Knowledge and Information Systems
DOI 10.1007/s10115-016-1020-2

An efficient algorithm for mining top-k on-shelf high utility itemsets

Thu-Lan Dam · Kenli Li · Philippe Fournier-Viger · Quang-Huy Duong

Received: Mar 09, 2016/Revised: Nov 24, 2016/Accepted: Dec 20, 2016
©Springer-Verlag London 2017

Abstract High on-shelf utility itemset (HOU) mining is an emerging data mining task which consists of discovering sets of items generating a high profit in transaction databases. The task of HOU mining is more difficult than traditional high utility itemset (HUI) mining, because it also considers the shelf time of items, and items having negative unit profits. HOU mining can be used to discover more useful and interesting patterns in real-life applications than traditional HUI mining. Several algorithms have been proposed for this task. However, a major drawback of these algorithms is that it is difficult for users to find a suitable value for the minimum utility threshold parameter. If the threshold is set too high, not enough patterns are found. And if the threshold is set too low, too many patterns will be found and the algorithm may use an excessive amount of time and memory. To address this issue, we propose to address the problem of top-k on-shelf high utility itemset mining, where the user directly specifies k, the desired number of patterns to be output instead of specifying a minimum utility threshold value. An efficient algorithm named KOSHU (fast top-K On-Shelf High Utility itemset miner) is proposed to mine the top-k HOUs efficiently, while considering on-shelf time periods of items, and items having positive and/or negative unit profits. KOSHU introduces three novel strategies, named efficient estimated co-occurrence maximum period rate pruning, period utility pruning and concurrence existing of a pair 2-itemset pruning to reduce the search space. KOSHU also incorporates several novel optimizations and a faster method for constructing utility-lists. An extensive performance study on real-life and synthetic datasets shows that the proposed algorithm is efficient both in terms of runtime and memory consumption, and has excellent scalability.

Keywords Data mining · High utility mining · On-shelf high utility mining · Top-k on-shelf high utility mining

T-L. Dam
College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China
Faculty of Information Technology, Hanoi University of Industry, Hanoi, Vietnam
E-mail: lanfict@gmail.com

K. Li (✉)
College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China
CIC of HPC, National University of Defense Technology, Changsha 410073, China
National Supercomputing Center in Changsha, Changsha , Hunan, 410082, China
E-mail: lkl@hnu.edu.cn

P. Fournier-Viger
School of Natural Sciences and Humanities, Harbin Institute of Technology, Shenzhen Graduate School, Shenzhen, Guangdong 518055, China
E-mail: philfv@hitsz.edu.cn

Q-H. Duong
Faculty of Information Technology, Mathematics and Electrical Engineering, Norwegian University of Science and Technology, Norway
E-mail: huydqyb@gmail.com

1 Introduction

Frequent Itemset Mining (FIM) [1] is a fundamental research topic in data mining that is essential to a wide range of applications. Given a transaction database, the objective of FIM is to discover itemsets appearing frequently in transactions. An itemset is said to be frequent if its support is no less than a given minimum support threshold. Hundreds of studies have been conducted on this topic [1,21,47, 29,19,20,10]. However, an important limitation of FIM is its assumptions that all items have the same importance to the user (e.g. unit profit or weight) and that items may not appear more than once in each transaction. These assumptions often do not hold in real life. For example, in transaction databases, items may have different unit profits, and items in transactions may be associated with different purchase quantities. Besides, in real-life applications, retailers may be more interested in finding itemsets that yield a high profit rather than discovering frequent itemsets.

To address these limitations, the problem of FIM has been redefined as High Utility Itemset Mining (HUIM). In HUIM, items can appear more than once in each transaction and a weight (e.g. unit profit) is assigned to each item. The goal of HUIM is to discover high utility itemsets (HUIs), that are itemsets generating a high profit rather than frequent itemsets. HUIM has become a popular research issue in recent years. It is widely recognized as more difficult than FIM because the powerful downward closure property used to prune the search space in FIM does not hold in HUIM. The downward closure property in FIM states that the support of an itemset is antimonotonic, i.e. supersets of an infrequent itemset are infrequent and subsets of a frequent itemset are frequent. But in HUIM, the utility of an itemset is neither monotonic nor anti-monotonic, that is a high utility itemset may have a superset or subset with lower, equal or higher utility. Thus, techniques to prune the search space developed in FIM cannot be directly applied in HUIM. To address this issue, overestimation methods such as the transaction weighted utility (TWU) [31,40] have been proposed.

Many studies have been carried out to develop efficient HUIM algorithms [40,38,15,46,14,28,27,39, 9]. However, most of them assume that the profits/weights of items are positive. While items having negative unit profits often occur in real-life transaction databases. For example, it is common that a retail store will sell items at a loss to increase the sale of related items, or simply to attract customers. Thus, transactions in a retail store may contain itemsets having negative unit profits. Traditional HUIM algorithms cannot handle databases where items have negative unit profits/weights. If they are applied on such database, they can generate an incomplete set of HUIs [8,12]. Besides, most HUIM algorithms assume that all items are always on sale. However, in real-life some products are only on shelf during specific time periods (e.g. the summer, or important holidays). Since traditional HUIM algorithms do not consider shelf time, they are biased toward finding itemsets with longer shelf time [24,23].

To address the two above limitations, the problem of HUIM has been respectively redefined as the problem of mining HUIs with negative/positive unit profits [8,12], and the problem of mining high on-shelf utility itemsets (HOUs) [24]. Recently, the problem of mining HOUs with negative/positive unit profits [23,16] has been proposed, which is a more general problem definition addressing both limitations. In this problem, an itemset is called a high on-shelf utility itemset if its relative utility is no less than a user-defined threshold.

The TS-HOUN algorithm was first proposed for this problem [23]. It uses a three phase approach that requires generating and maintaining a large amount of candidates in memory and performing multiple database scans. Then, a more efficient algorithm named FOSHU [16] (Faster On-Shelf High Utility itemset miner) was proposed. FOSHU discovers itemsets in a single phase without generating candidates and mines all times periods at the same time, thus avoiding the costly operation of merging patterns found in each time period. Furthermore, FOSHU has introduced novel strategies to handle negative unit profits efficiently.

Although FOSHU outperforms TS-HOUN, on-shelf HUI mining with positive/negative profits remains a time consuming task. It is more difficult than traditional high utility itemset (HUI) mining because the shelf time of items needs to be considered, and items with negative unit profits need to be treated differently. For this reason, techniques used in traditional HUIM and FIM cannot be directly applied to this problem.

Furthermore, an important limitation of traditional FIM, HUI and HOU mining algorithms is that it is difficult for users to find a suitable value for the minimum threshold parameter. If a user sets the threshold too high, not enough patterns are found. And if a user sets the threshold too low, a huge amount of patterns will be found, and the algorithm may use an excessive amount of time and memory. To find an appropriate threshold value, a user generally needs to run a mining algorithm several times using a trial and error approach, which can be very time-consuming.

The prominent solution to the problem of adjusting the thresholds in FIM and HUIM is to redefine these problems respectively as top-k FIM [17,6,42,36] and top-k HUIM [44,34,41,11], where users must specify k, the desired number of itemsets to be found instead of setting a minimum threshold. Specifying a number of patterns to be found has been shown to be much more intuitive for users than specifying a threshold [17,6,42,36,44,34,41,11]. Although top-k pattern mining is desirable, it is generally viewed as more difficult than discovering patterns using a threshold because top-k pattern mining algorithms need to at any time keep potential top-k patterns in memory, and use an internal minimum threshold initially set to 0 and gradually raise the threshold when exploring the search space to exactly find the top k patterns [17,6,42,36,44,34,41,11]. Designing a top-k algorithm thus requires to design efficient strategies for raising the internal minimum threshold as quickly as possible to reduce the search space without missing any desired itemset.

In this study, we address the difficulty of setting the relative utility threshold in HOU mining, by defining the problem of mining the top-k on-shelf high utility itemsets where both positive and negative item unit profits are taken into account, as well as the shelf time of each item. To the best of our knowledge, this is the first time that the problem is considered. Designing an efficient algorithm for this problem is difficult because few effective search space pruning techniques have been proposed for HOU mining. Moreover, it is not straightforward to adapt these pruning techniques and those used in HUIM for top-k HOU itemset mining with positive and negative unit profits.

To overcome these challenges, this study proposes an algorithm named top-K On-Shelf High Utility itemset miner with or without negative unit profits (KOSHU). KOSHU is a utility-list based algorithm, incorporating three novel itemset pruning strategies named EMPRP (Estimated Maximum Period Rate Pruning), PUP (Period Utility Pruning), and CE2P (Concurrence Existing of a pair 2-itemset Pruning). These strategies are easy to implement and effective. Furthermore, we also introduce an efficient method for constructing utility-lists. The proposed algorithm also employs effective strategies to raise the internal relative utility threshold. A rigorous experimental evaluation on several real and synthetic datasets was performed to evaluate the performance of the proposed algorithm and the effectiveness of the proposed strategies. Empirical results show that the proposed strategies are efficient, and that the performance of the proposed algorithm is close to the state-of-the-art on-shelf high utility itemset mining algorithm FOSHU [16] tuned with optimal minimum relative utility thresholds. The key contributions of this work are summarized as follows:

(1) An efficient algorithm is proposed for discovering the top-k on-shelf high utility itemsets in transaction databases. It relies on three efficient strategies named EMPRP, PUP and CE2P which can eliminate a large number of join operations and thus prune a large part of the search space.
(2) The algorithm utilizes several strategies for initializing and dynamically adjusting the internal minimum relative utility threshold during the top-k HOU mining process, named RIRU and RIRU2. These strategies can effectively raise the relative utility threshold, thus contributing to the effectiveness of the EMPRP strategy which is based on a novel Estimated co-occurrence Maximum Period Rate Structure (EMPRS).
(3) A low complexity procedure to construct utility-lists during the mining process is also introduced. It is implemented using a fast binary search method that remembers the index of the last search position so that the next search starts from this position instead of the first position. These techniques provide a considerable performance gain.
(4) Extensive experimental evaluations have been conducted on both real and synthetic datasets to evaluate the proposed techniques. Results show that the proposed algorithm is efficient both in terms of runtime and memory consumption.

The remainder of this paper is organized as follows. Section 2 reviews studies related to HUI, HOU and top-k pattern mining. Preliminaries and a formal problem definition are presented in Section 3. Section 4 presents the novel techniques employed in the proposed algorithm. Then, Section 5 describes the proposed KOSHU algorithm, designed for mining the top-k on-shelf high utility itemsets. It incorporates all the techniques presented in Section 4. Section 6 presents an extensive experimental evaluation. Finally, Section 7 draws the conclusion and discusses opportunities for future work.

2 Related Work

In this section, studies related to utility mining, on-shelf utility mining and top-k pattern mining are briefly reviewed.

2.1 High Utility Itemset Mining

In real life applications, customer transactions contain information about the quantities of items, and items may have different unit profits. Moreover, items may yield a high profit even if they have low selling frequencies. For these reasons, FIM can be viewed as inappropriate for market basket analysis. To overcome this issue, the problem of high utility itemset mining (HUIM) was proposed [3]. It considers both items' unit profits and their quantities in transactions. The goal of HUIM is to discover itemsets having utilities no less than a minimum utility threshold. These itemsets may be rare but still generate a high profit.

Many algorithms have been proposed for high utility itemset mining. A challenge in HUIM is that the utility measure is not monotonic or anti-monotonic (the downward closure property does not hold). To restore the downward closure property, the TWU model [31] was introduced. According to the TWU model, all supersets of an itemset having a TWU lower than the minimum utility threshold also have a TWU lower than that threshold. The disadvantage of using the TWU is that it is a loose upper bound on the utility of itemsets and thus a huge amount of candidates still need to be considered to find the final set of HUIs. Numerous HUI mining algorithms discover high utility itemsets in two phases using the TWU model. This approach is adopted by algorithms such as Two-Phase [31], UP-Growth, UP-Growth+ [40], PB [25] and BAHUI [37]. These algorithms first generate a set of candidate high utility itemsets by overestimating their utilities in Phase 1. Then, in Phase 2, these algorithms perform a database scan to calculate the exact utility of candidates and keep only those that are high utility itemsets. Despite having introduced new techniques to discover high utility itemsets, two-phase algorithms still suffer from the problem of generating too many candidates due to the use of the TWU model.

Recently, algorithms have been proposed to mine HUIs using a single phase. The HUI-Miner algorithm [30] introduced a new data structure called utility-list to maintain information about the utilities of itemsets. Once HUI-Miner has constructed the utility-list of each single item, it can create the utility-lists of larger itemsets without scanning the database, and can derive all HUIs and their exact utilities using utility-lists. HUI-Miner was shown to outperform previous algorithms. However, HUI-Miner still has to perform a costly join operation to obtain the utility-list of each itemset generated by its search procedure. To reduce the number of join operations, the FHM algorithm [15] was introduced. It integrates a strategy for pruning the search space using information about itemset co-occurrences. It was shown that this strategy can reduce the number of join operations by up to 95%, and that FHM can be up to six times faster than HUI-Miner.

