Expert Systems With Applications 117 (2019) 148–165

Contents lists available at ScienceDirect

Expert Systems With Applications

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

Mining top-k high utility itemsets with effective threshold raising strategies

Srikumar Krishnamoorthy

Indian Institute of Management, Ahmedabad, India

ARTICLE INFO

Article history:
Received 6 July 2018
Revised 25 September 2018
Accepted 25 September 2018
Available online 27 September 2018

Keywords:
Top-K high utility itemsets
Threshold raising strategies
Utility estimation method

ABSTRACT

Top-K High Utility Itemset (HUI) mining problem offers greater flexibility to a decision maker in specifying her/his notion of item utility and the desired number of patterns. It obviates the need for a decision maker to determine an appropriate minimum utility threshold value using a trial-and-error process. The top-k HUI mining problem, however, is more challenging and requires use of effective threshold raising strategies. Several threshold raising strategies have been proposed in the literature to improve the overall efficiency of mining top-k HUIs. This paper advances the state-of-the-art and presents a new Top-K HUI method (THUI). A novel Leaf Itemset Utility (LIU) structure and a threshold raising strategy is proposed to significantly improve the efficiency of mining top-k HUIs. A new utility lower bound estimation method is also introduced to quickly raise the minimum utility threshold value. The proposed THUI method is experimentally evaluated on several benchmark datasets and compared against two state-of-the-art methods. Our experimental results reveal that the proposed THUI method offers one to three orders of magnitude runtime performance improvement over other related methods in the literature, especially on large, dense and long average transaction length datasets. In addition, the memory requirements of the proposed method are found to be lower.

1. Introduction

High Utility Itemset (HUI) mining is one of the key problems in the data mining literature. The HUIs are useful in wide variety of applications such as retail (product assortment planning, promotion planning, shelf space allocations), e-commerce (personalization, review mining), software (error pattern detection) and business process mining. The HUI mining problem primarily involves determining the set of all itemsets that satisfy a user specified minimum utility threshold value. The utility of an individual itemset is defined in the literature (Liu & Qu, 2012; Liu, Liao, & Choudhary, 2005) as a function of the quantity of items purchased and the profitability of items. This problem is a generalized version of the frequent itemset mining problem (Agrawal & Srikant, 1994) and has been proven to be computationally challenging due to the lack of anti-monotonic property of HUIs (Liu & Qu, 2012; Liu et al., 2005; Zida, Fournier-Viger, Lin, Wu, & Tseng, 2017).

A standard HUI mining problem requires specification of a minimum utility threshold value. This is quite challenging as the choice of a threshold value is dependent on the underlying dataset characteristics. Setting too low a threshold value leads to generation of too many patterns that requires manual filtering of itemsets to determine actionable patterns. On the other hand, setting too high a threshold value often results in generation of a very few or zero patterns. For example, in a study conducted by Wu, Shie, Tseng, and Yu (2012), the authors demonstrate that a small change in minimum utility threshold value (0.03% to 0.02% in the chainstore dataset) leads to significantly long execution times. In essence, determining an appropriate threshold value is a highly inefficient trial and error process. Therefore, there is a need for alternate frameworks that can effectively address this problem. A decision maker is often interested in queries of the form: "What are the top-k interesting patterns that can be used for decision making?". Answering this query is non-trivial and has been the focus of recent research in the HUI mining literature (Duong, Liao, Fournier-Viger, & Dam, 2016; Tseng, Wu, Fournier-Viger, & Philip, 2016).

Several approaches have been proposed in the literature for efficiently mining top-k HUIs. These approaches can be broadly categorized as one-phase and two-phase methods. In a two-phase method, the potential top-k HUIs are mined in the first phase. Subsequently, in the second phase, the actual top-k HUIs are determined by computing the actual utility values and filtering the false positive top-k itemsets. TKU (Wu et al., 2012) and REPT (Ryang & Yun, 2015) are the two algorithms in the literature that adopt a two-phase method. The more recent algorithms (KHMC

E-mail address: srikumark@iima.ac.in

https://doi.org/10.1016/j.eswa.2018.09.051
0957-4174/

Table 1
Percentage of final minimum utility at different values of K.

Dataset	TKO (Tseng et al., 2016)		KHMC (Duong et al., 2016)	
	$K = 1$	$K = 30$	$K = 1$	$K = 30$
retail	100.00	45.36	100.00	45.36
chain	100.00	97.42	100.00	97.42
kosarak	100.00	45.29	100.00	45.29
accidents	60.39	41.41	60.39	41.41
mushroom	52.90	33.71	61.14	60.23
connect	34.23	22.59	34.23	22.59
chess	42.82	26.34	42.82	40.11
pumsb	30.75	21.04	30.75	21.04

(Duong et al., 2016) and TKO (Tseng et al., 2016)) in the literature adopt a one-phase method. These methods directly mine the top-k HUIs in one phase without generating the intermediate candidates or potential top-k HUIs.

KHMC (Duong et al., 2016) and TKO (Tseng et al., 2016) algorithms leverage a vertical utility list based data structure for efficiently mining top-k HUIs in one phase. These algorithms initially set the minimum utility threshold value to zero and mine the top-k HUIs in the following two sub-stages: (1) scan the database twice to construct 1-itemset utility lists (hereinafter referred as initial stage), and (2) use the generated 1-itemset utility lists to explore the itemset search tree and discover HUIs (hereinafter referred as growth stage). The minimum utility threshold value that is initially set to zero is gradually raised during different stages (i.e. initial or growth stage) of the mining process. It is quite intuitive to understand that the complexity of mining is in the growth stage, where a large itemset search tree is explored to mine the top-k HUIs. It is, therefore, imperative to raise the minimum utility threshold values as quickly as possible to reduce the total number of candidates generated in growth stages.

