* of a given itemset X as the promising utility of X.

4.2.2. Co-occurrence-based pruning

To reduce the cost of joining operations in the calculation of *utility-list* for an itemset *Pxy*, Fournier-Viger et al. (2014b) proposed the Estimated Utility Co-occurrence Pruning (EUCP) strategy to directly eliminate a low-utility extension of *Pxy* and all its transitive extensions without constructing their utility-list. They used the TWU pruning strategy (Liu et al., 2005) to eliminate a larger itemset of length more than three using the TWU value of all its subset of length two. The TWU value of each 2-itemsets is stored in a matrix, known as the Estimated Utility Co-Occurrence Structure (EUCS), which is defined as follows.

Definition 17 (*Estimated Utility Co-occurrence Structure (EUCS) (Fournier-Viger et al., 2014b)*). EUCS is a collection of tuples of the form $(a, b, c) \in I' \times I' \times R$ such that $TWU(ab) = c$.

The EUCS can be represented as a triangular matrix or a hash map. To achieve the memory efficiency in implementation, the hash map is used because EUCS is sparse in nature. Most of the entries of EUCS are zero, which indicates that very few items co-occurs with the rest items. The construction of EUCS is performed in a single database scan at the time of construction of *utility-list*. It occupies very less memory and is the order of $\mathcal{O}(|I'|^2)$. Table 7 shows the EUCS of the promising items of the database D given in Table 1.

Property 9 (*Estimated Utility Co-occurrence Pruning (EUCP) (Fournier-Viger et al., 2014b)*). *For a given itemset $X = \{x_1, x_2, x_3, \ldots, x_k\}$ and an item y, if there is no tuple (x_k, y, c) in EUCS such that $c <$ min_util, then Xy and no superset of Xy are high utility itemsets.*

This pruning strategy simply concludes that for the itemset $X = \{x_1, x_2, x_3, \ldots, x_k\}$, if the value of the entry (x_k, y) is less than the minimum utility, then there is no need to explore its any extension in the itemset space. Integrating EUCP strategy in HUI-Miner, called the algorithm as FHM (Fast High-Utility Miner) (Fournier-Viger et al., 2014b), leads to reduce the number of joining operations that are needed for formation of *utility-list* of a larger itemset. This strategy is actually performed before the joining operations to be performed. If the value is higher than the given *min_util*, the join operation is executed; otherwise, the larger itemset need not to be explored.

It is observed that in depth-first search strategy, the itemset search space problem is partitioned into non-overlapping sub-

Table 6
utility-list of each promising item.

Item	*Utility-list*
G	$\langle(7,2,7),(9,20,0)\rangle$
B	$\langle(3,16,8),(6,4,2)\rangle$
A	$\langle(1,12,13),(5,9,6)\rangle$
C	$\langle(1,5,8),(5,5,1)\rangle$
D	$\langle(2,2,9),(3,2,6),(4,2,8),(7,2,5),(8,14,3)\rangle$
F	$\langle(1,2,6),(2,5,4),(3,1,5),(4,6,2),(6,2,0),(7,4,1)\rangle$
E	$\langle(1,6,0),(2,4,0),(3,5,0),(4,2,0),(5,1,0),(7,1,0),(8,3,0)\rangle$

Table 7
EUCS of promising items.

Item	G	B	A	C	D	F
B	0					
A	0	0				
C	0	0	40			
D	9	24	0	0		
F	9	30	25	25	54	
E	9	24	40	40	71	79

problems such that the itemset search space of each item i is not included in the search space of the item j, with $i \prec j$ (Wu et al., 2011). In the procedure of HUI-Miner and FHM, the promising utility of item j does not contain the utility value of any item i such that $i \prec j$. The estimated utility co-occurrence value of item pair (j,k) can be further reduced after completion of the subproblem of item i such that $i \prec j$ and $i \prec k$. This means that the utility value contributed to the estimated utility co-occurrence values by i has no effect after the subproblem of the item i is processed. This idea motivate us to provide a Promising Utility Co-Occurrence Structure (PUCS) for each promising item to further reduce the number of candidates itemsets. We observe that this novel structure, FHM, can be made more faster (see Section 6 for details) by further reducing computational cost of join operation. We call this algorithm as FHIM (Fast High-utility Itemset Miner). We now define PUCS for a given item as follows.

Definition 18 (*Promising Utility Co-Occurrence Structure (PUCS)*). The Promising Utility Co-Occurrence Structure (PUCS) of a given item a, denoted by $PUCS_a$, is a set of tuples of the form $(b,c,v), a \prec b, a \prec c$, $\mathcal{D}_{ab} \neq \emptyset$ and $\mathcal{D}_{ac} \neq \emptyset$ such that $v = \sum_{e \in UL(a) \wedge e.tid \in \mathcal{D}_{abc}} (e.iutil + e.rutil)$.

In Definition 18, the value v in a tuple (b,c,v) is $v = TWU(bc)$ in the projected database of the item a. Table 8 shows the PUCS of items A and B. The PUCS takes less memory as it is only for a subproblem and then it is removed after the subproblem is solved. The proposed model generates high utility itemset faster than FHM. It seems that the PUCS for each item is constructed before beginning of the depth-first search, but it is not the case. In our implementation of the proposed algorithm, the PUCS for each item is not computed at the beginning at each time rather it is constructed and updated from 2-itemsets. For a given itemset $X = \{x,y,z\}$, the entry (y,z) of PUCS of x is calculated at the time of construction of the *utility-list* of the itemset X from xy and xz. However, a variable is used to accumulate the total promising utility z at the time of computing the *utility-list* of the itemset X. To further optimize the method, two construction procedures have implemented: one for construction of 3-itemset and another for the rest. This can significantly reduce time to compute the estimated utility for pruning. Further, in order to optimize the memory used, once a node is processed to generate its all extensions, it is then safe to remove it from memory, which is performed in Step 17 of Algorithm 1.

Our FHIM earns the novelty with the proposed pruning scheme, called PUCP (Promising Utility Co-occurrence Pruning), which is based on PUCS. This pruning scheme immediately eliminates all extensions of an itemset X without further constructing their *utility-list*. The proposed scheme is described as follows.

Theorem 5 (*PUCP Strategy*). *Let $P = \{x_1, x_2, x_3, \ldots, x_k\}$ be a k-itemset, where $2 \leqslant k < m$, and $x_1 \prec x_2 \prec x_3 \prec \ldots \prec x_k$. Let Px and Py be the itemsets formed by the combination of itemset P with items x and y, respectively, where $x \prec y$. If there is a tuple (x,y,v) in PUCS of x_1 such that $v < min\_util$, Pxy and all its extensions are pruned.*

Table 8
Promising Utility Co-Occurrence Structure of item A, B.

$PUCS_A$		
Item	C	F
F	25	
E	40	25
$PUCS_B$		
Item	D	F
F	24	
E	24	24

Proof. Let Y be an extension of Pxy. Since $|Pxy| > 3, |Y| > 3$. Any extension of Pxy is an extension of x_1. Then, $\{x_1\} \subset x_1xy \subset Pxy \subseteq Y$. Let $\mathcal{D}_{x_1}$ be the projected database of x_1. Let Z be a set of items after excluding x_1 in Y. That means $Z = Y/x_1$. Clearly, $Pxy/x_1 \subseteq Z \subseteq t/x_1$. If $Tidset(x_1)$ denotes the set of *tid*'s of $UL(x_1)$, we denote $Tidset(Pxy)$ for the set of *tid*'s of $UL(Pxy)$, and $Tidset(Y)$ for the set of *tid*'s of $UL(Y)$. Then, $Tidset(Y) \subseteq Tidset(Pxy) \subseteq Tidset(x_1) = \mathcal{D}_{x_1}$. Thus, the utility value of Y in $\mathcal{D}_{x_1}$ is given as

$$\begin{aligned}
u(Y) &= \sum_{t \in Tidset(Y)} u(Y,t) = \sum_{t \in Tidset(Y)} u(x_1,t) + u(Z,t) \\
&\leqslant \sum_{t \in Tidset(Y)} u(x_1,t) + u(t/x_1,t) \leqslant \sum_{t \in Tidset(Pxy)} u(x_1,t) + u(t/x_1,t) \\
&= \sum_{e \in UL(x_1) \wedge e.tid \in Tidset(Pxy)} e.iutil + e.rutil \\
&\leqslant \sum_{e \in UL(x_1) \wedge e.tid \in Tidset(x_1xy)} e.iutil + e.rutil \\
&= \sum_{e \in UL(x_1) \wedge e.tid \in \mathcal{D}_{x_1}} e.iutil + e.rutil = v, \text{ since } (x,y,v) \in PUCS_{x_1} \\
&< min_util.
\end{aligned}$$

Hence, the itemset Y is not a high utility itemset in $\mathcal{D}_{x_1}$. Since all the transactions containing Y are only in the projected database $\mathcal{D}_{x_1}$, so $u(Y, \mathcal{D}_{x_1}) = u(Y, \mathcal{D})$. Hence, the theorem follows. □

Note that the pruning PUCP strategy is applicable for the generation of the larger itemsets of length at least four. For the generation of itemsets having length 3 or less, the EUCP strategy is used (provided in Definition 17). Before constructing the *utility-list* of Pxy, except the pair (x,y), the rest pairs of items in Pxy have already been tested in previous recursions of the depth-first procedure. A genuine point about the PUCP strategy is to test the itemset Pxy the promising utility value of the pair (x,y) should be available in the corresponding PUCS structure. For example, before constructing the utility-list of the itemset $\{a,b,c,d\}$ from the itemset $\{a,b,c\}$ and $\{a,b,d\}$, the promising utility value of the pair (c,d) in $PUCS_a$ should be tested with *min_util*. For this reason, the promising utility value of the pair (c,d) in $PUCS_a$ needs to be computed before processing the itemset $\{a,b,c,d\}$. That means the utility-list of the itemset $\{a,c,d\}$ is computed first. In other words, all subsets of $\{a,b,c,d\}$ with prefix a will be generated before $\{a,b,c,d\}$ itself. This can be achieved by the reverse pre-order traversal on a sub-tree rooted at the root node discussed in Section 4.2.4.

4.2.3. Progressive expected utility pruning

It is observed that in the FHM and HUI-Miner procedures, an itemset Px is combined with another itemset Py, which lies right to it. The pruning strategy proposed by Liu and Qu (2012) is a straightforward strategy, which checks if the estimated utility of Px is greater than the threshold or not. In the EUCS strategy (Fournier-Viger et al., 2014b), before joining the utility-list of Px and Py, it checks if xy is a promising itemset or not. If it is so, it continues; otherwise, it stops searching in that sub-tree. By the above proposed strategy, the estimated utility of 2-itemsets is further decreased. But as the TWU obeys anti-monotone property the $TWU(Py)$ is less than or equal to xy. We thus have $TWU(Pxy) \leqslant TWU(Py) \leqslant TWU(xy)$. This inequality also holds for the projected database. For a given itemset Px, if $TWU(Py)$ is no more than the utility threshold, Py will not be combined with Px to form the itemset Pxy. However, in the depth-first search using the utility-list it is expensive to find $TWU(Py)$ in the corresponding projected database since it requires to sum up all *iutil* and *rutil* of the elements for the common transactions. As a result, we provide an estimated utility of Py, which says whether it can be combined

with the left itemsets or not. This, in turn, increases the performance, which is evident from Section 6.

Definition 19. For a given itemset Px, and an item Pz with $x \prec z$, the progressive estimated utility of the itemset Pxz is $PEUP(Pxz) = \sum_{tid \in Tidset(Pxz)}(u(Px, tid) + ru(Px, tid))$.

