Information Sciences 587 (2022) 654–678

Contents lists available at ScienceDirect

Information Sciences

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

TKN: An efficient approach for discovering top-*k* high utility itemsets with positive or negative profits

Mohamed Ashraf [a,*], Tamer Abdelkader [a], Sherine Rady [a], Tarek F. Gharib [a]

[a] *Information Systems Department, Faculty of Computer and Information Sciences, Ain Shams University, 11566 Cairo, Egypt*

ARTICLE INFO

Article history:
Received 20 June 2021
Received in revised form 8 December 2021
Accepted 10 December 2021
Available online 17 December 2021

Keywords:
Data mining
Pattern mining
Top-*k* high utility itemsets
Pattern growth approach
Negative unit profits
Threshold raising strategies

ABSTRACT

Top-*k* high utility itemsets (HUIs) mining permits discovering the required number of patterns - *k*, without having an optimal minimum utility threshold (i.e., minimum profit). Multiple top-*k* HUIs mining algorithms have been introduced with interesting results. However, these algorithms focus mainly on mining patterns from positive profit datasets, while few preliminary studies can handle datasets with negative profits. Moreover, conventional top-*k* HUI mining algorithms, that are meant for exploring positive profit datasets, perform poorly when mining top-*k* HUIs on highly dense and large datasets. In this paper, we propose **TKN** (efficiently mining **T**op-**K** HUIs with positive or **N**egative profits) which employs generalized and adaptive techniques to mine both positive and negative profit datasets effectively. The proposed approach adopts transactional projection and merging mechanisms to decrease the dataset traversing cost. Furthermore, several pruning and threshold elevating ideas are utilized to significantly narrow the exploration space. To highlight the reliability of the devised TKN, a series of extensive comparisons were conducted using two versions of six real datasets. The obtained results reveal that TKN is clearly superior in finding the required number of patterns, whether on positive or negative profit datasets, compared to the current cutting-edge competitors.

1. Introduction

Data mining technologies are broadly used for finding valuable and hidden information in vast amounts of data. One of the crucial data mining technologies is Frequent Itemset Mining (FIM) [1]. FIM is a common and essential mining method that is used mainly to disclose association relationships in customer transactional datasets. A typical FIM algorithm discovers all groups of items whose frequencies are not less than a predefined minimum frequency value (support). The process of generating frequent itemsets can be seen as the first phase in generating the association rules (ARs). Then, another measure is applied to assess the degree of certainty (confidence) of each generated rule. The estimation of association rules has proven its usefulness in various decision-making processes. Multiple methods have been presented over years to discover association rules and frequent itemsets efficiently. However, the discovered rules impose an unrealistic assumption that all items within the same transactional dataset are equivalent in importance and can not happen more than once in the same transaction. But in real life, an item may exist several times in the same transaction and every item can have different importance (profit or benefit). Hence, many discovered frequent patterns may then be infeasible to a decision maker.

Due to the previous limitations, a new framework known as Utility Pattern Mining (UPM) [11] has emerged. Inspired by the scientific understanding of the utility theory [29], UPM algorithms consider some useful aspects such as the quantity and profit

* Corresponding author.
E-mail address: mohamed.a.a.h.is@gmail.com (M. Ashraf).

https://doi.org/10.1016/j.ins.2021.12.024
0020-0255/

of items to find interesting and valuable patterns in transactional datasets. Primarily, it entails extracting itemsets producing utilities no lower than a predetermined utility value. The discovered information of HUIs has been integrated with multiple contexts such as clickstream analysis [15], travel pattern analysis [40], spatiotemporal pattern analysis [31], topic detection [4] and opinion analysis [16]. Many utility mining algorithms have been proposed with different data representations and pruning ideas to enhance the mining process [9,19,23,27,28,38,48], some of them are bio-inspired [13,25]. In the same vein, several works concerning the utility mining framework have been introduced to highlight its effectiveness, such as mining local and peak HUIs [10], closed HUIs [30], average HUIs [17], mining HUIs from uncertain data [3] and dynamic data [44].

In UPM, users usually specify a minimum utility threshold (**minUtil**) to retrieve from the datasets the collection of patterns that have a utility value no lower than this threshold. Thus, identifying an appropriate minUtil is significant for UPM algorithms. However, it is a challenging issue for most users to assign a proper value to the minUtil in different application domains; since it requires extensive background and knowledge about the data to be analyzed. Consequently, data analysts often follow a long trial-and-error approach to find an appropriate minUtil. The reason why they need to try multiple thresholds is that when minUtil is excessively high, multiple important patterns may be neglected, and when it is excessively low, a massive number of patterns are generated which consumes more time in filtering and analysis.

To resolve the aforementioned downside, top-*k* High Utility Itemset Mining (HUIM) [21] has arisen as a new framework to handle the sensitivity of traditional HUIM algorithms towards the prespecified utility threshold. In lieu of specifying a minUtil, the user will enter the number of the required patterns; that will be generated after the mining process, using a parameter termed as *k*. To some extent, assigning a value to parameter *k* is much easier than to the minUtil. Multiple approaches have been devised in the field of HUI mining to effectively discover top-*k* HUIs [7,22,32,36,39]. The most recent and efficient top-*k* HUI mining approaches are THUI [22] and TKEH [36]. THUI relies on Utility List (UL) and Leaf Itemset Utility (LIU) data structures to efficiently uncover the top-*k* HUIs. The author [22] reported that the approach provides up to 3 orders of magnitude enhancement in the mining performance when compared to TKO [39] and KHMC [7]. On the other hand, TKEH relies on transactional projection and Estimated Utility Co-Occurrence Structure (EUCS) to efficiently identify the top-*k* interesting patterns. The authors [36] reported that TKEH delivers up to 3 orders of magnitude better performance than TKO and KHMC. Still, after a thorough analysis of the performance of TKEH and THUI on various datasets, we observed the following issues:

- The mining performance of the two aforementioned algorithms was found to be limited on dense and large datasets due to the complex data structure used by THUI and the expensive coverage strategies used by TKEH.
- The aforementioned algorithms were created mainly to process items possessing positive profits, which may limit the applicability of these algorithms in genuine applications because, in real-life situations, retail business transactions may contain items with negative profits. For example, in many pricing strategies, supermarkets usually offer a free item when purchasing a package or a specific item. In such cases, the free item will possess a negative profit. Hence, concentrating purely on positive profits can be seen as a limitation since the algorithm will fail to find the correct set of HUIs if it encounters a negative profit value [37].
- The most recent top-*k* HUIM algorithms are unable to directly deal with items possessing negative profits as they depend on upper bounds and threshold raising strategies that are not applicable when there are items with negative utilities in the dataset.
- Some studies in the literature have tackled the idea of finding HUIs in presence of negative profits [5,20,26,35]. However, these studies still suffer the minUtil setting problem.

Thus far, there are few preliminary studies on capturing top-*k* HUIs while considering negative profits [12,37]. This study area is still at a very basic level of development and the techniques presented by these studies require major enhancements in terms of execution time, memory usage, weak patterns pruning, and scalability. Besides, these studies lack genericity since they are not optimized for situations where there are only positive profits. Therefore, this paper seeks a single generic solution by handling the problem from a global and dynamic perspective. For this purpose, we describe a novel algorithm to reveal the top-*k* HUIs in datasets involving positive profits only or positive and negative profits. Specifically, the major contributions of this work are summarized as follows:

- We address the problem of top-*k* HUIM by taking into account not only the positive profits but also the negative ones. To do so, a novel mining approach, named TKN, is suggested, which utilizes a pattern-growth methodology to obviate generating candidate itemsets that do not actually exist in the dataset and a horizontal dataset representation to facilitate merging any duplicate transactions in the projected datasets.
- We introduce a novel global search procedure to mine the patterns with positive or negative profits. We also adapt the LIU structure presented in THUI [22] and three threshold raising strategies to efficiently elevate the minUtil in the initial stages of mining.
- To narrow the exploration space effectively, key pruning properties (sub-tree utility and local utility) [48] are generalized. Additionally, two novel strategies are proposed. The first is an early pruning strategy to reduce the computational burden of building conditional datasets for candidates. And the second is an early abandoning strategy to reduce the number of evaluations needed to estimate the upper bounds of candidates. Also, an array-based mechanism is adopted to achieve maximum efficiency in estimating the utility of itemsets.

- To exhibit the performance and reliability of the proposed approach, a series of extensive comparisons are conducted using two versions of six real datasets. The comparisons include the state-of-the-art: negative top-*k* HUIM mining algorithm (THN [37]), conventional top-*k* HUIM algorithms (THUI [22] and TKEH [36]) and the optimal HUIM algorithms that consider negative profits (FHN [26] and GHUM [20]). To examine the influence of the adopted techniques, three variants from the proposed method; named TKN(PRIU), TKN(PSU) and TKN(TM), are developed. The obtained results prove that TKN provides better flexibility and usability than conventional top-*k* HUIM algorithms as it achieves more than an order of magnitude enhancement in the mining performance when compared to THN and up to 4 orders of magnitude enhancement when compared to TKEH and THUI, especially on highly dense and large datasets. Besides, it delivers very close or better performance when compared to FHN and GHUM in their optimal case.

In this regard, the generic behavior of TKN is inspired by multiple approaches in pattern mining literature. For instance, Ashraf and Nafis [2] introduced a generic method to discover fault-tolerant frequent patterns from both certain and uncertain datasets. Kim et al. [18] developed LIMHAUP method for quickly determining high average-utility patterns in static and dynamic environments. Sethi and Ramesh [33] proposed a generalized method that is suitable for mining high average-utility patterns with single or multiple minimum utility thresholds in acceptable time complexity. These works have motivated us to do the research introduced in the present paper, which studies the problem of top-*k* HUIM based on positive or negative profits.

The remainder of this work is listed as follows. The background and related literature are described in Section 2. Section 3 introduces problem statement and related notions. Section 4 illustrates the proposed TKN in detail. Section 5 evaluates the efficiency of the proposed algorithm. Eventually, Section 6 draws the conclusion.

2. Related literature

In this section, studies concerned with HUI mining and top-*k* HUI mining are concisely discussed.

2.1. High utility mining with positive profits

Since the introduction of the Two-Phase model [28], High Utility Itemset Mining (HUIM) has become an attractive research topic in the field of pattern mining. Motivated by its remarkable success in different areas [4,15,16,31,40], numerous studies have been conducted to improve the efficiency and effectiveness of the high utility mining process. From a computational perspective, HUIM is a costly operation since it does not subsume to the anti-monotonic property[1] that is broadly exploited in Frequent Itemset Mining (FIM) to prune the exploration space; because a low utility itemset can be part of a high utility superset, making pruning harder in HUIM. To address this vital challenge, several methods have been devised for mining HUIs incorporating different data representations and pruning properties. One of the pioneer studies on HUIM is the two-phase method [28]. This method comprises two phases to discover the whole set of HUIs. In phase one, all candidate HUIs are identified using a pruning property named transaction-weighted utility (*twu*). In phase two, the real utility value is estimated for each candidate itemset to identify the result patterns. Using the notion of *twu* and the Two-Phase paradigm, multiple algorithms have been introduced such as UMining [43] and FUM [24]. Later on, tree-based approaches were suggested, examples include UP-Growth [38] and MU-Growth [45]. However, despite the fact that the *twu* upper bound successfully established a boundary to the solution space, applying only this upper bound produces too many candidates because it highly overestimates the actual utility values of itemsets, thus the itemsets space is massive for mining the required HUIs. Moreover, the nature of the Two-Phase and tree-based approaches requires multiple database scans, which leads to performance and scalability issues.

To mitigate the previous drawbacks, Liu et al. presented HUI-Miner [27] method to reveal all HUIs without generating candidate itemsets in a dedicated phase and using only two database scans. Inspired by Eclat [46], HUI-Miner presented a vertical structure named Utility List (UL) which can save all the needed information for estimating the actual utility of itemsets. Subsequently, several algorithms adopted the HUI-Miner model, examples include FHM [9] and HUP-Miner [19]. Another one-phase algorithm is EFIM [48] which employs utility array, projection process and transaction merging to mine HUIs efficiently.

All the above-mentioned algorithms were mainly designed to enhance the mining efficiency in various types of datasets. Yet, these algorithms can not handle items with negative utilities.

2.2. High utility mining with negative profits

In 2009, HUINIV-Mine [5] was proposed based on the Two-Phase paradigm to process items with negative unit profits. This method presented a modified *twu* pruning property to diminish the itemsets tree and boost the mining process. However, it inherited the flaws of the Two-Phase paradigm. Based on the HUI-Miner model, Lin et al. introduced FHN [26] algorithm with a modified version of the UL structure, named PNU-List, to keep the positive and negative profits of the itemsests separated. This algorithm adapts multiple pruning ways presented in the literature to work with negative

[1] An infrequent itemset can not be part of a frequent superset.

utilities such as U-Prune [27], EUCS-Prune [9] and LA-Prune [19]. The efficiency of FHN was tested by comparing it to HUINIV-Mine. Recently, Krishnamoorthy proposed a new algorithm named GHUM [20]. The algorithm uses a more simple utility list than FHN to improve the performance of the mining method. Furthermore, GHUM adapts several pruning mechanisms like U-Prune and introduced N-Prune strategy which makes use of the negative utility values to perform early pruning for candidates. The author showed that GHUM offers a significant improvement in performance over FHN. Another recent study that tackles the problem of negative utility values is by Singh et al. They proposed a new method called EHIN [35] which extends the principles of the EFIM method to mine negative profit datasets efficiently. The method applies transaction consolidation and conditional datasets to enhance the performance of the mining procedure. It was shown that EHIN is more efficient than FHN by more than an order of magnitude in both runtime and memory consumption. Lastly, Singh et al. proposed EHNL[34] method that can discover HUIs while considering negative utilities and the length of the itemsets. Similar to EHIN, the method leverages transaction merging and projection mechanisms with length constraints to efficiently perform the mining operation. However, although EHNL can process itemsets with negative utilities, it is not straightly related to the former works as they do not apply length constraints during or before the mining process.

The key differences between the above-mentioned algorithms are shown in Table 1.