In recent years, several algorithms have also been proposed to mine HUIs in data streams. Those algorithms are based on assumptions that are different from those of traditional HUI mining. Data stream mining algorithms assume that data is continuously arriving at a very fast rate, that the amount of data is unknown or unbounded, and that it is impossible to store all the data permanently. Several algorithms for mining HUIs in data streams have been proposed such as THUI-Mine [7], MHUI-BIT [26], MHUI-TID [26], and SHU-Grow [35]. THUI-Mine, MHUI-BIT, and MHUI-TID are Apriori-based algorithms. They employ a level-wise generate-candidate-and-test approach to explore the search space of itemsets. Drawbacks of the Apriori approach is that it requires to perform numerous database scans and maintain a huge number of candidates in memory. The SHU-Grow algorithm addresses this issue by employing a pattern growth approach. SHU-Grow [35] utilizes the RGE (Reducing Global Estimated utilities) and RLE (Reducing Local Estimated utilities) techniques to decrease the overestimated utilities of itemsets. Moreover, SHU-Grow employs a SHU-Tree structure to maintain information about the data stream and high utility patterns. Experimental results have shown that SHU-Grow outperforms the existing sliding window-based methods in terms of runtime and scalability.

2.2 High On-shelf Utility Itemset Mining

Most HUIM algorithms assume that the unit profits of items are positive. However, in real-world applications, items often have negative unit profits. For example, retail stores often sell some products at loss (with a negative unit profit). It was demonstrated that if classical HUIM algorithms are applied on databases containing items with negative unit profits, they can produce an incomplete set of HUIs [8,12]. To address this limitation, the problem of HUIM has been redefined as the problem of mining HUIs with negative/positive unit profits [8,12]. The first algorithm proposed for this problem is HUINIV-Mine [8]. It extends the Two-Phase algorithm [31] by redefining the notion of transaction utility to avoid under-estimating the utility of items using the TWU model. However, HUIM with positive/negative unit profits remains a very expensive task in terms of runtime and memory usage. To

address this issue, a one-phase algorithm named FHN [12] was proposed to mine HUIs while considering both positive and negative unit profits by extending the FHM algorithm. FHN was shown to be up to 500 times faster than HUINIV-Mine and consume up to 250 times less memory.

However, these HUIM algorithms do not consider the on-shelf time periods of items. They assume that all items are always on sale. But in real-life retail stores, some items are only sold during some specific time periods. To address this problem, Lan et al. proposed a new issue named on-shelf utility mining [24] and an algorithm named TP-HOUN for this problem. TP-HOUN is a two-phase algorithm, which only considers items having positive unit profits. Recently, the authors of TP-HOUN introduced the more general problem definition of mining HOUs with negative/positive unit profits [23], and proposed the TS-HOUN algorithm for this problem. This latter uses a three phase approach that requires generating and maintaining a large amount of candidates in memory and performing multiple database scans. Furthermore, TS-HOUN mines patterns in each time period separately and then combines the results found in each time period using a costly merge operation. Therefore, its performance deteriorates when a large number of time periods are used.

Recently, the FOSHU algorithm [16] was proposed to mine HUIs while considering both positive and negative item unit profits and on-shelf time periods. FOSHU is a one-phase algorithm, which employs the utility-list structure. It mines all times periods at the same time and introduces novel strategies to handle negative values efficiently. Results show that FOSHU can be more than 1000 times faster and uses up to 10 times less memory than TS-HOUN. It also performs well on dense databases as well as databases containing many time periods. Although these approaches are able to discover HOUs, they still suffer from long execution times and consume a large amount of memory. They are also not designed to discover top-k HOUs.

2.3 Top-k Pattern Mining

An important drawback of HUIM is that specifying the minimum utility threshold is usually difficult for the user. Users do not know in advance which threshold value will result in the desired amount of patterns. But choosing an appropriate threshold value is crucial and difficult. To address this issue, top-k HUIM was proposed, where the user specifies a parameter k representing the number of patterns to be found instead of setting a minimum utility threshold. Several algorithms have been proposed for this problem. TKU [41] is the first top-k HUIM algorithm. It extends the UP-Growth algorithm [40] with several efficient threshold raising strategies. Zihayat and An [48] proposed an highly efficient algorithm named T-HUDS for mining top-k HUIs over data streams. Recently, an algorithm called REPT [34] was proposed. REPT also relies on the UP-Tree structure proposed in the UP-Growth algorithm [40,41]. But REPT utilizes threshold raising strategies that are different from those used in TKU. However, a drawback of REPT is that in addition to the parameter k, users have to set a value for a parameter N, used by its RSD (Raising by Support Descending order) strategy. Specifying an appropriate value for the parameter N to obtain good performance is difficult for users [41]. The strategies proposed in TKU and REPT may raise the internal minimum utility threshold effectively during the mining process. However, the number of candidates generated by these algorithms is still huge because they are two-phase algorithms. They repeatedly scan the database to obtain the exact utility of candidates and identify the actual top-k HUIs. This process is very expensive both in terms of time and memory, especially when a huge number of candidates are found, or when datasets contain long transactions and many items. Therefore, the authors of TKU also introduced a one-phase algorithm named TKO [41]. TKO was shown to generally outperform both TKU and REPT. The TKO algorithm employs the utility-list structure to store information about the utilities of itemsets in a database. It integrates novel strategies named RUC (Raising threshold by Utility of Candidates), RUZ (Reducing estimated utility values by using Z-elements) and EPB (Exploring the most Promising Branches first) that greatly improve its performance. Very recently, a highly efficient algorithm for mining the top-k HUIs named kHMC [11] was proposed. The kHMC algorithm discovers the top-k high utility itemsets in a single phase. It employs three strategies named RIU (Real Item Utilities), CUD (Co-occurrence with Utility Descending order), and COV (COVerage) to raise its internal minimum utility threshold effectively, and thus reduce a large part of the search space. The COV strategy introduces a novel concept of coverage. This concept can be employed to prune the search space in high utility itemset mining, or to raise the threshold in top-k high utility itemset mining. Furthermore, kHMC relies on a novel co-occurrence pruning technique named EUCPT (Estimated Utility Co-occurrence Pruning Strategy with Threshold) to avoid performing costly join operations for calculating the utilities of itemsets. Moreover, a novel pruning strategy named TEP (Transitive Extension Pruning) is proposed for reducing the search space. In an experimental evaluation, kHMC was shown to outperform the state-of-the-art TKO and REPT

algorithms for top-k high utility itemset mining both in terms of memory consumption and execution time.

In frequent itemset mining, several efficient algorithms have been proposed to mine the top-k frequent itemsets in transaction databases [17,6,42,36]. Moreover, several algorithms have been designed to discover the top-k frequent itemsets in data streams [18,43,45,4]. TOPSIL-Miner [45] mines the top-k significant itemsets in data streams using a prefix-tree structure. It is an approximate algorithm, i.e. it does not guarantee that the exact set of top-k frequent itemsets is found. Another recent algorithm is MSWTP [4], which employs a prefix-pattern tree structure, named SWTP-tree, to efficiently store patterns and calculate their true frequencies in a data stream. MSWTP filters out insignificant patterns by applying the Chernoff bound. Several algorithms have also been designed to mine the top-k frequent items in data streams [33,32,22]. However, they are approximate algorithms, designed for finding the most popular k elements in a data stream. Hence, they only consider single items rather than itemsets. But mining frequent itemsets or the top-k frequent itemsets is much more challenging than counting the frequencies of single items due to the very large number of itemsets that can be obtained by combining single items [5]. Hence, it is difficult or impossible to simply adapt these approaches to HOUs mining, though these algorithms are efficient.

Despite recent advances, algorithms for on-shelf high utility itemset mining still generally suffer from long execution times and large memory consumption. The reason is that it is difficult to extend the effective pruning techniques developed in HUIM to this problem [16,24,23]. Hence, it is necessary to design novel pruning strategies for on-shelf high utility itemset mining. Moreover, specifying an appropriate value for the relative utility threshold used by these algorithms is also very difficult for users, as it requires prior knowledge about the transaction database taken as input. To address these two issues, the present study redefines the problem of HOU mining as the problem of mining the top-k on-shelf high utility itemsets, where the user directly specifies the number of patterns k to be found, instead of specifying a minimum relative utility threshold. Furthermore, an effective algorithm is proposed to solve this problem. The proposed algorithm employs several novel search space pruning strategies to efficiently discover the top-k HOUs, and can consider databases containing positive and negative unit profits.

3 Preliminaries And Problem Definition

This section presents preliminaries related to on-shelf high utility itemset mining and defines the problem of top-k on-shelf high utility itemset mining. Basic definitions are presented as follows.

3.1 Problem Definition

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of items. Each item $i_j \in I$ is associated with a positive or negative number $p(i_j)$, called its external utility or unit profit. Let D be a transaction database containing a set of n transactions $D = \{T_1, T_2, \ldots, T_n\}$ such that $T_d \subseteq I (1 \leq d \leq n)$, and T_d has a unique identifier d called its Tid (Transaction id). Moreover, a positive number $q(i, T_d)$ is associated to each item $i \in T_d$, which represents the purchase quantity (internal utility) of item i in T_d. Let PE be a set of positive integers representing time periods. A time period is a period of time during which some products were sold. For example, a time period could be a given day, week, month, or season. Each transaction $T_d \in D$ is associated with a time period $pt(T_d) \in PE$, representing the time period during which the transaction occurred. As in previous work [16], it is assumed that each time period is associated to at least one transaction, and that each item is sold at least once during each time period where it is on the shelves. An itemset X is a set of l distinct items $\{i_1, i_2, \ldots, i_l\}$ where $i_j \in I; 1 \leq l \leq m$, and l is said to be the length of X. An l-itemset is an itemset of length l. In the rest of this paper, for the sake of simplicity, each itemset may be denoted by the concatenation of its items. For example, $\{x, y, z\}$ is denoted as xyz. Similarly, the union of two itemsets X and Y will be denoted as XY or $X \cup Y$.

Example 1 Consider the transaction database shown in Table 1, which will be used as running example. This database contains nine transactions $(T_1, T_2, \ldots, T_9)$ and three time periods (1, 2, 3). Transaction T_3 occurred in time period 1 and contains four items a, b, f and g, while the numbers 7, 1, 2 and 1 represent the purchase quantities of these items, respectively. The external utilities of items are presented in Table 2. Consider the items sold in T_3, the external utilities of these items a, b, f and g are respectively 1, 5, -2 and 1. Thus, item f is sold at a loss.

Table 1: A transaction database with time periods

TID	Transactions	Period
T_1	$(a,5)(b,1)(e,2)(d,1)$	1
T_2	$(b,3)(c,2)(f,1)$	1
T_3	$(a,7)(b,1)(f,2)(g,1)$	1
T_4	$(b,3)(c,2)$	2
T_5	$(a,1)(b,1)(c,1)(d,1)(f,1)(g,1)$	2
T_6	$(a,5)(d,2)$	2
T_7	$(a,6)(c,2)(d,2)$	3
T_8	$(d,7)(e,2)(f,4)$	3
T_9	$(c,1)(d,3)(e,3)$	3

Table 2: External utility values (Unit profit)

Item	a	b	c	d	e	f	g
Profit	1	5	-3	4	3	-2	1

Definition 1 The utility of an item $i \in I$ in a transaction T_d is denoted as $u(i,T_d)$ and defined as $u(i,T_d) = p(i) \times q(i,T_d)$. The utility of an itemset X in a transaction T_d is defined as $u(X,T_d) = \sum_{i \in X} u(i,T_d)$. It represents the profit generated by items in X in transaction T_d [16].

Definition 2 The time periods (shelf time) of an itemset $X \subseteq I$, are the set of time periods where X was sold, defined as $pi(X) = \{pt(T_d) | T_d \in D \wedge X \subseteq T_d\}$ [16].

Definition 3 The utility of an itemset $X \subseteq I$ in a time period $h \in pi(X)$ is denoted as $u(X,h)$ and defined as $u(X,h) = \sum_{T_d \in D \wedge h = pt(T_d)} u(X,T_d)$. The utility of an itemset $X \subseteq I$ in a database D is defined as $u(X) = \sum_{h \in pi(X)} u(X,h)$ [16].

Note that, $u(X)$ may be negative, which means that itemset X have generated a loss (a negative profit) over a time period. For example, if X is a single item with negative unit profit, then $u(X)$ will always be negative.

Example 2 The utility of item b in T_3 is $u(b,T_3) = 1 \times 5 = 5$. The utility of the itemset $\{b,g\}$ in T_3 is $u(\{b,g\},T_3) = u(b,T_3) + u(g,T_3) = 1 \times 5 + 1 \times 1 = 6$. The time periods of itemset $\{b,g\}$ are $pi(\{b,g\}) = \{1,2\}$. The utility of $\{b,g\}$ in periods 1 and 2 are respectively $u(\{b,g\},1) = 6$ and $u(\{b,g\},2) = 6$. The utility of $\{b,g\}$ in the database is $u(\{b,g\}) = 6 + 6 = 12$.

Definition 4 The transaction utility (TU) of a transaction T_d in a database D is the sum of the utility of the items from T_d in T_d. i.e. $TU(T_d) = \sum_{i \in T_d} u(i,T_d)$. Given an itemset X, the total utility of the time periods of X is defined as $to(X) = \sum_{h \in pi(X) \wedge T_d \in D \wedge h = pt(T_d)} TU(T_d)$ [16]. It is easy to see that $to(X)$ can be positive, negative or equal to zero.

Definition 5 The relative utility of an itemset $X \subseteq I$ in a database D is defined as $ru(X) = u(X)/|to(X)|$, if $to(X) \neq 0$, and is defined as 0 otherwise. The relative utility of an itemset X represents how large the profit/loss generated by X is compared to the total profit/loss generated during the time periods where X was sold. The relative utility measure is useful for retailers as it is an indicator of the relative selling performance (profit/loss) of an itemset during time periods where it was on the shelves. It can thus be used to compare the selling performance of various itemsets in terms of their relative contribution to the overall profit of a retail store, to determine which itemsets are the most profitable.