Theorem 6 (*PEUP Strategy*). *Let P be an itemset and Py an 1-extension itemset. If $EU(Py) < min\_util$, there is no extension Y of P such that $Py \subset Y$, which is a high utility itemset.*

Proof. Let Y be an extension of P and $Py \subset Y$. Let $Tidset\ (P)$ denote the set of *tids*'s of $UL(P)$, $Tidset(Py)$ the set of *tids*'s of $UL(Py)$, and $Tidset(Y)$ the set of *tids*'s of $UL(Y)$. Then, $Tidset(Y) \subseteq Tidset(Py)$. For all transactions t such that $Y \subseteq t$, we have,

$$\begin{aligned} u(Y,t) &= u(Py,t) + u((Y - Py),t) = u(P,t) + u(y,t) + u((Y - Py),t) \\ &= u(P,t) + u((Y - P),t) = u(P,t) + u((Y/P),t) \\ &= u(P,t) + \sum_{i \in Y/P} u(i,t) \leqslant u(P,t) + \sum_{i \in t/P} u(i,t) \\ &= u(P,t) + ru(P,t). \end{aligned}$$

Now,

$$\begin{aligned} u(Y) &= \sum_{t \in Tidset(Y)} u(Y,t) \leqslant \sum_{t \in Tidset(Y)} u(P,t) + ru(P,t) \\ &\leqslant \sum_{t \in Tidset(Py)} u(P,t) + ru(P,t) = EU(Py,t) < min\_util. \end{aligned}$$

Hence, the theorem follows. □

The PEUP pruning strategy simply states that the join operation between Px and Py will not be performed, if the estimated utility of Py with Px is less than the minimum utility. Before performing the join operation, the itemset Pxy and the sub-tree rooted at node Pxy are removed from the search space. Observe that the estimated utility of an itemset Pxy is calculated at the construction procedure using Definition 19. A variable is also used to accumulate the estimated utility of each utility list. Integrating both the strategies, the proposed method yields a significant improvement in performance. From the experiments, it is evident that the proposed method takes less memory or comparable memory for different thresholds. The reason is that we reduce the search space as the corresponding utility-list is not generated and as a result, it consumes less memory.

4.2.4. Reverse pre-order traversal

The HUI-Miner is the first HUI mining method that combines the vertical representation of a database with the depth-first search based on traversal of the Eclat algorithm of frequent itemset mining. It traverses the search space from left-to-right in the set enumeration tree. FHM adopts the searching procedure of HUI-Miner to the exclusive generation of the HUIs. To reduce the cost of join operation, it relies on a technique, called EUPS strategy, that checks whether any superset of the itemset is a HUI or not by considering all its subsets of TWU values as they obey the anti-monotone property. To apply our pruning strategies, it must ensure that while before an itemset is generated, all the expected co-occurrence utility of each two-pair items need to be computed. This suggests that all the subsets of length 3 be generated before the generation of the itemset. The technique presented in Talky-G (Szathmary et al., 2009), called the reverse pre-order traversal, ensures that all the subsets of a given itemset are in right to the node in the set enumeration tree. This traversal also ensures that all the subsets of an itemset X are generated before the generation of X. We adopt this traversal for the sub-tree rooted at the child of the root node. The complete traversal is shown in Fig. 2.

4.2.5. The FHIM algorithm

In this section, we describe an efficient algorithm, called FHIM, for mining high utility itemsets. The utility database $\mathcal{D}$ and the minimum utility threshold *min_util* are taken as inputs in the main algorithm, FHIM, provided Algorithm 1.

Algorithm 1 *FHIM*($\mathcal{D}$, *min_util*)

Input: $\mathcal{D}$: database, *min_util*: minimum utility threshold.
Output: High utility itemsets.
1 Scan $\mathcal{D}$ to get *TWU* of each item.;
2 Let C be the set of items x such that $TWU(x) \geqslant min\_util$.;
3 Sort the items in C in increasing order of *TWU* value.;
4 Remove the items from the database whose $TWU < min\_util$.;
5 Construct the utility-list *UL* as $UL = \{UL(x) \mid TWU(x) \geqslant min\_util\}$ and build the EUCS structure.;
6 **for each** *item* $x \in$ C
7 **if** $(SUM(UL(x).iutils) \geqslant min\_util)$
8 Write $\{x\}$, $\{x\}$.utility, $\{x\}$.support; `// High utility 1-itemset`
9 **if** $(SUM(UL(x).iutils) + SUM(UL(x).rutils) \geqslant min\_util)$
10 Set $tail = \emptyset$.;
11 **for each** item $y \in$ C, $x \prec y$ from left to right
12 **if** $\exists(x, y, v) \in EUCS$ such that $v \geqslant min\_util$
13 Compute $X = \{x\} \cup \{y\}$;
14 Compute $UL(X) = Construct(\emptyset, \emptyset, UL(x), UL(y))$;
15 Compute $tail = tail \cup UL(X)$;
16 Call $Rsearch(x, tail, min\_util)$;
17 Remove $UL(X)$ from UL;

Procedure *Construct*($UL(P), UL(Px), UL(Py)$)

Input: a: item a for the subproblem, $UL(P)$: Utility-list of parent P, $UL(Px)$: Utility-list of Px, $UL(Py)$: Utility-list of Py.
Output: $UL(Pxy)$: Utility-list of the itemset Pxy.
1 Set $UL(Pxy) = \emptyset$;
2 Set $Pxy.EU = 0$; `// Expected utility of Pxy for PUEP strategy`
3 Set $eu = 0$; // `Expected co-occurrence utility of Pxy for PUCP strategy if` $|Pxy| = 3$
4 **for each** element $e_x \in UL(Px)$
5 **if** $\exists e_y \in UL(Py)$ and $e_x.tid == e_y.tid$
6 **if** $UL(P) \neq \emptyset$
7 Find $e \in UL(P)$ such that $e.tid == e_x.tid$;
8 Compute $exy = (e_x.tid, e_x.iutil + e_y.iutil - e.iutil, e_y.rutil)$;
9 Compute $Pxy.EU = Pxy.EU + e_x.iutil + e_x.rutil$;
10 **else**
11 Compute $exy = (e_x.tid, e_x.iutil + e_y.iutil, e_y.rutil)$;
12 Compute $Pxy.EU = Pxy.EU + e_x.iutil + e_x.rutil$;
13 Compute $eu = eu + e.iutil + e.rutil$; // `For PUCP strategy if` $|Pxy| = 3$
14 Add exy to $UL(Pxy)$;
15 **if** $|Pxy| == 3$
16 Add eu to $PUCS_a$;
17 **return** $UL(Pxy)$;
18 End Procedure

At first, the *TWU* of all items are calculated by a database scan in Step 1. If the *TWU* of any item is less than the specified *min_util*, the item is discarded according to Property 2 for subsequent mining process. The items, whose *TWU* are higher than *min_util* (known as *promising* items), are sorted according to the order relation Ω and stored in a global *TWU_Table* (Liu et al., 2005). In the next pass, the transactions are sorted according to Ω, and at the same time the initial *utility-list* for each promising item and the EUCS structure are constructed. These procedures are performed during Steps 2–5. In Steps 7–8, it finds the high utility 1-itemsets. All promising 2-itemsets and their utility-list are generated using Property 8 and the EUCP strategy in Step 9–14. Step 13 calls the *Construct* procedure to construct the utility-list of 2-itemsets. One can find the details of this procedure in Liu and Qu (2012). In this construction procedure, the progressive expected utility of the utility-list is calculated in Steps 9–12. Also, in this procedure, the PUCS structure of an item is updated. The complete modified *Construct* procedure of HUI-Miner is given in the procedure *Construct*. After this, the reverse depth-first search procedure, Rsearch, is explored at Step 15 to generate all the itemsets of the subproblem of an item *x*.

Procedure *Rsearch*$(a, tail, min\_util)$

Input: a: item for the subproblem, *tail*: Set of $utility-list, min\_util$: minimum utility threshold.
Output: High utility itemsets.

```
1  If (tail == ∅) return;
2  for each Px ∈ tail from right to left
3      if (SUM(UL(Px).iutils) ⩾ min_util)
4          Write Px, Px.utility, Px.support
5      if SUM(UL(Px).iutils) + SUM(UL(Px).rutils) ⩾ min_util)
6          Set tail_Pxy = ∅ ;
7          for each Py ∈ tail with x ≺ y from left to right
8              if (JOIN(a, Px, Py, min_util)==true)
9                  Compute Pxy = Px ∪ Py;
10                 Compute UL(Pxy) = Construct(a, P, Px, Py);
11                 Compute tail_Pxy = tail_Pxy ∪ UL(Z);
12         Call Rsearch(a, tail_Pxy, min_util);
13 End Procedure
```

In the *Rsearch* procedure, the reverse traversal searching is applied in Step 2. This generates the high utility itemset of length more than 1 in Steps 3–4. Steps 7–13 are same as the FHM except Step 8. Step 8 contains the EUCP, PUCP and PUEP strategies in another procedure, called *JOIN*. This procedure verifies whether the joining between two itemsets *Px* and *Py* is performed or not. If the progressive expected utility of *Py* is below the minimum utility threshold, *JOIN* procedure returns false, otherwise it checks for the promising utility value in EUCS or PUCS matrix depending on the size of *Px*. If this sub procedure returns true, the itemsets *Px* and *Py* are combined to form *Pxy* in Step 9 of the Rsearch procedure. The utility-list of this itemset is constructed in Step 10 by calling the *Construct* procedure. The procedure calls itself recursively until $tail \neq \emptyset$. Otherwise, it returns to the main procedure to perform the exploration on the next subproblem.

Procedure *JOIN*(a, Px, Py, min_util)

```
if EU(Py) ⩾ min_util
    if | Px |== 2
        if ∃(x, y, v) ∈ EUCS such that v < min_util
            return false;
    else
        if ∃(x, y, v) ∈ PUCS_a such that v < min_util
            return false;
    return true;
else return false;
End procedure
```

4.3. Deriving high utility closed itemsets and generators

The HUCI-Miner algorithm given in Algorithm 2 identifies the high utility closed itemsets and generators among the high utility itemsets. It enables to the efficient generation of the non-redundant association rules among the high utility itemsets using the utility-confidence framework. The algorithm outputs the resultant set, *CH*, which contains the high utility closed itemsets H_k, where each set $H_k, 1 \leqslant k \leqslant max$, has all high utility *k*-itemsets and *max* is the size of the longest high utility itemset. This algorithm generates the high utility closed itemsets and generators from the high utility itemsets using Theorems 2, 3 and Corollary 3. Note that no additional database scan is required in order to find out the *utility unit array* of each closed itemset, which is used to calculate the local utility value of any subset. By scarifying a little more memory consumption, this can be calculated from the *utlity-list* of the constituent items of an itemset.

The pseudo-code of the HUCI-Miner algorithm, provided in Algorithm 2, is a level-wise procedure. It identifies all the high utility itemsets successively as the high utility closed itemset or generator in each set $H_k, 1 \leqslant k \leqslant max$, contains the high utility itemset of length *k*. It derives the sets $CH_k, 1 \leqslant k \leqslant max$, containing the closed itemsets and their supports, utility values and the corresponding generators. At first, it finds all the high utility itemsets using the FHIM algorithm. After this exploration, the algorithm examines each high utility *k*-itemset, $k \geqslant 2$, which is a generator of a high utility *k*-itemset $(k \geqslant 2)$ by considering the supports of all its subsets of length $k-1$. The algorithm then verifies if it is a closed itemset by examining the supports of all its subsets of length $k-1$. Two boolean variables *closed* and *key* are used in order to identify whether an itemset is a high utility closed itemset or a generator. If H_k is empty and H_{k-1} is nonempty, by consequence of Property 5 the elements of H_{k-1} are closed and it is performed in Steps 15–16. Conversely, if H_k is nonempty and H_{k-1} is empty, all itemsets in H_k are generators, and no extra step is needed as all itemsets are initially marked as generators.

Table 9
HGB basis extracted from Table 1 with *min_util* = 20 and *min_uconf* = 80%.

Rule	$(utility, uconf(\%))$
B → F	(23, 100)
D → E	(37, 100)
F → E	(36, 90)
E → F	(36, 81)
A → CE	(38, 100)
F → DE	(36, 80)
B → DFE	(24, 80)
AFE → C	(25, 100)

Table 10
Characteristics of datasets.