2.3. Top-k high utility mining with positive profits

To overcome the issue of determining a precise and proper minUtil, top-*k* HUIM methods were presented. These methods initially set the minUtil to zero. After that, the minUtil is automatically increased using threshold raising strategies. Ideally, these strategies should guarantee that no top-*k* HUIs are lost, as well as, the minUtil is raised to a value near to the optimal value. This is quite tough and necessitates highly efficient threshold raising techniques. In reality, one can classify the existing top-*k* HUIM algorithms according to their number of phases as follows:

2.3.1. Two-phase algorithms

TKU [41] is one of the first studies that early handled the task of finding top-*k* HUIs. This method relies on a tree-based structure and entails three major steps. In the first step, the itemsets tree is created by traversing the dataset twice. In the second step, potential top-*k* HUIs (PKHUIs) are determined. In the last step, the PKHUIs are verified by traversing the dataset once to identify the true top-*k* HUIs. TKU applies five threshold increasing techniques during the previous three steps to enhance the overall performance. To optimize the performance of TKU, Ryang et al. proposed REPT [32] which utilizes the same tree structure as TKU but with improved threshold raising strategies to generate fewer candidate itemsets. REPT was found to be superior when compared to TKU. However, both the aforementioned algorithms generate too many PKHUIs and require several dataset scans as they are subjected to the Two-Phase paradigm. To alleviate the drawbacks of these algorithms, one-phase algorithms have been suggested.

2.3.2. One-phase algorithms

TKO [39] extends the HUI-Miner approach to uncover the top-*k* HUIs without excessive dataset scans by using the UL structure and novel pruning strategies. The method showed more promising results in both runtime and memory usage than those that follow the Two-Phase paradigm. As an enhancement, Duong et al. introduced KHMC [7] by extending the FHM approach to discover the desired number of HUIs. KHMC employs the notion of coverage to raise the minUtil using two threshold raising strategies (CUV & COD) along with a modified EUC structure and LA-Prune property to limit the number of join operations between the ULs of the candidate itemsets. KHMC was compared to REPT and TKO to show the efficacy of its adopted ideas. Recently, a projection-based algorithm called TKEH [36] was proposed. TKEH uses the pruning and mining techniques of EFIM [48] as well as the notion of coverage to explore the top-*k* interesting patterns with higher efficiency. The experimental evaluation showed that TKEH delivers up to 3 orders of magnitude better performance than KHMC. Another recent work was proposed by Krishnamoorthy. The author proposed THUI [22] algorithm which utilizes the leaf itemset utility structure (LIUS) and extends the basic idea of HUI-Miner with LA pruning to boost the top-*k* mining process. Based on a predetermined order, the concept of leaf itemset utility involves capturing the utility information of each ordered sequence of items in the dataset. The captured information is then used to raise the minUtil effectively using two LIUS-based strategies (LIU-E & LIU-LB). Experiments conducted by the author proved that the LIUS-based strategies are more effective than the coverage-based strategies utilized by the KHMC method.

The core differences between the previously-mentioned top-*k* mining algorithms are outlined in Table 2.

2.4. Top-k high utility mining with negative profits

Up to date, few attempts have been conducted to address the problem of top-*k* HUIM while considering negative profits. TopHUI [12] is the first method for mining top-*k* HUIs based on negative profits. This method utilizes the concepts introduced in FHN [26] to find the required number of patterns without specifying a utility threshold. It employs four threshold increasing techniques (RIU, RTU, RTWU and RUC) in different stages of the mining operation. Additionally, four pruning techniques are used to restrict the solution space. But, TopHUI suffers the drawbacks of the FHN model as it requires huge memory space for its relatively complex PNU-list structure. Another method is THN [37] which follows the concepts presented in EHIN [35] to mine top-*k* HUIs on negative profit datasets. THN relies on a dedicated procedure to extend positive HUIs with

Table 1
A brief view for HUIM algorithms with negative profits.

Algorithm	Dataset representation	Pruning strategy	Base algorithm
HUINIV	Horizontal	twu	Two-Phase
FHN	Vertical (Utility-list)	twu, EUCP, U, LA	HUI-Miner
GHUM	Vertical (Utility-list)	twu, EUCP, U, LA, N	HUI-Miner
EHIN	Horizontal (Utility-array)	twu, LU, SU	EFIM
EHNL	Horizontal (Utility-array)	twu, SU, Length-constraint	EFIM

Table 2
A brief view for top-*k* HUIM algorithms with positive profits.

Algorithm	Dataset representation	Threshold raising strategy	Pruning strategy	Base algorithm
TKU	Horizontal (UP- Tree)	PE,MD,NU,MC,SE	DGU, DGN, DLU, DLN	UP-Growth
REPT	Horizontal (UP- Tree)	RIU,PUD,RSD,NU,MC,SEP	twu, DGN, DLU, DLN	MU-Growth
TKO	Vertical (Utility-list)	PE,RUC	DGU, RUZ, EPB, U	HUI-Miner
KHMC	Vertical (Utility-list)	RIU,CUD,COV,RUC	twu, EUCP, U, LA, TEP	FHM
THUI	Vertical (Utility-list)	RIU,LIU-E,LIU-LB,RUC	twu, U, LA	HUI-Miner
TKEH	Horizontal (Utility-array)	RIU,CUD,COV,RUC	twu, EUCP, SU	EFIM

items having negative profits and uses only one threshold raising strategy (RTWU) to elevate the utility threshold in the early stages of mining. The authors stated that THN performs better than FHN [26] and HUINIV-Mine [5] algorithms on dense and large datasets. However, our evaluation revealed that using a standalone procedure for mining HUIs with negative profits incurs a performance penalty and generates much more candidate patterns. In fact, a major limitation of the aforementioned algorithms is that they rely on weak strategies to increase the minUtil. More importantly, these algorithms may not find the complete solution because they utilize RTWU values for raising the minUtil, which can lead to unwitting pruning for candidates since RTWU values are overestimated and may not reflect actual utility values in the dataset.

2.5. Top-k high utility mining extensions

Recently, multiple approaches have been suggested to extend the basic idea of top-*k* HUIM. Zhang et al. suggested TKUS [47] method which can identify the top-*k* high utility patterns in sequential datasets. This approach utilizes Sequence Utility Raising (SUR) as a threshold raising strategy and two pruning techniques (TDE & EUI) to limit the search space. The usefulness of the utilized techniques was experimentally evaluated on two sparse and four dense benchmark datasets. Dam et al. [6] incorporates the idea of on-shelf utility mining with the top-*k* framework by presenting KOSHU. The method accommodates the UL data structure to compactly save the important information for finding top-*k* on-shelf HUIs. It also applies three pruning mechanisms and two threshold increasing techniques to improve the mining process. The authors showed that KOSHU offers a promising performance when compared to the corresponding non-top-*k* methods. TKAU [42] integrates the notion of average utility with the concept of top-*k* mining to find top-*k* high average utility itemsets. This method utilizes a UL-based structure named AUO-List with two pruning methods (EMUP & EA) and three average-utility threshold increasing mechanisms (RIU, CAD & EPBF). TKAU was compared with the corresponding non-top-*k* average utility algorithms and showed a competitive to optimal performance. Lastly, PTM [14] discovers top-*k* HUIs in massive datasets. It utilizes a prefix-based partitioning technique to store the dataset transactions and a new sub-tree-based pruning strategy to explore the required number of patterns efficiently. PTM first divides the transaction dataset into partitions based on the prefix item in every transaction. After that, the average transaction utility is calculated for every partition. Subsequently, the partitions are processed in the non-increasing order of the average transaction utility of each partition so that the minUtil can reach faster to the optimal value. Eventually, the top-*k* valuable patterns are found when all partitions are processed in a depth-first mode.

2.6. Differences from prior works

Based on the aforementioned review, it can be concluded that our suggested approach (TKN) is distinct from the existing top-*k* HUIs mining methods in multiple aspects. On the one hand, the suggested approach does not involve a candidate generation phase like TKU [41] and REPT [32] methods. Our approach includes pattern growth and dataset reduction mechanisms to eliminate the downsides of the utility list data structure utilized by TKO [39], KHMC [7] and THUI [22] methods. The presented approach does not use the inefficient coverage-based threshold elevating mechanisms utilized by TKEH [36] method. All the aforementioned methods have a major problem that they can extract the top-*k* HUIs only from positive profit datasets, while TKN can mine both positive and negative profit datasets. On the other hand, the suggested approach is different from the existing negative top-*k* HUIM methods since it does not utilize the inefficient PNU-list structure like TopHUI [12] method and uses a global search procedure for mining top-*k* HUIs with positive and negative utilities in contrast to THN [37] method. In addition, TKN is optimized to mine datasets involving only positive profits. The efficiency of the suggested approach is evaluated and demonstrated through multiple experiments on several positive and negative profit datasets.

3. Problem statement and related notions

In this section, we present key notions and terms that are frequently used in utility mining literature.

Definition 1. (Transaction dataset). Consider the transaction dataset D in Table 3. D comprises six transactions $(t_1, t_2, \ldots, t_6)$. Every transaction entails a list of distinct items. E.g., in Table 3, items c, d, and e were purchased in transaction t_6 with quantities 2, 1, and 1 respectively. Table 4 outlines the profit value for every item within D.

Definition 2. (Internal utility of items iu). Let x be an item within a transaction $t_k \in D$. The quantity of x expresses the internal utility value of x in t_k. E.g., in Table 3, $iu(b, t_4) = 3$.

Definition 3. (External utility of items eu). Consider an item $x \in D, eu(x)$ declares the profit of selling item x, which can be positive or negative. E.g., in Table 4, $eu(b) = 3$.

Definition 4. (Utility of an item in a transaction). Consider an item $x \in t_k, u(x)$ refers to the utility of x in t_k and is computed as next, $u(x, t_k) = iu(x, t_k) * eu(x)$. E.g., in Table 3, $u(b, t_4) = iu(b, t_4) * eu(b) = 3 * 3 = 9$.

Definition 5. (Utility of an itemset in a transaction). Consider an itemset $X \subseteq t_k, u(X)$ in t_k can be estimated as next, $u(X, t_k) = \sum_{x_i \in X \wedge X \subseteq t_k} u(x_i, t_k)$. E.g., in Table 3, $u(\{ce\}, t_6) = u(c, t_6) + u(e, t_6) = 2 + 2 = 4$.

Definition 6. (Utility of an itemset in a dataset). Consider an itemset X in the dataset D, the overall utility of X in D can be estimated using the following formula: $u(X) = \sum_{X \subseteq t_k \wedge t_k \in D} u(X, t_k)$. E.g., in Table 3, $u(\{ce\}) = u(\{ce\}, t_2) + u(\{ce\}, t_3) + u(\{ce\}, t_5) + u(\{ce\}, t_6) = 7 + 12 + 13 + 4 = 36$.

Definition 7. (Transaction utility tu). Consider any transaction t_k in the example dataset D, the transaction utility of t_k can be estimated as follows: $tu(t_k) = \sum_{x \in t_k} u(x, t_k)$. E.g., in Table 3, $tu(t_6) = u(c, t_6) + u(d, t_6) + u(e, t_6) = 2 - 1 + 2 = 3$.

Definition 8. (Transaction-weighted utility *twu*). Consider an itemset X, we can estimate $twu(X)$ as next, $twu(X) = \sum_{X \subseteq t_k \wedge t_k \in D} tu(t_k)$. E.g., $twu(\{abc\}) = tu(t_1) + tu(t_2) = 11 + 18 = 29$.

The *twu* has served as a pruning criterion to eliminate the unpromising itemsets in all conventional top-k HUIM algorithms by overestimating the actual utility values of candidates [7,22,32,36,39]. Nevertheless, it has been confirmed that this upper bound is no longer applicable if the dataset comprises items possessing negative profits since some HUIs can be missed, underestimated or incorrectly pruned during the mining process [5,26].

3.1. Properties to deal with negative profits

Next, we demonstrate how TKN is developed to encounter the problem of mining top-k HUIs on datasets encompassing negative profits. For simplification purpose, we refer to items with positive profits as positive items and the ones with negative profits as negative items.

Property 1. (The relation between positive and negative items). Given the itemset X which contains positive and negative items. Let $Putil(X)$ indicates the aggregation of utilities of positive items, and $Nutil(X)$ represents the aggregation of utilities of negative items within X. It is possible to estimate the exact utility of X as follows: $u(X) = Putil(X) + Nutil(X)$, such that the relation $Putil(X) \geqslant u(X) \geqslant Nutil(X)$ holds [26].

Table 3
An example of a transaction dataset.

Tid	Transaction	tu
t_1	(a,1), (b,2), (c,2), (d,1)	11
t_2	(a,1), (b,3), (c,3), (d,2), (e,2)	18
t_3	(a,1), (c,6), (e,3)	16
t_4	(b,3), (d,5), (e,2)	8
t_5	(b,1), (c,5), (d,1), (e,4)	15
t_6	(c,2), (d,1), (e,1)	3

Table 4
Item profits.

Item	a	b	c	d	e
Profit $ per unit (EU)	4	3	1	−1	2

Rationale. Items with positive profits permanently raise $u(X)$, whereas items with negative profits permanently lower $u(X)$. Thus, the property $Putil(X) \geqslant u(X) \geqslant Nutil(X)$ holds.

Property 2. (Positive utility upper bound). Given an itemset X, since $Putil(X)$ can not be lower than both $u(X)$ and $Nutil(X)$, then it can be said that $Putil(X)$ is an upper bound utility on X [26].

Definition 9. (Positive transaction utility Ptu). The Ptu of a transaction t_k is the aggregation of utilities of positive items in t_k. The Ptu for any transaction t_k is defined as follows: $\text{Ptu}(t_k) = \sum_{x \in t_k \wedge u(x) \geqslant 0} u(x, t_k)$. E.g., in Table 3, $\text{Ptu}(t_2) = 4 + 9 + 3 + 0 + 4 = 20$.

Definition 10. (Positive transaction-weighted utility Ptwu). Given an itemset X, the Ptwu of X in the dataset D can be expressed as next, $\text{Ptwu}(X) = \sum_{X \in t_k \wedge t_k \in D} Ptu(t_k)$. E.g., $\text{Ptwu}(\{ab\}) = Ptu(t_1) + Ptu(t_2) = 12 + 20 = 32$. Table 5 shows the Ptwu values for all items in D.

Property 3. (Overestimating an itemset using Ptwu). Given an itemset X, if $\text{Ptwu}(X)$ is larger than the exact utility of X, more specifically, $\text{Ptwu}(X) \geqslant u(X)$, then X is said to be overestimated [26].

Property 4. (Pruning an itemset using Ptwu). Given an itemset X, if the positive transaction-weighted utility of X is lower than minUtil, to be specific, $\text{Ptwu}(X) < minUtil$, then X cannot be included for further processing and can be safely discarded from the exploration space [26].

Using this adaptation, we restore the pruning property of the transaction-weighted utility *twu*. However, it heavily overestimates the actual utility values of itemsets. Since exploring top-k HUIs in the existence of positive and negative items is a difficult computing task, there is thus a need to develop more efficient pruning strategies. To avert huge exploration space, this paper presents two tighter pruning properties to boost the algorithm speed and efficiently limit the exploration space.