We argue that the state-of-the-art methods on top-k HUI mining are not very effective in raising the threshold values, especially during the early stages of mining, for dense databases. In order to validate this claim and motivate the need for the current work, we conducted an experimental analysis of the effectiveness of threshold raising strategies of the state-of-the-art methods. We assessed the effectiveness, for a given value of K, as a fraction of the minimum utility threshold value at the end of the initial stage of mining to the final minimum utility threshold value. This measure of effectiveness is quite intuitive and relevant as a higher value obtained during initial stages directly translates to efficient mining during the later growth stages. Our evaluation results of effectiveness of the state-of-the-art algorithms are presented in Table 1. The results are given for mining top-1 and top-30 HUIs across different benchmark datasets. The table shows the minimum utility threshold value at the end of the initial stage of mining HUIs as a percentage of the final minimum utility value. It is evident from the table that the state-of-the-art methods are quite effective in quickly raising the threshold values on sparse datasets such as retail, chain, and kosarak. As the density of the dataset increases (e.g. mushroom, connect, chess, pumsb), the performance of these methods in raising threshold values during early stages significantly declines. It is to be noted that the dense datasets are likely to produce large number of HUIs (in comparison to sparse datasets). Therefore, there is a need to design effective threshold raising strategies for dense datasets and improve the overall efficiency of top-k HUI mining.

Our primary motivation in this paper is to design better threshold raising strategies to significantly improve the performance of top-k HUI mining for dense datasets. We show through rigorous experimental evaluation that our method can raise threshold values close to 100% during early stages of mining, even for highly dense datasets. We also demonstrate that our early threshold raising strategy significantly reduces the number of candidates generated in the growth stage and improves the efficiency of mining top-k HUIs.

Our key contributions in this paper are as follows:

1. We present a new method (THUI) for efficiently mining top-k high utility itemsets from transactional databases. Our method uses a novel Leaf Itemset Utility (LIU) data structure for storing utility information of itemsets. The proposed LIU structure requires an extremely small memory footprint for storing itemset information.
2. A new threshold raising strategy (LIU-Exact utilities) is proposed to effectively raise the minimum utility threshold value during the early stages of mining. The proposed strategy leverages the information stored in LIU for raising the minimum utility value.
3. A novel utility lower bound estimation method (LIU-LB) is proposed. This method aids in significantly increasing the minimum utility threshold value without computing the actual utility value of long itemsets.
4. Rigorous experimental evaluation against two state-of-the-art methods (KHMC (Duong et al., 2016) and TKO (Tseng et al., 2016)) is conducted to demonstrate the utility of the proposed ideas. Our experimental evaluation of the effectiveness of threshold raising strategies shows very promising results. The results also reveal that our method is one to three orders of magnitude faster compared to other methods in the literature, especially on dense datasets.

The rest of the paper is organized as follows. Section 2 reviews the related work and outlines the key differences of the current work. Section 3 describes the key definitions and notations and presents the problem statement. The proposed threshold raising strategy, lower bound estimation method and the new top-k HUI mining algorithm (THUI) are described in Section 4. Subsequently, the experimental results are presented and discussed in Section 5. A comparative evaluation of THUI against the state-of-the-art TKO (Tseng et al., 2016) and KHMC (Duong et al., 2016) methods is also made in Section 5. Finally, the concluding remarks and future research directions are presented in Section 6.

2. Related work

In this section, we review the literature on HUI mining, top-k HUI mining and their variants. We also clearly highlight the key differences of the proposed method from prior works in the literature.

It is to be noted that several n-most (or top-k) algorithms have been proposed in the frequent itemset mining literature (Cheung & Fu, 2004; Quang, Oyanagi, & Yamazaki, 2006; Salam & Khayal, 2012). These algorithms discover the top-k frequent patterns without using any user defined minimum support value. The proposed top-k mining problem is quite different and has also been shown to be more challenging compared to top-k frequent pattern mining problems (Duong et al., 2016; Tseng et al., 2016). Hence, we restrict our review primarily to HUI mining related works in the literature.

2.1. High utility itemset mining

The two-phase algorithm (Liu et al., 2005) is one of the earliest works in the area of high utility itemset mining. The authors introduced the HUI mining problem and presented a two phase approach to efficiently mine HUIs. The algorithm mined HUIs in two phases, namely: (1) candidate HUI generation phase, and (2) actual HUI determination phase. A Transaction Weighted Utilization (TWU) model was proposed to estimate the utility upper bound

and limit the search space of mining HUIs. Two-phase algorithm and other related methods that primarily relied on TWU model generated too many candidates and suffered from performance and scalability issues (Liu & Qu, 2012; Tseng, Wu, Shie, & Yu, 2010).

Tree-based methods such as UP-Growth (Tseng et al., 2010) and UP-Growth+ (Tseng, Shie, Wu, & Yu, 2013) aimed to address the limitations of the TWU model with the use of improved utility upper bounds for pruning unpromising candidates. Some of the key properties introduced by UP-Growth based methods (Tseng et al., 2013; Tseng et al., 2010) include: Discarding Global Unpromising (DGU) items, Decreasing Global Node (DGN) utilities, Discarding Local Unpromising (DLU) items, Decreasing Local Node (DLN) utilities, Discarding local unpromising items and estimated Node Utilities (DNU) and Decreasing local Node utilities by estimated utilities of descendant Nodes (DNN).