Example 3 The transaction utility of transactions $T_1, T_2, \ldots, T_6$ are $TU(T_1) = 20$, $TU(T_2) = 7$, $TU(T_3) = 9$, $TU(T_4) = 9$, $TU(T_5) = 6$ and $TU(T_6) = 13$. The total utility of the time periods of $\{b,g\}$ is $to(\{b,g\}) = 64$. The relative utility of $\{b,g\}$ is $ru(\{b,g\} = u(\{b,g\})/to(\{b,g\}) = 12/64 = 0.19$.

An itemset X is a *high on-shelf utility itemset* (HOU) if its relative utility $ru(X)$ is no less than a user-specified minimum utility threshold *minutil* given by the user. Otherwise, X is a *low on-shelf utility itemset*. The problem of on-shelf high utility itemset mining is to discover all HOUs in a database where external utilities are positive [24]. The problem of on-shelf high utility itemset mining with negative values [16,23] is to discover all HOUs in a database where external utilities are positive or negative.

Definition 6 Let there be a transaction database D with time periods, where external utilities of items may be positive or negative. The problem of mining the top-k on-shelf high utility itemset is to discover the k on-shelf itemsets having the highest relative utilities in the database, where k is a parameter specified by the user.

Example 4 If $k = 5$ then the set of top-k high on-shelf utility itemsets in the running example is $\{\{d, e\} : 0.739, \{b\} : 0.703, \{a, b, d, e\} : 0.556, \{d\} : 0.552, \{d, e, f\} : 0.5\}$, where the number after the colon for each itemset indicates its relative utility.

The problem of mining top-k HOUs is interesting for market basket analysis, as it discovers the k itemsets that yield the highest profit relatively to the total profit/loss during the time periods when they were sold. Thus, it can be used to discover the k itemsets that had the best relative selling performance. This give a useful insight to retailers that may then use this information in various ways to adapt their business strategies. For example, a retailer may decide to increase the shelf time of the top-k HOU as they are the best performing products relative to the time periods when they are sold.

It can be demonstrated that the (relative) utility measure is not monotonic or anti-monotonic [24, 23, 30, 31, 41]. In other words, an itemset may have a utility lower, equal or higher than the utility of its subsets. Therefore, the powerful pruning strategies that are used in FIM to prune the search space based on the anti-monotonicity of the support cannot be directly applied to discover HOUs. To restore this property in high on-shelf utility mining, the TWU measure has been adapted to handle time periods [24, 23]. The TWU measure assumes that all items have positive external utility values. The TWU measure, adapted for time periods, is defined as follows.

Definition 7 For a given time period h, the transaction-weighted utilization of an itemset X is denoted as $TWU(X, h)$ and defined as the sum of the utilities of transactions in time period h containing X, that is $TWU(X, h) = \sum_{T_d \in D \wedge X \subseteq T_d \wedge pt(T_d)=h} TU(T_d)$ [23].

The TWU measure has important properties that TS-HOUN uses to prune the search space [23, 31]. However, these properties only hold if external utility values of items are positive [8]. Subsection 3.4 will explain how to handle items with negative unit profits.

Definition 8 The utility of a time period h is denoted as $pto(h)$ and defined as the sum of the transaction utilities of all transactions in period h, that is $pto(h) = \sum_{T_d \in D \wedge h=pt(T_d)} TU(T_d)$. The relative utility of an itemset X in a time period h is $ru(X, h) = u(X)/pto(h)$ [23].

Property 1 The TWU of an itemset X in a time period h is an upper bound on the utility of X in period h, that is $TWU(X, h) \geq u(X, h)$ [23].

Property 2 For a period h, the TWU upper bound is anti-monotonic. Let X and Y be two itemsets, if $X \subset Y$ then $TWU(X, h) \geq TWU(Y, h)$ [23].

Property 3 The TWU of an itemset X in a period h divided by the total utility of the time period h is an upper bound on the relative utility of X in period h, i.e. $TWU(X, h)/pto(h) \geq ru(X, h)$ [23].

Property 4 Let X be an itemset, if there exists no time period $h \in PE$ such that $TWU(X, h)/pto(h) \geq minutil$, then X is not a high on-shelf utility itemset. Otherwise, X may or may not be a high on-shelf utility itemset [16].

3.2 Utility-list Structure

The utility-list structure was proposed in the HUI-Miner [30] algorithm to mine high utility itemsets directly, and avoid repeatedly scanning the database to calculate the utility of itemsets. The utility-list structure is used to maintain information about the utility of itemsets. The utility of an itemset can be quickly calculated by making join operations with utility-lists of smaller itemsets. The definition of utility-lists and their properties are presented next.

Definition 9 (Utility-list [30]) Without loss of generality, let $\succ$ be a total order on items from I. The *utility-list* of an itemset X in a database D is denoted as $ul(X)$. It is a set of tuples, containing a tuple $(tid, iutil, rutil)$ for each transaction T_{tid} containing X. The $iutil$ element of a tuple is the utility of X in T_{tid}. i.e., $u(X, T_{tid})$. The $rutil$ element of a tuple is defined as $\sum_{i \in T_{tid} \wedge i \succ x \forall x \in X} u(i, T_{tid})$.

Property 5 Let there be an itemset X. The utility of X denoted as $u(X)$ is the sum of *iutil* values in its utility-list $ul(X)$ [30].

Property 6 Let there be an itemset X. The sum of *iutil* and *rutil* values in its utility-list $ul(X)$ is an upper bound on $u(X)$. Moreover, it was shown that this upper bound is tighter than $TWU(X)$ [30].

Property 7 Let there be an itemset X. Let the *extensions* of X be the itemsets that can be obtained by appending an item y to X such that $y \succ i, \forall i \in X$. The utilities of transitive extensions of X can only be less than or equal to the sum of *iutil* and *rutil* values in $ul(X)$ [30].

3.3 Handling Time Periods

This subsection describes how properties of utility-lists have been adapted to handle the case of databases having time period information. These modified properties are given next.

Property 8 Let there be an itemset X and a time period h. Let $sumIUtil(X,h)$ and $sumRUtil(X,h)$ respectively denote the sum of *iutil* and *rutil* values in the utility-list $ul(X)$ for the time period h. An upper bound on $u(X,h)$ is $sumIUtil(X,h) + sumRUtil(X,h)$. This upper bound is tighter than $TWU(X,h)$, i.e. $sumIUtil(X,h) + sumRUtil(X,h) \geq u(X,h)$ [16].

Property 9 Let X and Y be two itemsets. If $X \subset Y$ then $sumIUtil(X,h)+sumRUtil(X,h) \geq sumIUtil(Y,h)+sumRUtil(Y,h)$ [16].

Property 10 For an itemset X and a period h, the value $sumIUtil(X,h) + sumRUtil(X,h)$ divided by the total utility of time period $pto(h)$ is an upper bound on the relative utility of X in period h, that means $(sumIUtil(X,h) + sumRUtil(X,h))/pto(h) \geq ru(X,h)$ [16].

Property 11 Let X be an itemset. If there exists no time period h such that $(sumIUtil(X,h) + sumRUtil(X,h))/pto(h) \geq minutil$ then X is not a high on-shelf utility itemset as well as any transitive extensions of X according to the total order $\succ$. Recall that an extension of X is an itemset obtained by appending an item y to X such that $y \succ x, \forall i \in X$ [16].

3.4 Handling Negative Unit Profits

Traditional HUI mining and HOU mining [24] assume that all items have a positive unit profit. But in real-life, this assumption often does not hold as items are often sold at a loss in retail stores to promote the sale of other items or attract customers. It was shown that if traditional HUI mining or HOU mining algorithms are applied on databases containing items with negative unit profits, an incomplete set of results may be found [8,23]. Thus, an important question is how to handle items with negative unit profits, to ensure that a complete set of results is found.

There is no trivial solution to this problem. It is tempting to think that this problem could be simply solved by just shifting all unit profit values of items by a constant so that they would all become positive, and that a normal high utility itemset mining algorithm could then be applied to discover the itemsets. However, this approach would result in an incomplete algorithm. We show this with an example. Consider that the retail store sells an item f 1000 times with a negative unit profit of -2$, and that an item g is sold 10 times with a positive unit profit of 1$. If we shift the unit profit of these items using a constant $x > 2$ so that the unit profits become positive, then the itemset $\{f\}$ will be viewed as generating a higher profit than $\{g\}$, which is incorrect. Thus, we cannot use a simple approach of shifting unit profits to solve this problem, and it is thus necessary to design a specific way of addressing negative unit profit values.

To handle items having negative unit profits, the TWU was redefined as follows [23] based on previous work [8].

Definition 10 The redefined transaction utility (RTU) of a transaction T_d is the sum of the utilities of items in T_d having positive external utilities, that is $RTU(T_d) = \sum_{x \in T_d \wedge p(x) > 0} u(x, T_d)$. The redefined transaction-weighted utilization (RTWU) of an itemset X in a time period h is defined as $RTWU(X,h) = \sum_{T_d \in D \wedge X \subseteq T_d \wedge pt(T_d) = h} RTU(T_d)$ [23].

Using the RTWU instead of the TWU restores Property 4 for on-shelf high utility mining. This allows TS-HOUN[23] to prune the search space, and still find the complete set of HOUs.

Furthermore, to handle both time periods and items with negative unit profits, properties have been further adapted as follows [16]. Let the terms "positive items" and "negative items" denote items respectively having positive and negative unit profits. The total order $\succ$ is defined such that negative items always succeed all positive items. By using this order, positive items are always used to extend an itemset first before appending negative items. This total order is used to define some new pruning properties. Let $up(X)$ and $un(X)$ respectively refer to the set of all positive items and the set of all negative items in X.

Property 12 Let there be an itemset X. It can be found that $u(X,h) \leq u(up(X),h)$ holds [16].

Property 13 Let X be an itemset and z be a negative item such that $z \notin X$. It follows that $u(up(X \cup \{z\}),h) \leq u(up(X),h)$ [16].

Property 14 Let X be an itemset. For any itemset Y resulting from transitive extensions of X with negative items, we have $u(up(Y),h) \leq u(up(X),h)$ [16].

Property 15 Let X be an itemset. If only negative items can be appended to X according to the total order $\succ$ and there exists no time period h such that $u(up(X),h)/pto(h) \geq minutil$, then X and any transitive extension Y of X are not high on-shelf utility itemsets [16].

Then, to calculate $u(up(X),h)$ easily, FOSHU [16] separates the *iutil* value in utility-lists as two values *iputil* and *inutil*, respectively representing the sum of positive utilities and the sum of negative utilities of X in all transactions containing X. For a given transaction T_d, the *iputil* and *inutil* values are respectively defined as $u(up(X),T_d)$ and $u(un(X),T_d)$. Additionally, let the notation $sumIPUtil(X,h)$ denotes the sum of the *iputil* values of X in period h.

To prune the search space without missing any HOU, only utilities of positive items are used to calculate upper bounds on the utility of itemsets and their extensions. The pruning property 11 of utility-lists is thus rewritten as follows.

Property 16 Let X be an itemset. If there exists no time period h such that $(sumIPUtil(X,h) + sumRUtil(X,h))/pto(h) \geq minutil$, then X is not a high on-shelf utility itemset as well as any transitive extensions of X according to the total order $\succ$ [16].

4 Proposed Techniques For Mining The Top-k High On-shelf Utility Itemsets

One of the most important challenge in HOU mining is to design effective search space pruning techniques to avoid considering a large amount of candidates. In particular, top-k pattern mining algorithms all rely on an internal minimum threshold value initially set to 0, which is raised gradually during the search for itemsets, to prune the search space. It is thus a challenge to develop strategies for raising the threshold as quickly as possible while avoiding pruning HOUs.

This section address this challenge by proposing three novel pruning strategies to reduce the search space, and two strategies to raise the internal minimum threshold. Moreover, it introduces a procedure for quickly constructing utility-lists.

4.1 Search Space Pruning Strategies

This subsection describes novel pruning strategies and key structures for the problem of mining HOUs.

4.1.1 Estimated Maximum Period Rate Pruning Strategy

The first pruning strategy is named Estimated Maximum Period Rate Pruning Strategy (EMPRP). It is based on a novel structure called the Estimated Maximum Period Rate Structure (EMPRS), which is defined as follows.

Definition 11 The Estimated Maximum Period Rate Structure of a dataset D is denoted as $EMPRS_D$ and defined as $EMPRS_D = \{((a;b;m), m = max(RTWU(\{ab\},h)/pto(h)), \forall h \in pi(ab)) \in I^* \times I^* \times \mathbb{R}^+\}$, where I^* is the set of all items having a relative utility no less than *minutil* in D.

In the problem of top-k HOU mining, the *minutil* threshold is not set by the user. Instead, an internal *minutil* threshold is initialized to zero and increased during the search process by threshold raising strategies, which will be presented in section 4.2. Similarly to other one-phase algorithms for HOU mining, the proposed algorithm only scans the database twice to compute the utilities of each single item and of all 2-itemsets to construct the *EMPRS* structure. The first database scan is done to calculate the *RTWU* and the utility of each item. The *RTWU* values of items are then used to establish a total order on items from the transaction database, defined as the ascending order of *RTWU* values such that all negative items succeed positive items. The second database scan is performed to construct the *EMPRS*. During this database scan, items in transactions are reordered using the total order. For instance, consider the database D shown in Table 1. Items are reordered by ascending order of *RTWU* values such that all negative items succeed positive items. The *EMPRS* is implemented as a triangular matrix, as shown in Fig. 1.