Definition 11. (Total order of items $\prec$). It is assumed in this paper that all items in the dataset are processed according to the total order $\prec$ as follows: (1) positive items are arranged in Ptwu ascending order, (2) negative items always succeed all positive items, and (3) negative items are arranged in Ptwu ascending order. In other words, all sub-trees of positive items are explored first before extending them with negative items. Considering the current example, items are processed as follows: $a \prec b \prec c \prec e \prec d$. Table 6 depicts the arranged set of items in every transaction of the sample dataset D.

Definition 12. (Positive remaining utility for an itemset). Consider an itemset $X \subseteq t_k$. The positive remaining utility of X, termed by $\text{Pru}(X, t_k)$, is the aggregation of utilities of all positive items after X within t_k, to be specific, $\text{Pru}(X, t_k) = \sum_{i \in t_k \wedge X \prec i \wedge u(i,t_k) > 0} u(i, t_k)$.

Definition 13. (Top-k high utility itemset). Given the dataset D and the user-preferred value for k, the top-k HUIs represent the collection of itemsets that have the largest utility values in D.

Problem statement. Given a positive or negative profit dataset D and the user-preferred number of patterns k in advance, this method is intended to identify the k itemsets with the largest utility values in D. Considering the current transactional dataset D, in Table 3, the top-k HUIs when k = 4 are {{be}:37, {ce}:36, {bce}:32, {bced}:29}.

Table 5
Ptwu values.

Item	a	b	c	d	e
Ptwu	48	61	68	65	69

Table 6
An ordered transaction dataset using the total order $\prec$.

Tid	Transaction	Ptu
t_1	(a,1), (b,2), (c,2), (d,1)	12
t_2	(a,1), (b,3), (c,3), (e,2), (d,2)	20
t_3	(a,1), (c,6), (e,3)	16
t_4	(b,3), (e,2), (d,5)	13
t_5	(b,1), (c,5), (e,4), (d,1)	16
t_6	(c,2), (e,1), (d,1)	4

The key target of this research is to devise a novel top-*k* HUIM method that is able to handle transactional datasets involving items with negative profits while keeping in view the likelihood that a transactional dataset may have only positive profits.

4. The proposed TKN

In this section, we institute a precise study for the suggested algorithm as follows. Section 4.1 portrays the exploration space. Section 4.2 describes the adopted mechanisms to decrease the dataset traversing cost. Section 4.3 presents the adopted LIU structure that is used to raise the minUtil. Section 4.4 covers the proposed threshold increasing strategies. Section 4.5 covers the pruning properties of the suggested algorithm. Section 4.6 presents the adopted technique to estimate the utility of itemsets. Section 4.7 provides a demonstration for the working way of the proposed algorithm using pseudo code. Finally, Section 4.8 introduces an illustrative example for the proposed algorithm.

4.1. The exploration space

The exploration space for the problem of mining top-*k* HUIs with positive or negative profits can be depicted as an enumeration tree with items arranged using the proposed order $\prec$ (refer to Definition 11). The exploration space of items a, b, c, e and d can be seen in Fig. 1. The whole 2^nitemsets can be generated using this tree structure, where n represents the number of distinct items in D. Starting from the root, TKN explores the search space and generates larger itemsets in a depth-first way as shown in Fig. 1. During this process, the proposed algorithm recursively extends each itemset by another successor item according to the total order $\prec$ on items. We provide below various related definitions to the tree exploration process in a depth-first fashion.

Definition 14. (Extension of an item). For an itemset α, the group of items that can be used to extend α during the depth-first search is referred to as: $E(\alpha)$ and defined as: $E(\alpha) = \{z | z \in I \Lambda z \succ i, \forall i \in \alpha\}$.

Definition 15. (Extension of an itemset). Let α be an itemset. An itemset Y extends α if it exists in the sub-tree of α such that $Y = \alpha \cup Z$, where itemset $Z \in 2^{E(\alpha)}$. Moreover, If $Y = \alpha \cup \{z\}$, where item $z \in E(\alpha)$, then Y is a single-item extension of α (a descendent node of α in its sub-tree) [26]. Considering the current example, assuming $\alpha = \{c\}$, the set $E(\alpha)$ is equal to $\{e, d\}$ as per the total order of items $\prec$. The single-item extensions of α are $\{ce\}$ *and* $\{cd\}$.

Definition 16. (Extension of negative item). Let α and X be itemsets. X contains one or more item with a negative utility value such that $X \subseteq E(\alpha)$[26].

Rationale. The total number of transactions involving $\alpha \cup X$ can not exceed that involving α. Extending α with positive items may increase the utility of α. But, extending α with negative items can only reduce the utility of α as per Property 1. As a result, from the above properties, if $u(\alpha) >$minUtil then an itemset X with negative items can be used to extend itemset α. Subsequently, if $\alpha \cup X$ has a utility no less than minUtil then the extended itemset is a high utility pattern.

4.2. Dataset scanning techniques

The suggested TKN employs transactional projection and merging mechanisms to decrease the required time and memory for scanning the dataset.

Definition 17. (Transaction merging). Let there be transactions $t_1, t_2, \ldots, t_n$ such that these transactions are similar (have the same items with different or same quantities). These similar transactions can be substituted by one new transaction $t_m = t_1 = t_2..t_n$ using the transaction merging mechanism. The quantity of every item $x \in t_m$ is calculated as follows: $q(x, t_m) = \sum_{i=1..n} q(x, t_i)$.

The transaction consolidation concept is also applied on the projected datasets to fulfill more dataset size reductions as the projected transactions tend to be similar.

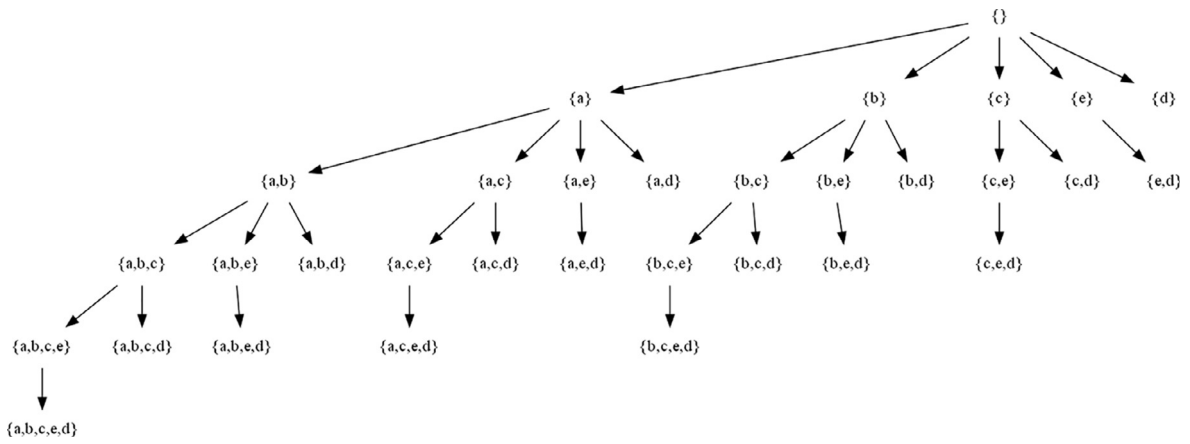

Fig. 1. The exploration space when I = {a, b, c, e, d}.

Definition 18. (Projected dataset). Let be an itemset α, the projection of transaction t w.r.t[2] α is termed as $\alpha - t$ and is defined as: $\alpha - t = \{i | i \in t \wedge i \in E(\alpha)\}$. For the current dataset D, the projection of D w.r.t α is termed as $\alpha - D$ and is expressed as the multi-set $\alpha - D = \{\alpha - t | t \in D \wedge \alpha - t \neq \varnothing\}$.

Definition 19. (Projected transaction merging). Let be a projected dataset $\alpha - D$ which involves the similar transactions $t_{p1}, t_{p2}, .., t_{pn}$. These transactions can be substituted by one new transaction $t_{pm} = t_{p1} = t_{p2} = t_{pn}$ and the quantity of each item $x \in t_{pm}$ can be estimated using the formula: $q(x, t_{pm}) = \sum_{i=1..n} q(x, t_i)$, such that n represents the number of the similar transactions in $\alpha - D$.

Considering the current example, the projected dataset $\alpha - D$, when $\alpha = \{c\}$, involves some similar projected transactions such as $\alpha - t_2 = \{e, d\}$, $\alpha - t_5 = \{e, d\}$ and $\alpha - t_6 = \{e, d\}$. These transactions can be merged into a new projected transaction with the aid of pseudo projection [48]. Thereby, the three previous transactions are combined and replaced by the new transaction $t_{2\_5\_6} = \{e, d\}$, where $q(e, t_{2\_5\_6}) = 7$ and $q(d, t_{2\_5\_6}) = 4$. The previous example indicates that as we explore longer itemsets, the conditional datasets tend to be smaller.

From an algorithmic perspective, the transaction merging technique is highly effective in decreasing the size of the conditional datasets. Nonetheless, a major challenge is how to merge identical transactions with the lowest possible processing cost. To enhance the performance of the merging process, we adopt the transaction sorting technique proposed by EFIM [48]. This sorting technique is performed only once and has a low computational cost.

4.3. The concept of Leaf Itemset Utility LIU

A crucial concern in the top-k HUIs mining problem is to develop effective mechanisms to increase the minUtil in the initial levels of the mining process. These mechanisms should have a low complexity regarding memory and runtime. In the proposed algorithm, we employ LIU structure-based strategies to quickly increase the minUtil to the optimal value and limit the exploration space. This structure was introduced by THUI [22] algorithm to discover top-k HUIs from datasets having only positive utility values. In principle, based on a prespecified order, the LIUS captures the utilities of itemsets consisting of an ordered group of items and saves them in a compact manner. In this paper, we adjust the LIUS to work with positive and negative utility values. Thus, the LIUS-based strategies can be used to raise the minUtil efficiently.

Definition 20. (LIU structure). Let be a specific order of items, the LIUS is a triangular matrix that can be used to store the total utility of every ordered group of items in each transaction in the dataset. More formally,

$$LIU(x, y) = \sum_{z=(x..y) \subseteq t_j \wedge t_j \in D} u(z, t_j) \text{ [22].}$$

where $(x \ldots y)$ denotes the contiguous and ordered sequence of items from x to y. In essence, the LIU structure holds the utilities of the long-ordered sequences of items so that it can be used to raise the minUtil effectively. In the proposed algorithm, we adapt the LIUS to store only the utilities of sequences having no items with negative utility value. Considering the current example, the order of positive items according to the total order of items $\prec$ assumed in this paper is: $a \prec b \prec c \prec e$. Therefore, the contiguous sequence $\{a \ldots e\} = \{a, b, c, e\}$. Similarly, $\{b \ldots e\} = \{b, c, e\}$. The utility values of theses sequences are calculated and stored in $LIU(a, e)$ and $LIU(b, e)$ respectively. The LIU values for all contiguous sequences of positive items, after processing the first and all transactions, are shown in Fig. 2

[2] with respect to

4.4. Threshold raising strategies

4.4.1. Positive Real Item Utility strategy

The RIU strategy was first introduced by REPT algorithm [7] and has been adopted in multiple top-k algorithms [7,22,36]. In the proposed TKN, this strategy is generalized to elevate the minUtil for the first time from zero. During the first dataset scan, a positive RIU value is computed for each positive item in the dataset using the following formula:

$$PRIU(x) = \sum_{x \in t_j \wedge t_j \in D} u(x, t_j)$$

After calculating PRIU values, the PRIU strategy elevates the minUtil to the k^{th} largest PRIU value in a sorted list. Considering the current example, the PRIU values for items a, b, c and e are $12, 27, 18$ and 24 respectively.

Algorithm 1: PRIU strategy

Input: PRIU-list, k: the required number of HUIs.
Output: Increased minUtil.
1 Sort PRIU-list in descending order.
2 Set minUtil to the k^{th} value in the PRIU-list.
2 Return minUtil;

4.4.2. Positive LIU-Exact strategy

We utilize LIU-E strategy [22] to raise the minUtil using the values stored in the LIU structure. After the LIUS has been constructed using only positive sequences, all values no lower than the minUtil are kept in a priority queue *PIQU_LIU* of size k. Then, the minUtil is increased to the k^{th} largest value in the *PIQU_LIU*.

Algorithm 2: PLIU_E strategy

Input: LIUS, *PIQU_LIU*, k: the required number of HUIs.
Output: Raised minUtil.
1 Keep the k largest values of the LIUS in priority queue *PIQU_LIU*.
2 Set minUtil to the k^{th} largest value in *PIQU_LIU* iff it is greater than the current minUtil.
3 Return minUtil;

4.4.3. Positive LIU-Lower Bound strategy

We adopt the LIU-LB strategy [22] to elevate the minUtil for the third time. This strategy uses the LIU structure to calculate lower bound utility values for each stored sequence. These values are then combined with the ones already exist in *PIQU_LIU* to elevate the minUtil to the k^{th} largest value among them. It is to be noted that the LIUS has only utilities of positive sequences (has no items with negative utilities). Hence, this strategy can be harmlessly applied to raise the utility threshold. Given any two items x and y in the LIU structure, the lower bound of the superset sequence $(x \ldots y)$ can be calculated using the formula:

$$LIULB(x, y, q) = LIU(x, y) - u(q) \text{ [22].}$$

where x and y are the first and last item in the sequence and q is the set of items removed from the sequence. It is worth mentioning that a massive number of subsets can be produced from the superset sequences. To limit the number of generated subsets, the largest size of q is set to 4. More precisely, four items at most can be removed from any superset sequence. It is noteworthy that THUI [22] limits the maximum size of q to 3 to obviate generating a huge number of

	b	c	e
a			
b			
c			4

	b	c	e
a	23	28	20
b		28	32
c			36

Fig. 2. (a) LIUS after processing t_1 (b) After processing t_1 to t_6.

subsets, which can degrade the performance. However, when there are negative items in the dataset, TKN considers only positive items to create the LIUS. As a result, the number of sequences stored in the LIUS is lower than that of THUI. In such a case, it can be argued that raising the number of subsets exploits the existence of negative utility values in the dataset and increases the efficiency of the strategy. To avoid generating the same subset multiple times, the first and last items in the sequence are fixed and only intermediate items are removed. For the current example, the utility lower bound of itemset $\{abe\}$; which is subset of the superset sequence $\{abce\}$ with one item removed from it, can be computed that way: $LIULB(a,e,c) = LIU(a,e) - u(c) = ULB(abe)$. After processing all sequences in the LIU matrix, the lower bound utilities can be used to increase the minUtil.