The more recent methods in the literature utilize a vertical database representation for efficiently mining HUIs (HUI-Miner (Liu & Qu, 2012), FHM (Fournier-Viger, Wu, Zida, & Tseng, 2014b), and HUP-Miner (Krishnamoorthy, 2015)). These methods are proven to be superior compared to the earlier level-wise (Liu et al., 2005) and tree-based (Tseng et al., 2013) methods. They apply efficient utility list based intersections and pruning strategies (such as U-Prune (Liu & Qu, 2012), EUCS-Prune (Fournier-Viger et al., 2014b), LA-Prune (Krishnamoorthy, 2015)) to significantly improve the performance of HUI mining. d^2HUP (Liu, Wang, & Fung, 2012) introduced a hyper-linked data structure, named CAUL, to efficiently mine HUIs.

EFIM (Zida et al., 2017) algorithm uses database projection, transaction merging and fast utility computation methods to improve the efficiency of mining HUIs. EFIM is one of the state-of-the-art methods for efficiently mining HUIs. The authors use a horizontal database representation for combining duplicate transactions in the database. The algorithm is shown to be highly effective (two to three orders of magnitude superior) on dense benchmark datasets. HMiner (Krishnamoorthy, 2017) uses a compact utility list data structure to combine duplicate transactions present in the database. It also uses a few new pruning strategies to improve the efficiency of mining HUIs. The authors demonstrate that their method offers better runtime performance over EFIM on multiple benchmark sparse and dense datasets.

Several variants of the basic HUI mining problem have also been extensively studied in the literature. Some of these variants include: On-shelf utility patterns (Fournier-Viger & Zida, 2015; Lan, Hong, & Tseng, 2011), HUIs with negative unit profits (Lin, Fournier-Viger, & Gan, 2016a), HUIs with multiple minimum utility thresholds (Lin, Gan, Fournier-Viger, Hong, & Zhan, 2016b), data stream mining (Ryang & Yun, 2016), sequential patterns (Fournier-Viger, Lin, Kiran, Koh, & Thomas, 2017), up-to-date high utility patterns (Lin, Gan, Hong, & Tseng, 2015) and high average utility patterns (Lin et al., 2017). A comprehensive survey of high utility itemset mining methods can be referred in Gan et al. (2018).

All of the above methods require specification of a minimum utility threshold value for mining HUIs. But, the proposed method uses the desired number of HUIs as input and starts with a minimum utility threshold value of zero.

2.2. Top-K high utility itemset mining

Wu et al. (2012) introduced the top-k HUI mining framework and presented a two-phase TKU algorithm. The TKU algorithm, in the first phase, constructs an UP-Tree (Tseng et al., 2010) and generates the Potential top-K HUIs (PKHUI)s. Subsequently, in the second phase, the unpromising PKHUIs are filtered to determine the actual top-k HUIs. The TKU algorithm uses five different threshold raising strategies namely, Raising the threshold by MIU of Candidates (MC), Pre-Evaluation (PE), Raising the threshold by Node Utilites (NU), Raising the threshold by MIU of Descendants (MD), and Sorting candidates and raising the threshold by the Exact utilites (SE). Each of these strategies is applied at different phases of mining to improve the overall efficiency. More specifically, the PE strategy is applied during the first database scan of phase one. The NU and MD strategies are then applied during the second database scan of phase one. The MC strategy is applied during the growth stage of mining. Subsequently, the SE strategy is applied during the second phase to determine the top-k HUIs from the potential top-k HUIs. The authors extended their work and introduced two new algorithms: TKU and TKO in Tseng et al. (2016). While TKU is a two-phase algorithm, TKO is a one-phase utility list based approach.

The TKO algorithm uses a utility list data structure (Liu & Qu, 2012) for maintaining itemset information during the mining process. The algorithm initially scans the database to compute the TWU and utility values of items. A pre-evaluation (PE) matrix is then constructed and used to raise the minimum utility threshold value. Subsequently, the algorithm constructs 1-itemset utility lists and explores the itemset search tree to mine the top-k HUIs. The TKO algorithm raises the minimum utility threshold value by using the Raising threshold by Utilities of Candidates (RUC) strategy. In order to reduce the number of candidates generated during mining, the TKO algorithm also employed Reducing Utilities by using Z-elements (RUZ) and Exploring the most Promising Branches first (EPB) strategies. The one-phase TKO method has been shown to be superior compared to the baseline two-phase TKU method.

REPT (Ryang & Yun, 2015) is another two phase method for mining top-k HUIs. The algorithm utilizes six different threshold raising strategies (PUD, RIU, RSD, NU, MC and SEP) to quickly raise the minimum utility threshold value. In the first phase, REPT uses the PUD and RIU strategies to raise the utility threshold value. The RSD and NU strategies are then applied to efficiently mine PKHUIs. In the last phase, the SEP strategy is applied to efficiently mine the actual top-k HUIs from PKHUIs. Their approach is proven to be superior compared to the related two-phase TKU method (Tseng et al., 2016). Both REPT and TKU methods utilize UP-Tree (UP-T) data structure for storing utility information.