Dataset	| T |	| I |	AvgL	MaxL	Type
Foodmart	4141	1559	4.4	14	Sparse
Chess	3196	75	37	37	Dense
Mushroom	8124	119	23	23	Dense
Connect	67,557	129	43	43	Dense
T10I4D100K	100,000	870	10.1	29	Sparse
Retail	88162	16,470	10.3	76	Sparse
Chain-store	1,112,949	46,086	7.2	170	Sparse

Algorithm 2. *HUCI-Miner*($\mathcal{D}$, *min_util*)

Input: $HUI : \{H_1, H_2, \ldots, H_{max}\}$, where H_k is the set of huis of length k, max is the size of largest high utility itemset.
Output: High utility closed itemsets with their generators.

```
1 Set CH ← ∅, HG ← ∅ // HG: Set of high utility generators
2 Compute H = FHIM(D, min_util) // H = {H1, H2, ... Hmax}, Hk Set of high utility k-itemset
3 for each itemset h ∈ H1 do
4     Set h.closed ← true; h.key ← true;
5 end
6 for (k = 2; k ⩽ max; k++) do
7     if Hk ≠ ∅
8         for each itemset h ∈ Hk do
9             Set h.key ← true; h.closed ← true;
10            for all subsets h' ∈ HK−1 of h do
11                if (supp(h') == supp(h))
12                    Set h.key ← false; h'.closed ← false;
13            end
14        end
15        Set CHk−1 ← {h ∈ Hk−1 | h.closed = true};
16        Get_generators(CHk−1, Hk);
17    else
18        Set CHk−1 ← {h ∈ Hk−1 | h.closed = true};
19        Get_generators(CHk−1);
20 end
21 Set CHk ← Hk;
22 Get_generators(CHk);
23 Set CH ← ∪k CHk with generators;
24 Calculate utility unit array of each closed itemset;
25 End procedure
```

An itemset c is identified as a generator during Steps 8–12 in the algorithm *HUCI-Miner*. If the support of c is same as one of its subsets having length $k-1$ in H_{k-1}, then c is not a generator, and conversely, it is not closed. In Steps 15–18, all the closed itemsets of length $k-1$ are added to the set CH_{k-1}. Step 21 discovers the closed itemset of the maximum length. In Steps 16, 19 and 22, the *Get_generators* procedure is called in order to update the global list of generators and assign the generators to the respective closed itemset. It takes the set CH_k as input. For each closed itemset $ch \in CH_k$, its proper subsets in the global set of generators HG are then removed and then added to the generators of ch (Steps 1–5 in *Get_generators* procedure). This procedure updates the global set of generators HG by the itemsets, which are not closed but they are generators before the starting of the next iteration. If the set of generators of a given closed itemset is empty, it indicates that the closed itemset is the generator of itself. For example, considering the database given in Table 1 with $min\_util = 20\%$. The HUCIs are $G(), F(), E(), BF(B), DE(D), FE(), ACE(A), DFE(DF), BDFE(BE)$ and $ACFE(AFE)$, where $X(Y)$ means X is the closed itemset and Y is its generator.

Procedure *Get_generators*(CH_{k-1}, H_k)

Input: CH_{k-1}: high utility closed itemset of length $(k-1)$
Output: Assign the generators to each closed itemsets of CH_{k-1}

```
1 for each itemset ch ∈ CHk do
2     for all subsets c' ∈ HG of ch do
3         Add c' in ch.generator;
4     end
5 end
6 Compute
  HG = HG ∪ {h ∈ Hk | h.key = true ∧ h.close = false};
```

Theorem 7. *For a given minimum utility threshold, the proposed HUCI-Miner algorithm generates all high utility closed itemsets with their generators correctly.*

Proof. The correctness of our proposed HUCI-Miner algorithm is based on Theorems 2 and 3. Theorem 2 determines a high utility itemset h_{k-1}, which is a high utility closed itemset, by comparing its support with the supports of the high utility k-itemset h_k containing the itemset h_{k-1}. Theorem 3 enables if a high utility k-itemset h_k is a generator by examining its support with the supports of the high utility $(k-1)$-itemsets, which are included in h_k. Since a high utility closed itemset cannot be a generator for the large itemsets, they are not in the global list of generators. Again, the closure of a non-generator itemset c is the smallest superset of c in the set of the high utility closed itemsets. This proves that the identification of the generators of a closed itemset is correct. □

4.4. Complexity analysis

The time complexity of the utility-list construct procedure is in the order of $O(s \log s)$, where s is the maximum support among all itemsets and the binary search algorithm is used to get an element from a utility-list. The complexity of FHIM and FHM are proportional to the cost of total number of candidate high utility itemsets generated during the mining process. Thus, the time complexity of FHIM and FHM are $O(Can_{FHIM})$ and $O(Can_{FHM})$, respectively, where Can_{FHIM} and Can_{FHM} are the number of candidate high utility itemsets generated using the respective algorithms. FHIM is based on the pruning strategies, PUCP and PUEP, is improved versions of FHM algorithm. The proposed pruning strategies reduce the generation of the number of candidate high utility itemsets than that of FHM (See Table 13). In general, it can be shown that $Can_{FHIM} \leqslant Can_{FHM}$, which makes FHIM is more efficient than FHM. If N_{max} is the maximum number of high utility itemsets presents in some H_K and max is the maximum length of high utility itemset obtained, the time complexity of HUCI-Miner algorithm is $\mathcal{O}(N_{max}^2 \times max)$ and it is straight forward as it is based on the subset checking in a level-wise manner and due to the dominating three for loops in Steps 6, 8 and 10.

5. Utility based non-redundant association rule

From the experiments in Section 6.5, it is observed that a large number of rules with high utility and high utility confidence are generated and it grows rapidly. In this subsection, we use high utility closed itemsets to eliminate redundant association rule. A small number of rules can be generated and provided to the user for the decision making, from which all the valid rules can be derived by using the high utility closed itemsets and the utility unit array. Before proposing our non-redundant rule generation algorithm, we start with an example to show the existence of redundancy followed by a definition provided in Definition 20.

Example 6. Consider an association rule $R : X \rightarrow Y$ with utility u and utility-confidence c on an utility-confidence framework. The semantic interpretation of such a rule is that the utility value of the itemset $X \cup Y$ is u and the utility-confidence c reflects the contribution of utility of itemset X to the utility of $X \cup Y$ from its total utility value. This means the fraction of utility of X is covered by $X \cup Y$. The manager can decide the importance of the rule, if u and c satisfy the specified thresholds. Consider the database given in Table 1. Let the manager want to promote the items for increasing the sells. The question is that "what is the least set of items from which the manager can take his decision to promote

the sells?". Consider the valid rules $R_1 : B \rightarrow DFE(24, 80\%)$, $R_2 : BE \rightarrow DF(24, 100\%), R_3 : BDE \rightarrow F(24, 100\%)$, and $R_4 : BFE \rightarrow D(24, 100\%)$. The rules R_2, R_3, and R_4 are not useful for taking decision for promoting the itemset *BDFE* from the least set of items, since the decision for the itemset *BDFE* has already taken from the rule R_1, which has minimal antecedent. We say the rules R_2, R_3, and R_4 are redundant to the rule R_1, and R_1 is a non-redundant rule.

Definition 20. Let $R_1 : X_1 \rightarrow Y_1$ and $R_2 : X_2 \rightarrow Y_2$ be two valid association rules on utility-confidence framework. We say that the rule R_2 is redundant to the rule R_1, if $X_2 \cup Y_2 \subseteq X_1 \cup Y_1, R_1.utility \geqslant R_2.utility, supp(R_1) = supp(R_2)$ and $X_1 \subseteq X_2, Y_2 \subseteq Y_1$, where $R_i.utility$ is the utility of the rule R_i for $i = 1, 2$.

Definition 20 simply states that an association rule R is a valid non-redundant rule in utility-confidence framework, if there does not exist any other valid rule R' with same support, having more utility, and an antecedent that is a subset of the antecedent of R, and a consequent which is a super set of the consequent of R. In general, a non-redundant association rule is a rule with minimal antecedent and maximal consequent. We call a non-redundant rule in utility-confidence framework as a high utility generic rule (hgr) and denote the set of such non-redundant rules as *HGR*. Based on this definition, for the generation of all hgrs, we have proposed the high utility generic basis (*HGB*) in Section 5.1.

5.1. High utility generic basis (HGB)

It can be noticed that all the rules $R_i : X_i \rightarrow Y_i$ having the same support and $\{X_i \cup Y_i\} \cap \{X_j \cup Y_j\} \neq \emptyset, i \neq j$, have the same closure $\gamma(X_i \cup Y_i)$. So, if g be a generator of $\gamma(X_i \cup Y_i)$, all the itemsets in the interval $[g, \gamma(X_i \cup Y_i)]$ have the same support. Thus, if the rule $R : g \rightarrow \gamma(g) \setminus g$ is a valid non-redundant rule, all the rules $R_i : l \rightarrow \gamma(g) \setminus l, g \subset l$, are redundant to the rule R, which has minimal antecedent and maximal consequent. If it is not a valid non-redundant rule and no subset of g also generates a valid rule, by concatenating the items of $\gamma(g) \setminus g$ valid non-redundant rules can be formed. This explanation extends to define the following basis given in Definition 21 that contains all the elements of *HGR* defined in Definition 20.

Definition 21. Let *HUCI* be the set of high utility closed itemsets. Then the high utility generic basis is defined by $HGB = \{R : g \rightarrow h \setminus g \mid h \in HUCI \wedge g \neq \emptyset,\ g \subset h,\ uconf(R) \geqslant min\_uconf \wedge \nexists\, g' : g' \subset g \wedge uconf(g' \rightarrow h \setminus g') \geqslant min\_uconf\}$.

Consider the database $\mathcal{D}$ in the running example given in Table 1. From Fig. 1, it is clear that *DFE* is a high utility closed itemset with utility 36. Consider all valid rules from this itemset. Let $min\_uconf = 80\%$. It is easy to verify that the high utility subsets of *DFE* are $D(22), F(20), E(22), DE(37), DF(24), FE(36)$, where utilities are specified within the parenthesis. The itemsets for which $uconf(X \rightarrow DFE) \geqslant min\_uconf, X \subseteq DFE$ are $F(20)$ and $DF(24)$. Hence, $F \rightarrow DE$ is a HGB rule with utility 36 and utility-confidence 0.8.

In Definition 21, the condition $g \subset h$, with $g \neq \emptyset$, states that there is no rule, which is either with empty antecedent or empty consequent. For the formation of basis from high utility closed itemsets, *HUCI* ensures that the eliminated redundant rule has the same support. The non existence of subsets of an antecedent of a valid rule in the basis *HGB* indicates that it is the rule with smallest antecedent and since it is derived from the closed itemset, it has maximum consequent. Theorem 8 concludes that the construction of *HGB* leads to extraction of all non-redundant rules on utility-confidence framework.

Theorem 8. *Let h and h' be two high utility closed itemsets such that $h' \subset h$. Let HG'_h be the set containing the generators of h' and $g \in HG'_h$ such that $uconf(R : g \rightarrow h \setminus g) \geqslant min\_uconf$. If there does not exist any subset g' of g such that $uconf(R' : g' \rightarrow h \setminus g') \geqslant min\_uconf$, then $R : g \rightarrow h \setminus g \in HGB$.*

Proof. It directly follows from Definition 21. $R : g \rightarrow h \setminus g$ is a valid rule and there does not exist any subset g' of g such that $uconf(g' \rightarrow h \setminus g') \geqslant min\_uconf$. Hence, R is generated from h containing the smallest antecedent and $R \in HGB$. □

We now provide an approach that constructs *HGB* basis by examining all high utility closed patterns level-by-level in Algorithm 3. The *HGB* construction algorithm takes as input the high utility closed itemsets with their generators, the set *HUCI* and the user-provided threshold values *min_uconf*, and *min_util*. The main idea of this proposed algorithm is based on Theorem 8. A basis is added into the *HGB* set only if there does not exist any valid rule with smaller antecedent and of same support. According to Theorem 6, if for a high utility closed itemset h we have a valid rule with some generator of a subset itemset $h' \subset h$, that rule is a *HGB* rule. If such a valid rule does not exist, we can join the items level-by-level to obtain an antecedent with smaller length. While adding items to a generator, the algorithm checks whether the resulting antecedent is a high utility itemset or not. A set L_{ma} is used to keep the minimal antecedent for a valid rule, discovered so far, which is an empty set at the beginning of each iteration. The algorithm starts searching for the minimal antecedent, and as soon as it is found, it is added to L_{ma} if there does not exist a subset in L_{ma}. The supersets of the currently added set are removed from L_{ma}. Once all the closed subsets of a closed itemset are processed, the *HGB* rules are generated by taking the elements of L_{ma} as antecedent. The algorithm continues to generate *HGB* rules until no more high utility closed itemsets are processed. Finally, all elements of *HGB* basis are generated.