Item Name	g	a	b	e	d	f	c
RTWU	24	71	74	75	113	73	76

Period	**Total utility**
1	36
2	28
3	52

Item	g	a	b	e	d	f
a	0.39					
b	0.39	0.92				
e	0	0.56	0.56			
d	0.39	0.86	0.56	1.06		
f	0.39	0.39	0.78	0.65	0.65	
c	0.39	0.39	0.93	0.4	0.67	0.42

Fig. 1: Items are reordered by ascending order of RTWU values such that negative items succeed positive items (top). The total utility of each time period (bottom-left). The EMPRS matrix (bottom-right).

Property 17 (pruning) Let X be an itemset, if $ru(X,h) < minutil$ then $\forall Y, ru(XY,h) < minutil$.

Proof By Property 2 we have that $TWU(XY,h) \leq TWU(X,h)$. Therefore $TWU(XY,h)/pto(h) \leq TWU(X,h)/pto(h)$. Furthermore, by Property 1, $u(XY,h) \leq TWU(XY,h)$. Thus, $ru(XY,h) < minutil$.

The EMPRP strategy employs Property 17 to immediately discard extensions of an itemset as follows. Let there be two items a and b. If there exists a tuple $\{(a;b;m)\}$ in the EMPRS where $m = max(RTWU(\{ab\},h)/pto(h)), \forall h \in pi(ab))$ such that $m < minutil$, then any superset $\{abX\}$ of $\{ab\}$, has a period relative utility $ru(\{abX\},h)$ lower than *minutil*. Therefore, the algorithm does not need to continue the search process with this pair of items according to Property 4. The purpose of this pruning strategy is to reduce the number of costly utility-list join operations.

Example 5 Consider the transaction database in Table 1 and $k = 5$. Assume that the search-space of all itemsets can be represented as a tree, as depicted in Fig. 2[1]. Each node X in this search tree is annotated with its relative utility $ru(X)$. Consider that the process of top-k HOU mining explores the search space using a depth-first search, that it is currently considering extensions of node b (its descendant nodes in the tree), and that *minutil* is equal to 0.44. The proposed algorithm can generate itemset bfc by combining nodes bf and bc. But by applying the EMPRP strategy, it is found that the maximum period rate of fc as in Fig. 1 is $0.42 < 0.44 = minutil$. Thus, the node bfc (and its subtree) is pruned.

4.1.2 Concurrence Existing Of A Pair 2-itemset Pruning Strategy

The second proposed search space pruning strategy is named Concurrence Existing of pair 2-itemset Pruning (CE2P). It is based on the following observation. In Table 1 and the EMPRS in Fig. 1, it can be observed that some pairs of items do not appear together in any transactions (e.g. $\{e,g\}$). To store information about which 2-itemsets appear together, the CE2P strategy utilizes a bit matrix. If two items appear concurrently at least once (in at least one transaction) then the corresponding bit for this 2-itemset is set to 1. Otherwise, it is set to 0. Then, the utility-list of any 2-itemsets will only be constructed if this bit value is 1. This pruning strategy is used to avoid performing numerous utility-list join operations. This strategy is especially effective when the number of items in a dataset is very large or the dataset is sparse.

[1] It should be noted that this tree representation is used for illustration purpose in this article. The proposed algorithm does not create an explicit tree structure in memory to explore the search space.

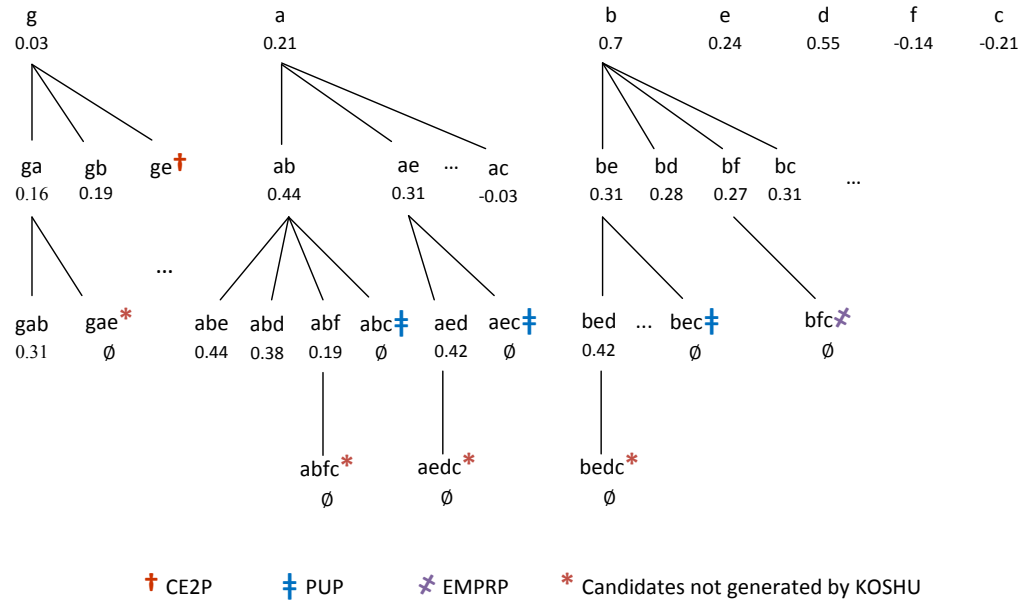


Fig. 2: A sample tree illustrating the different pruning strategies.

Example 6 Consider node *ge* in Fig. 2. It can be seen that these items do not concurrently exist in any transaction of the example database. Therefore, this node as well as its extensions are pruned.

4.1.3 Period Utility Pruning Strategy

The third proposed search space pruning strategy is named Period Utility Pruning (PUP). It is based on the following property.

Property 18 Let P be an itemset. Moreover, let Px and Py be extensions of P with some items x and y respectively such that $x \succ y$. If there exists no time period h such that $(sumIPUtil(x,h) + sumRUtil(x,h))/pto(h) \geq minutil$, where $h \in pi(Pxy)$, then all extensions of Px that contain y are not on-shelf high utility itemsets.

This property can be easily derived from Property 9 and Property 11. Let there be two itemsets X' and Y' such that $x \in X'$ and $y \in Y'$. If the condition of Property 18 holds, then three facts can be inferred: (i) $Pxy \notin$ top-k HOUs, (ii) $PxY' \notin$ top-k HOUs and (iii) $PX'Y' \notin$ top-k HOUs. Therefore, this pruning strategy can effectively reduce the number of utility-list join operations performed, and the search space.

Example 7 Consider node *abc*, which can be obtained by combining nodes *ab* and *ac*. Assume that at this time, $minutil = 0.44$. By applying the PUP strategy, it is found that $pi(abc) = 2$, and $(sumIPUtil(abc,2) + sumRUtil(abc,2))/pto(2) = 10/28 = 0.36 < 0.44 = minutil$. Therefore, the node *abc* and its descendants are pruned without constructing their utility-lists. For example, the node $abfc$ will not be generated because the node *abc* has been pruned.

4.2 Effective Threshold Raising Strategies

The proposed algorithm is designed to mine top-k on-shelf high utility itemsets. Therefore the optimal *minutil* value for obtaining k patterns is not known in advance. The proposed algorithm initializes an internal *minutil* threshold to zero. Then, it employs two effective threshold raising strategies to raise the *minutil* threshold. These two strategies are described next.

4.2.1 The Real 1-itemset Relative Utilities Threshold Raising Strategy

The first threshold raising strategy is named the Real 1-Itemset Relative Utilities threshold raising strategy (RIRU). It is inspired by the RIU strategy used in top-k HUI mining [34]. The RIRU strategy is performed after the first database scan. During the first database scan, the relative utilities of all 1-itemsets are calculated. The relative utility of a 1-itemset i in a transaction database D is denoted as $ru(i)$, and is calculated by the formula in Definition 5. The RIRU strategy utilizes the real relative utilities of 1-items to raise the value of *minutil* as follows. Let I={i_1, i_2, ..., i_m} be a set of items in a database D, and $R = \{ru_1, ru_2, \ldots, ru_m\}$ be the list of relative utilities of items in I in total order. Let ru_k denote the k-th highest value in R. Then the RIRU strategy increases the *minutil* threshold to the value $ru_k(1 \leq k \leq m)$. This new value is then used for the rest of the mining process until the threshold is increased to a greater value by another threshold raising strategy.

Table 3: Real item relative utility table

Item Name	g	a	b	e	d	f	c
Real relative utility	0.03	0.21	0.7	0.24	0.55	-0.14	-0.21

Example 8 Consider the database in Table 1. The real relative utilities of 1-itemsets are presented in Table 3. If $k = 5$, the fifth highest real relative utility is $ru(\{g\}) = 0.03$. Therefore, *minutil* is set to 0.03.

4.2.2 The Real 2-itemset Relative Utilities Threshold Raising Strategy

The second threshold raising strategy is named the Real 2-Itemset Relative Utilities threshold raising strategy (RIRU2). This strategy is applied after the RIRU strategy and the second database scan. The relative utility of a 2-itemset xy is calculated by the equation $ru(xy) = u(xy)/|to(xy)|$ as in Definition 5. The values $to(xy)$ and $u(xy)$ are respectively calculated during the first and second database scans. The RIRU2 strategy raises the internal minimum threshold to the k-th largest value by considering 1-items obtained after applying the RIRU strategy, in addition to the relative utilities of 2-itemsets. The RIRU2 strategy is performed in the same way as RIRU, presented in the previous subsection.

Example 9 Consider the database in Table 1, assume that $k = 5$, and that $minutil = 0.03$ after applying RIRU as in Example 8. The real relative utilities of 1-items and 2-itemsets are: {b}:0.7, {d}:0.55, {e}:0.24, {a}:0.21, {g}:0.03, {ga}:0.16, {gb}:0.19, {gd}:0.18, {ab}:0.44, {ae}:0.31, {ad}:0.35, {be}:0.31, {bd}:0.28, {bf}:0.27, {bc}:0.31, {ed}:0.74, {ec}: 0.12, {df}:0.28, {dc}:0.15. The fifth highest relative utility among these values is $ru(\{ad\}) = 0.35$. Thus, *minutil* is set to 0.35 by the RIRU2 strategy.

An important consideration is whether to build the EMPRS structure before of after applying RIRU2. If the EMPRS is built after RIRU but before RIRU2, this latter could use the EMPRP strategy to eliminate some 2-itemsets. However, it is better to apply RIRU2 before building the EMPRS because the EMPRS is built only once and then it is employed during the whole mining process. Thus, it is preferable to raise the *minutil* threshold as high as possible before constructing this structure. Hence, building the EMPRS structure is done after applying the RIRU and RIRU2 strategies, in the proposed algorithm.

4.3 An Efficient Utility-list Construction Method

In utility-list based algorithms such as HUI-Miner [30], FHM [15], TKO [41] and the proposed algorithm, a key operation is the construction of utility-lists. In this subsection, we propose an improved utility-list construction procedure that has a lower complexity than the ones used in previous algorithms. This optimization greatly reduces the runtime of the proposed algorithm. The utility-list structure is used to maintain information about the utility of itemsets. The utility of an itemset can be quickly constructed by making join operations with utility-lists of smaller itemsets. Definition and properties of this structure have been given in subsection 3.2. As proposed in the HUI-Miner algorithm [30], the utility-lists of larger

Algorithm 1 The standard utility-list construction procedure

Input:
$P.UL$: the utility-list of itemset P;
$Px.UL$: the utility-list of itemset Px;
$Py.UL$: the utility-list of itemset Py;
Output:
$Pxy.UL$: the utility-list of itemset Pxy;

```
1: Pxy.UL = NULL;
2: for each (tuple ex ∈ Px.UL) do
3:    if (∃ey ∈ Py.UL and ex.tid==ey.tid) then
4:       if (P.UL is not empty) then
5:          Search element e ∈ P.UL such that e.tid = ex.tid;
6:          exy ⟵ (ex.tid; ex.iutil + ey.iutil - e.iutil; ey.rutil);
7:       else
8:          exy ⟵ (ex.tid; ex.iutil + ey.iutil; ey.rutil);
9:       end if
10:      Pxy.UL ⟵ Pxy.UL ∪ exy;
11:   end if
12: end for
13: return Pxy.UL;
```

itemsets can be obtained by intersecting utility-lists of smaller itemsets. For example, let P, Px and Py be itemsets, such that Px and Py are extensions of P with items x and y, respectively. The utility-list of itemset Pxy is obtained by applying Algorithm 1. For each element in $ul(x)$, the algorithm checks whether the element exists in $ul(y)$ or not, if yes then a binary search is performed in the utility-list of P. Therefore, the complexity of the procedure is $O(mlogn)$, where m and n are respectively the number of entries in $ul(x)$ and $ul(y)$.

Because Tids in utility-lists are ordered by increasing Tid values, a better way for identifying transactions that are common to two utility-lists $ul(x)$ and $ul(y)$ is to read the two utility-lists at the same time by reading Tids sequentially in each utility-list. The complexity of this searching method is $O(m+n) < O(mlogn)$. Note that if *ul(P)* is not empty, the Tid list in *ul(Pxy)* is a subset of the Tid list in *ul(P)*. Furthermore, to speed up the binary search in the utility-list of P, $ul(P)$, the proposed construction procedure remembers the last *index* considered by the binary search. Then, the next binary search in $ul(P)$ is started from that *index* position instead of from the first element at index 0. Thus, an intersection procedure for constructing utility-lists, having a complexity of $O(m+n)$, has been introduced in this subsection. The pseudo-code is presented in Algorithm 2.