KHMC (Duong et al., 2016) is the most recent top-k HUI mining method, to the best of our knowledge, that adopts a one-phase utility list based approach. The algorithm first scans the database to compute the TWU and utility values of items. An RIU strategy is then applied to increase the minimum utility threshold value. During the second scan of the database, the algorithm constructs EUCS, CUDM and utility list data structures. The CUD and COV strategies are then applied to raise the threshold value. Subsequently, the 1-item utility lists generated at the end of the second scan of the database is used to explore the search space and mine the top-k HUIs. The RUC strategy is applied during the search tree exploration process to efficiently mine HUIs. The KHMC algorithm also uses three pruning properties during the growth stage of mining namely, utility prune (U-Prune), early abandonment (EA/LA) and transitive extension pruning (TEP). The authors compared their method against the related one-phase TKO method (Tseng et al., 2016) to demonstrate the utility of their core ideas.

2.3. Top-K high utility itemset mining variants

Several variants of the top-k mining problem have also been studied in the recent literature. We briefly review each of these methods in this section.

TUS algorithm (Yin, Zheng, Cao, Song, & Wei, 2013) efficiently mines top-k high utility sequential patterns using two new threshold raising strategies (pre-insertion and sorting) and one pruning

Table 2
Comparison of THUI against prior works in the literature.

Dimension	TKU	REPT	TKO	KHMC	THUI
No. of phases	Two	Two	One	One	One
Data structures	UP-T	UP-T	UL	UL	UL
				EUCS	LIU
				CUDM	
Threshold raising strategy					
Phase I					
Scan 1	PE	PUD	RUC	RIU	RIU
		RIU			
Scan 2	MD	RSD	PE	CUD	LIU-E
	NU	NU		COV	LIU-LB
Growth	MC	MC	RUC	RUC	RUC
Phase II	SE	SEP	–	–	–
Pruning strategy	DGU	TWU	DGU	TWU	TWU
	DGN	DGN	RUZ	EUCP	U-Prune
	DLU	DLU	EPB	U-Prune	LA
	DLN	DLN	U-Prune	LA	
				TEP	

strategy (sequence reduced utility). The TUS algorithm is shown to be superior compared to the state-of-the-art TUSNaive method.

T-HUDS algorithm (Zihayat & An, 2014) discovers top-k high utility patterns over data streams. The algorithm utilizes a HUDS-tree data structure and a new utility estimation method (PrefixUtil) to efficiently mine top-k HUIs. The T-HUDS algorithm follows a two-phase approach. In the first phase, a set of potential top-k HUIs are mined from a HUDS-tree. The actual top-k HUIs are then mined, in the second phase, by calculating the exact utilities of potential itemsets. A one-phase algorithm for top-k HUI mining over data streams was recently proposed by Dawar, Sharma, and Goyal (2017). The authors demonstrate that their approach is superior compared to a two-phase T-HUDS (Zihayat & An, 2014) algorithm on both sparse and dense benchmark datasets.

KOSHU algorithm (Dam, Li, Fournier-Viger, & Duong, 2017) mines top-k on-shelf high utility patterns considering both positive and negative unit profits. The algorithm works in two phases. In the first phase, 1-itemset utility lists are constructed. Subsequently, the 1-itemset utility lists are used to explore the itemset search space and mine all the on-shelf top-k HUIs. The algorithm uses three pruning strategies (EMPRP, PUP, CE2P) and two threshold raising strategies (RIRU, RIRU2). The utility of the proposed ideas are experimentally evaluated on both real and synthetic datasets.

The key ideas proposed in this paper on basic top-k HUI mining can be easily adapted to the above top-k sequential, streaming, and on-shelf variants. A comprehensive survey of top-k HUI mining methods can be referred in Krishnamoorthy (2018).

2.4. Differences from previous works

From the foregoing discussions, it is evident that the proposed work is directly related to four different top-k HUI mining methods, namely TKU (Wu et al., 2012), TKO (Tseng et al., 2016), REPT (Ryang & Yun, 2015) and KHMC (Duong et al., 2016). We tabulate the differences of these methods from the proposed work in Table 2. The table highlights the differences on four key dimensions: (1) number of phases, (2) underlying data structures utilized, (3) threshold raising strategies adopted, and (4) pruning strategies used for limiting the search space. It is evident from the table that our work is closely related to two other one-phase approaches TKO and KHMC. The key differences lie in the use of new data structures and threshold raising strategies.

We propose two new threshold raising strategies, namely LIU-E and LIU-LB, to significantly raise the utility threshold value compared to other state-of-the-art methods. We conduct substantial experiments to demonstrate the utility of the proposed data structures and threshold raising strategies.

Table 3
Sample transaction database.

TID	Transaction	Purchase Qty (IU)	Utility (U)	TU
T_1	a, c, d, e, f	1, 1, 1, 2, 2	5, 1, 2, 6, 2	16
T_2	a, c, e, g	2, 6, 2, 5	10, 6, 6, 5	27
T_3	a, b, c, d, e, f	1, 2, 1, 6, 1, 5	5, 4, 1, 12, 3, 5	30
T_4	b, c, d, e	4, 3, 3, 1	8, 3, 6, 3	20
T_5	b, c, e, g	2, 2, 1, 2	4, 2, 3, 2	11
T_6	a, c, d, e, f	3, 3, 3, 3, 3	15, 3, 6, 9, 3	36
T_7	a, b, c, d, f	1, 1, 1, 2, 3	5, 2, 1, 4, 3	15
T_8	a, b, c, e, f	1, 2, 2, 1, 1	5, 4, 2, 3, 1	15

Table 4
Item profits.