Algorithm 3. *HGB* construction

Input: *HUCI*: High utility closed itemset with utility unit array, *min_uconf*: minimum utility-confidence and *min_util*.
Output: *HGB*: High utility Generic Basis.

```
1  for each itemset h ∈ HUCI do
2      Set L_ma = {}; // ma: minimal antecedent
3      for each h' ⊆ h in increasing order of size do
4          Set L_temp = {};
5          for each g ∈ HG_h' and g ≠ h do
6              if ( lu v(g,h) / u(g) ⩾ min_uconf and ∄ g_s ∈ L_ma | g_s ⊂ g )
7                  Compute L_ma = L_ma ∪ g;
8              else
9                  Compute L_temp = L_temp ∪ g;
10         end
11         for each g ∈ L_temp do
12             Compute A_1 = {i_1, i_2, ..., i_k}, where each
               i_j ∈ h' \ g;
13             for (j = 1; A_j ≠ ∅; and (i ⩽ k); i++) do
14                 for all l ∈ A_j do
15                     if ( lu v({gl},h) / u({gl}) ⩾ min_uconf and
                       ∄ g_s ∈ L_ma | g_s ⊂ {gl}) do
16                         Remove all l' ⊃ {gl} from L_ma;
17                         Compute L_ma = L_ma ∪ {gl} ;
18                 end
19                 Compute A_{j+1} = Apriorigen(A_j, min_util) ;
                   // without pruning techniques
20             end
21         end
```

(*continued*)

Algorithm 3. *HGB* construction

```
    end
    for each g_s ∈ L_ma do
        Compute R = g_s → h \ g_s;
        Compute R.utility = h.utility;
        Compute R.uconf = lu v(g_s,h) / u(g_s);
        Compute HGB = HGB ∪ R;
    end
end
```

Each high utility closed itemset consists of high utility closed itemset itself, its generators, support, utility unit array and utility. The algorithm returns *HGB* containing all non-redundant rules with utility and utility-confidence. At first, *HGB* is initialized with the empty set and then closed itemsets are passed in increasing order of their sizes. For an itemset, its closed subsets are processed in increasing order of their size in Step 3. The algorithm processes the generators of the closed subsets h' for generating a valid *HGB* rule in Step 5. If a generator g satisfies the rule generation criteria in Step 6, it is added to the set L_{ma} in Step 7. Otherwise, it is added to a temporary set L_{temp} for further expansion in Step 9. After processing all the generators of a subset h', if the set L_{temp} is nonempty, the algorithm finds other elements of the equivalence class $[h']$ level-by-level by simply adopting the *Apriorigen* procedure (Agrawal & Srikant, 1994) without pruning strategy with items of $h' \setminus g$ using Steps 12–23. The *Apriorigen* procedure is used for generating the k-itemset from $k-1$-itemset. The expansion of g is done by combining with the elements of the itemsets returned by *Apriorigen* procedure. The resulting itemsets are tested for *HGB* conditions, and if the conditions are satisfied, they are added to the set L_{ma} during Steps 15–20. After processing all the subsets, the algorithm processes each element of L_{ma}, which contains the minimal antecedent to generate valid rules from a closed itemset, and the rule is generated and added to the global set *HGB* using Steps 25–30. Finally, after processing all the high utility closed itemsets, the algorithm returns *HGB* basis.

Example 7. Consider again the database $\mathcal{D}$ given in Table 1. Further, consider $min\_util = 20$ and $min\_uconf = 80\%$. The extracted *HGB* basis is shown in Table 9 using Algorithm 3.

Theorem 9. *All rules in the HGB basis are high utility generic rules. In other words, if $R \in HGB$, then $R \in HGR$ and vice versa.*

Proof. Let $R : g \rightarrow h \setminus g \in HGB$. By Definition 21, h is a closed itemset. We need to justify whether $R : g \rightarrow h \setminus g \in HGR$ or not. On contrary, assume that $R : g \rightarrow h \setminus g \notin HGR$. Then there exists a valid association rule $R' : g' \rightarrow h' \setminus g' \in HGR$ such that $supp(R) = supp(R'), g' \subseteq g$, and $h \subseteq h'$. If $h = h'$ and $g = g'$, then $R = R'$. If $h \subset h'$, we have $\gamma(h) = h \subset h' \subseteq \gamma(h')$. Then, $f(\gamma(h)) \supset f(\gamma(h')) \Rightarrow supp(\gamma(h)) > supp(\gamma(h')) \Rightarrow supp(h) > supp(h') \Rightarrow supp(R) > supp(R')$. Again, if $g' \subset g$, there is a valid rule $R' : g' \rightarrow h' \setminus g'$ which has the smaller antecedent than the antecedent g of the rule R. Thus, by Definition 21, $R \notin HGB$. This leads to a contradiction, and hence, $R \in HGR$. The converse part follows straightforwardly from Definition 21 itself. □

Theorem 10. *Our proposed algorithm given in Algorithm 3 is complete and sound.*

Proof. From Theorem 9, any high utility non-redundant association rule is in *HGB* and vice versa. As a result, the proposed algorithm is complete as well as sound. □

5.2. Deriving all high utility association rules from HGB basis

Note that each element in the proposed high utility generic basis (*HGB*) given in Algorithm 3 is a valid association rule. Since the high utility closed itemsets with *utility unit array* are loss-less (Wu et al., 2011), we can derive all high utility itemsets with their actual utility values without accessing the datasets. As a result, all the valid rules can be derived from *HGB* basis using *HUCI* by the antecedent expansion of a basis element and consequent diminution by items using Theorem 11.

Algorithm 4. All high utility association rules generation from *HGB*

```
Input: HGB, min_util, min_uconf.
Output: HAR: set of all valid high utility association rules.
1  Set HAR = ∅;
2  for all rule R_1 : X → Y ∈ HGB do
3      Compute
   HAR = HAR ∪ {R_1 : X → Y, R_1.utility, R_1.uconf};
4      for all Z such that Z ⊂ Y, Z ≠ ∅ do
5          find_utility(X ∪ Z); // Calculates the utility of
   X ∪ Z from the utility unit array of γ(X ∪ Z).
6          if ({X ∪ Z}.utility ⩾ min_util)
7              if ( lu v({X∪Z},{X∪Y}) / u({X∪Z}) ⩾ min_uconf )
8                  Compute
   HAR = HAR ∪ {R_2 : X ∪ Z → Y \ Z, R_1.utility, R_2.uconf};
9      end
10 end
11 for all rules R_1 : X → Y ∈ HAR do
12     for all Z such that Z ⊂ Y, Z ≠ ∅ do
13         find_utility(X ∪ Z);
14         if ({X ∪ Z}.utility ⩾ min_util)
15             if R_2 : X → Z ∉ HAR
16                 Compute
   HAR = HAR ∪ {R_2 : X → Z, R_2.utility, R_2.uconf};
17     end
18 end
```

Algorithm 4 provides the required procedure to generate all valid association rules from *HGB* basis, which takes the inputs as *HGB*, *min_util*, and *min_uconf*. The utility value of any itemset is calculated with the help of the *utility unit array* of each closed itemset. We are also able to reconstitute all high utility closed itemsets by joining antecedent and the consequent parts of a *HGB* rule. Since the support of an itemset is evaluated from the smallest closed itemset containing it, the utility and utility-confidence of all the rules can be derived exactly.

Theorem 11. *Let HUCI be the set of all high utility closed itemsets and HGB the high utility generic basis. If $R : X \rightarrow Y \in HGB$, for all subsets Z of Y, then $R' : X \rightarrow Z$ is also a valid association rule, where XUZ is a high utility itemset.*

Proof. Let $R : X \rightarrow Y \in HGB$ and *HUCI* be the set of all high utility closed itemsets. The utility confidence of the rule $R : X \rightarrow Y$ is given by $uconf(R) = \frac{lu v(X,Y)}{u(X)}$. Since $Z \subset Y$, we have $lu v(X,Y) \leqslant lu v(X,Z)$.

Hence, $\frac{lu\nu(X,Z)}{u(X)} \geqslant \frac{lu\nu(X,Y)}{u(X)} \geqslant min\_uconf$. As a result, the rule $R' : X \rightarrow Z$ is valid as XUZ is a high utility itemset. □

Corollary 4. *Let $R : X \rightarrow Y$ be a valid association rule on a utility-confidence framework. Then for any subset $Z \subset Y$, if $X \cup Z$ is a high utility itemset, $R' : X \rightarrow Z$ is also a valid association rule.*

Theorem 12. *The proposed inference algorithm for generating all valid high utility association rules given in Algorithm 4 is complete.*

Proof. Let $R : X \rightarrow Y$ is a *HGB* basis. Steps 2–10 of Algorithm 4 generate all valid association rules by the augmentation on antecedent by the subset of consequent parts of a *HGB* basis. Since all *HGB* basis are also valid rules, they are added to the global set *HAR*, which is used to keep all the high utility association rules in Step 3. After generation of all valid rules by antecedent expansions, the algorithm generates the valid association rules by applying decomposition on the generated association rules using Theorem 11 and Corollary 4 during Steps 11–17. Hence, the proposed inference algorithm for generating all valid high utility association rules is complete. □

6. Performance evaluation

In Section 6.1, we first perform the simulation of our proposed utility-confidence framework on a small dataset. We then demonstrate the computational performance of our proposed FHIM, HUCI-Miner, and HGB construction algorithms using both synthetic (T10I4D100K) and real datasets (Foodmart, Chess, Mushroom, Connect, Retail, Chain-store) in Sections 6.2–6.5. In our experiments, seven datasets are selected in order to indicate the strength of our proposed methods on the different characteristics vary from the less number of transactions to the large number of transactions, from the less number of items to more number of items, and also from fully sparse to fully dense.

The datasets T10I4D100K, Chess, Mushroom, Connect and Retail are obtained from frequent itemset mining dataset repository (FIMI, 2003), Chain-store is obtained from NU-MineBench 2.0 (Pisharath et al., 2005), and Foodmart is from Microsoft foodmart 2000 database. Table 10 shows the characteristics regarding these datasets in terms of the number of transactions ($|T|$), the number of distinct items ($|I|$), the average number of items in a transaction (*AvgL*), the maximum length of transaction *MaxL*, and its type: dense or sparse. Except Chain-store and Foodmart, the other four remaining considered datasets do not provide unit profits of each item (external utility) and item count for each transaction (internal utility). As in Ahmed et al. (2009, 2011), Liu and Qu (2012), Liu et al. (2005) and Tseng et al. (2010), we assign in each transaction of T10I4D100K, Mushroom, Chess, Connect and Retail, the internal utilities randomly between 1 and 10, and the external utilities of each item are randomly generated using a log-normal distribution between the range 0.01 and 10. To show the performance of our proposed algorithms, we compare these with the existing well-known algorithms: FHM (Fournier-Viger et al., 2014b), CHUD (Wu et al., 2011, 2014), UP-Growth (Tseng et al., 2010) and UP-Growth$^+$ Tseng et al. (2013). The FHM implementation is downloaded from the SPMF framework (Fournier-Viger, Gomariz, Soltani, & Gueniche, 2014a). We implement all the experiments using the Java programming language and run with the Windows 7 operating system on a machine with CPU clock rate 3.20 GHz and intel core i5 processor with 3.41 GB of main memory. To enhance the performance of all algorithms, the discovered candidate itemsets and high utility itemsets are stored in the main memory. In all the experiments we have used the minimum utility threshold *min_util*, as the percentage of total transaction utility values of the database.