Algorithm 3: PLIU_LB strategy

```
Input: LIUS, PIQU_LIU, PIQU_LB_LIU, k: the required number of HUIs.
Output: Raised minUtil.
foreach entry i, j ∈ LIUS do
    pos_start_item = succ(i);
    pos_end_item = pred(j);
    foreach x ∈ (pos_start_item...pos_end_item) do
        util_LB = LIUS[i, j] − u(x);
        if util_LB > minUtil then
            add util_LB to PIQU_LB_LIU;
        end if
        foreach y = succ(x) ∈ (pos_start_item...pos_end_item) do
            util_LB = LIUS[i, j] − u(x) − u(y);
            if util_LB > minUtil then
                add util_LB to PIQU_LB_LIU;
            end if
            foreach z = succ(y) ∈ (pos_start_item...pos_end_item) do
                util_LB = LIUS[i, j] − u(x) − u(y) − u(w);
                if util_LB > minUtil then
                    add util_LB to PIQU_LB_LIU;
                end if
                foreach w = succ(z) ∈ (pos_start_item...pos_end_item) do
                    util_LB = LIUS[i, j] − u(x) − u(y) − u(w) − u(z);
                    if util_LB > minUtil then
                        add util_LB to PIQU_LB_LIU;
                    end if
                end foreach
            end foreach
        end foreach
    end foreach
end foreach
if |PIQU_ALL = PIQU_LIU ∪ PIQU_LB_LIU| ≥ k then
    if k^th largest value in PIQU_ALL > minUtil then
        minUtil = k^th largest value in PIQU_ALL;
    end if
end if
Return minUtil;
```

4.5. Pruning strategies

To develop an efficient top-k mining algorithm that can consider both positive and negative profits, a core challenge is how to early identify the unpromising branches in the exploration tree. In the light of this challenge, we introduced one upper bound (Ptwu) to decrease the number of possible HUIs. To further improve the robustness of TKN, we introduce two more upper bounds with early pruning and abandoning strategies to greatly narrow the exploration space and reach higher efficiency. These strategies are applied while exploring the sub-trees of the candidate itemsets in a depth-first fashion.

4.5.1. Pruning the exploration space using positive sub-tree utility

Definition 21. (Positive Sub-tree Utility PSU). Given the itemset α and an item $y \in E(\alpha)$, which can extend α while exploring its sub-tree in a depth-first order. To estimate the PSU of item y w.r.t α we use the following formula:

$$PSU(\alpha, y) = \sum_{t_k \in (\alpha \cup \{y\})} [u(\alpha, t_k) + u(y, t_k) + \sum_{i \in t_k \ \Lambda \ i \in E(\alpha \cup \{y\})} u(i, t_k)]$$ [48].

This pruning property can be viewed as a generalized version of the sub-tree pruning property introduced by EFIM. Considering three cases. Case 1: when y is a positive item, the strategy subsumes to the sub-tree property proposed by EFIM. Case 2: when y is positive but all items in $E(\alpha \cup y)$ are negative, $\sum_{i \in t_k \Lambda i \in E(\alpha \cup \{y\})} u(i, t_k)$ becomes zero. Case 3: when y is a negative item, $u(y, t_k) + \sum_{i \in t_k \Lambda i \in E(\alpha \cup \{y\})} u(i, t_k)$ becomes zero. The application of this strategy requires that items in transactions are ordered according to $\prec$ (positive items precede negative items). This is one of the essential pruning properties proposed in this paper due to its supreme effectiveness in removing the weak candidates early from the search space. Considering the current example, when $\alpha = \{b\}$, PSU (α, c) = (6 + 2 + 0) + (9 + 3 + 4 + 0) + (3 + 5 + 8 + 0) = 40 and PSU (α, e) = (9 + 4 + 0) + (9 + 4 + 0) + (3 + 8 + 0) = 37. More specifically, we accumulate only positive utilities according to Property 2.

Property 5. (PSU overestimation). Given the itemset α and an item $y \in E(\alpha)$. If PSU (α, y) < minUtil then the sub-tree of $\alpha \cup \{y\}$ cannot have a high utility itemset. This means that the whole sub-tree of $\alpha \cup \{y\}$ within the exploration space can be safely removed from the exploration space without missing any HUIs [48].

4.5.2. Pruning the exploration space using positive local utility

Definition 22. (Positive Local Utility PLU). Given an itemset α and an item $y \in E(\alpha)$. To estimate the PLU of item y w.r.t α we use the following formula:

$$PLU(\alpha, y) = \sum_{t_k \in (\alpha \cup \{y\})} [u(\alpha, t_k) + re(\alpha, t_k)$$ [48].

The local utility property is generalized to support both positive and negative profits. When there are no negative items within the transaction, the strategy subsumes to the local utility property presented by EFIM. In case y is negative, the second part of the equation $re(\alpha, t_k)$ becomes zero. This is always true as we add only positive utilities according to Property 2. Moreover, the transactions are ordered as per the total order $\prec$. That is, all positive items are processed before the negative ones. Considering the current example, when $\alpha = \{b\}$, PLU(α, c) = 40 and PLU (α, e) = 45.

Property 6. (PLU overestimation). Given an itemset α and an item $y \in E(\alpha)$. If PLU (α, y) < minUtil then any itemset contains α and y can not be HUI. In short, item y can be neglected when traversing the whole sub-tree of α [48].

The relationship between the generalized upper bounds (Ptwu, PLU, PSU) proposed in this paper and the vital upper bound REU (Remaining Utility) proposed by HUI-miner [27] and adapted in FHN [26] and GHUM [20] is the following. The computation of the REU is based on a vertical dataset representation. The introduced PSU is a generalization of the Sub-tree Utility (SU) upper bound proposed by EFIM [48] so that it can handle both positive and negative utilities. SU is computed for each itemset while traversing the itemsets tree in a depth-first order. In the same way, the proposed upper bound PSU is computed at itemset α during the depth-first search instead of computing it at the descendant nodes of α. Thereby, if PSU (α, y) < minUtil then the itemset $\alpha \cup \{y\}$ and all its descendant nodes can be pruned. Conversely, the REU upper bound is computed at the descendant nodes of the itemset $\alpha \cup \{y\}$. Hence, only descendant nodes are pruned. Therefore, using PSU rather than REU achieves more effectiveness in pruning the exploration space.

From Property 3, Property 5 and Property 6 it can be clearly observed that the relation Ptwu (α) $\geqslant$ PLU (α, y) $\geqslant$ PSU (α, y) holds. In the remainder of this paper, items with PSU and PLU no lower than minUtil are referred to as primary and secondary items respectively.

Definition 23. (Primary and Secondary items). Consider an itemset α, the primary items of α are the group of items denoted as Primary(α) = $\{w | w \in E(\alpha) \ \Lambda \ PSU(\alpha, w) \geqslant minUtil\}$. The secondary items of α are the group of items denoted as Secondary (α) = $\{w | w \in E(\alpha) \ \Lambda \ PLU(\alpha, w) \geqslant minUtil\}$. Considering the current example, when $\alpha = \{a\}$, Primary(α)= $\{b, c\}$ and Secondary (α)= $\{b, c, e, d\}$.

4.5.3. Early Pruning (EP) strategy

To achieve an efficient level of mining, a novel strategy is introduced to accelerate the mining process by reducing the number of the processed transactions in the projected dataset of the prefix itemset. Consider an itemset α and a negative item $y \in E(\alpha)$. During building the projected dataset of $\alpha \cup \{y\}$, the utility of the prefix itemset α can be viewed as an upper bound utility for $\alpha \cup \{y\}$; since negative items can only reduce the utility of α according to Property 1. While processing the transactions of the projected dataset of α, the prefix utility u(α) is reduced by the negative utility of the extended item u(y) in the transaction being processed. Thereby, if u(α) has been reduced below the minUtil then we can safely stop building the projected dataset of $\alpha \cup \{y\}$ and proceed to the next extended item. Applying this strategy requires that both primary and

secondary items are arranged as per the total order $\prec$. That is, a negative item is always followed by another negative item. The proposed EP strategy is applied iteratively during the dataset projection operation to achieve early pruning.

4.5.4. Early Abandoning (EA) strategy

This strategy is based on the following important observation, negative items always reduce the utility of the itemsets (refer to Property 1). Assuming an itemset α, if u(α) is less than minUtil and all items in $E(\alpha)$ are negative items (which indicates that the sub-tree of α cannot have an HUI) then we can safely stop exploring the whole sub-tree of α without the need to calculate its primary and secondary sets of items. This strategy requires that items are ordered as per the total order $\prec$. The proposed EA strategy can be used to speed up the exploring process by avoiding the unnecessary dataset scans required to calculate the upper bounds (PLU and PSU).

4.6. Compute upper bounds efficiently using an array-based mechanism

Until now, we have introduced three generalized upper bounds (UBs) to limit the number of possible high utility itemsets. Next, we demonstrate how the values of these UBs can be estimated in a linear time using an array-based mechanism.

Definition 24. (Utility array UA). Let there be a collection of distinct items I in the transaction dataset D. The UA for D is an array of size $|I|$ and has an entry UA[w] for every item $w \in I$. We call every entry a UA and we use it to capture a utility value [48].

Assuming that there are n transactions in D, we can leverage the UA to compute the proposed UBs in O(n) time complexity as follows:

Computing Ptwu of all distinct items. At first, all entries in the UA are given a zero value. Then, the UA[w] for every item $w \in t_k$ is computed as follows: UA[w] = UA[w] + Ptu(t_k) for every transaction $t_k \in D$. After the termination of the dataset scanning, for each item $y \in I$, the UA[y] contains Ptwu(y). Considering the current example, the Ptwu is calculated for each item in the sample dataset D as shown in Fig. 3. First, we make the size of the UA the same as the number of items in the transaction dataset such that each entry corresponds to one distinct item in the dataset. Then, we set an initial value equal to 0 for each entry in the UA as presented in iteration 1 in Fig. 3. Iteration 2 scans transaction t_1 and updates the UA with the Ptu value. Transaction t_1 contains items a, b, c, and d; hence, only entries correspond to these items are updated with Ptu value 12. Iteration 3 scans the transaction t_2 which contains all the distinct items in D, hence, all entries are updated by adding Ptu (t_2) to the previously existing values in the UA. Similarly, the rest of the transactions are scanned and the UA is updated accordingly. In the end, we have calculated a Ptwu value for every distinct item in D.

Computing PSU of itemset α. Initially, entries in the UA are set to 0. Then, the UA[y] for every item $y \in t_k \cap E(\alpha)$ is computed as: UA[y]= UA[y]+$u(\alpha, t_k) + u(y, t_k) + \sum_{i \in t_k \wedge i \in E(\alpha \cup \{y\})} u(i, t_k)$ for every transaction t_k in the dataset D. After reading all transactions, each entry UA[y] stores PSU(α, y) $\forall y \in I$ such that each item $y \in E(\alpha)$[48].

Computing PLU of itemset α. Initially, entries in the UA are set to 0. Then, the UA[y] for every item $y \in t_k \cap E(\alpha)$ is computed as: UA[y]=UA[y] + $u(\alpha, t_k) + re(\alpha, t_k)$ for every transaction t_k in the dataset D. After the whole set of transactions have been processed, each entry UA[y] stores PLU (α, y) $\forall y \in I$ [48].

4.7. TKN algorithm

In this section, we present a novel algorithm for mining top-k HUIs with the consideration of negative utility values. We have employed many novel ideas that have been demonstrated previously to enhance the overall mining process. We have also adopted an array-based mechanism to compute both the utility and UBs of candidate patterns in a linear time.

Algorithm 4: TKN algorithm: Main

Input: D: A transation dataset, k: the required number of HUIs.
Output: topkHUIs.

1 $\alpha \leftarrow \varnothing$;
2 Let η be the set of promising negative items in D.
3 Let TKHQ be a priority queue of size k.
4 Set minUtil to 0.
5 Scan D to compute Ptwu for all 1-itemsets using UA[x].
6 Compute PRIU values for all positive items $x \notin \eta$ and store them in PRIU_list.
7 Apply PRIU strategy to raise minUtil to k^{th} highest value in PRIU_list. // {refer Algorithm 1}
8 Calculate secondary(α) = $\{x | x \in I \Lambda Ptwu(x) \geqslant minUtil\}$.
9 Sort Secondary(α) as per the total order $\prec$. // {refer Definition 11}
10 Scan D to remove item $x \notin Secondary(\alpha)$ from each transaction t_k and delete empty transactions.
11 Sort the remaining transactions in D according to $\prec$.

12 Scan D to calculate PSU (α, x) for each item $x \in Secondary(\alpha)$ using UA[x].
13 Build LIUS using positive items $x \notin \eta$ in D. [22]
14 Apply PLIU_E strategy. // {refer Algorithm 2}
15 Apply PLIU_LB strategy. // {refer Algorithm 3}
16 Calculate $Primary(\alpha) = \{x | x \in Secoundry(\alpha) \Lambda PSU(\alpha, x) \geqslant minUtil\}$.
17 topkHUIs = Search (α, 0, η, D, Primary(α), Secondary(α), minUtil, TKHQ). // {refer Algorithm 5}

The main procedure of TKN is provided in (Algorithm 4). It requires two parameters as input: 1) a transaction dataset, D; and 2) a user specified parameter k. It outputs the top-k HUIs of D. Line 1 initially sets the current itemset α as empty. Line 2 and 3 create a set for negative items η and a priority queue for the top-k HUIs respectively. Line 4 sets the initial value of the minUtil to 1. Line 5 scans the dataset once to estimate the positive transaction-weighted utility for every item w.r.t α using a UA. It is to be noted that, when α is empty, the Ptwu of an item is equal to its PLU. PRIU values are also calculated for each positive item and stored in a list as explained in Section 4.4.1. PRIU strategy is applied in line 7 to increase the threshold from 0 (Algorithm 1). Line 8 identifies the set of secondary items for itemset α. Line 9 sorts the secondary items of α using the proposed total order of items $\prec$. Line 10 scans the dataset D to eliminate the items that are not exist in secondary(α) and also delete any empty transactions in D. Line 11 sorts all the remaining transactions according to the total order $\prec$. The dataset D is scanned to estimate the PSU for each item in Secondary(α) in line 12. Line 13 builds the LIUS using only positive items $x \notin \eta$ [22]. Line 14 and 15 apply PLIU_E_strategy (Algorithm 2) and PLIU_LB_strategy (Algorithm 3) respectively to raise the minUtil. Line 16 identifies the set of primary items of α. Line 17 invokes the global search procedure (Algorithm 5) to extend itemset α with positive and negative items by running a depth-first search recursively.