Item	a	b	c	d	e	f	g
Profit $ per unit (EU)	5	2	1	2	3	1	1

3. Preliminaries and problem statement

In this section, we define the key terms used in high utility mining (Ahmed, Tanbeer, Jeong, & Lee, 2009; Liu & Qu, 2012; Liu et al., 2005; Yao & Hamilton, 2006) and formally state the top-k HUI mining problem.

Let $I = \{i_1, i_2 \ldots i_m\}$ be a set of distinct items. A transaction $T_j = \{x_l | l = 1, 2 \ldots N_j, \forall x_l \in I\}$, where N_j is the number of items in transaction T_j. A transaction database D has a set of transactions, $D = \{T_1, T_2 \ldots T_n\}$, where n is the total number of transactions in the database. A sample transaction database D is given in Table 3 and item profit details are provided in Table 4. A set $X = \{x_1, x_2 \ldots x_p\} \subseteq I$ is referred as a p-itemset.

Definition 1. Each item $x_i \in I$ is assigned an external utility value (e.g. profit), referred as $EU(x_i)$. For example, in Table 4, $EU(b) = 2$.

Definition 2. Each item $x_i \in T_j$ is assigned an internal utility value, referred as $IU(x_i, T_j)$. For example, in Table 3, $IU(b, T_4) = 4$.

Definition 3. The utility of an item $x_i \in T_j$, denoted as $U(x_i, T_j)$ is computed as the product of external and internal utilities of items in the transaction, T_j. That is,

$$U(x_i, T_j) = EU(x_i) \cdot IU(x_i, T_j) \tag{1}$$

For example, in Table 3, $U(b, T_4) = EU(b) \cdot IU(b, T_4) = 2 \cdot 4 = 8$.

Definition 4. The utility of an itemset X in transaction T_j ($X \subseteq T_j$) is denoted as $U(X, T_j)$.

$$U(X, T_j) = \sum_{x_i \in X} U(x_i, T_j) \tag{2}$$

For example, in Table 3, $U(\{adef\}, T_1) = 5 + 2 + 6 + 2 = 15$.

Definition 5. The utility of an itemset X in database D is denoted as $U(X)$.

$$U(X) = \sum_{X \subseteq T_j and T_j \in D} U(X, T_j) \tag{3}$$

For example, $U(\{adef\}) = U(\{adef\}, T_1) + U(\{adef\}, T_3) + U(\{adef\}, T_6) = 15 + 25 + 33 = 73$.

Definition 6. The transaction utility, $TU(T_j)$ for transaction T_j is defined as

$$TU(T_j) = \sum_{x_i \in T_j} U(x_i, T_j) \tag{4}$$

Table 5
Transaction weighted utility.

Item	g	b	f	d	a	e	c
TWU	38	91	112	117	139	155	170

Table 6
Ordered transaction history.

TID	Transaction	Utility (U)	TU
T_1	f, d, a, e, c	2, 2, 5, 6, 1	16
T_2	g, a, e, c	5, 10, 6, 6	27
T_3	b, f, d, a, e, c	4, 5, 12, 5, 3, 1	30
T_4	b, d, e, c	8, 6, 3, 3	20
T_5	g, b, e, c	2, 4, 3, 2	11
T_6	f, d, a, e, c	3, 6, 15, 9, 3	36
T_7	b, f, d, a, c	2, 3, 4, 5, 1	15
T_8	b, f, a, e, c	4, 1, 5, 3, 2	15

For example, $TU(T_3) = U(a, T_3) + U(b, T_3) + U(c, T_3) + U(d, T_3) + U(e, T_3) + U(f, T_3) = 30$

Definition 7. The transaction-weighted utility of an itemset X, denoted as $TWU(X)$, is defined by

$$TWU(X) = \sum_{X \subseteq T_j and T_j \in D} TU(T_j) \tag{5}$$

The TWU values for the sample transactional database in Table 3 is provided in Table 5.

Definition 8. Let T_j/X denote the set of all items after X in T_j (Liu & Qu, 2012). For example, in Table 3, $T_1/\{ac\} = \{def\}$, $T_7/\{ac\} = \{df\}$.

Definition 9. The remaining utility of an itemset X in transaction $T_j(X \subseteq T_j)$ is denoted as $RU(X, T_j)$ (Liu & Qu, 2012) and is computed as,

$$RU(X, T_j) = \sum_{x_i \in (T_j/X)} U(x_i, T_j), \tag{6}$$

For example, in Table 3, $RU(\{ac\}, T_1) = 2 + 6 + 2 = 10$, $RU(\{ac\}, T_7) = 4 + 3 = 7$.

Definition 10. The remaining utility of an itemset X in database D is denoted as RU(X) (Liu & Qu, 2012).

$$RU(X) = \sum_{X \subseteq T_j and T_j \in D} RU(X, T_j) \tag{7}$$

For example, in Table 3, $RU(\{ac\}) = 10 + 11 + 20 + 18 + 7 + 4 = 70$.

Definition 11. *(Ordering of items).* The items in the transaction database are processed using total order $\prec$ such that the items are sorted in TWU ascending order. For the running example, the ordering of items are: g $\prec$ b $\prec$ f $\prec$ d $\prec$ a $\prec$ e $\prec$ c. For the sample database in Table 3, the ordered set of items in individual transactions are presented in Table 6.