Algorithm 2 The proposed *i*Construct utility-list construction procedure

Input:
$P.UL$: the utility-list of itemset P;
$Px.UL$: the utility-list of itemset Px;
$Py.UL$: the utility-list of itemset Py;
Output:
$Pxy.UL$: the utility-list of itemset Pxy;

```
1: Pxy.UL = NULL; Let i, j = 0;
2: while (i < Px.UL.size and j < Py.UL.size) do
3:    if ( Px.UL[i].tid < Py.UL[j].tid) then
4:       i++;
5:    else if ( Px.UL[i].tid > Py.UL[j].tid) then
6:       j++;
7:    else
8:       if (P.UL is not empty) then
9:          Search element e ∈ P.UL such that e.tid = ex.tid using the improved binary search method;
10:         exy ⟵ (ex.tid; ex.iutil + ey.iutil - e.iutil; ey.rutil);
11:      else
12:         exy ⟵ (ex.tid; ex.iutil + ey.iutil; ey.rutil);
13:      end if
14:      Pxy.UL ⟵ Pxy.UL ∪ exy;
15:      i++;
16:      j++;
17:   end if
18: end while
19: return Pxy.UL;
```

4.4 Exploring The Most Promising Branches First Strategy

Lastly, another strategy named Exploring the most Promising Branches first (EPB) is adopted in the proposed algorithm. EPB was first proposed [41] for top-k HUI mining. The idea of EPB is to always try to extend itemsets having the largest estimated utility value first, based on the hypothesis that it is more likely to generate itemsets having higher utilities. Thus, the threshold can be raised more quickly for pruning the search space. For the problem of top-k on-shelf high utility itemset mining, this strategy is adapted to always explore the candidate itemset having the highest relative utility first. The reason is that if itemsets with high relative utilities are found earlier, the proposed KOSHU algorithm can raise its *minutil* threshold more quickly, to prune the search space.

5 The Proposed Algorithm

This section presents the proposed KOSHU algorithm for mining the top-k on-shelf high utility itemsets. KOSHU employs the novel EMPRS structure to prune the search space using the EMPRP strategy. This strategy is used to avoid performing numerous join operations and the construction of several utility-lists. Besides the EMPRP strategy, two other search space pruning strategies are used, namely the CE2P and PUP strategies. Furthermore, two threshold raising strategies are applied to raise the *minutil* threshold, which are the RIRU and RIRU2 strategies. This threshold is also dynamically raised by the KOSHU algorithm when the exact relative utilities of potential candidates are calculated. To raise the *minutil* threshold higher and earlier, KOSHU also employs the EPB strategy. In addition, a novel procedure that constructs utility-lists with a complexity of $O(m+n)$ is integrated in the proposed algorithm. These techniques are described in the previous section. The main procedure of KOSHU is shown in Algorithm 3.

Algorithm 3 The KOSHU algorithm

Input:
D: a transaction database;
k : the desired number of patterns;
Output:
The top-k on-shelf high utility itemsets;

1: Initialize global variables: $minutil \longleftarrow 0$; $R_{topk} \longleftarrow \varnothing$;
2: Scan D to:
3: *1.* Calculate $RTWU(\{i\})$ and $pi(\{i\})$ of each item i;
4: *3.* Calculate the list of all time periods PE;
5: *2.* Calculate the utility $pto(h)$ of each period $h \in PE$;
6: *4.* Calculate the real relative utility of each item i (required by the RIRU strategy);
7: Let I be the list of single items sorted by ascending order of $RTWU$ and negative items succeed positive items;
8: $minutil \longleftarrow$ the k-th highest relative utility value in $RIRU$; //RIRU strategy
9: Scan D to build the utility-list of each item $i \in I$;
10: $minutil \longleftarrow$ the k-th highest relative utility value in $RIRU2$; //RIRU2 strategy
11: Let $I^* = \{i | \exists h \in PE \wedge RTWU(\{i,h\})/pto(h) \geq minutil\}$;
12: Build the *EMPRS* structure;
13: **Search** ($\emptyset$, I^*, $\emptyset$, *EMPRS*);
14: Output the k itemsets having the highest relative utilities in R_{topk};

The main procedure of KOSHU is performed as follows. The value of *minutil* threshold is set to 0 and the set of top-k candidates R_{topk} is initialized to empty set (line 1). Then KOSHU scans the database the first time to obtain the $RTWU(\{i\})$ and $pi(\{i\})$ of each item i, where $RTWU(\{i\}) = \sum_{h \in pi(\{i\})} RTWU(\{i\}, h)$. The set of all time periods PE and the utility $pto(h)$ of each period $h \in PE$ are computed. Moreover, the real relative utilities ($RIRU$) of single items are calculated during this database scan (lines 2-6). The algorithm then sorts the list of single items in I by ascending order of RTWU such that negative items succeed all positive items (line 7), and raises the value of *minutil* threshold using the RIRU strategy (line 8). KOSHU then scans the database again to build the utility-list of each single item (line 9). The threshold raising strategy RIRU2 is then applied (line 10). Each item that does not satisfy Property 4 is discarded (line 11) before building the EMPRS structure (line 12). Then, the *Search* procedure is called to search high on-shelf utility candidates, which are then inserted in the set of top-k candidates R_{topk} (line 13). At the end, the real top-k on-shelf high utility itemsets in R_{topk} are output (line 14).

Algorithm 4 The Search Procedure

Input:
$ul(P)$: the utility-list of an itemset P;
$Class[P]$: a set of itemsets that are single item extensions of P;
$ULS[P]$: a set of utility-lists with respect to the itemset P;
$EMPRS$: the $EMPRS$ structure;
Output:
The set of top-k on-shelf high utility itemset candidates;

1: **for each** itemset $Px \in Class[P]$ **do**
2: **if** $(sumIUtil(Px)/|to(Px)| \geq minutil)$ **then**
3: $R_{topk} \longleftarrow R_{topk} \cup Px$;
4: **if** $(R_{topk}.size >= k)$ **then**
5: $minutil \leftarrow$ the k-highest utility in R_{topk} ;
6: Keep only the k itemsets having the highest utilities in R_{topk};
7: **end if**
8: **end if**
9: **if** $\exists h \in pi(Px)$ such that $(sumIPUtil(Px,h) + sumRUtil(Px,h))/pto(h) \geq minutil$ **then**
10: $Class[Px] \leftarrow \varnothing$;
11: $ULS[Px] \leftarrow \varnothing$;
12: **for each** itemset $Py \in Class[P]$ and $y \succ x$ **do**
13: **if** $concurrent\_bit(x,y) = 0$ **then**
14: continue;// CE2P strategy
15: **end if**
16: **if** $(EMPRS(x,\ y) < minutil)$ **then**
17: continue; // EMPRP strategy
18: **end if**
19: **if** $(\nexists h$ such that $(sumIPUtil(Px,h) + sumRUtil(Px,h))/pto(h) \geq minutil,\ h \in pi(Pxy)\ )$ **then**
20: continue; // PUP strategy
21: **end if**
22: $Pxy \longleftarrow Px \cup Py$;
23: $ul(Pxy) \longleftarrow$ ***i*Construct**(P, Px, Py);
24: **if** $(\exists h \subset PE$ such that $RTWU(Pxy,h)/pto(h) \geq minutil)$ **then**
25: $Class[Px] \leftarrow Class[Px] \cup Pxy$;
26: $ULS[Px] \leftarrow ULS[Px] \cup ul(Pxy)$;
27: **end if**
28: **end for**
29: **Search** $(Px,\ Class[Px],\ ULS[Px],\ EMPRS)$;
30: **end if**
31: **end for**

The *Search* procedure (Algorithm 4) is applied as follows. For each itemset Px in the set of extensions of P, the *Search* procedure first scans the utility-list of Px to calculate $ru(Px) = sumIUtil(Px)/|to(Px)|$. If $ru(Px)$ is no less than $minutil$, then Px is an on-shelf high utility itemset and it is added to the set of top-k candidates R_{topk}. Then, $minutil$ is raised to the k-highest value in R_{topk} if there are k entries in R_{topk} (lines 2-8). Then, if there exists a time period $h \in pi(Px)$ such that $(sumIUtil(Px,h) + sumRUtil(Px,h))/pto(h)$ is no less than $minutil$, by Property 11, it means that extensions of Px should be explored (line 9). Before merging Px with all extensions Py of P such that $y \succ x$ to form larger extensions of the form Pxy, pruning conditions are checked: (i) if x and y concurrently exist (lines 13-15); (ii) if there is a tuple in the EMPRS for x and y such that this tuple value is greater than $minutil$ (lines 16-18); (iii) if the PUP condition is respected (lines 19-21). If these three conditions are passed, the utility-list of Pxy is constructed by calling the *iConstruct* procedure (cf. Algorithm 2) (line 26). Then, a check is performed to determine if Pxy and its extensions may be on-shelf high utility itemsets by using the RTWU measure based on Property 4 (line 27). If Pxy is a promising itemset, it will be added to a priority queue for further exploration (lines 28-29). Note that entries in this queue are sorted by descending order of their estimated relative utility value. Then, the *Search* procedure is recursively called with the top entry Px in that queue. The procedure stops when the queue is empty.

The Search procedure starts from single items, and recursively explores the search space of itemsets by appending single items. By the proposed properties, it can be easily seen that the algorithm is correct and complete to discover top-k on-shelf high utility itemsets.

6 Performance Study

To evaluate the performance of the proposed algorithm, a series of experiments have been conducted. Extensive experiments have also been performed to evaluate the proposed strategies and methods. All algorithms were implemented by extending the SPMF open-source java library [13] using the J2SDK

1.7.0. The experiments were executed on a computer equipped with an Intel core i3 processor 2.4 GHz and 4 GB of RAM, running Windows 7 as operating system.

6.1 Experimental Design

Both real and synthetic datasets having varied characteristics were used in the experiments. They are standard datasets used in the HUIM literature for evaluating HUIM algorithms. The characteristics of these datasets are described in Table 4, where #Transactions, #Distinct items and Avg. trans. length indicate the number of transactions, the number of distinct items and the average transaction length, respectively.

Table 4: Characteristics of the datasets.

Dataset	#Transactions	#Distinct items	Avg. trans. length
BMS-POS	515,597	1657	6.5
Chainstore	1,112,949	46,086	7.2
Chess	3196	75	37
Foodmart	4141	1559	4.4
Mushroom	8124	119	23.0
Retail	88,162	16,470	10.3
T10I4D500K	500,000	870	10

For these experiments, we prepared a synthetic dataset named T10I4D500K, which was generated using the IBM Quest synthetic data generator[2], where the numbers after T, I, and D represent the average transaction size, average size of maximal potentially frequent patterns, and the number of transactions, respectively. We also used two real-world customer transaction datasets named Chainstore[3] and Foodmart[4]. Chainstore is a very large dataset consisting of transaction data from a Californian retail store, while Foodmart is a small dataset of customer transactions obtained from the Microsoft Food-Mart 2000 database. These two datasets already contain real unit profits and purchase quantities.

The remaining four datasets are also real datasets. The BMS-POS[5] dataset is a large dataset, which was used in the KDD CUP 2000. It contains several years worth of point-of-sale data from a large electronics retailer. Retail[6] is a sparse dataset containing customer transactions from a Belgian retail store. Lastly, two dense datasets were used named Chess[7] and Mushroom[8]. Although these two datasets are not retail data, they are often used in the pattern mining literature as benchmark datasets to evaluate the performance on dense data. Chess is especially a quite challenging dataset for most mining algorithms because it contains many long itemsets.

For all datasets, external utilities for items were generated randomly in the range of -1000 to 1000, and quantities of items were generated randomly in the [1, 5] interval, similar to the settings of [24, 23, 16, 30], except for the Foodmart and Chainstore datasets because these datasets already contain information about purchase quantities and unit profits of items. The number of time periods was set to 5 as in [23, 16].

The datasets used in the experiments have been selected because they contain real customer transaction data (Chainstore, Foodmart and Retail) or because they are typical itemset benchmark datasets representing different types of data in terms of density, number of items and transaction length (BMS-POS, Chess, Mushroom and T10I4D500K). Using different types of data allows to see the performance of the algorithm in various situations.

[2] http://www.almaden.ibm.com/cs/quest/syndata.html
[3] http://cucis.ece.northwestern.edu/projects/DMS/MineBenchDownload.html
[4] https://www.microsoft.com/en-us/download/details.aspx?id=51958
[5] http://www.kdd.org/kdd-cup/view/kdd-cup-2000
[6] http://fimi.cs.helsinki.fi/data/
[7] http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php
[8] http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php

6.2 The Efficiency Evaluation Of KOSHU

In this subsection, the performance of the KOSHU algorithm is evaluated. Because KOSHU is the first algorithm for mining the top-k HOUs, it is not possible to compare its performance with another algorithm for the same problem. In general, the standard way of evaluating a top-k algorithm for a new problem is to compare its performance with the corresponding non top-k algorithms, set with optimal threshold values [17,6,42,36,44,34,41]. In the case of KOSHU, the corresponding non top-k algorithms are TS-HOUN [23] and FOSHU [16], for on-shelf high utility itemset mining with negative values. It has already been established in [15] that FOSHU can be more than 1000 times faster than TS-HOUN.

Because TS-HOUN and FOSHU were not designed for mining top-k HOUs, they cannot be directly compared with the proposed KOSHU algorithm. As it is typically done for evaluating top-k algorithms for new problems, we have thus considered the scenario where users would choose the optimal minimum utility threshold for FOSHU and TS-HOUN (denoted as FOSHU_Opt and TS-HOUN_Opt) to produce the same amount of patterns as KOSHU.