The global search procedure requires 8 parameters as input: 1) the current itemset α to be appended with positive and negative items; 2) the prefix utility μ of the current prefix itemset α; 3) the set of negative items η; 4) the projected dataset of the current itemset; 5) primary items of α; 6) secondary items of α; 7) the current minUtil; and 8) priority queue TKHQ. For lines 1–19, a loop is carried out to extend α with each item $z \in primary(\alpha)$. In line 2, β is initialized with $\alpha \cup \{z\}$ to be a single-item extension of α. Line 3 creates the projected dataset of β and computes its utility while applying the EP strategy; in case z is a negative item, using the prefix utility μ. Line 4 checks if the utility of β is no less than the minUtil. If so, β is added to TKHQ and another check is made to raise the minUtil to the k^{th} highest value in TKHQ using the RUC strategy [39]. In line 10, we find the next item that can be used to extend β by scanning the secondary items of α using a binary search method. Line 11 initializes w with the first item to extend β. Line 12 checks if β is low utility and w is a negative item. If so, then no need to compute the UBs of β (EA strategy). Line 15 computes the PSU and PLU of β. Line 16 and 17 identify the primary and secondary set of items of β. Finally, in line 18, the search procedure is recursively called to further extend β with more promising items in a depth-first mode.

Algorithm 5: The global search procedure

Input: α: the prefix itemset, μ: the prefix utility, η: the set of promising negative items in D, $\alpha - D$: projected dataset of α, Primary(α): the primary items of α, Secondary(α): the secondary items of α, minUtil, TKHQ: priority queue of k items.
Output: topkHUIs with prefix α.

1 **foreach** *item* $z \in Primary(\alpha)$ **do**
2 $\beta \leftarrow \alpha \cup \{z\}$;
3 Scan $\alpha - D$, compute U(β) and create $\beta - D$. // {EP strategy}
4 **if** $U(\beta) \geq minUtil$ **then**
5 Add β to TKHQ;
6 **if** $|TKHQ| == k$ **then**
7 raise minUtil to the utility of the top element. // {RUC strategy}
8 **end if**
9 **end if**
10 Find the index i of item z in Secondary (α) using a binary search method.
11 $w \leftarrow$ item at $i + 1$ in Secondary (α).
12 **if** $U(\beta) < minUtil \wedge w \in \eta$ **then**
13 Continue; // {EA strategy}
14 **end if**
15 Scan $\beta - D$ to compute PLU (β, w) and PSU (β, w) where item $w \in$ Secondary (α) using two UAs.
16 Primary(β) = $\{w | w \in Secondary(\alpha) \wedge PSU(\beta, w) \geq minUtil\}$.
17 Secoundry(β) = $\{w | w \in Secondary(\alpha) \wedge PLU(\beta, w) \geq minUtil\}$.
18 Search($\beta, U(\beta), \eta, \beta - D, Primary(\beta), Secondary(\beta), minUtil$,TKHQ). // {explore primary items of β in DFS manner}
19 **end foreach**

	UA[a]	UA[b]	UA[c]	UA[d]	UA[e]
(Iteration 1) Initialization	0	0	0	0	0
(Iteration 2) After reading t_1	12	12	12	12	0
(Iteration 3) After reading t_2	32	32	32	32	20
(Iteration 4) After reading t_3	48	32	48	32	36
(Iteration 5) After reading t_4	48	45	48	45	49
(Iteration 6) After reading t_5	48	61	64	61	65
(Iteration 7) After reading t_6	48	61	68	65	69

Fig. 3. Estimate Ptwu using an array structure.

4.8. Illustrative example

In this section, we provide a straightforward example to elaborate how the suggested TKN is able to discover the top-*k* HUIs with positive and negative profits using the dataset *D* given in Table 3. Suppose that we want to find the top-4 HUIs (i.e., k = 4) and the minUtil is initially set to 0. The algorithm first scans *D* and calculates Ptwu values for each distinct item in *D*. Moreover, PRIU values are computed as well as negative items are determined and stored in a dedicated set. Based on the calculated PRIU values, the PRIU strategy is applied.

Considering the current example, there are four positive items {a, b, c, e} with PRIU values {27, 24, 18, 12}. As the 4^{th} highest value is 12, the PRIU strategy will raise the minUtil to 12. Items with Ptwu larger than minUtil are identified as secondary items. The Ptwu values are shown in Table 5. Since all items have Ptwu more than the current minUtil, then Secondary(α)= {a, b, c, d, e}. Next, all items in transactions are sorted as per $\prec$. The ordered transactions are shown in Table 6. The sorted Secondary(α)= {a, b, c, e, d}. After that, any item does not belong to Secondary(α) is removed and blank transactions are deleted from *D*. Thereafter, the suggested TKN scans *D* again to compute PSU for each item in Secondary(α) and build the LIUS using only positive items in transactions. The LIUS of the current example is depicted in Fig. 2. Subsequently, TKN applies the PLIU_E strategy to raise the minUtil using the values stored in the LIU structure. The 4^{th} highest value in the LIU structure after processing all transactions is 28 which is larger than the current minUtil. Hence, the minUtil will be elevated to 28. Afterwards, the PLIU_LB strategy is executed to elevate the minUtil. In the running example, the minUtil is not changed from 28 because it is already close to the optimal minUtil which is 29. Next, only items having PSU no less than minUtil are considered primary items. In the current example, Primary(α)= {a, b, c}. Finally, TKN explores the primary items using a depth-first search approach. The value of the minUtil is increased during exploring the sub-trees of the primary items using the RUC strategy until finding the k itemsets with the largest utilities in the dataset.

5. Performance evaluation

In this section, we exhibit the performance evaluation of the proposed algorithm based on four series of efficiency comparisons. In the first series of comparisons, we check the efficiency of TKN against the state-of-the-art negative top-k HUIM method (THN) by using three versions of the proposed method; so that we can better show the influences of the adopted techniques. In the second series of comparisons, we check the efficiency of TKN against the most recent negative optimal methods (FHN and GHUM). In the third series of comparisons, the efficiency of TKN is checked against the most recent conventional top-*k* HUIM methods (TKEH and THUI). Finally, we check the ability of TKN to scale while the sizes of the datasets are growing compared to the other competitors.

5.1. Experimental environment & datasets

The conducted experiments entail two versions of six real-world datasets that are commonly used in HUI mining literature, namely retail, mushroom, accidents, chess, pumsb, and connect. The first version contains positive and negative utility values whereas the second contains only positive utility values. These datasets were acquired from the data mining library SPMF [8]. We provide a brief information for these datasets in Table 7. From this table one can observe that the adopted datasets have different sizes and nature (sparse, dense, and large).

Through the experiments, we compared TKN against THN method and the most recent optimal methods which can work on datasets having negative profits (FHN and GHUM) using the first version of the datasets. Then, we conducted experiments to investigate the efficiency of TKN against the most recent conventional top-*k* HUIM methods (THUI and TKEH) using the second version of the datasets.

All algorithms were written using java language by extending the SPMF library. Evaluations were conducted on an intel-based machine having a core i7 processor at 2.1 GHz clock speed, 6 GB RAM and windows 7 OS. We ran all algorithms on the datasets while increasing the value of *k* until they took too much time to execute, depleted all available memory or an obvious dominator was observed.

Table 7
Characteristics of the datasets.

Dataset	#Trans	#Items	Avg. length	Max. length
retail	88,162	16,470	10.3	76
mushroom	8,124	119	23	23
accidents	340,183	468	33.8	51
chess	3,196	75	37	36
pumsb	49,046	2113	74	74
connect	67,557	129	43	43

5.2. Performance analysis of TKN against THN

In the first series of experiments, we compared the efficiency of TKN to the most recent negative top-*k* HUIM algorithm (THN) using the negative version of the six benchmark datasets. We used a modified version of THN by replacing the RTWU strategy with the PRIU strategy; since the former can result in an incomplete solution. Also, we examined the influence of the adopted techniques by developing three variants of TKN. The three variants are TKN(PRIU), TKN(PSU) and TKN(TM). TKN (PRIU) uses all the adopted techniques except the LIUS-based threshold raising strategies. TKN(PSU) utilizes all except the transaction merging technique, while TKN(TM) indicates applying all except the PSU pruning strategy. Fig. 4 presents the runtime performance results of the compared methods. The results indicate that the suggested approach is obviously superior compared to THN on all the studied datasets. It can also be observed that the three variants of TKN can not provide higher speeds than TKN in most cases. For retail dataset, TKN is found to be 20 times faster than THN, while TKN(PSU) gives better performance (2 times faster) than TKN since the transaction merging technique is less effective on datasets with highly sparse values. On mushroom, TKN provides 2 times better performance than THN, while TKN(PSU) underperforms both TKN and TKN(TM) which indicates the efficacy of the transaction merging technique on datasets with moderately long transactions like mushroom. For chess dataset, TKN gives almost an order of magnitude better performance than THN, while TKN and TKN(PRIU) show similar trends at higher k values. On accidents, pumsb and connect datasets, TKN gives more than an order of magnitude better performance than THN. It can be observed that TKN clearly dominates the other three variants on the highly dense datasets (pumsb and connect). This can be attributed to the effective threshold raising strategies utilized by TKN. Overall, the main reason behind the efficiency of TKN over THN is that the proposed method utilizes the global search procedure which makes use of the generic pruning properties to skip a huge number of candidates by checking both positive and negative single-item extensions in each iteration rather than iterating through all negative items for each positive HUI in a dedicated procedure as performed in THN. In addition, TKN applies powerful threshold increasing strategies in the early stages of mining, which brings a significant speed-up over THN. Besides, it employs two effective strategies (EP and EA) to reduce both the computational burden of building the projected datasets and the number of evaluations needed to compute the upper bounds respectively. To better see the efficacy of the devised search procedure, we tabulated the number of evaluations needed to estimate the upper bounds for TKN, TKN(PRIU) and THN in Table 8. It is interesting to observe that, even without the LIUS-based strategies, the number of evaluations for THN is significantly more than that of TKN and TKN (PRIU). Fig. 5 presents the results of our memory usage experiments. On retail dataset, it can be observed that TKN, TKN

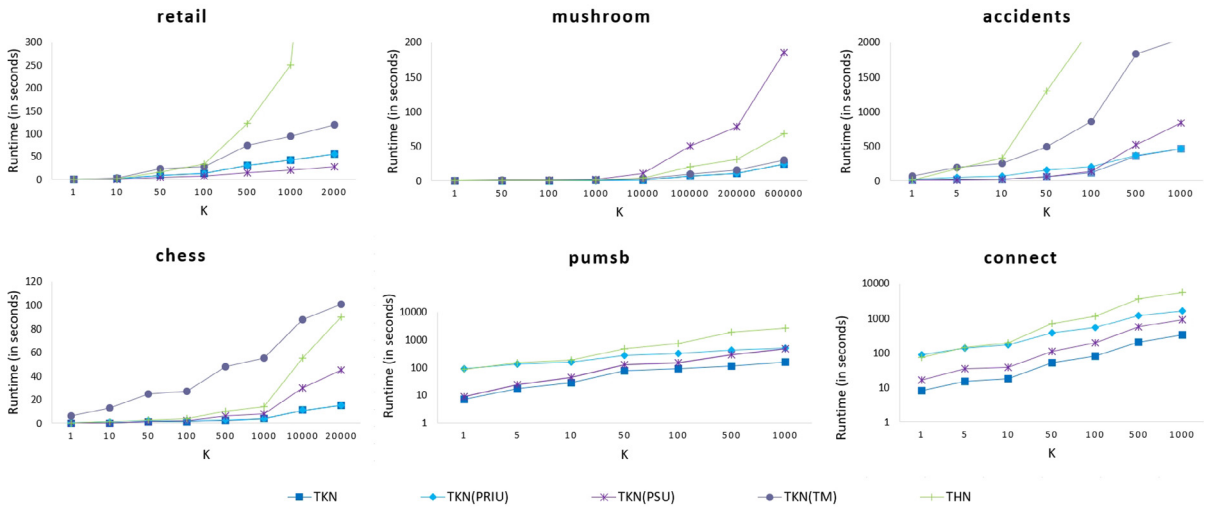

Fig. 4. Runtime performance of TKN variations against THN.

Table 8
Number of evaluations needed to estimate the upper bounds for TKN, TKN(PRIU) and THN.

K	retail			mushroom			accidents			chess			pumsb			connect		
	TKN	TKN(PRIU)	THN	TKN	TKN(PRIU)	THN	TKN	TKN(PRIU)	THN	TKN	TKN(PRIU)	THN	TKN	TKN(PRIU)	THN	TKN	TKN(PRIU)	THN
1	1	1	1	11	58	58	13	123	567	249	1314	1305	240	15205	15175	540	14435	14307
5	26	26	601	102	427	1372	99	677	6137	619	3313	6860	1087	25256	49383	1313	27038	50006
10	58	58	3175	249	705	2523	158	1721	18242	875	4570	12316	2168	33129	84877	1536	36941	84675
30	515	515	65573	1248	2645	11969	449	5093	84437	2353	17797	58792	5668	64013	236940	3924	92574	326239
50	947	947	152382	2104	6611	31086	939	8734	163681	3649	27883	94770	6829	100596	421692	4924	163623	738258
100	1713	1713	440094	6416	17285	75597	3439	25102	615051	6533	40133	145771	9288	223381	1185518	8756	314585	1710837
500	5540	5540	4806872	64791	64791	235318	99476	229043	3566848	56446	112167	412096	42852	1403893	10677742	28319	961093	6372993
1000	9870	9870	13700732	114578	114578	382794	400128	400128	5249795	184685	184685	663898	111705	2844806	26417815	47555	1598588	11237366

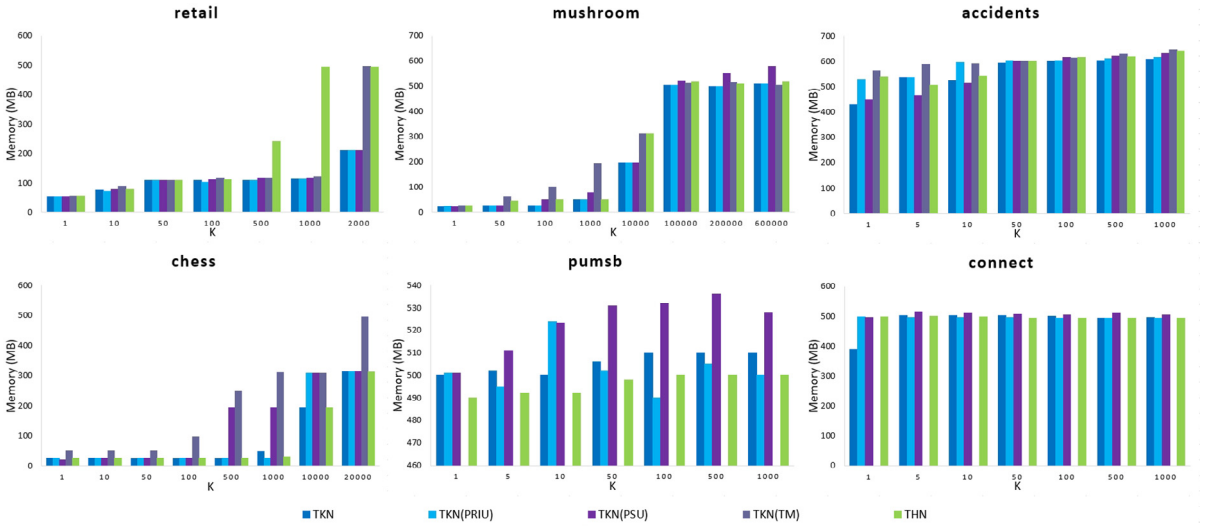

Fig. 5. Memory consumption of TKN variations against THN.