Definition 12. The absolute minimum utility value is denoted as δ. The δ value is set to zero, by default. This value is raised during the mining process to generate the top-k HUIs.

Definition 13. The absolute minimum utility threshold value at the end of the initial stage of HUI mining (i.e. before the growth stage) is denoted as δ_I.

Definition 14. (High Utility Itemset) An itemset X is referred as a HUI iff its utility $U(X)$ is greater than or equal to the minimum utility threshold value δ.

Definition 15. (Top-K High Utility Itemset) The set of all K HUIs with the highest utilities in D are denoted as *top-kHUI*.

Definition 16. The optimal minimum utility threshold value, denoted as δ_F, is defined as

$$\delta_F = min\{U(X) | X \in top\text{-}kHUI\} \tag{8}$$

Problem statement Given a transactional database D and the desired number of HUIs (K), the problem of top-k high utility mining involves determining exactly K HUIs in D that have the highest utilities.

It is to be noted that there are slightly varying definitions of top-k HUIs in the literature (exactly K (Duong et al., 2016) vs flexible K HUIs with support for tie cases (Tseng et al., 2016)). We follow the stricter definition used in the most recent KHMC algorithm (Duong et al., 2016) and apply it consistently across all our implementations.

For the transaction database in Table 3, when $K = 6$,
$top\text{-}kHUI = \{\{aec\} : 80, \{fdaec\} : 78, \{fdae\} : 73, \{fdac\} : 73, \{faec\} : 69, \{daec\} : 68\}$.

Our main objective in this paper is to design and develop a new method to efficiently mine top-k HUIs from the transaction database D.

4. Top-K high utility itemset (THUI) mining

In this section, we first introduce the LIU data structure in Section 4.1 and illustrate its key benefits. We then present two new threshold raising strategies (LIU-E and LIU-LB) in Section 4.2. Subsequently, we describe the proposed THUI algorithm in Section 4.3 and provide an illustrative example.

4.1. Leaf itemset utility (LIU) data structure

Let $(x_i..x_j)$ denote the contiguous and ordered sequence of items between x_i and x_j. That is, $(x_i..x_j) = \{x_i, x_{i+1}, x_{i+2} \ldots x_j\}$. For the running example, the ordered set of items are: g $\prec$ b $\prec$ f $\prec$ d $\prec$ a $\prec$ e $\prec$ c. Therefore, the contiguous sequence $(f..c) = \{f, d, a, e, c\}$. Similarly, $(b..a) = \{b, f, d, a\}$.

The LIU structure is a triangular matrix that holds the utility information of a contiguous set of items in compact form. More formally,

$$\begin{aligned} LIU(x_i, x_j) &= LIU((x_i..x_j)) \\ &= LIU(\{x_i, x_{i+1}, x_{i+2} \ldots x_j\}) \\ &= \sum_{X=(x_i..x_j) \subseteq T_s and T_s \in D} U(X, T_s) \end{aligned} \tag{9}$$

For example,

$$\begin{aligned} LIU(a, c) &= LIU((a..c)) \\ &= LIU(\{a, e, c\}) \\ &= \sum_{X=\{a,e,c\} \subseteq T_s and T_s \in D} U(X, T_s) \\ &= U(\{a, e, c\}, T_1) + U(\{a, e, c\}, T_2) + U(\{a, e, c\}, T_3) \\ &\quad + U(\{a, e, c\}, T_6) + U(\{a, e, c\}, T_8) \\ &= 12 + 22 + 9 + 27 + 10 \\ &= 80 \end{aligned}$$

Similarly, $LIU(d, c) = \sum_{X=\{d,a,e,c\} \subseteq T_s and T_s \in D} U(X, T_s) = 68$.

The LIU values for every pair of contiguous sequence of items are generated from the transaction database. Let us consider the transaction T_1 in Table 6 with items *f, d, a, e, c*. We first consider the item *c* as the last item in the sequence. The contiguous sequence that can be generated with *c* as the last item are: *ec, aec, daec*, and *fdaec*. We compute the utility values of each of these sequences and store the utility values respectively in $LIU(e, c)$, $LIU(a, c)$, $LIU(d, c)$, $LIU(f, c)$. The processed utility values are shown in

	b	f	d	a	e	c
g						
b						
f			4	9	15	16
d				7	13	14
a					11	12
e						7

	b	f	d	a	e	c
g	6					
b		19	30	40	29	30
f			37	67	73	78
d				54	63	68
a					67	80
e						51

Fig. 1. LIU matrix after processing transactions (1) T_1, (2) T_1 to T_8.

Algorithm 1 UpdateLIU.

Input: item position i in T, transaction T

1: nextItem = x_i
2: cumUtil = $U(x_i, T)$
3: **for** each $j \in [0, i-1]$ in reverse order **do**
4: currItem = x_j, $x_j \in T$
5: **if** currItem $\prec$ nextItem and |(currItem .. nextItem)| = 2 **then**
6: Increment cumUtil by $U(currItem, T)$
7: Increment $LIU(currItem, x_i)$ by cumUtil
8: nextItem = currItem
9: **else**
10: return
11: **end if**
12: **end for**

Fig. 1, with the column titled *c*. Next, we treat *e* as the last item in the sequence, and generate contiguous sequences *ae, dae, fdae*. The processed utility values are shown in Fig. 1, with the column titled *e*. This process is repeated with items *a* and *d* as the last item. The final set of utility values after processing transaction T_1 is shown in Fig. 1. Now, let us consider transaction T_2 that contains items *g, a, e, c*. We start with item *c* as the last item and generate contiguous sequence *ec, aec*. Note that the sequence *gaec* is not generated as it is not contiguous. The LIU values are then incremented in the matrix for the entries *LIU*(*e, c*) and *LIU*(*a, c*). Subsequently, the item *e* is treated as the last item and the contiguous sequence *ae* is generated. No more contiguous item sequences can be generated from transaction T_2. Let us now consider transaction T_4 as another example with items *b, d, e, c*. Only one sequence *ec* can be generated from this transaction as there are no other contiguous sequences.