However, it is important to note that performing this type of comparison between top-k and non top-k algorithms is not fair for the designed top-k algorithm because a top-k problem is always more difficult than the corresponding non top-k problem [17,6,42,36,44,34,41,11]. The reason is that the top-k algorithm has to start from $minutil = 0$, and then to gradually increase its internal threshold until it finds the optimal threshold value to find k patterns, while the non top-k algorithms are directly set with the optimal value, and can thus ignore a large part of the search space. Thus, it is important to understand that in general one should not expect a top-k algorithm to perform better than a non top-k algorithm when this latter is set with an optimal threshold [17,6,42,36,44,34,41]. It should be clear that the goal of comparing a top-k algorithm with a standard algorithm is rather to see how close the top-k algorithm's performance can be to that of a non top-k algorithm [17,6,42,36,44,34,41]. In fact, a user of a non top-k algorithm typically has to run the algorithm many times using a trial and error process to find k patterns, which can be very time-consuming [17,6,42,36,44,34,41,11]. In this experiment, we do not consider the cost of running these non top-k algorithms several times to obtain the optimal threshold. We assume that they are directly set with the optimal threshold, which gives an advantage to the non top-k algorithms.

Thus, we ran KOSHU on each dataset using various values of k. We then ran TS-HOUN and FOSHU on each dataset with *minutil* equal to the smallest relative utility among the k itemsets found by KOSHU. The runtime results are presented in Figs. 3 - 9 where FOSHU_Opt and TS-HOUN_Opt mean that FOSHU and TS-HOUN were applied with the optimal minimum threshold, respectively. Runtimes include the time for reading the input file, discovering the patterns, and writing results to an output file. It should be noted that we also compared the performance of KOSHU with the TS-HOUN algorithm when it is set with the optimal minimum utility threshold. However, TS-HOUN failed to terminate within 3 hours on most datasets. Thus, some results are missing for the TS-HOUN algorithm.

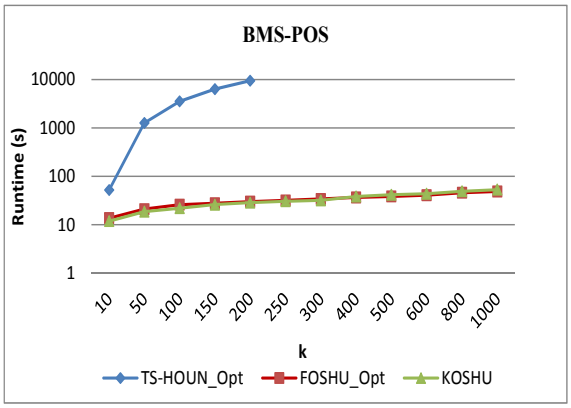


Fig. 3: The runtime on BMS-POS.

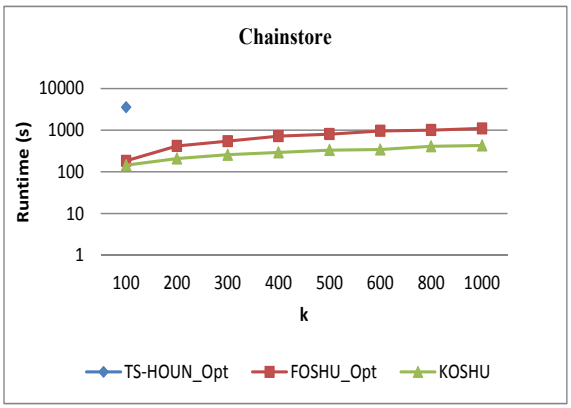


Fig. 4: The runtime on Chainstore.

Fig. 3 compares the runtimes of the compared algorithms on the BMS-POS dataset. It can be observed that when k is set to values less than 400, KOSHU is faster than FOSHU_Opt. When k is set to larger values, KOSHU is slightly slower than FOSHU_Opt. However, there is not a big difference

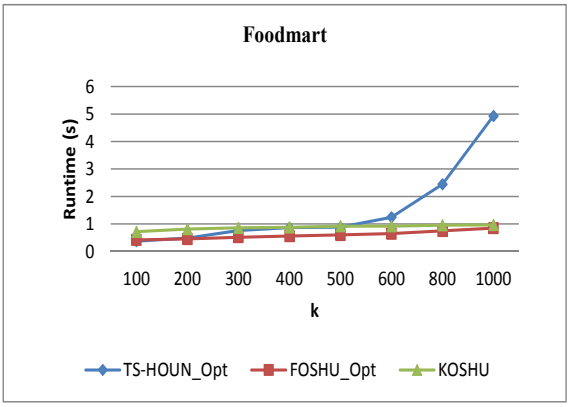


Fig. 5: The runtime on Foodmart.

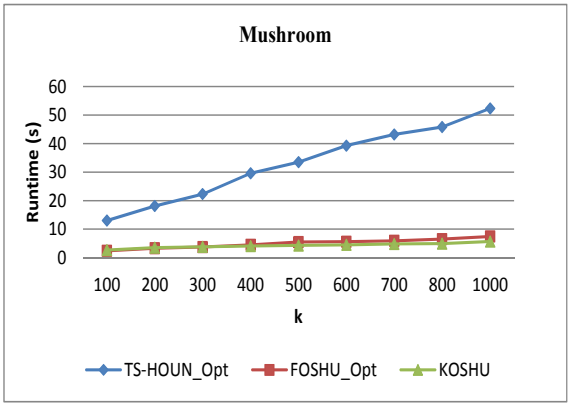


Fig. 6: The runtime on Mushroom.

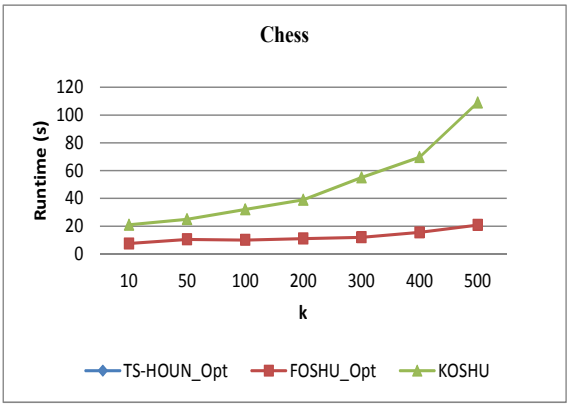


Fig. 7: The runtime on Chess.

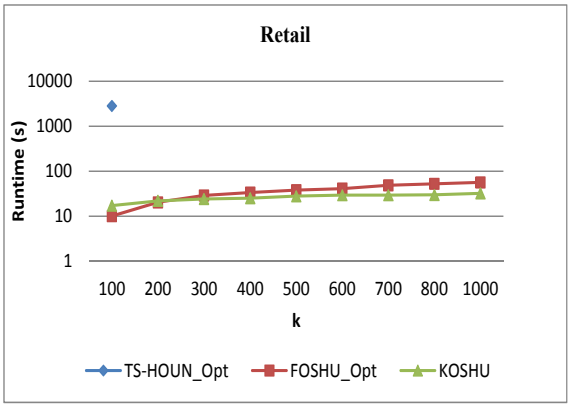


Fig. 8: The runtime on Retail.

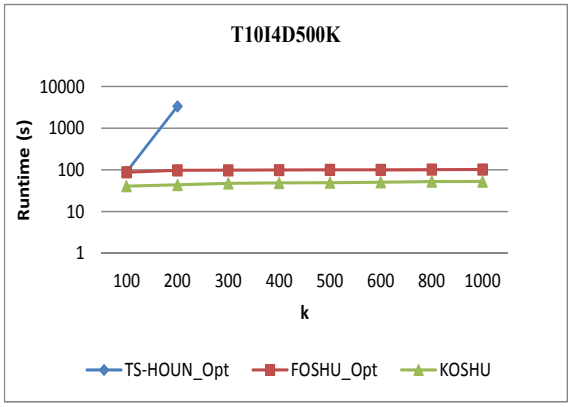


Fig. 9: The runtime on T10I4D500K.

between their runtimes. This is considered as an excellent result, since as previously mentionned, a top-k algorithm should not be expected to be faster than a non top-k algorithm. On BMS-POS, TS-HOUN_Opt failed to terminate within three hours when k was set to values larger than 200 and it took an enormous amount of time for the other values of k. Fig. 4 shows the execution times for the Chainstore dataset. TS-HOUN_Opt only terminated in less than three hours for $k = 100$. KOSHU is the fastest for all values of k. KOSHU and FOSHU_Opt have almost the same performance on the Foodmart dataset (Fig. 5). For the Mushroom and Retail datasets (Figs. 6 and 8), with small values of k, KOSHU is slower than FOSHU_Opt. However, for the remaining values of k, KOSHU has better performance.

The runtimes of the algorithms on the Chess dataset are shown in Fig. 7. Chess is a dataset containing long patterns and highly similar transactions. Therefore, the number of candidates for top-k mining on this dataset is huge. For this reason, there is a large gap between the compared algorithms. On this dataset, TS-HOUN_Opt failed to terminate within three hours even for the smallest values of k. Fig. 9 shows the runtimes of all algorithms on the T10I4D500K dataset. In this figure, KOSHU has the best performance for all values of k, and there is a big gap between KOSHU and FOSHU_Opt. The TS-HOUN_Opt algorithm failed to terminate within three hours for $k > 200$ and it took a considerable amount of time for $k \leq 200$. It is clear that the proposed approach has excellent performance as it always outperforms the TS-HOUN algorithm. Moreover, it is very close to or faster than the FOSHU algorithm on six out of seven datasets, when those algorithms are set with an optimal threshold.

On overall, these results are excellent because the minimal relative utility of FOSHU_Opt was chosen optimally. But in real-life, users rarely choose an optimal value for this threshold as it requires extensive background knowledge about the dataset. Generally, users find an appropriate value for the minimum utility threshold by trial and error, which is very time-consuming. If the minimal relative utility threshold of FOSHU is not chosen optimally, it can be much slower than KOSHU. In these experiments, the time that would be required for running the TS-HOUN and FOSHU algorithms several times by trial and error to obtain the same number of patterns as KOSHU has not been considered in the runtimes.

Table 5: Comparison of maximum memory usage (MB).

Dataset	TS-HOUN_Opt	FOSHU_Opt	KOSHU
BMS-POS	-	689	1410
Chainstore	-	1443.5	2553.3
Chess	-	812	1788
Foodmart	33.8	33	19
Mushroom	541.2	492	389
Retail	-	481	575
T10I4D500K	-	791	916

Table 5 shows the peak memory consumptions of the compared algorithms on six datasets for the largest value of k. All memory measurements were done using the standard Java API. Note that the memory usage of TS-HOUN_Opt is only shown for the Foodmart and Mushroom datasets because for the other datasets, TS-HOUN_Opt failed to terminate whithin three hours for all values of k. By consulting this table, it can be found that KOSHU consumes more memory than FOSHU_Opt on the BMS-POS, Chainstore, Chess and T10I4D500K datasets while their memory usage is similar for the Retail and Mushroom datasets when k is large. For the Foodmart dataset, KOSHU consumes almost the same amount of memory for all values of k and it uses less memory than FOSHU_Opt when k is set to values larger than 400. FOSHU and KOSHU are single phase algorithms that do not need to maintain candidate high on-shelf utility itemsets in memory. However, KOSHU needs to store other structures for top-k HOU mining such as the EMPRS structure, and the current top-k patterns found until now. In general, it is normal that a top-k algorithm uses more memory than a non top-k algorithm because a non top-k algorithm does not need to keep track of the current top-k patterns found until now, and does not need to keep additional structures in memory related to dynamically adjusting the threshold [17,6,42, 36,44,34,41,11]. Hence, considering this, it can be concluded that KOSHU has reasonable performance in terms of memory usage.

6.3 Pruning Effectiveness Analysis

The influence of the two main pruning strategies EMPRP and PUP was also evaluated. For each dataset and various k values, the pruning strategies were individually applied. In utility-list based algorithms, it is well-known that the join operation to construct utility-lists is the costliest operation. Therefore, we evaluate the proposed pruning strategies in terms of percentage of join operations that are avoided. Results on the studied datasets are presented in Table 6. In general, the PUP strategy provides a greater reduction than EMPRP. These results show that candidate pruning can be very effective. Up to 93% of candidates are pruned by PUP and up to 89% by EMPRP.

Figs. 10 - 13 illustrate the effect of individual pruning strategies on the overall execution performance of KOSHU. The relative performance of EMPRP and PUP was found to vary from one dataset to

Table 6: Reduction of join operations by the EMPRS and PUP strategies as percentages

Dataset	Pruning Strategy	k							
		100	200	300	400	500	600	800	1000
BMS-POS	EMPRP	2	4	6	7	8	8	10	11
	PUP	93	92	91	91	91	90	90	89
Foodmart	EMPRP	88	87	86	85	85	85	87	89
	PUP	88	85	83	82	81	82	84	72
Mushroom	EMPRP	26	30	32	32	34	34	33	32
	PUP	63	64	63	60	60	58	55	51
Retail	EMPRP	8	16	22	29	34	37	43	49
	PUP	71	77	79	83	85	86	89	91

another. However, the overall best performance was observed when combining pruning strategies on all the datasets. The above analysis confirms the effectiveness of the two proposed pruning strategies.