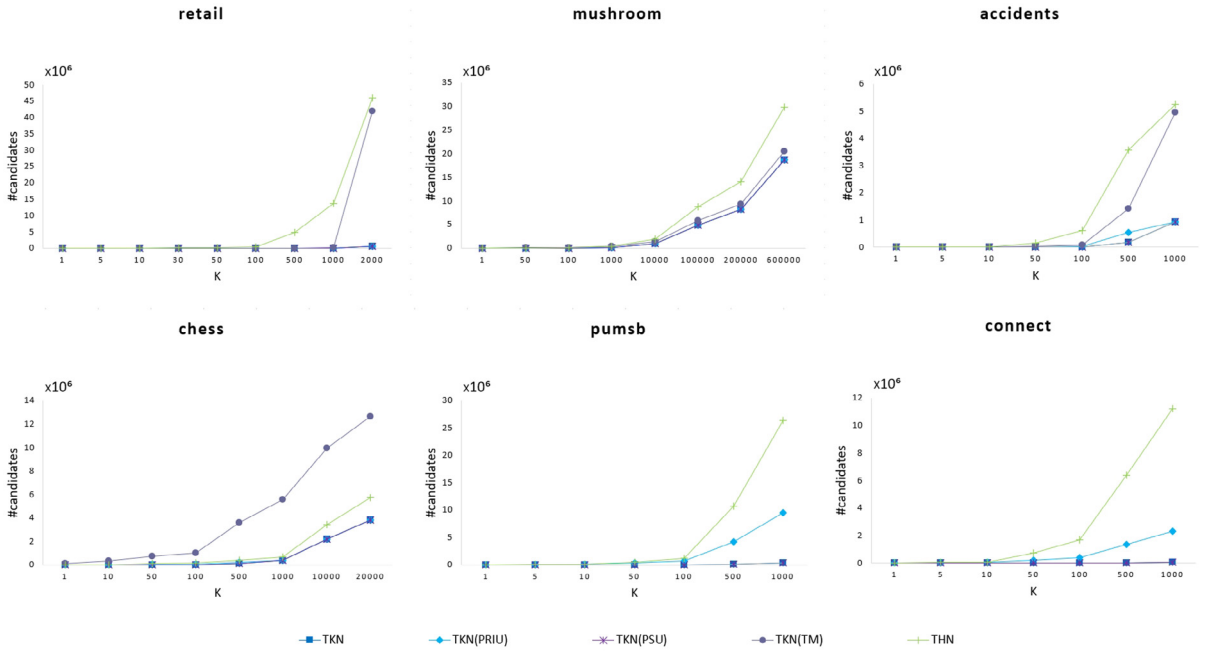

Fig. 6. Performance analysis of candidate sizes of TKN variations against THN.

Table 9
Percentages of candidates that were early pruned at different k values by the proposed EP strategy.

Dataset	K						
	100	200	300	400	500	1000	2000
retail	1	3	4	5	5	7	82
mushroom	53	56	55	55	55	54	52
accidents	50	51	51	52	53	70	70
chess	49	51	50	52	16	33	55
pumsb	61	72	77	80	81	82	82
connect	18	21	23	24	25	28	31

Table 10
Percentages of upper bounds evaluations reductions at different k values by the proposed EA strategy.

Dataset	K						
	100	200	300	400	500	1000	2000
retail	1.6	1.7	2	2.6	2.8	3.8	1
mushroom	0.5	0.3	0.2	0.2	0.2	0.2	0.2
accidents	24.6	23.9	25	27.5	27.3	14.6	14.4
chess	5	7.8	9.4	9.3	8.5	5	5.5
pumsb	1.5	2	2.2	2.1	2.1	1.8	3
connect	0.4	0.7	1	1.2	1.3	1.7	2

(PRIU) and TKN(PSU) consume much less memory than THN and TKN(TM). This can be blamed to the inefficient search procedure of THN, which incurs a lot of computational overhead, and the low effectiveness of the transaction merging technique on highly sparse datasets like retail. For chess and pumsb datasets, the memory consumption of TKN(PSU) is quite higher than the other competitors, which indicates that transaction merging is so effective on highly dense datasets. Fig. 6 compares the number of candidates generated by each method. It can be viewed that TKN always generates the least number of candidates which means that all the adopted techniques contribute to the efficiency of TKN. Table 9 provides the percentage of candidates that can be pruned using the EP strategy on each studied dataset. The estimated percentages indicate that TKN can perform early pruning for up to 82% of candidates at different k values, which can save a considerable amount of time. Table 10 provides the percentages of the upper bounds evaluations that can be skipped using the proposed EA strategy.The results emphasize that the EA strategy can enhance the overall performance of TKN on some datasets. In general, consulting the EA strategy before evaluating the upper bounds can reduce up to 28% of the number of evaluations needed to compute the upper bounds.

5.3. Performance analysis of TKN against negative optimal algorithms

In the second series of experiments, we compared the efficiency of TKN to the corresponding non-top-*k* methods (FHN and GHUM) that can handle items with negative profits using the negative version of the six benchmark datasets. Since FHN and GHUM were not developed for the top-*k* mining, we cannot compare TKN directly with them. To conduct the comparison, we examined the case when users select the optimal minUtil to generate the same number of HUIs as TKN. Fig. 7 presents the runtime performance results of the compared methods. For accidents dataset, TKN gives better performance than FHN and close performance to GHUM. On chess dataset, TKN gives similar performance to GHUM and better performance than FHN. For mushroom, pumsb and connect datasets, TKN delivers better performance than the compared algorithms. On pumsb dataset, TKN is found to be twofold faster than GHUM and up to an order of magnitude faster than FHN. For retail dataset, TKN shows less competitive runtime performance at higher k values. The reason why TKN needs more time on retail dataset is that it contains too many distinct items, which makes the transaction merging technique adopted by TKN less effective and takes much time. As expected, the proposed algorithm has better performance on dense

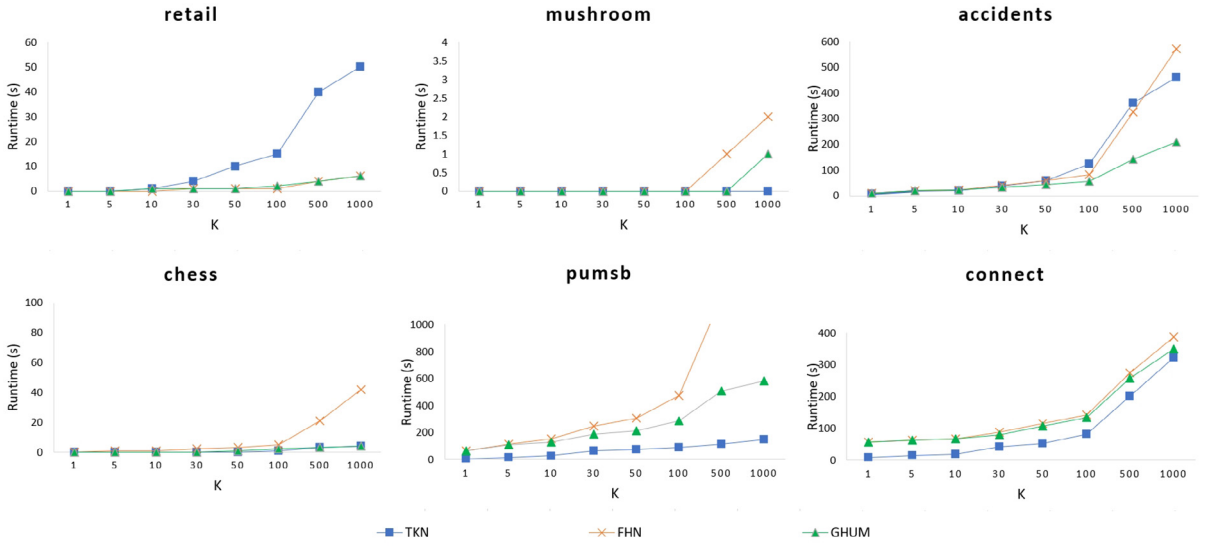

Fig. 7. Runtime performance of TKN against FHN and GHUM.

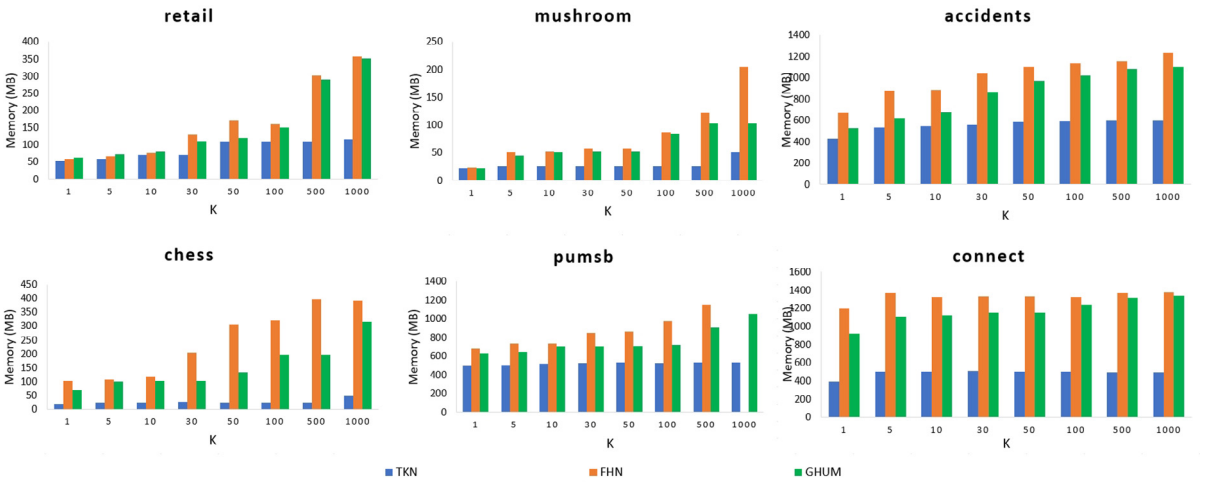

Fig. 8. Memory consumption of TKN against FHN and GHUM.

datasets as the average and maximum length of transactions are closer. Hence, there are higher opportunities to discover identical transactions and longer transactions can be merged in the projected datasets. Fig. 8 presents the results of our memory usage experiments. These results reflect the remarkable efficiency of the proposed TKN. On all benchmark datasets, the proposed algorithm demands lower memory than the most recent non-top-*k* methods in their optimal case. The main reason is that, unlike FHN and GHUM, TKN does not use a complex data structure to maintain the utility information of itemsets. Moreover, the proposed threshold increasing techniques and pruning properties allow TKN to eliminate a huge part of the exploration space. Furthermore, the utilized transaction merging mechanism is very effective in reducing the sizes of the projected datasets.

5.4. Performance analysis of TKN against conventional top-k HUIM algorithms

In the third series of experiments, we compared the efficiency of TKN to the most recent positive top-*k* methods (TKEH and THUI) using the positive version of the benchmark datasets. Fig. 9 presents the runtime performance results of the compared algorithms. The results obviously confirm the substantial enhancement in performance that TKN attains over TKEH and THUI on dense and large datasets. On accidents dataset, TKN is 2 times faster than TKEH and almost 100 times faster

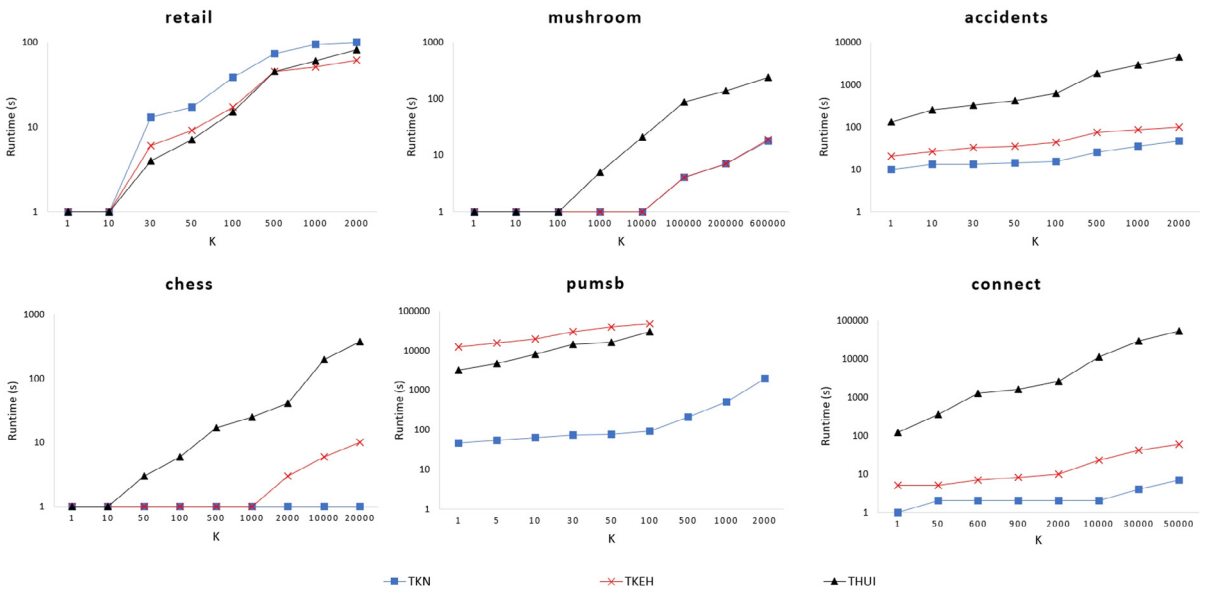

Fig. 9. Runtime performance of TKN against TKEH and THUI.