6.1. Simulation

We have performed the simulation for our proposed utility-confidence framework on a sample dataset, available in Microsoft SQL Server 2012 Data Mining Add-ins for Microsoft Office. This dataset contains 13050 number of transactions and 37 items. The selling cost of each item provided in this dataset is considered as the utility of each item and the internal quantity is assigned randomly between 1 and 10, in order to demonstrate the usefulness of our utility-confidence framework. Since utility value is normally larger than support, a comparison of the two constraints is not signified with the same thresholds. As an alternative, we have considered the same confidence and utility-confidence thresholds and generates the top 100 rules with respect to support and utility, respectively. To extract the top 100 association rules in support confidence framework the TopKRules (Fournier-Viger, Wu, & Tseng, 2012) implementation in SPMF is used. The minimum confidence is set to 0.4, and after all rules are extracted the top 14 association rules with respect to the support are presented in Table 11. Additionally, Table 11 contains the utility value and utility-confidence information of the extracted association rules. We then simulate our proposed algorithm for utility-confidence framework and compare it with traditional association rule mining. The top 14 rules with respect to utility are extracted and shown in Table 12.

Table 11
Top 14 rules with respect to support in support–confidence framework.

Rule No.	Rule	Support	Confidence	Utility value	Utility-confidence
1	Mountain Bottle Cage → Water Bottle	998	0.83	82,295	0.83
2	Road Bottle Cage → Water Bottle	897	0.89	70,427	0.9
3	Mountain Tire Tube → Sport-100	749	0.42	245,572	0.4
4	HL Mountain Tire → Mountain Tire Tube	552	0.68	121,680	0.67
5	Touring Tire Tube → Touring Tire	507	0.57	94,737	0.87
6	Touring Tire → Touring Tire Tube	507	0.87	94,737	0.57
7	ML Mountain Tire → Mountain Tire Tube	435	0.66	84,810	0.66
8	ML Road Tire → Road Tire Tube	363	0.68	66,636	0.68
9	Half-Finger Gloves → Sport-100	352	0.41	154,524	0.42
10	Touring-1000 → Sport-100	344	0.42	4,532,612	0.53
11	Mountain-200, Mountain Bottle Cage → Water Bottle	344	0.8	4,426,705	0.81
12	Water Bottle, Mountain-200 → Mountain Bottle Cage	344	1	4,426,705	1
13	LL Road Tire → Road Tire Tube	334	0.55	57,495	0.56
14	HL Road Tire → Road Tire Tube	326	0.7	78,795	0.72

Table 12
Top 14 rules with respect to utility in utility-confidence framework.

Rule No.	Rule	Utility value	Utility-confidence	Support	Confidence
1	Touring-1000 → Sport-100	4,532,612	0.57	344	0.42
2	Water Bottle, Mountain-200 → Mountain Bottle Cage	4,426,705	1	344	1
3	Mountain Bottle Cage, Mountain-200 → Water Bottle	4,426,705	0.81	344	0.8
4	Mountain Tire Tube, Mountain-200 → HL Mountain Tire	2,734,660	1	204	1.0
5	HL Mountain Tire, Mountain-200 → Mountain Tire Tube	2,734,660	0.67	204	0.65
6	Road-550-W → Sport-100	2,610,225	0.73	264	0.43
7	Touring-1000, Water Bottle → Road Bottle Cage	2,322,753	1	195	1.0
8	Road Bottle Cage, Touring-1000 → Water Bottle	2,322,753	0.9	195	0.9
9	Road-750, Water Bottle → Road Bottle Cage	1,742,757	1	278	1.0
10	Road-750, Road Bottle Cage → Water Bottle	1,742,757	0.95	278	0.87
11	Mountain Bottle Cage, Sport-100 → Mountain-200	1,718,844	0.4	135	0.4
12	Patch kit, Mountain-200 → HL Mountain Tire	1,373,430	0.61	89	0.46
13	Touring Tire Tube, Touring-1000 → Touring Tire	1,319,548	1	95	1.0
14	Touring Tire, Touring-1000 → Touring Tire Tube	1,319,548	0.87	95	0.89

Table 13
Number of candidates generated with different *min_util* of different datasets.

Dataset	*min_util* (%)	# of Cand. for FHM	# of Cand. for FHIM	% of Cand. pruned (%)
Foodmart	0.05	2,862,896	1,198,011	58.15
	0.06	15,98,864	1,180,953	26.13
	0.07	1,252,117	1,175,749	6.09
	0.08	1,187,833	1,173,032	1.25
Chess	20	1,251,083	1,238,568	1
	22	500,398	494,654	1.15
	24	199,649	197,239	1.21
	26	81,662	80,436	1.5
Mushroom	1	41,169,325	38,416,343	6.68
	2	8,230,798	7,702,670	6.42
	3	2,788,719	2,611,890	6.34
	4	1,152,048	1,079,207	6.32
T10I4D100K	0.004	89,455,922	31,066,876	65.27
	0.005	52,524,204	24,037,538	54.23
	0.006	34,750,540	19,176,083	44.82
	0.007	24,972,505	15,598,576	37.54
Retail	0.01	71,880,396	50,381,512	29.9
	0.02	31,073,941	29,086,264	6.39
	0.03	20,662,033	20,058,672	2.92
	0.04	14,535,985	14,299,784	1.62
Chain-store	0.04	13,217,269	13,211,125	0.05
	0.05	6,271,300	6,268,374	0.05
	0.06	3,001,556	2,999,339	0.07
	0.07	1,311,632	1,310,693	0.07

Observe that the two confidence measures accord on most of the association rules, though they both differ in some rules. Again, most of the rules ranking are different in both the frameworks. In essence, the traditional association rule mining framework misses two types of rules:

- Rules have low frequent and high utility values.
- Rules with small confidence but contributes more profit share to the utility value of the antecedent.

For example, in Table 12 the rule *Touring Tire, Touring-1000 → Touring Tire Tube* is of less frequent but high utility value than the rule *HL Road Tire → Road Tire Tube* of Table 11 and is of first type. Again, the utility-confidence of the rule *Patch kit, Mountain-200 → HL Mountain Tire* is 0.61 in Table 12, which is larger than its confidence value. It indicates that 61% profit of *Patch kit, Mountain-200* is obtained from the composite selling of *Patch kit*, *Mountain-200* and *HL Mountain Tire*, which is more acceptable than its frequency of selling. This information is not qualified by support–confidence framework, and this rule is of second type.

From Tables 11 and 12, it is observed that the high support of a rule does not necessarily imply a high utility itemset, because there exist less frequent items, which are more profitable than highly occurrence itemsets. From the confidence and utility-confidence, it is also clear that the confidence indicates the percentage of simultaneous occurrence of the antecedent with the consequent, while the utility confidence indicates the percentage of profit of antecedent contributed towards to its total profit selling with the consequent. Hence, using both the measures simultaneously, one can improve the financial decisions.

6.2. Impact on number of candidate high utility itemsets

In this part of experiment, we compare the number of candidate high utility itemsets generated in FHM and FHIM methods. The comparison with other methods is not shown as FHM generates less candidate than HUI-Miner algorithm, which generates less number of candidates than UP-Growth and UP-Growth$^+$. The number of potential candidates in both FHM and FHIM algorithms indicates the number of recurrences in the respective methods. For each dataset, we vary the value of *min_util* as the percentage of total utility and we calculate the number of candidate itemsets. The results of most datasets are shown in Table 13. The fifth column of this table represents the percentage of candidate pruned by FHIM. We observe that the number of candidate patterns in our proposed method, FHIM, is less than the number of patterns generated by FHM algorithm. The experimental results show that the proposed pruning strategy enables FHIM in reducing the number of candidate patterns as compared to that for FHM algorithm. Also, in most datasets, it is observed that the percentage of candidates pruned increases with the decrease of the minimum utility value. This makes FHIM more faster than FHM.

6.3. Impact on execution time, memory consumption and scalability

We have evaluated the efficiency of our proposed methods in terms of running time, and memory by varying the minimum utility *min_util* value against FHM, CHUD, UP-Growth and UP-Growth$^+$ algorithms. When the HUCI-Miner algorithm is composite with FHIM, the utility unit array of an itemset is calculated from utility list. For the experiments of UP-Growth and UP-Growth$^+$, the generated candidate itemsets are stored in a file. To calculate their actual utility values in phase-II, the generated candidate itemsets are loaded in main memory. To optimize the phase-II of UP-Growth and UP-Growth$^+$ the original dataset is transformed to the revised dataset that only contains the promising items and the database scan is performed on the revised dataset as in Liu and Qu (2012). The phase-II of CHUD algorithm is also performed on the revised

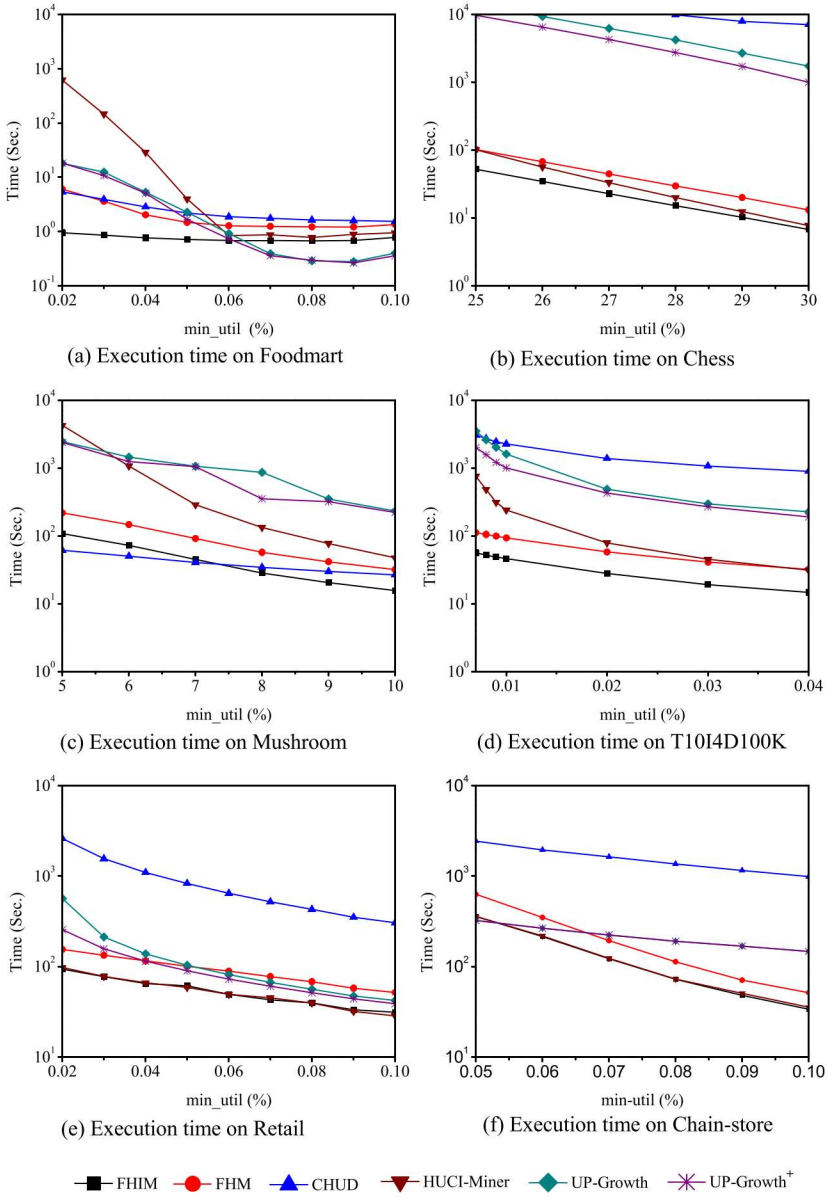


Fig. 3. Execution time comparison on the datasets mentioned in Table 10.

transactions with the optimization technique provided in Wu et al. (2011).