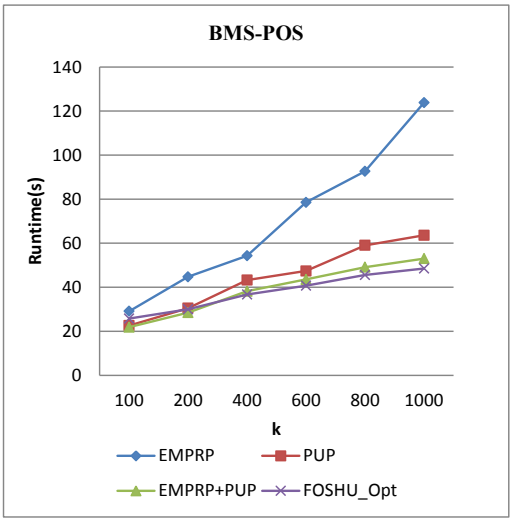


Fig. 10: Effect of pruning strategies on BMS-POS.

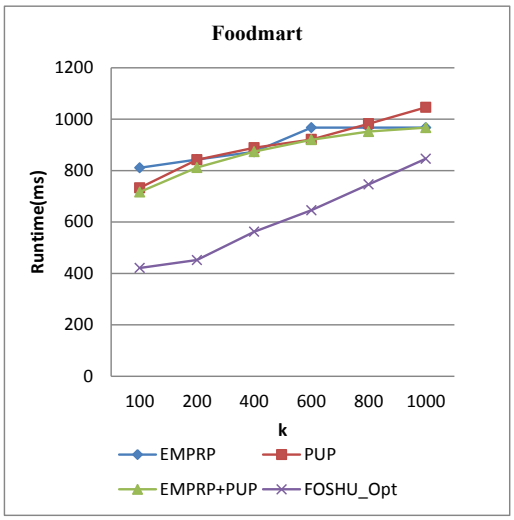


Fig. 11: Effect of pruning strategies on Foodmart.

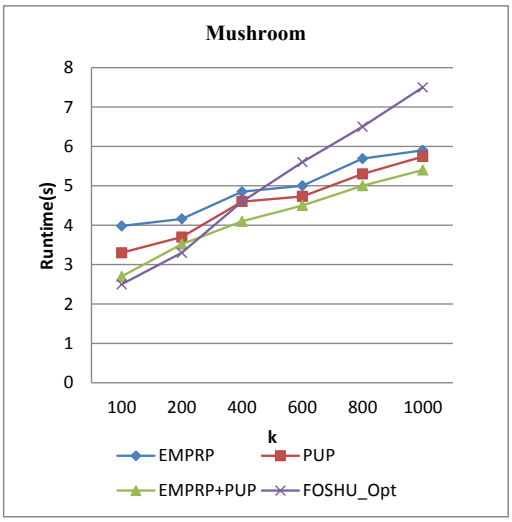


Fig. 12: Effect of pruning strategies on Mushroom.

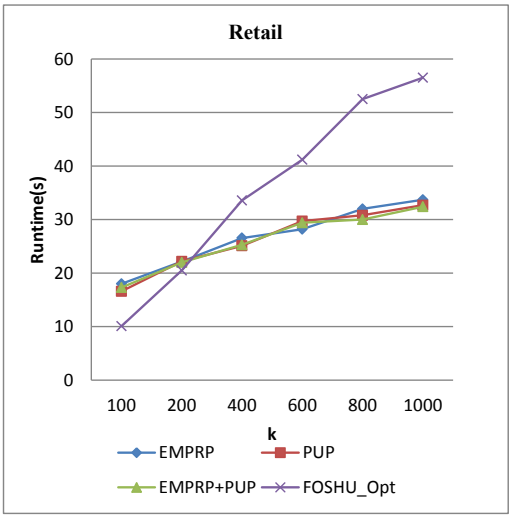


Fig. 13: Effect of pruning strategies on Retail.

6.4 Efficiency Of The Proposed Utility-list Construction Method

The effectiveness of the proposed method for constructing utility-lists was also assessed. As mentioned, the utility-list of an l-itemset such that $l > 1$ can be obtained by intersecting utility-lists of smaller itemsets. This process is the most expensive operation performed by utility-list based algorithms. Hence, an experiment was carried out to compare the efficiency of the improved utility-list construction procedure with the traditional construction procedure. The comparison of execution times is shown in Fig. 14 for the studied datasets. In this figure, KOSHU_iConstruct denotes KOSHU using the proposed utility-list construction method while KOSHU_construct represents KOSHU employing the traditional construction method. As we can see in this figure, the improved method is very efficient. It is faster than the traditional method, specifically it can reduce execution time by up to 41%.

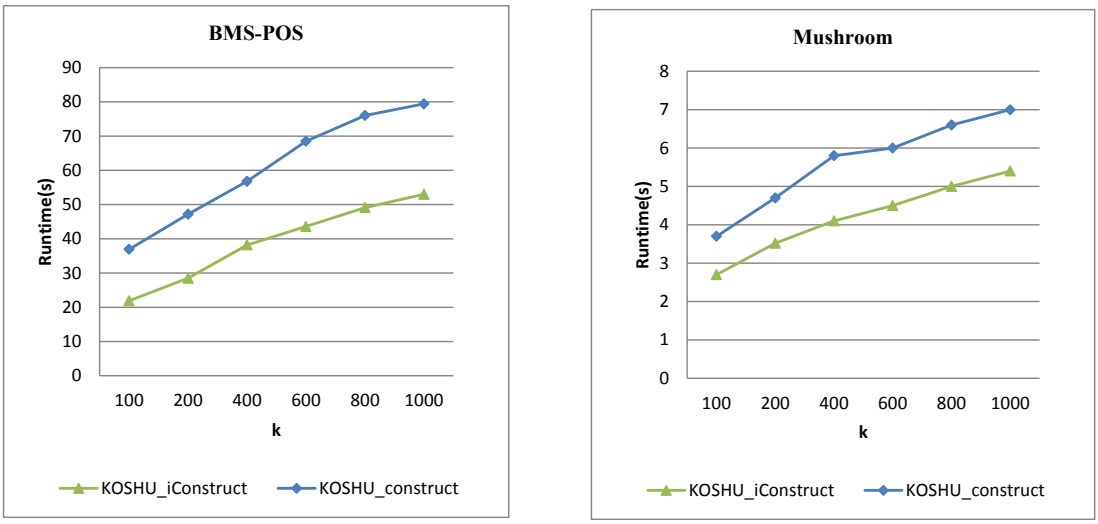


Fig. 14: Effect of using the iConstruct utility-list construction procedure.

6.5 Influence Of The Number Of Time Periods

In the previous experiments, the number of time periods was fixed. In this experiment, the number of time periods was varied to assess the influence of the number of time periods on the execution time of KOSHU. Transactions were randomly grouped into 5, 25 and 50 time periods. Figs. 15 and 16 show the results of this experiment on the studied datasets. In this experiment, TS-HOUN_Opt failed to terminate within 3 hours on the Mushroom dataset for k >200 when the number of time periods is 50. It can be observed that KOSHU has excellent scalability with respect to the number of time periods. For all datasets, the runtime remains almost the same when the number of time periods is set to 5, 25 and 50 time periods. In general, KOSHU is only a little slower when a dataset has more time periods. The reason why KOSHU has excellent scalability with respect to the number of time periods is that it mines all time periods at the same time.

6.6 Scalability Of KOSHU Under Different Parameter Settings

We also performed experiments to assess the scalability of the proposed algorithm with respect to the number of transactions and the number of items. Synthetic datasets have been generated using the IBM Quest dataset generator [2]. For this experiment, k was set to 5000, the number of distinct items was varied from 2K to 10K items and the database size was varied from 100K to 500K transactions. Fig. 17 shows the runtimes of the compared algorithms on the T10I4NXKD100K and T10I4N4KDXK datasets, and Fig. 18 compares the memory usage of the algorithms on these same datasets. Note that TS-HOUN failed to terminate whithin three hours or run out of memory in these scalability tests. By consulting Fig. 17, it can be concluded that the proposed algorithm has linear scalability with respect

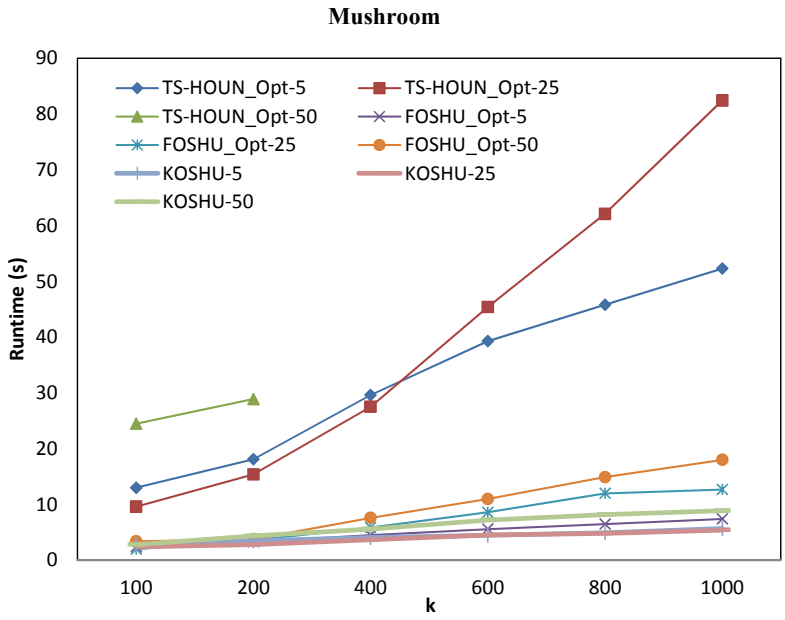


Fig. 15: Execution time w.r.t. time period count on Mushroom.

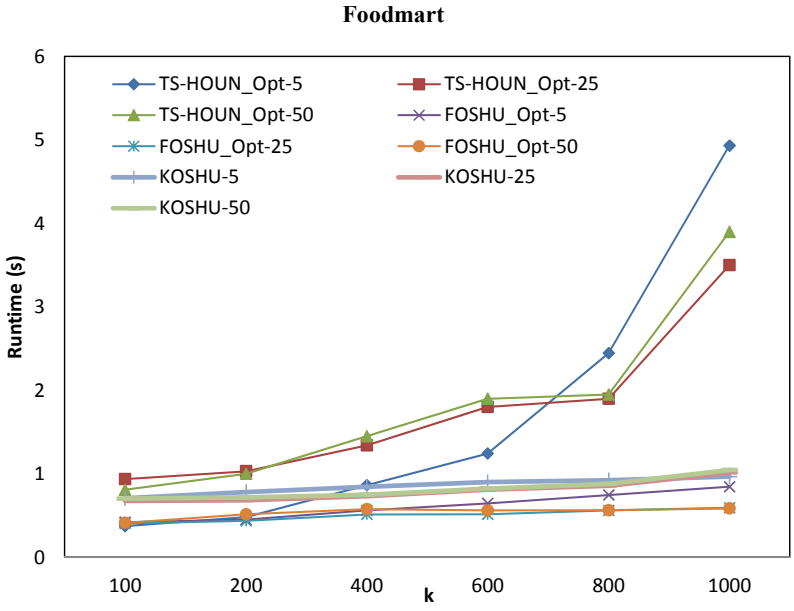
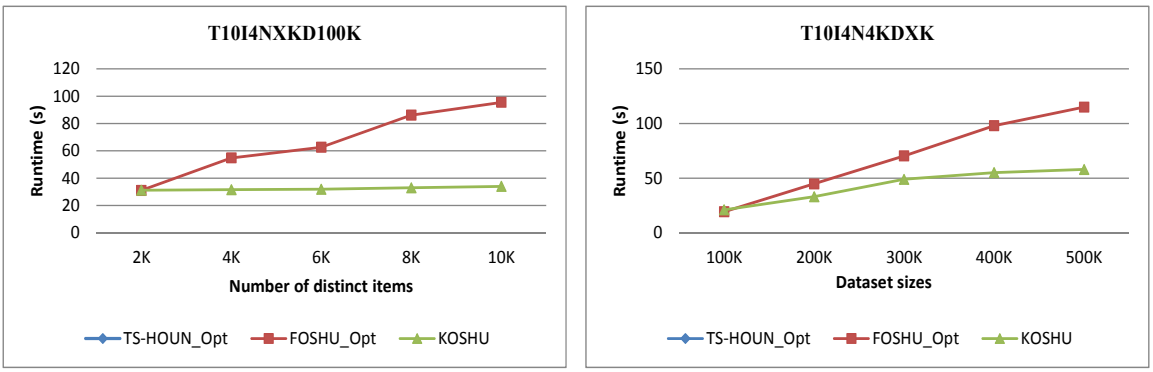


Fig. 16: Execution time w.r.t. time period count on Foodmart.