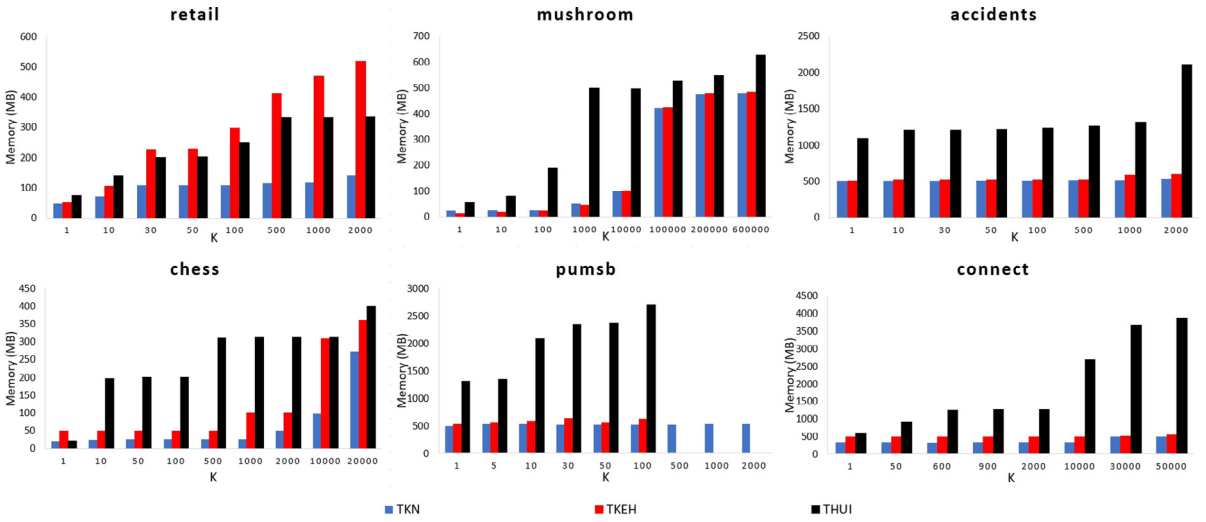

Fig. 10. Memory consumption of TKN against TKEH and THUI.

than THUI. On chess dataset, TKN shows a superior performance (almost 1000 times faster) compared to TKEH and THUI. On the highly dense datasets pumsb and connect, the performance of TKN is far superior (almost 10000 times faster) compared to TKEH and THUI. The extremely downgrade performance of TKEH on the two previous datasets is a consequence of the low effectiveness of the coverage-based threshold raising strategies utilized by TKEH. Although THUI utilizes the LIU structure to raise the minUtil, its performance is found to be very poor when compared to the designed TKN. The main reason is that THUI is a utility-list-based algorithm and thus it suffers the expensive utility list construction process. On mushroom dataset, TKN delivers an order of magnitude better performance than THUI and a similar performance to TKEH. The LIUS-based strategies are found to be less effective on mushroom dataset because it contains too many non-contiguous itemsets. As for retail dataset, the performance of TKEH is observed to be slightly ahead of TKN. This is because TKEH algorithm uses coverage-based threshold increasing strategies. However, the application of these strategies is costly. Although THUI utilizes

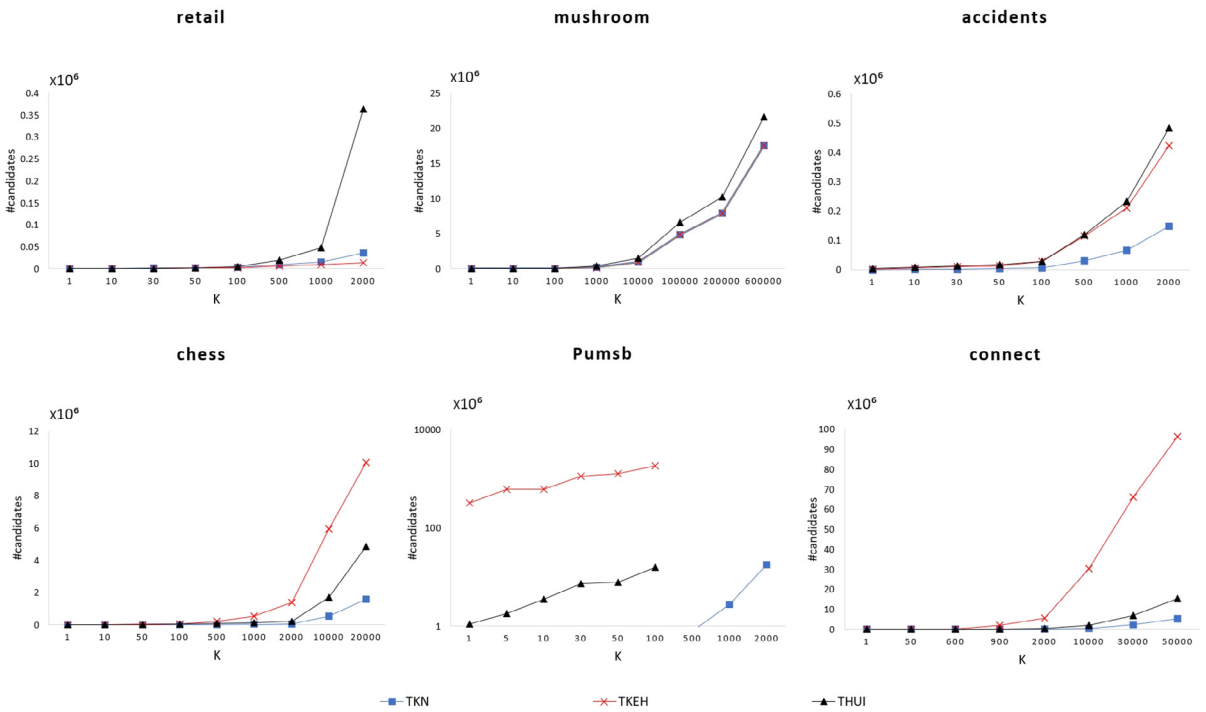

Fig. 11. Performance analysis of candidate sizes of TKN against TKEH and THUI.

Table 11
Enhancements of TKN over THUI and TKEH at the highest k values evaluated.

Dataset	K	Speedup		Frac. of mem.		Frac. of cand.	
		TKEH	THUI	TKEH	THUI	TKEH	THUI
retail	2000	0.61	0.82	0.27	0.42	2.62	0.09
mushroom	600000	1.05	13.05	0.95	0.72	1	0.81
accidents	2000	2.13	94.59	0.89	0.25	0.348	0.3
chess	20000	10	380	0.55	0.68	0.16	0.33
pumsb	100	515.6	318.1	0.84	0.19	0.0001	0.01
connect	50000	8.57	7617.85	0.928	0.13	0.055	0.347

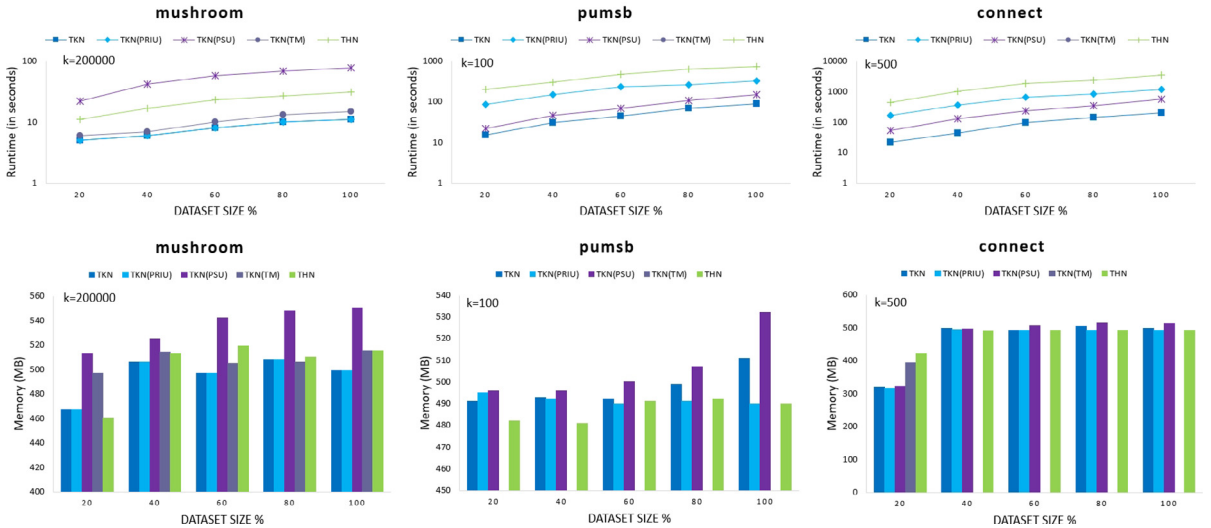

Fig. 12. Scalability experiments of TKN variations against THN.

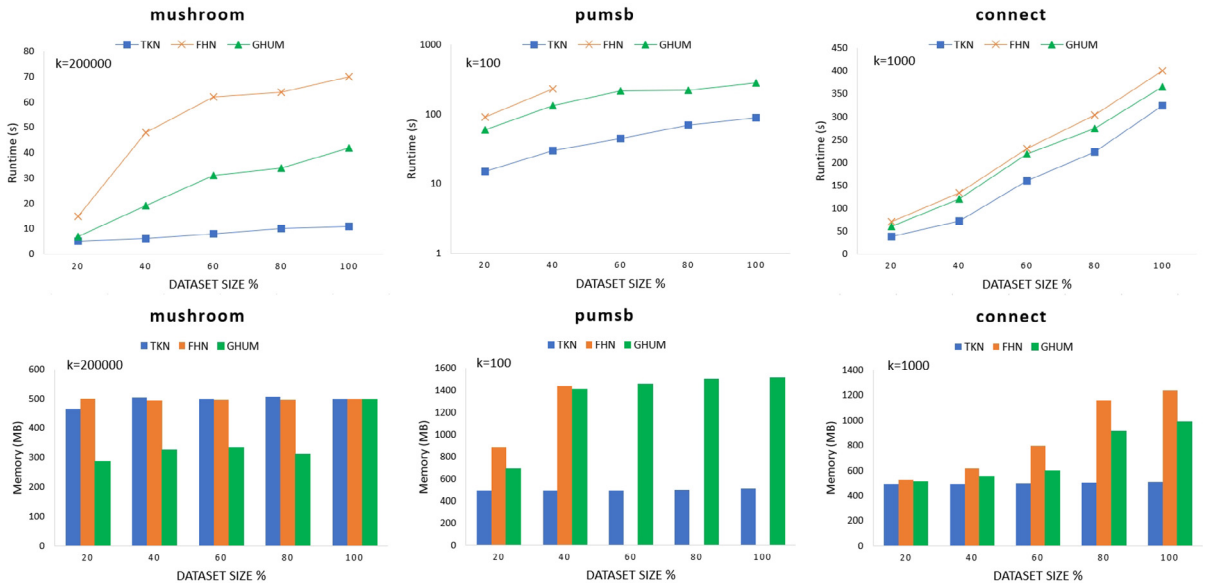

Fig. 13. Scalability experiments of TKN against the negative optimal algorithms.

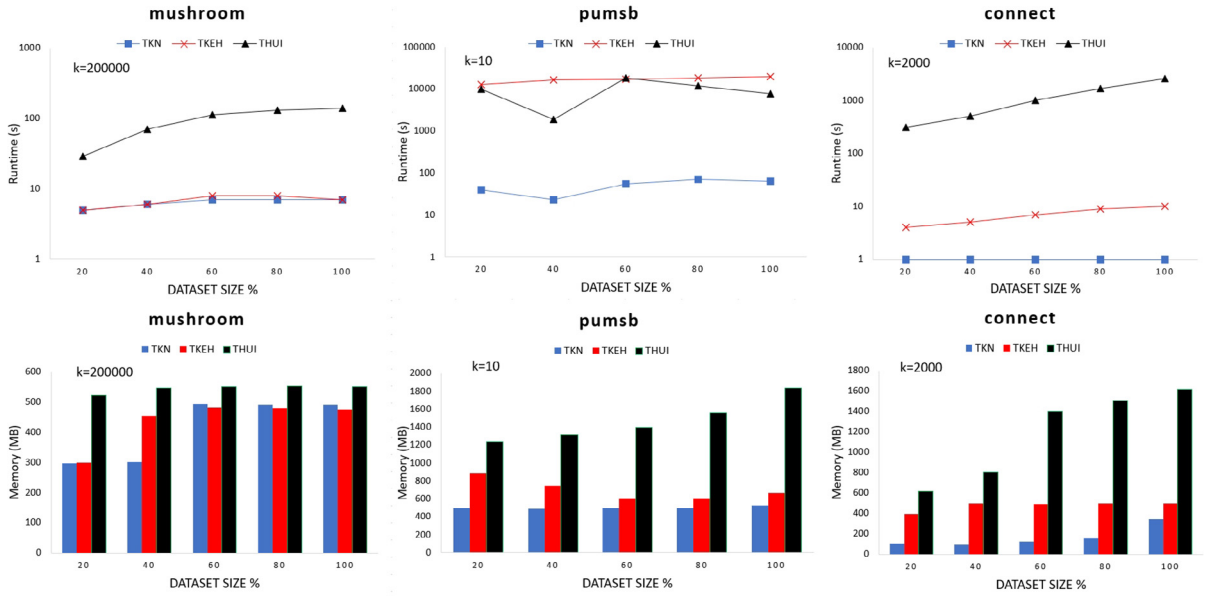

Fig. 14. Scalability experiments of TKN against the conventional top-*k* algorithms.

LIU-based threshold raising strategies like TKN, its performance is found to be slightly better on retail dataset. This is because of the low efficacy of the transaction merging technique adopted by TKN on highly sparse datasets like retail. Fig. 10 presents the results of our memory usage experiments. It can be noted that TKN is the most memory-efficient algorithm on all datasets. The cause is that TKEH algorithm requires an extra memory space to construct and maintain its EUC structure, whereas THUI algorithm requires a large memory space for its utility list. Finally, we performed experiments to measure the candidates sizes of the compared algorithms. From Fig. 11 we can consider that TKN produces the least number of candidates on almost all datasets which clearly reveals the utility of the key strategies adopted in this paper.

Table 11. gives a summary of the performance enhancements of TKN over TKEH and THUI at the highest *k* values. The third column of Table 11 shows that the enhancement in speed is very significant for highly dense datasets. The fraction of memory consumed and candidate patterns produced are provided in the fourth and fifth columns in Table 11. It can be observed that the fraction of memory and candidates are very small on almost all the adopted datasets which implies the efficiency of the strategies proposed in this paper.