The above process is repeated for each and every transaction in the database and the final processed LIU matrix is shown in the bottom part of Fig. 1. The complete pseudo-code for the LIU construction procedure is described in Algorithm 1. The key idea is to iteratively record the cumulative utility values of items until the immediate predecessor (refer to step 5 of Algorithm 1) condition is satisfied.

A graphical illustration of the LIU structure is given in Fig. 2. All the square marked entries in the figure represent contiguous sequences with *c* as the last item. The oval shaped entries represent contiguous sequences with *e* as the last item. Similarly, the double rectangle and diamond shaped entries represent contiguous sequences with *a* and *d* as the last item respectively. One can visualize the above structure as layers of leaf nodes of the itemset tree for a given last item (e.g. *c, e*). Hence, we refer to this novel data structure as Leaf Itemset Utility (LIU) structure. The bottom part of the figure gives the details of the itemset along with their utility values. It also provides the rank ordered utility values numbered from 1 to 6. It is to be noted that for the running example, we are interested in discovering only the top-6 HUIs from the sample database.

We have explained the LIU structure using a matrix notation for illustration purpose. In the actual implementation, a hash map structure is utilized to optimize space usage. Theoretically, the space required for the LIU structure is of the order of square of the total number of items (i.e. $O(m^2)$). However, in practice, the space required will be extremely small as the LIU structure captures only the long contiguous itemsets in transactions. Let us consider transaction T_5 with items *g, b, e, c* as an example. It can be verified that only two contiguous item sequence *gb, ec* will be generated from T_5. Similarly, from transaction T_4, just one contiguous sequence *ec* will be generated. We also show through our experimental evaluation that the number of entries in LIU structure are of the order of square of the average length of transactions (L) in the database (i.e. $O(L^2)$).

From the foregoing discussions, it is evident that the LIU structure aims to capture utilities of long contiguous sequence of items in a compact form. EUCS (Fournier-Viger et al., 2014b) data structure that is commonly used in the literature, on the other hand, stores utility information of every pair of items. While the EUCS structure is useful for mining basic HUIs, we argue that the proposed LIU structure is extremely effective in raising threshold values for top-k HUI mining. We provide illustrative examples and conduct rigorous experimental evaluation to demonstrate our ideas.

4.2. Threshold raising strategies

The THUI algorithm uses four key threshold raising strategies, namely RIU, RUC, LIU-E, and LIU-LB. Two of these strategies, namely RIU (raising threshold for 1-itemsets) and RUC (similar to RIU but generalized for k-itemsets), are used in the past literature and the details of these strategies can be referred in Ryang and Yun (2015), Duong et al. (2016), Krishnamoorthy (2018). In this section, we describe our two novel threshold raising strategies: LIU-E and LIU-LB.

4.2.1. LIU-Exact threshold raising strategy

After generating the LIU matrix, the values in the matrix that are no less than δ are stored in a priority queue *PQ_LIU*. More formally, $PQ\_LIU = \{LIU(x, y) | LIU(x, y) \geq \delta\}$.

Property 1. *LIU-Exact (LIU-E) If* $|PQ\_LIU| \geq K$, *then* δ *can be raised to the Kth highest value in PQ.*

Proof. *LIU*(*x, y*) holds the exact utility information about the itemset sequence (*x..y*). The priority queue primarily accumulates the utility values of all itemsets in *LIU* that are greater than δ. Moreover, the set of itemsets in *PQ_LIU* are essentially subsets of overall HUIs at the current δ value. Therefore, raising the δ value to the Kth highest utility value in PQ does not raise it beyond the optimal δ_F value required to mine top-k HUIs. Hence, the proof. □

For the running example, there are 16 LIU values shown in Figs. 1 and 2. For $K = 6$, the δ value can be raised to 67 by applying the LIU-E property. This value is almost close to (but less than) the optimal value ($\delta_F = 68$) for mining top-6 HUIs from the sample database.