Fig. 3 plots the comparison results on running time for Foodmart, Chess, Mushroom, T10I4D100K, Retail and Chain-store datasets. The minimum utility value threshold *min_util* in percentage is taken on X-axis as liner scale and the running time in second is taken on Y-axis as log scale. All the six algorithms are executed for five times each for fixed *min_util* value and the average running time is plotted for each *min_util*. From Fig. 3, we observe that the running time for all the datasets is more for low *min_util* value, and it grows fast as *min_util* value decreases. Fig. 3(a) and (c) show the comparison time on datasets, Foodmart and Mushroom, respectively. Note that in Foodmart dataset for high utility value, HUCI-Miner is more efficient than CHUD, but the performance of HUCI-Miner degrades as *min_util* value decreases. In case of Mushroom dataset, CHUD achieves better performance than HUCI-Miner. The reason is HUCI-Miner performs more activity for the generation of high utility itemset generators, which scans over the all high utility itemsets. The running time comparison on Chess, Retail and Chain-store are shown in Fig. 3(b), (e) and (f), respectively. The plot of these figures shows that HUCI-Miner takes less time than CHUD. In this case, the intersection operation among the Tidlists in CHUD takes more time than the subset testing among high utility itemsets in HUCI-Miner. Also, in the datasets Retail and Chain-store the CHUD method takes more time than UP-Growth as these datasets contain more items and transactions. The intersection operation among Tidlists is costly. In the dataset T10I4D100K, for high minimum utility threshold, HUCI-Miner is more efficient, but as the *min_util* value decreases the plot of HUCI-Miner increases exponentially. Again, in all datasets the FHIM method dominates the state-of-the-art high utility mining algorithm FHM. The proposed FHIM method is two times faster than FHM method for low *min_util* value. It is clear that FHIM is faster than other existing methods. For all most datasets, and *min_util* values, the superiority of FHIM demonstrates the effectiveness of the proposed pruning strategies.

We have shown the memory consumption of six algorithms on Foodmart, Chess, Mushroom, T10I4D100K, Retail and Chain-store

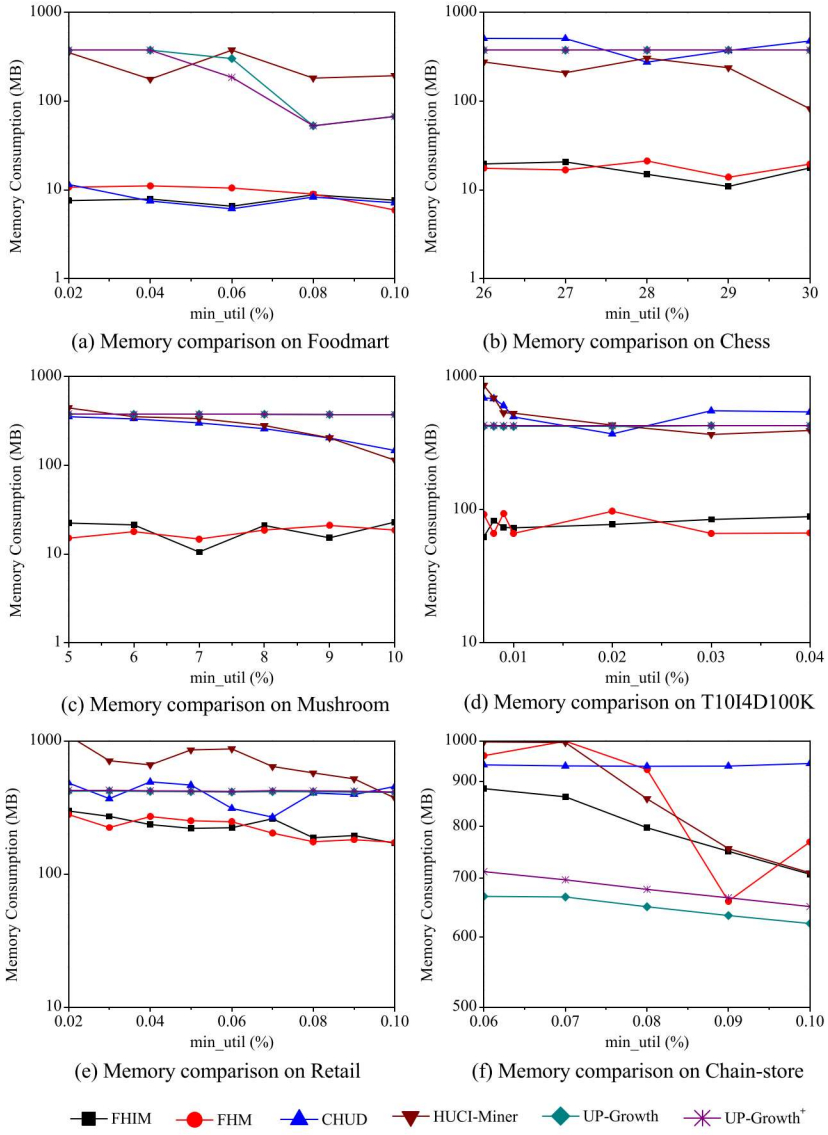


Fig. 4. Memory Consumption comparison on the datasets mentioned in Table 10.

datasets. The average maximum memory used by each algorithm is calculated by varying *min_util* and executing five times. Fig. 4(a) plots the graph for memory used on the sparse dataset, Foodmart. This figure shows when the *min_util* value is less than 0.04%, the memory used increases very fast in HUCI-Miner. CHUD consumes less memory than FHM and almost same memory used by FHIM. For high utility value, FHIM dominates CHUD. Fig. 4(b) plots the memory consumption graph for the dense dataset, Chess. The growth rate of memory used by CHUD is slightly more than the memory used in HUCI-Miner as more number of intersection operations is executed in CHUD. The memory consumption growth rate of HUCI-Miner is similar to the growth of CHUD in Mushroom and T10I4D100K datasets, which is evident from Fig. 4(c) and (d). The graph on Retail datasets shown in Fig. 4(e) reflects that the memory used by UP-Growth and UP-Growth$^+$ is almost constant, but for others it grows linearly. The memory consumption of UP-Growth is very less in the sparse dataset Chain-store as shown in Fig. 4(f). In all the plots, it can be observed that the memory consumption of FHIM and FHM are comparable. The memory consumed by both FHIM and FHM vary alternatively in Retail and Chain-store datasets. The memory consumed by the local co-occurrence table is less than the memory used by the utility-list of the extra candidate itemsets generated by FHM. In general, as the memory used in each algorithm is proportional to the number of candidate itemsets generated, FHIM consumes less memory than FHM.

We have performed the experiments on the extremely dense dataset, Connect. For this dataset, our implementation of UP-Growth, UP-Growth$^+$, and CHUD could not be executed (showing out of memory) for the considered minimum utility percentage. So, we have executed and plotted the graphs for FHIM, FHM and HUCI-Miner for the execution time and memory used in each algorithm shown in Fig. 5. We vary the minimum utility threshold from 27% to 30%. Fig. 5(a) shows the plot of running time comparison and Fig. 5(b) shows the graph for memory consumption on this dataset. The FHM algorithm consumes more time than FHIM and HUCI-Miner, but the growth rate of these algorithms are linear. The memory consumed by the HUCI-Miner algorithm is more than other two as it maintains all the extracted high utility itemsets in the main memory for generating generators and high utility closed itemsets.

We have performed the experiments on the scalability, which is measured in terms of the amount of main memory used and execution time by an algorithm. The study of scalability of FHIM and

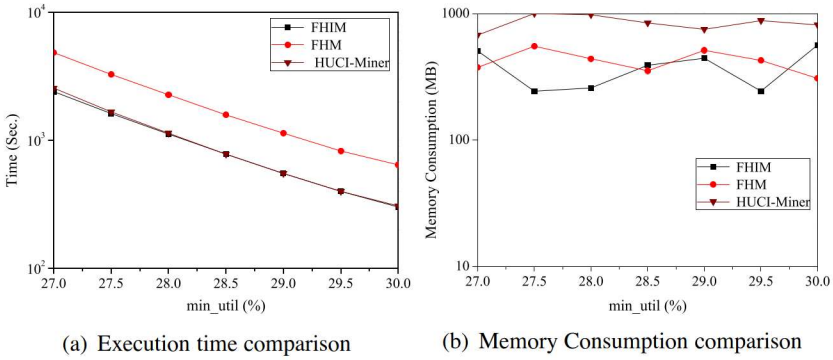


Fig. 5. Experiment on Connect dataset with varying minimum utility threshold (*min_util*) in %.

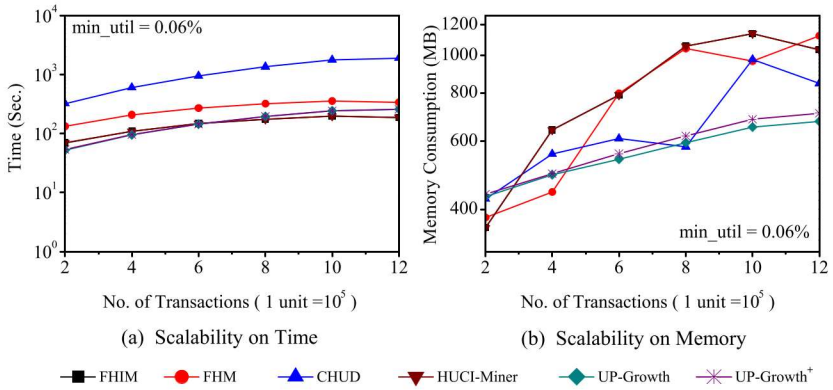


Fig. 6. Scalability study by varying number of transactions on Chain-store.

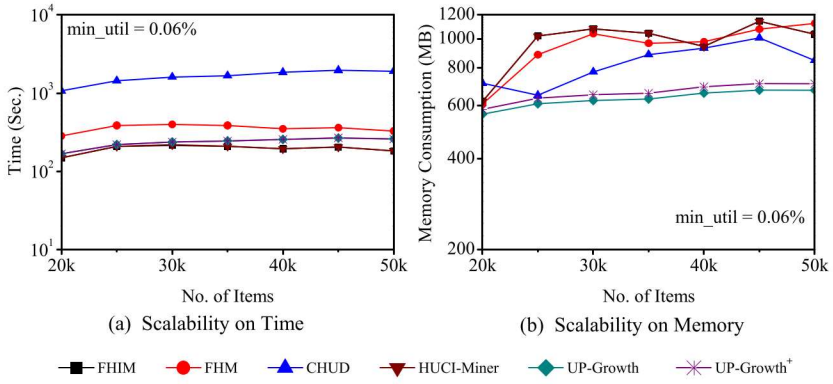
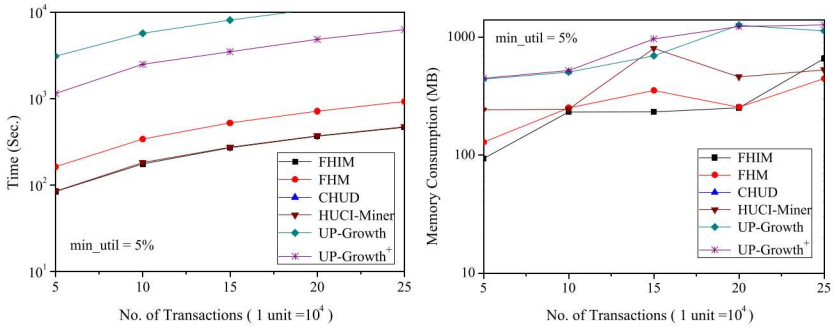


Fig. 7. Scalability study by varying number of items on Chain-store.