5.5. Scalability

In the last series of experiments, we investigated the effect of changing the dataset size on the efficiency of the suggested method. For these experiments, we prepared two versions from three scalable datasets and measured the running time and memory. These scalable datasets were created by varying the sizes of mushroom, pumsb and connect datasets from 20% to 100%. Fig. 12 and Fig. 13 provide the results of our experiments on the negative version of the scalable datasets. The results clearly confirm that TKN has very good scalability compared to THN and the other competitors. Fig. 14 provides the results of our experiments on the positive version of the scalable datasets. From Fig. 14, it can be observed that TKN scales much better on the large and dense datasets compared to TKEH and THUI.

6. Conclusion

Discovering top-*k* HUIs provides the decision-maker great flexibility in identifying the required number of patterns without having to go through a long trial-and-error process to find an appropriate minimum utility threshold. From the current literature of the top-*k* HUIM framework, it appears clearly that most researches focus mainly on mining top-*k* HUIs on positive profit datasets, while few preliminary studies can handle datasets with negative profits. It also comes out that the state-of-the-art algorithms of this framework suffer degrading in performance while running on very dense and large datasets. In this paper, we addressed these challenges by developing a generic top-*k* HUIM approach (TKN) that is efficient in both disciplines. In the proposed approach, dataset projection and transaction merging were employed to compress duplicate transactions and decrease the computational cost of dataset scanning. The concept of leaf itemset utility was leveraged as part of the adapted threshold raising strategies. Moreover, existing pruning properties (sub-tree utility and local utility) were generalized to efficiently narrow the exploration space. In addition, early pruning and abandoning strategies were proposed to

reduce the number of the processed transactions during building the projected datasets of candidates, and avoid unnecessary dataset scans required to estimate the pruning properties. To achieve maximum efficiency in estimating the utility of the itemsets, an array-based mechanism was utilized. Experimental results on two versions of six real datasets showed that TKN provides better flexibility and usability than conventional top-*k* HUIM algorithms as it achieves more than an order of magnitude enhancement in the mining performance when compared to the most recent negative top-*k* HUIM algorithm (THN) and up to 4 orders of magnitude enhancement when compared to the most recent positive top-*k* HUIM algorithms (TKEH and THUI), especially on very dense and large datasets. Furthermore, it delivers very close or better performance when compared to the most recent optimal methods (FHN and GHUM).

For future work, the suggested mining approach can be extended to operate on different top-*k* utility mining variations, including top-*k* local and peak HUIM, periodic HUIM, and closed HUIM.

CRediT authorship contribution statement

Mohamed Ashraf: Conceptualization, Methodology, Software, Visualization, Investigation, Writing - original draft, Writing - review & editing. **Tamer Abdelkader:** Conceptualization, Validation, Writing - review & editing. **Sherine Rady:** Conceptualization, Validation, Writing - review & editing. **Tarek F. Gharib:** Conceptualization, Validation, Writing - review & editing, Supervision.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] R. Agrawal, R. Srikant, in: Fast algorithms for mining association rules in large databases in: Proceedings of the 20th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994, pp. 487–499, https://doi.org/10.5555/645920.672836.
[2] S.A. Ashraf, M.T. Nafis, Fault tolerant frequent patterns mining in large datasets having certain and uncertain records, Adv. Comput. Sci. Technol. 10 (2017) 2381–2394.
[3] C.M. Chen, L. Chen, W. Gan, L. Qiu, W. Ding, Discovering high utility-occupancy patterns from uncertain data, Inf. Sci. 546 (2021) 1208–1229, https://doi.org/10.1016/j.ins.2020.10.001.
[4] H.J. Choi, C.H. Park, Emerging topic detection in twitter stream based on high utility pattern mining, Expert Syst. Appl. 115 (2019) 27–36, https://doi.org/10.1016/j.eswa.2018.07.051.
[5] C.J. Chu, V.S. Tseng, T. Liang, An efficient algorithm for mining high utility itemsets with negative item values in large databases, Appl. Math. Comput. 215 (2009) 767–778, https://doi.org/10.1016/j.amc.2009.05.066.
[6] T.L. Dam, K. Li, P. Fournier-Viger, Q.H. Duong, An efficient algorithm for mining top-k on-shelf high utility itemsets, Knowl. Inf. Syst. 52 (2017) 621–655, https://doi.org/10.1007/s10115-016-1020-2.
[7] Q.H. Duong, B. Liao, P. Fournier-Viger, T.L. Dam, An efficient algorithm for mining the top-k high utility itemsets, using novel threshold raising and pruning strategies, Knowl.-Based Syst. 104 (2016) 106–122, https://doi.org/10.1016/j.knosys.2016.04.016.
[8] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C.W. Wu, V.S. Tseng, SPMF: a java open-source pattern mining library, J. Mach. Learn. Res. 15 (2015) 3389–3393.
[9] P. Fournier-Viger, C.W. Wu, S. Zida, V.S. Tseng, Fhm: Faster high-utility itemset mining using estimated utility co-occurrence pruning, in: International symposium on methodologies for intelligent systems, Springer, 2014. pp. 83–92. doi: 10.1007/978-3-319-08326-1_9..
[10] P. Fournier-Viger, Y. Zhang, J. Chun-Wei Lin, H. Fujita, Y.S. Koh, Mining local and peak high utility itemsets, Inf. Sci. 481 (2019) 344–367, https://doi.org/10.1016/j.ins.2018.12.070.
[11] W. Gan, J.C.W. Lin, P. Fournier-Viger, H.C. Chao, V.S. Tseng, S.Y. Philip, A survey of utility-oriented pattern mining, IEEE Trans. Knowl. Data Eng. 33 (2019) 1306–1327, https://doi.org/10.1109/TKDE.2019.2942594.
[12] W. Gan, S. Wan, J. Chen, C.M. Chen, L. Qiu, Tophui: Top-k high-utility itemset mining with negative utility, in: 2020 IEEE International Conference on Big Data (Big Data), 2020, pp. 5350–5359, https://doi.org/10.1109/BigData50022.2020.9378288.
[13] R. Gunawan, E. Winarko, R. Pulungan, A BPSO-based method for high-utility itemset mining without minimum utility threshold, Knowl.-Based Syst. 190 (2020), https://doi.org/10.1016/j.knosys.2019.105164.
[14] X. Han, X. Liu, J. Li, H. Gao, Efficient top-k high utility itemset mining on massive data, Inf. Sci. 557 (2021) 382–406, https://doi.org/10.1016/j.ins.2020.08.028.
[15] H.M. Huynh, L.T. Nguyen, B. Vo, A. Nguyen, V.S. Tseng, Efficient methods for mining weighted clickstream patterns, Expert Syst. Appl. 142 (2020), https://doi.org/10.1016/j.eswa.2019.112993 112993.
[16] W. Jentner, D.A. Keim, Extracting potentially high profit product feature groups by using high utility pattern mining and aspect based sentiment, Analysis 51 (2019) 303–337, https://doi.org/10.1007/978-3-030-04921-8.
[17] H. Kim, U. Yun, Y. Baek, J. Kim, B. Vo, E. Yoon, H. Fujita, Efficient list based mining of high average utility patterns with maximum average pruning strategies, Inf. Sci. 543 (2021) 85–105, https://doi.org/10.1016/j.ins.2020.07.043.
[18] J. Kim, U. Yun, E. Yoon, J.C.W. Lin, P. Fournier-Viger, One scan based high average-utility pattern mining in static and dynamic databases, Future Gener. Comput. Syst. 111 (2020) 143–158, https://doi.org/10.1016/j.future.2020.04.027.
[19] S. Krishnamoorthy, Pruning strategies for mining high utility itemsets, Expert Syst. Appl. 42 (2015) 2371–2381, https://doi.org/10.1016/j.eswa.2014.11.001.
[20] S. Krishnamoorthy, Efficiently mining high utility itemsets with negative unit profits, Knowl.-Based Syst. 145 (2018) 1–14, https://doi.org/10.1016/j.knosys.2017.12.035.
[21] S. Krishnamoorthy, A comparative study of top-k high utility itemset mining methods, in: High-Utility Pattern Mining, 2019. Springer, pp. 47–74. doi: 10.1007/978-3-030-04921-8_2..
[22] S. Krishnamoorthy, Mining top-k high utility itemsets with effective threshold raising strategies, Expert Syst. Appl. 117 (2019) 148–165, https://doi.org/10.1016/j.eswa.2018.09.051.
[23] G.C. Lan, T.P. Hong, V.S. Tseng, An efficient projection-based indexing approach for mining high utility itemsets, Knowl. Inf. Syst. 38 (2014) 85–107, https://doi.org/10.1007/s10115-012-0492-y.

[24] Y.C. Li, J.S. Yeh, C.C. Chang, Isolated items discarding strategy for discovering high utility itemsets, Data Knowl. Eng. 64 (2008) 198–217, https://doi.org/10.1016/j.datak.2007.06.009.
[25] J.C.W. Lin, Y. Djenouri, G. Srivastava, U. Yun, P. Fournier-Viger, A predictive ga-based model for closed high-utility itemset mining, Appl. Soft Comput. 108 (2021), https://doi.org/10.1016/j.asoc.2021.107422 107422.
[26] J.C.W. Lin, P. Fournier-Viger, W. Gan, FHN: an efficient algorithm for mining high-utility itemsets with negative unit profits, Knowl.-Based Syst. 111 (2016) 283–298, https://doi.org/10.1016/j.knosys.2016.08.022.
[27] M. Liu, J. Qu, Mining high utility itemsets without candidate generation, ACM International Conference Proceeding Series 55–64 (2012), https://doi.org/10.1145/2396761.2396773.
[28] Y. Liu, W.K. Liao, A. Choudhary, A fast high utility itemsets mining algorithm. Proceedings of the 1st International Workshop on Utility-Based Data Mining, UBDM '05, 90–99, 2005. doi: 10.1145/1089827.1089839..
[29] A. Marshall, From Principles of Economics, 2005. pp. 195–215. doi: 10.1142/9789812701275_0021..
[30] L.T. Nguyen, V.V. Vu, M.T. Lam, T.T. Duong, L.T. Manh, T.T. Nguyen, B. Vo, H. Fujita, An efficient method for mining high utility closed itemsets, Inf. Sci. 495 (2019) 78–99, https://doi.org/10.1016/j.ins.2019.05.006.
[31] P.P. Reddy, R.U. Kiran, K. Zettsu, M. Toyoda, P.K. Reddy, M. Kitsuregawa, Discovering spatial high utility frequent itemsets in spatiotemporal databases, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 11932 LNCS, 287–306, 2019, https://doi.org/10.1007/978-3-030-37188-3_17.
[32] H. Ryang, U. Yun, Top-k high utility pattern mining with effective threshold raising strategies, Knowl.-Based Syst. 76 (2015) 109–126, https://doi.org/10.1016/j.knosys.2014.12.010.
[33] K.K. Sethi, D. Ramesh, High average-utility itemset mining with multiple minimum utility threshold: A generalized approach, Eng. Appl. Artif. Intell. 96 (2020), https://doi.org/10.1016/j.engappai.2020.103933 103933.
[34] K. Singh, A. Kumar, S.S. Singh, H.K. Shakya, B. Biswas, EHNL: an efficient algorithm for mining high utility itemsets with negative utility value and length constraints, Inf. Sci. 484 (2019) 44–70, https://doi.org/10.1016/j.ins.2019.01.056.
[35] K. Singh, H.K. Shakya, A. Singh, B. Biswas, Mining of high-utility itemsets with negative utility, Exp. Syst. 35 (2018) e12296.
[36] K. Singh, S.S. Singh, A. Kumar, B. Biswas, TKEH: an efficient algorithm for mining top-k high utility itemsets, Appl. Intell. 49 (2019) 1078–1097, https://doi.org/10.1007/s10489-018-1316-x.
[37] R. Sun, M. Han, C. Zhang, M. Shen, S. Du, Mining of top-k high utility itemsets with negative utility, J. Intell. Fuzzy Syst. (2021) 1–16.
[38] V.S. Tseng, B.E. Shie, C.W. Wu, P.S. Yu, Efficient algorithms for mining high utility itemsets from transactional databases, IEEE Trans. Knowl. Data Eng. 25 (2013) 1772–1786, https://doi.org/10.1109/TKDE.2012.59.
[39] V.S. Tseng, C.W. Wu, P. Fournier-Viger, P.S. Yu, Efficient Algorithms for Mining Top-K High Utility Itemsets, IEEE Trans. Knowl. Data Eng. 28 (2016) 54–67, https://doi.org/10.1109/TKDE.2015.2458860.
[40] H.Q. Vu, G. Li, R. Law, Discovering highly profitable travel patterns by high-utility pattern mining, Tour. Manage. 77 (2020), https://doi.org/10.1016/j.tourman.2019.104008 104008.
[41] C.W. Wu, B.E. Shie, V.S. Tseng, P.S. Yu, in: Mining top-k high utility itemsets, in Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Association for Computing Machinery, New York, NY, USA, 2012, pp. 78–86, https://doi.org/10.1145/2339530.2339546.
[42] R. Wu, Z. He, Top-k high average-utility itemsets mining with effective pruning strategies, Appl. Intell. 48 (2018) 3429–3445, https://doi.org/10.1007/s10489-018-1155-9.
[43] H. Yao, H.J. Hamilton, Mining itemset utilities from transaction databases, Data Knowl. Eng. 59 (2006) 603–626, https://doi.org/10.1016/j.datak.2005.10.004.
[44] U. Yun, H. Nam, J. Kim, H. Kim, Y. Baek, J. Lee, E. Yoon, T. Truong, B. Vo, W. Pedrycz, Efficient transaction deleting approach of pre-large based high utility pattern mining in dynamic databases, Future Gener. Comput. Syst. 103 (2020) 58–78, https://doi.org/10.1016/j.future.2019.09.024.
[45] U. Yun, H. Ryang, K.H. Ryu, High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates, Expert Syst. Appl. 41 (2014) 3861–3878, https://doi.org/10.1016/j.eswa.2013.11.038.
[46] M.J. Zaki, Scalable algorithms for association mining, IEEE Trans. Knowl. Data Eng. 12 (2000) 372–390, https://doi.org/10.1109/69.846291.
[47] C. Zhang, Z. Du, W. Gan, P.S. Yu, Tkus: Mining top-k high utility sequential patterns, Inf. Sci. 570 (2021) 342–359, https://doi.org/10.1016/j.ins.2021.04.035.
[48] S. Zida, P. Fournier-Viger, J.C.W. Lin, C.W. Wu, V.S. Tseng, Efim: a highly efficient algorithm for high-utility itemset mining, in: Mexican international conference on artificial intelligence, Springer, 2015, pp. 530–546, https://doi.org/10.1007/978-3-319-27060-9_44.