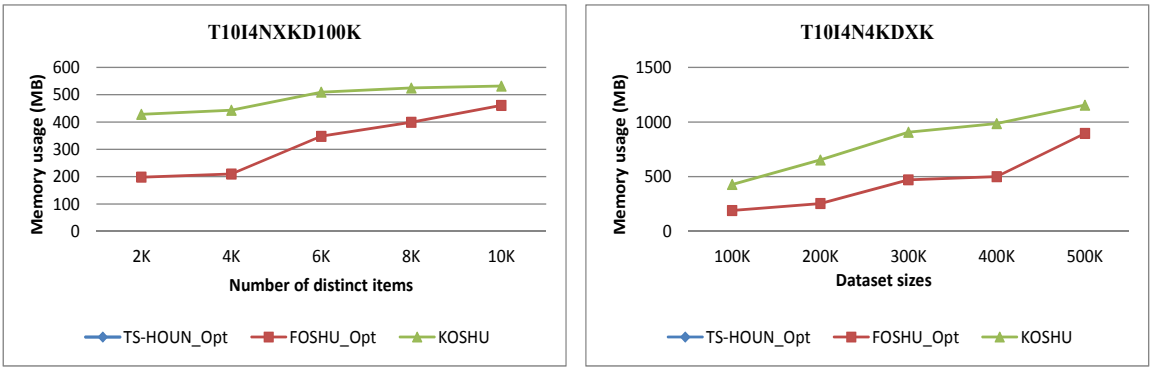
to the number of items and good scalability with respect to the number of transactions. On Fig. 18, we observe that the memory consumption of KOSHU increases almost linearly when the number of items increases and that it increases slowly when the number of transactions increases. These results indicate that KOSHU scales well under different parameter settings.

From the above performance studies, the experimental results suggest that the proposed techniques are effective, and the proposed algorithm is highly efficient in terms of execution time and memory usage as well as has excellent scalability.



(a) Varied number of items (b) Varied dataset sizes

Fig. 17: Execution times of KOSHU under different parameter settings



(a) Varied number of items (b) Varied dataset sizes

Fig. 18: Memory usage of KOSHU under different parameter settings

7 Conclusion

This work has studied the new research issue of top-k on-shelf high utility itemset mining where negative profit values are allowed, shelf time of items are considered, and where the user can specify the number of patterns to be found k instead of the minimum utility threshold.

To discover the top-k high on-shelf utility itemset efficiently, a novel single phase algorithm named KOSHU was proposed. By using utility-lists, KOSHU does not maintain candidates in memory and mines HOUs in all time periods at the same time. KOSHU also incorporates three novel strategies to reduce the number of join operations when mining on-shelf high utility itemsets using the utility-list data structure, namely EMPRP, CE2P and PUP. A novel intersection procedure for intersecting utility-lists with complexity $O(m + n)$ was also introduced.

An extensive experimental evaluation was conducted to evaluate the proposed algorithm KOSHU and the proposed strategies. Results have shown that KOSHU is highly efficient, as its performance is always very close to or better than FOSHU, when FOSHU is applied using an optimal minimum threshold value. Moreover, it was shown that the PUP and EMPRP strategies can reduce the number of join operations by up to 93% and 89%, respectively, and that the proposed utility-list construction procedure can reduce run time by up to 41%. These proposed strategies are interesting as they could be used for other problem related to high on-shelf utility pattern mining, where the number of effective strategies is limited.

Although the proposed algorithm was shown to be efficient in terms of runtime and memory consumption, it remains the first algorithm for top-k HOU mining. Hence there is still room for further research and improvements. More specifically, there are several interesting research directions that can be considered. One of them is to design an approximate version of the algorithm that would return an approximate list of top-k HOUs. This would provide a trade-off to reduce the memory usage and execution time by sacrificing completeness. This idea would be innovative since to the best of our knowledge,

there does not exist an approximate algorithm for mining HUIs in static transaction databases. Another interesting research direction is to develop a parallel and distributed version of the proposed algorithm to address the case of big data so that it could be run on massive datasets. Another interesting possibility for future work is to extend this algorithm to other on-shelf high utility pattern mining problems such as closed and maximal on-shelf high utility itemset mining, and also to consider other extensions such as handling non static unit profit values.

Acknowledgements

This study was funded by the National Natural Science Foundation of China (Grant Nos. 61133005, 61432005, 61370095, 61472124, 61202109, and 61472126), and the International Science & Technology Cooperation Program of China (Grant Nos. 2015DFA11240 and 2014DFBS0010). T-L. Dam was also partially supported by science research fund of Hanoi University of Industry, Hanoi, Vietnam.

Compliance with Ethical Standards

Funding: This study was funded by the National Natural Science Foundation of China (Grant Nos. 61133005, 61432005, 61370095, 61472124, 61202109, and 61472126) and the International Science & Technology Cooperation Program of China (Grant Nos. 2015DFA11240 and 2014DFBS0010). T-L. Dam was also partially supported by science research fund of Hanoi University of Industry, Hanoi, Vietnam.

Conflict of Interest: The authors declare that they have no conflict of interest.

Ethical approval: This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. VLDB pp. 487–499 (1994)
2. Agrawal, R., Srikant, R.: Quest Synthetic Data Generator. Available at. (<http://www.almaden.ibm.com/cs/quest/syndata.html>) (1994)
3. Chan, R., Yang, Q., Shen, Y.D.: Mining high utility itemsets. In: Third IEEE International Conference on Data Mining (ICDM 2003), pp. 19–26 (2003)
4. Chen, H.: Mining top-k frequent patterns over data streams sliding window. Journal of Intelligent Information Systems **42**(1), 111–131 (2014)
5. Cheng, J., Ke, Y., Ng, W.: A survey on algorithms for mining frequent itemsets over data streams. Knowledge and Information Systems **16**(1), 1–27 (2008)
6. Cheung, Y.L., Fu, A.C.: Mining frequent itemsets without support threshold: with and without item constraints. IEEE Transactions on Knowledge and Data Engineering **16**(9), 1052–1069 (2004)
7. Chu, C.J., Tseng, V.S., Liang, T.: An efficient algorithm for mining temporal high utility itemsets from data streams. Journal of Systems and Software **81**(7), 1105 – 1117 (2008)
8. Chu, C.J., Tseng, V.S., Liang, T.: An efficient algorithm for mining high utility itemsets with negative item values in large databases. Applied Mathematics and Computation **215**(2), 767 – 778 (2009)
9. Dam, T.L., Li, K., Fournier-Viger, P., Duong, Q.H.: CLS-Miner: efficient and effective closed high utility itemset mining. Frontiers of Computer Science pp. 1–27 (2016). DOI 10.1007/s11704-016-6245-4
10. Dam, T.L., Li, K., Fournier-Viger, P., Duong, Q.H.: An efficient algorithm for mining top-rank-k frequent patterns. Applied Intelligence **45**(1), 96–111 (2016)
11. Duong, Q.H., Liao, B., Fournier-Viger, P., Dam, T.L.: An efficient algorithm for mining the top-k high utility itemsets, using novel threshold raising and pruning strategies. Knowledge-Based Systems **104**, 106–122 (2016)
12. Fournier-Viger, P.: FHN: Efficient Mining of High-Utility Itemsets with Negative Unit Profits. In: Advanced Data Mining and Applications, *Lecture Notes in Computer Science*, vol. 8933, pp. 16–29. Springer International Publishing (2014)
13. Fournier-Viger, P., Gomariz, A., Gueniche, T., Soltani, A., Wu, C.W., Tseng, V.S.: SPMF: A java open-source pattern mining library. Journal of Machine Learning Research **15**, 3569–3573 (2014)
14. Fournier-Viger, P., Lin, J.C.W., Gueniche, T., Barhate, P.: Efficient incremental high utility itemset mining. In: Proceedings of the ASE BigData & Social Informatics 2015, ASE BD & SI '15, pp. 53:1–53:6. ACM, New York, NY, USA (2015)
15. Fournier-Viger, P., Wu, C.W., Zida, S., Tseng, V.: 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 (2014)
16. Fournier-Viger, P., Zida, S.: FOSHU: Faster On-shelf High Utility Itemset Mining – with or Without Negative Unit Profit. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing, SAC '15, pp. 857–864. ACM, New York, NY, USA (2015)

17. Fu, A.W.C., Kwong, R.W.w., Tang, J.: Mining N-most Interesting Itemsets. In: Proceedings of the 12th International Symposium on Foundations of Intelligent Systems, ISMIS '00, pp. 59–67. Springer-Verlag, London, UK (2000)
18. Golab, L., DeHaan, D., Demaine, E.D., Lopez-Ortiz, A., Munro, J.I.: Identifying frequent items in sliding windows over on-line packet streams. In: Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement, IMC '03, pp. 173–178. ACM, New York, NY, USA (2003)
19. Grahne, G., Zhu, J.F.: Fast algorithms for frequent itemset mining using FP-trees. IEEE Transactions on Knowledge and Data Engineering **17**(10), 1347–1362 (2005)
20. Han, J., Cheng, H., Xin, D., Yan, X.: Frequent pattern mining: current status and future directions. Data Mining and Knowledge Discovery **15**(1), 55–86 (2007)
21. Han, J.W., Pei, J., Yin, Y.W.: Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach. Data Mining and Knowledge Discovery **8**(1), 53–87 (2004)
22. Homem, N., Carvalho, J.P.: Finding top-k elements in data streams. Information Sciences **180**(24), 4958 – 4974 (2010)
23. Lan, G.C., Hong, T.P., Huang, J.P., Tseng, V.S.: On-shelf utility mining with negative item values. Expert Systems with Applications **41**(7), 3450 – 3459 (2014)
24. Lan, G.C., Hong, T.P., Tseng, V.S.: Discovery of high utility itemsets from on-shelf time periods of products. Expert Systems with Applications **38**(5), 5851 – 5857 (2011)
25. Lan, G.C., Hong, T.P., Tseng, V.S.: An efficient projection-based indexing approach for mining high utility itemsets. Knowledge and Information Systems **38**(1), 85–107 (2014)
26. Li, H.F., Huang, H.Y., Lee, S.Y.: Fast and memory efficient mining of high-utility itemsets from data streams: with and without negative item profits. Knowledge and Information Systems **28**(3), 495–522 (2011)
27. Lin, J.C.W., Gan, W., Fournier-Viger, P., Hong, T.P.: RWFIM: Recent weighted-frequent itemsets mining. Engineering Applications of Artificial Intelligence **45**, 18 – 32 (2015)
28. Lin, J.W., Gan, W., Hong, T.P.: Maintaining the discovered high-utility itemsets with transaction modification. Applied Intelligence pp. 1–13 (2015)
29. Liu, G., Lu, H., Lou, W., Xu, Y., Yu, J.: Efficient mining of frequent patterns using ascending frequency ordered prefix-tree. Data Mining and Knowledge Discovery **9**(2), 249–274 (2004)
30. Liu, M., Qu, J.: 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, New York, NY, USA (2012)
31. Liu, Y., Liao, W.k., Choudhary, A.: A two-phase algorithm for fast discovery of high utility itemsets. In: Advances in Knowledge Discovery and Data Mining, *Lecture Notes in Computer Science*, vol. 3518, pp. 689–695. Springer Berlin Heidelberg (2005)
32. Manerikar, N., Palpanas, T.: Frequent items in streaming data: An experimental evaluation of the state-of-the-art. Data & Knowledge Engineering **68**(4), 415 – 430 (2009)
33. Metwally, A., Agrawal, D., Abbadi, A.E.: An integrated efficient solution for computing frequent and top-k elements in data streams. ACM Transactions on Database Systems **31**(3), 1095–1133 (2006)
34. Ryang, H., Yun, U.: Top-k high utility pattern mining with effective threshold raising strategies. Knowledge-Based Systems **76**(0), 109 – 126 (2015)
35. Ryang, H., Yun, U.: High utility pattern mining over data streams with sliding window technique. Expert Systems with Applications **57**, 214 – 231 (2016)
36. Salam, A., Khayal, M.: Mining top-k frequent patterns without minimum support threshold. Knowledge and Information Systems **30**(1), 57–86 (2012)
37. Song, W., Liu, Y., Li, J.: BAHUI: Fast and Memory Efficient Mining of High Utility Itemsets Based on Bitmap. International Journal of Data Warehousing and Mining **10**(1), 1–15 (2014)
38. Song, W., Liu, Y., Li, J.: Mining high utility itemsets by dynamically pruning the tree structure. Applied Intelligence **40**(1), 29–43 (2014)
39. Song, W., Zhang, Z., Li, J.: A high utility itemset mining algorithm based on subsume index. Knowledge and Information Systems pp. 1–26 (2015)
40. Tseng, V., Shie, B.E., Wu, C.W., Yu, P.: Efficient algorithms for mining high utility itemsets from transactional databases. IEEE Transactions on Knowledge and Data Engineering **25**(8), 1772–1786 (2013)
41. Tseng, V., Wu, C.W., Fournier-Viger, P., Yu, P.: Efficient algorithms for mining top-k high utility itemsets. IEEE Transactions on Knowledge and Data Engineering **28**(1), 54–67 (2016)
42. Wang, J.Y., Han, J.W., Lu, Y., Tzvetkov, P.: TFP: An efficient algorithm for mining top-k frequent closed itemsets. IEEE Transactions on Knowledge and Data Engineering **17**(5), 652–664 (2005)
43. Wong, R.C.W., Fu, A.W.C.: Mining top-k frequent itemsets from data streams. Data Mining and Knowledge Discovery **13**(2), 193–217 (2006)
44. Wu, C.W., Shie, B.E., Tseng, V.S., Yu, P.S.: Mining top-k high utility itemsets. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, pp. 78–86. ACM, New York, NY, USA (2012)
45. Yang, B., Huang, H.: TOPSIL-Miner: an efficient algorithm for mining top-K significant itemsets over data streams. Knowledge and Information Systems **23**(2), 225–242 (2010)
46. Yun, U., Ryang, H., Ryu, K.H.: High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates. Expert Systems with Applications **41**(8), 3861 – 3878 (2014)
47. Zaki, M.J., Gouda, K.: Fast vertical mining using Diffsets. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 326–335. ACM (2003)
48. Zihayat, M., An, A.: Mining top-k high utility patterns over data streams. Information Sciences **285**, 138–161 (2014)