HUCI-Miner is performed by varying the transaction size and number of items with fixed *min_util*. We have used the Chain-store dataset, which is a sparse dataset with a large number of transactions and distinct items. As this dataset contains 1,112,949 number of transactions, another 87,051 random transaction are added to make it a dataset of 1,200,000 number of transaction. Again as this dataset contains 46,086 number of items, another 3914 number of items are added to this dataset with their utility value are chosen randomly from the existing utility value distribution of Chain-store. Then the modified dataset is processed for scalability testing. Initially, 200,000 random numbers of transactions are selected from this modified Chain-store dataset and processed. In each process, another 200,000 random numbers of transactions are selected and added to the previous processed dataset to increase the transaction size. The performance results of the experiments for execution time and memory requirement by varying the number of transactions are reported in Fig. 6. Fig. 6(a) shows the execution time of our FHIM and HUCI-Miner algorithms and existing CHUD, FHM, UP-Growth and UP-Growth$^+$ algorithms as a function of the transaction size. Fig. 6(b) shows the memory requirement versus the number of transactions. It is clear that the overall mining time and memory consumption increases when the number of transactions increases. However, FHM, FHIM, HUCI-Miner, UP-Growth and UP-Growth$^+$ show a linear increase in run-time and memory utilization when the number of transactions increases.



(a) Scalability on execution time (b) Scalability on memory

Fig. 8. Scalability study by varying number of transactions on T30I1D500K.

Table 14
Number of high utility closed itemset (HUCI) of different datasets.

Dataset	*min_util* in (%)	# of HUI	# of HUCI	# HG	# of HUCI + HG
Foodmart	0.07	637	605	22	627
	0.06	1487	771	304	1075
	0.05	6276	1076	1581	2667
	0.04	20,766	1762	4686	6448
Chess	28	1982	1519	433	1952
	27	3874	2776	962	3738
	26	7187	4910	1937	6847
	25	13,331	8700	3811	12,511
Mushroom	8	28,764	982	3254	4236
	7	51,188	1470	5453	6923
	6	108,797	2216	10,161	12,377
	5	222,742	3308	17,303	20,611
T10I4D100K	0.009	93,155	66,467	9828	76,226
	0.008	111,754	76,226	13,166	89,392
	0.007	141,382	90,187	18,851	109,038
	0.006	192,673	111,544	28,837	140,381
Retail	0.05	1876	1874	2	1876
	0.04	2810	2802	8	2810
	0.03	4848	4828	20	4848
	0.02	9355	9303	52	9355
Chain-store	0.08	118	118	–	118
	0.07	151	151	–	151
	0.06	193	193	–	193
	0.05	260	260	–	260

In order to perform the scalability by varying the number of items, the same modified database Chain-store is chosen. Initially 20,000 items are randomly chosen, such that each transaction of Chain store contains at least one item of these 20,000 items. The modified database is processed for a fixed *min_util*. In each process, another 5000 number of random items are selected and added to the transaction of the previous processed database randomly. The execution and memory consumption by varying number of items with a fixed *min_util* is shown in Fig. 7. The graphs of this figure shows that both FHIM and HUCI-Miner grow linearly as the number of items increases as in FHM.

In order to strengthen our experiments, we have performed the scalability on a synthetic dense dataset, T30I1D500K, which is generated using the code of IBM Quest Synthetic Data Generator. This dataset contains 100 items with the average transactional length 30 and 500,000 transactions. The execution of CHUD implementations is out of memory for this dataset for the chosen minimum utility threshold. To test the scalability, initially the first 50,000 transactions are chosen and for the subsequent process next 50,000 transactions are added. The plots of the rest algorithms on scalability are shown in Fig. 8. The execution time taken by each algorithm grows linearly and it is evident from Fig. 8(a). Although the memory consumed by each algorithm in Fig. 8(b) follows some zigzag path, the graphs are nearly linear as the number of transactions increases.

Table 15
Number of high utility association rules (HAR) and HGB of different datasets.

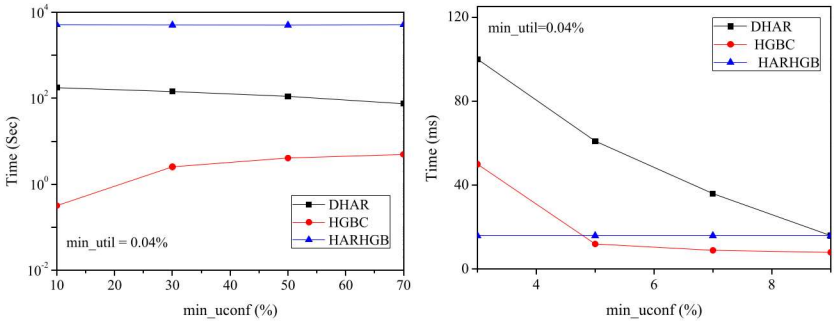
Dataset (*min_util* (%))	*min_uconf* in (%)	# of HAR	# of HGB	δ in (%)
Foodmart (0.04)	70	810,488	11,895	98.53
	50	811,876	10,630	98.69
	30	812,353	7360	99.09
	10	851,635	1599	99.81
Mushroom (11)	40	778,056	1962	99.75
	30	827,627	1574	99.81
	20	881,346	1045	99.88
	10	881,350	1041	99.88
Chess (25)	90	347,346	58,208	83.24
	80	1,080,743	92,831	91.41
	70	1,953,364	108,552	94.44
	60	2,466,563	84,271	96.58
Chain-store (0.03)	9	230	213	7.39
	7	277	257	7.22
	5	348	313	10.06
	3	411	367	10.71

6.4. Impact on number of high utility closed itemsets versus the number of high utility itemsets

Table 14 reports the number of high utility itemsets (HUI), high utility closed itemsets (HUCI), and high utility generators (HG) extracted by HUCI-Miner in various datasets with different *min_util* values. The itemset, which is a generator of itself, is not counted in HG. The symbol '–' represents that there is no generator except those, who are generators of themselves. From Table 14, we see that the total number of HUCI and HG is less than or equal to the total number of high utility itemsets. Since all HUIs can be generated from HUCIs with the help of utility unit array, HGs help in finding non-redundant rules in support–confidence framework of high utility mining. Note that our proposed HUCI-Miner algorithm achieves a great reduction in compressing the number of high utility itemsets.

6.5. Performance of HGB with different min_uconf

We have evaluated the compactness of our proposed *HGB* basis. The major issue on extracting association rule on the utility-confidence framework is concerned with the size of the extracted rule



(a) Execution time on Foodmart (b) Execution time on Chain-store

Fig. 9. Evaluation of utility-based association rule generation methods.

sets. We have evaluated the number of rules in *HGB* basis and the total number of high utility rules on different datasets by keeping the fixed *min_util* value and varying *min_uconf*, and the results are reported in Table 15. Let | *HAR* | be the total number of high utility association rules and | *HGB* | the total number of *HGB* rules. The compression ratio (δ) gained by the basis *HGB* against the total rules HAR is given by $\delta = 1 - \frac{|HGB|}{|HAR|}$.

Compression gain indicates that there is less number of HGB rules generated as compared to the total rules. In Foodmart, Mushroom and Chess datasets, the compression gain is more as compared to the compression gain in Chain-store dataset. It can be observed that except Chess and Chain-store dataset, the number of HARs are increased with the decrease of *min_uconf*. On the other hand, the number of HGB rules are increased with the increase of *min_uconf*. In Chain-store dataset, there is very less number of high utility itemsets of larger size with the *min_util* value as compared to other datasets.

To show the execution time of HGB and all high utility association rules generation process, we have used some notations as follows. The direct generation of high utility association rules from the high utility itemsets using the traditional rule generation method is denoted as DHAR. The HGB-construction algorithm is denoted as HGBC and the generation of all high utility association rules from HGB is denoted as HARHGB. The execution time of each rule generation method is executed for a fixed minimum utility value and by varying the utility-confidence value. The running time does not count the time taken by HUCI-Miner. Although various experiments on each dataset are performed, we have plotted for the Foodmart and Chain-store datasets given in Fig. 9. In case of Foodmart dataset, HARHGB takes more time as compared to the direct method, DHAR. The inference mechanism consumes more time on checking a rule present in a set currently discovers the rules set or not. Moreover, the time complexity of DHAR is a function of the number of HUIs, but the complexity of HARHGB is a function of the number of rules discovered so far. In general, the number of rules is exponential to the number of HUIs. Hence, it is not surprising to see that it takes more time as compared to the traditional rule generation method. The rules compressed by HGB is designed to provide fewer non-redundant high utility association rules. In Chain-store datasets, for the considered minimum utility value, DHAR takes more time. In both cases, HGBC consumes less execution time as compared to the direct generation method.

7. Conclusion

Association rule mining techniques are being studied exhaustively by researchers to discover the relationships between itemsets using statistical measure support and confidence. However, they do not provide any semantic implication between itemsets, such as relationship with respect to utility. To provide a semantic relationship between itemsets, we have used the utility-confidence framework to mine association rules in high utility itemset mining. We have addressed the problem of redundancy in association rules extracted from high utility itemset by proposing a compact representation of the rule set, called the high utility generic basis (HGB) containing rules with minimal antecedent and maximal consequent. To the best of our knowledge, this is the first study on mining non-redundant association rules extracted from the high utility itemsets. This is a two step process: finding all high utility itemsets and then extracting all valid rules. To extract the non-redundant rule set, the high utility closed itemset and generator are mined first by the proposed algorithm HUCI-Miner that identifies the high utility itemsets, high utility closed itemsets and associates the high utility generators to the corresponding closed itemset. We have proposed an algorithm to mine all non-redundant rules under this proposed framework. We have further proposed an inference algorithm to derive all valid association rules under this framework. We have then studied the performance of HUCI-Miner with UP-Growth, CHUD and FHM algorithms on various datasets and shown that FHIM and HUCI-Miner almost outperforms than other existing approaches. Furthermore, experimental results show significantly better achievement in compactness of the proposed rule mining algorithms.

In this work, we have addressed the importance of a utility-based association rule and their redundancies using the semantic measure utility-confidence. However, there are other utility-based interestingness measures and subjective interesting measures (Geng & Hamilton, 2006), which are not incorporated in the high utility itemset mining. Incorporating various interesting measures of association rules in high utility itemset mining with the inspiration of this work is an interesting future research direction. Furthermore, the proposed approach can be used in web-recommendation systems by considering both page weights and the number of times a user visited to page in a stipulated period of a web transaction. The proposed concepts and framework could be extended for discovering rules with negative item utilities and considering the different types of databases such as temporal databases and dynamic databases. In addition, in future we like to integrate the proposed framework to associative classification models.

Acknowledgements

The authors would like to acknowledge the many helpful suggestions of the anonymous reviewers, the Editor and the Editor-in-Chief, Dr. Binshan Lin, which have improved significantly the content and the presentation of this paper.

References

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules in large databases. *Proceedings of the 20th international conference on very large data bases. VLDB'94* (Vol. 1215, pp. 487–499). Morgan Kaufmann Publishers Inc..

Ahmed, C., Tanbeer, S., Jeong, B.-S., & Lee, Y.-K. (2009). Efficient tree structures for high utility pattern mining in incremental databases. *IEEE Transactions on Knowledge and Data Engineering, 21*(12), 1708–1721.

Ahmed, C. F., Tanbeer, S. K., Jeong, B.-S., & Lee, Y.-K. (2011). Huc-prune: An efficient candidate pruning technique tomine high utility patterns. *Applied Intelligence, 34*(2), 181–198.

Alonso, F., Martinez, L., Perez, A., & Valente, J. P. (2012). Cooperation between expert knowledge and data mining discovered knowledge: Lessons learned. *Expert Systems with Applications, 39*(8), 7524–7535.

Balcázar, J. L. (2013). Formal and computational properties of the confidence boost of association rules. *ACM Transactions on Knowledge Discovery from Data, 7*(4), 19:1–19:41.

Barber, B., & Hamilton, H. J. (2001). Parametric algorithms for mining share frequent itemsets. *Journal of Intelligent Information Systems, 16*(3), 277–293.

Barber, B., & Hamilton, H. J. (2003). Extracting share frequent itemsets with infrequent subsets. *Data Mining and Knowledge Discovery, 7*(2), 153–185.

Cai, C. H., Fu, A.-C., Cheng, C., & Kwong, W. W. (1998). Mining association rules with weighted items. In *Proceedings of international database engineering and applications symposium (IDEAS'98)* (pp. 68–77). IEEE.

Carter, C., Hamilton, H. J., & Cercone, N. (1997). Share based measures for itemsets. In *Principles of data mining and knowledge discovery. Lecture notes in computer science* (Vol. 1263, pp. 14–24). Berlin Heidelberg: Springer.

Chan, R., Yang, Q., & Shen, Y.-D. (2003). Mining high utility itemsets. In *Third IEEE international conference on data mining (ICDM 2003)* (pp. 19–26). IEEE.

Cheng, J., Ke, Y., & Ng, W. (2008). Effective elimination of redundant association rules. *Data Mining and Knowledge Discovery, 16*(2), 221–249.

Chen, Y., Zhao, Y., & Yao, Y. (2007). A profit-based business model for evaluating rule interestingness. In *Advances in artificial intelligence. Lecture notes in computer science* (Vol. 4509, pp. 296–307). Berlin Heidelberg: Springer.

FIMI, (2003). Fimi: The frequent itemset mining dataset repository. Accessed on February 2012. <<http://fimi.cs.helsinki.fi/data/>.

Fournier-Viger, P., Gomariz, A., Soltani, A., & Gueniche, T. (2014). SPMF: Open-source data mining library. Available at <http://www.philippe-fournier-viger.com/spmf/>. Accessed on August 2014.

Fournier-Viger, P., Wu, C., & Tseng, V. S. (2012). Mining top-k association rules. In *Advances in artificial intelligence. Lecture notes in computer science* (Vol. 7310, pp. 61–73). Berlin Heidelberg: Springer.

Fournier-Viger, P., Wu, C.-W., & Tseng, V. (2014c). Novel concise representations of high utility itemsets using generator patterns. In *Advanced data mining and applications. Lecture notes in computer science* (Vol. 8933, pp. 30–43). Springer International Publishing.

Fournier-Viger, P., Wu, C., Zida, S., & Tseng, V. S. (2014b). FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning. In *Foundations of intelligent systems. Lecture notes in computer science* (Vol. 8502, pp. 83–92). Springer International Publishing.

Geng, L., & Hamilton, H. J. (2006). Interestingness measures for data mining: A survey. *ACM Computing Surveys, 38*(3), 9.

Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation. *SIGMOD Record, 29*(2), 1–12.

Hilderman, R. J., Hamiliton, H. J., Carter, C. L., & Cercone, N. (1998). Mining association rules from market basket data using share measures and characterized itemsets. *International Journal on Artificial Intelligence Tools, 07*(02), 189–220.

Huynh-Thi-Le, Q., Le, T., Vo, B., & Le, B. (2015). An efficient and effective algorithm for mining top-rank-k frequent patterns. *Expert Systems with Applications, 42*(1), 156–164.

Kryszkiewicz, M. K. (1998). Representative association rules. In *Research and development in knowledge discovery and data mining. Lecture notes in computer science* (Vol. 1394, pp. 198–209). Springer Berlin Heidelberg.

Lan, G.-C., Hong, T.-P., & Tseng, V. S. (2014). An efficient projection-based indexing approach for mining high utility itemsets. *Knowledge and Information Systems, 38*(1), 85–107.

Lee, D., Park, S.-H., & Moon, S. (2013). Utility-based association rule mining: A marketing solution for cross-selling. *Expert Systems with Applications, 40*(7), 2715–2725.

Lin, C.-W., Hong, T.-P., & Lu, W.-H. (2011). An effective tree structure for mining high utility itemsets. *Expert Systems with Applications, 38*(6), 7419–7424.

Liu, Y., Liao, W.-K., & Choudhary, A. (2005). A fast high utility itemsets mining algorithm. In *Proceedings of the 1st international workshop on utility-based data mining (UBDM'05)* (pp. 90–99). ACM.

Liu, M., & Qu, J. (2012). Mining high utility itemsets without candidate generation. In *Proceedings of the 21st ACM international conference on information and knowledge management (CIKM'12)* (pp. 55–64). ACM.

Li, Y.-C., Yeh, J.-S., & Chang, C.-C. (2008). Isolated items discarding strategy for discovering high utility itemsets. *Data & Knowledge Engineering, 64*(1), 198–217.

Luna, J. M., Romero, J. R., Romero, C., & Ventura, S. (2014). Reducing gaps in quantitative association rules: A genetic programming free-parameter algorithm. *Integrated Computer-Aided Engineering, 21*(4), 321–337.

Luna, J. M., Romero, J. R., & Ventura, S. (2012). Design and behavior study of a grammar-guided genetic programming algorithm for mining association rules. *Knowledge and Information Systems, 32*(1), 53–76.

Mata, J., Alvarez, J., & Riquelme, J. (2001). Mining numeric association rules via evolutionary algorithm. In *Proceedings of the 5th international conference on artificial neural networks and genetic algorithms (ICANNGA'01)* (pp. 264–267).

Mishra, D., Das, A. K., & Mukhopadhyay, S. (2014). A secure user anonymity-preserving biometric-based multi-server authenticated key agreement scheme using smart cards. *Expert Systems with Applications, 41*(18), 8129–8143.

Park, J., Chen, M.-S., & Yu, P. (1995). An effective hash-based algorithm for mining association rules. *SIGMOD Record, 24*(2), 175–186.

Pasquier, N., Bastide, Y., Taouil, R., & Lakhal, L. (1999). Efficient mining of association rules using closed itemset lattices. *Information Systems, 24*(1), 25–46.

Pasquier, N., Taouil, R., Bastide, Y., Stumme, G., & Lakhal, L. (2005). Generating a condensed representation for association rules. *Journal of Intelligent Information Systems, 24*(1), 29–60.

Pears, R., Koh, Y. S., Dobbie, G., & Yeap, W. (2013). Weighted association rule mining via a graph based connectivity model. *Information Sciences, 218*(0), 61–84.

Pei, J., Han, J., Lu, H., Nishio, S., Tang, S., & Yang, D. (2001). H-mine: Hyper-structure mining of frequent patterns in large databases. In *Proceedings of the 2001 IEEE international conference on data mining (ICDM'01)* (pp. 441–448). IEEE.

Pisharath, J., Liu, Y., Liao, W.K., Choudhary, A., Memik, G., & Parhi, J. (2005). Nu-minebench version 2.0 dataset and technical report. Available at <http://cucis.ece.northwestern.edu/projects/DMS/MineBench.html>. Accessed on June 2013.

Ramkumar, G.D., Ramkumar, S., & Shalom, T. (1998). Weighted association rules: Model and algorithm. In *Proceedings of fourth ACM international conference on knowledge discovery and data mining* (pp. 661–666).

Sadik, A. T. (2008). Premises reduction of rule based expert systems using association rules technique. *International Journal of Soft Computing, 3*(3), 195–200.

Sahoo, J., Das, A. K., & Goswami, A. (2014). An effective association rule mining scheme using a new generic basis. *Knowledge and Information Systems*, 1–30.

Salleb-Aouissi, A., Vrain, C., & Nortet, C. (2007). Quantminer: A genetic algorithm for mining quantitative association rules. In *Proceedings of the 20th international joint conference on artificial intelligence (IJCAI'07)* (pp. 1035–1040). Morgan Kaufmann Publishers Inc.

Shie, B.-E., Tseng, V. S., & Yu, P. S. (2010). Online mining of temporal maximal utility itemsets from data streams. In *Proceedings of the 2010 ACM symposium on applied computing (SAC'10)* (pp. 1622–1626). ACM.

Shie, B.-E., Yu, P. S., & Tseng, V. S. (2012). Efficient algorithms for mining maximal high utility itemsets from data streams with different models. *Expert Systems with Applications, 39*(17), 12947–12960.

Shie, B.-E., Yu, P. S., & Tseng, V. S. (2013). Mining interesting user behavior patterns in mobile commerce environments. *Applied Intelligence, 38*(3), 418–435.

Song, W., Liu, Y., & Li, J. (2014). Mining high utility itemsets by dynamically pruning the tree structure. *Applied Intelligence, 40*(1), 29–43.

Szathmary, L., Napoli, A., & Kuznetsov, S. (2007). Zart: A multifunctional itemset mining algorithm. In *Proceedings of the fifth international conference on concept lattices and their applications (CLA'07)* (pp. 26–37).

Szathmary, L., Valtchev, P., Napoli, A., & Godin, R., et al. (2008). An efficient hybrid algorithm for mining frequent closures and generators. In: *6th International conference on concept lattices and their applications (CLA'08)* (pp. 47–58).

Szathmary, L., Valtchev, P., Napoli, A., & Godin, R. (2009). Efficient vertical mining of frequent closures and generators. In *Advances in intelligent data analysis VIII* (pp. 393–404). Berlin Heidelberg: Springer.

Tao, F., Murtagh, F., & Farid, M. (2003). Weighted association rule mining using weighted support and significance framework. In *Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining (KDD'03)* (pp. 661–666). ACM.

Tseng, V. S., Shie, B.-E., Wu, C.-W., & Yu, P. S. (2013). Efficient algorithms for mining high utility itemsets from transactional databases. *IEEE Transactions on Knowledge and Data Engineering, 25*(8), 1772–1786.

Tseng, V. S., Wu, C.-W., Shie, B.-E., & Yu, P. S. (2010). Up-growth: An efficient algorithm for high utility itemset mining. In *Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining (KDD'10)* (pp. 253–262). ACM.

Wang, W., Yang, J., & Yu, P. S. (2000). Efficient mining of weighted association rules (war). In *Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining (KDD'00)* (pp. 270–274). ACM.

Wang, K., Zhou, S., & Han, J. (2002). Profit mining: From patterns to actions. In *Advances in database technology – EDBT 2002. Lecture notes in computer science* (Vol. 2287, pp. 70–87). Berlin Heidelberg: Springer.

Webb, G. (2006). Discovering significant rules. In *Proceedings of the twelfth ACM SIGKDD international conference on knowledge discovery and data mining (KDD'06)* (pp. 434–443). ACM.

Wu, C. W., Fournier-Viger, P., Yu, P. S., & Tseng, V. S. (2011). Efficient mining of a concise and lossless representation of high utility itemsets. In *Proceedings of the 2011 IEEE 11th international conference on data mining (ICDM'11)* (pp. 824–833). IEEE.

Wu, C. W., Fournier-Viger, P., Yu, P. S., & Tseng, V. S. (2014). Efficient algorithms for mining the concise and lossless representation of closed+ high utility itemsets. *IEEE Transactions on Knowledge and Data Engineering, 99*(PrePrints), 1.

Xu, Y., Li, Y., & Shaw, G. (2011). Reliable representations for association rules. *Data & Knowledge Engineering, 70*(6), 555–575.

Yahia, S., Gasmi, G., & Nguifo, E. (2009). A new generic basis of "factual" and "implicative" association rules. *Intelligent Data Analysis, 13*(4), 633–656.

Yan, L., & Li, C. (2006). Incorporating pageview weight into an association-rule-based web recommendation system. In *AI 2006: Advances in Artificial Intelligence* (pp. 577–586). Berlin Heidelberg: Springer.

Yan, X., Zhang, C., & Zhang, S. (2005). Armga: Identifying interesting association rules with genetic algorithms. *Applied Artificial Intelligence, 19*(7), 677–689.

Yao, H., Hamilton, H.J., & Butz, C.J. (2004). A foundational approach to mining itemset utilities from databases. In *Proceedings of the fourth SIAM international conference on data mining* (Vol. 4, pp. 215–221).

Yao, H., Hamilton, H. J., & Geng, L. (2006). A unified framework for utility-based measures for mining itemsets. In *Proceedings of ACM SIGKDD 2nd workshop on utility-based data mining* (pp. 28–37). ACM.

Yun, U. (2007). Efficient mining of weighted interesting patterns with a strong weight and/or support affinity. *Information Sciences, 177*(17), 3477–3499.

Yun, U., Ryang, H., & Ryu, K. H. (2014). High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates. *Expert Systems with Applications, 41*(8), 3861–3878.

Zaki, M. (2004). Mining non-redundant association rules. *Data Mining and Knowledge Discovery, 9*(3), 223–248.

Zhang, H., Padmanabhan, B., & Tuzhilin, A. (2004). On the discovery of significant statistical quantitative rules. In *Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining (KDD'04)* (pp. 374–383). ACM.