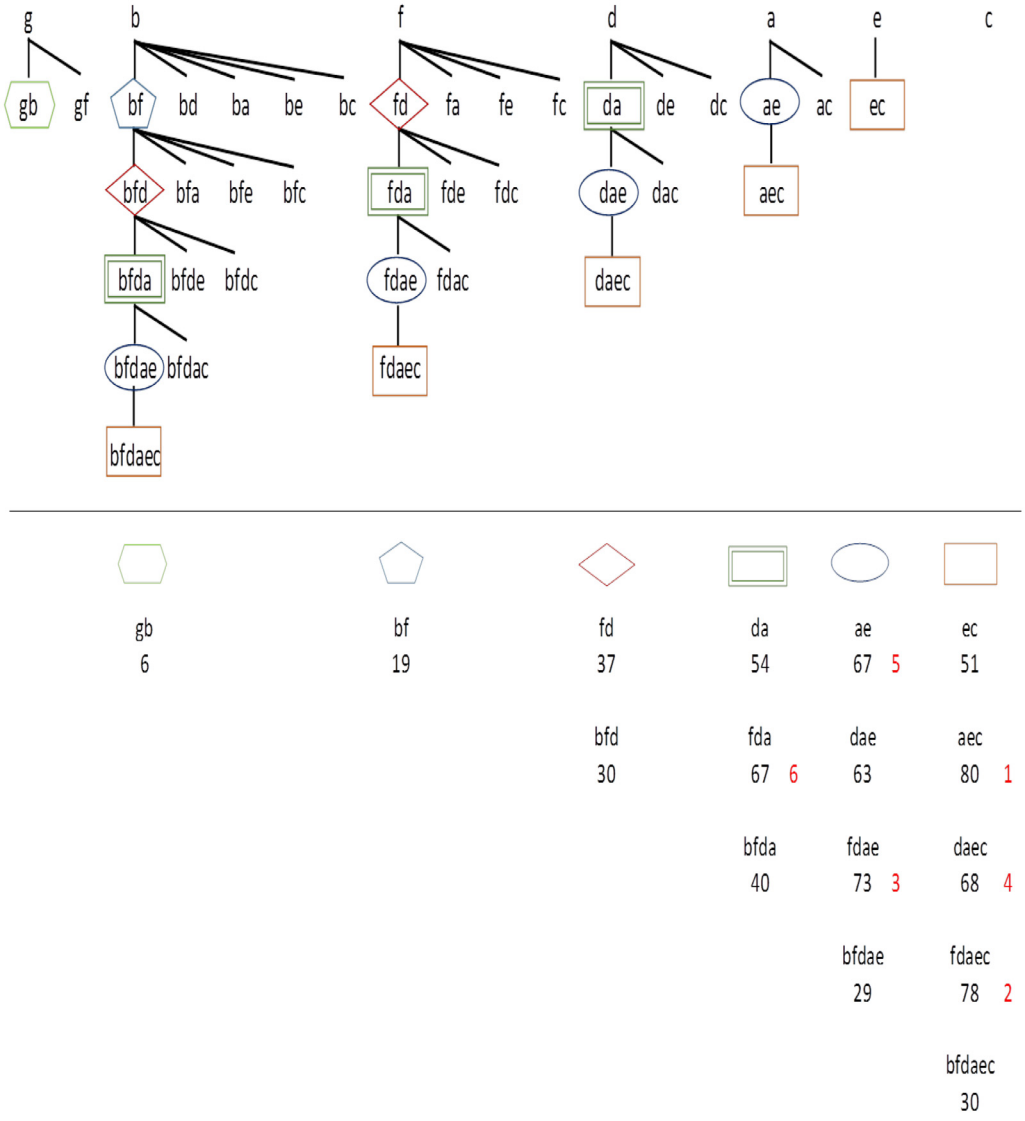

Fig. 2. Sample itemset tree with utility values.

4.2.2. LIU-LB Threshold raising strategy

A new utility lower bound estimation procedure is proposed in this paper. The proposed approach uses the information stored in LIU matrix for a few contiguous itemsets to estimate utility lower bound value for other related itemsets. We describe our estimation method by introducing a few properties and notations in the following pages.

Property 2. *(Cumulative utility property) Let X and Y be any two disjoint itemsets that are subsets of I. The property* $U(X)+U(Y) \geq U(X \cup Y)$ *holds for any given X and Y.*

Proof. The LHS of the property can be expanded as

$$U(X)+U(Y) = \sum_{X\cup Y\subseteq T_j} U(X,T_j) + \sum_{X\cup Y\not\subseteq T_j, X\subseteq T_j} U(X,T_j) + \sum_{X\cup Y\subseteq T_j} U(Y,T_j) + \sum_{X\cup Y\not\subseteq T_j, Y\subseteq T_j} U(Y,T_j)$$

Similarly, the RHS of the property can be expanded as

$$U(X\cup Y) = \sum_{X\cup Y\subseteq T_j} U(X) + \sum_{X\cup Y\subseteq T_j} U(Y)$$

Comparing the expanded terms of the LHS and RHS, one can easily verify that LHS will always be greater than or equal to RHS. Hence, the proof. □

Let us consider two itemsets $X=\{d\}$ and $Y=\{faec\}$. As per the property $U(X)+U(Y) \geq U(XY)$ i.e. $U(d)+U(faec) \geq U(fdaec)$. It can be verified from Table 6 that $U(d)=30$, $U(faec)=69$ and $U(fdaec)=78$. Therefore, $U(d)+U(faec) \geq U(fdaec)$ implies $30+69 >= 78$ and satisfies the property.

Property 3. *(Utility lower bound property) Let us denote the utility lower bound of an itemset Y as ULB(Y). Given any itemset* $X \subset I$*, the ULB(Y) can be computed as follows:*

$$\begin{aligned} ULB(Y) &= U(X\cup Y) - U(X) \\ &= U(X\cup Y) - \sum_{x_i\in X} U(x_i) \end{aligned} \quad (10)$$

Proof. The first line of Eq. (10) is a direct application of property 2 described earlier. The second line of the equation computes the utility value of X as the sum of utilities of individual items. As per property 2, the sum of the individual item utility value is no less than the utility value of X. Therefore, the *ULB*(Y) is never overestimated. Hence, the proof. We use this property to

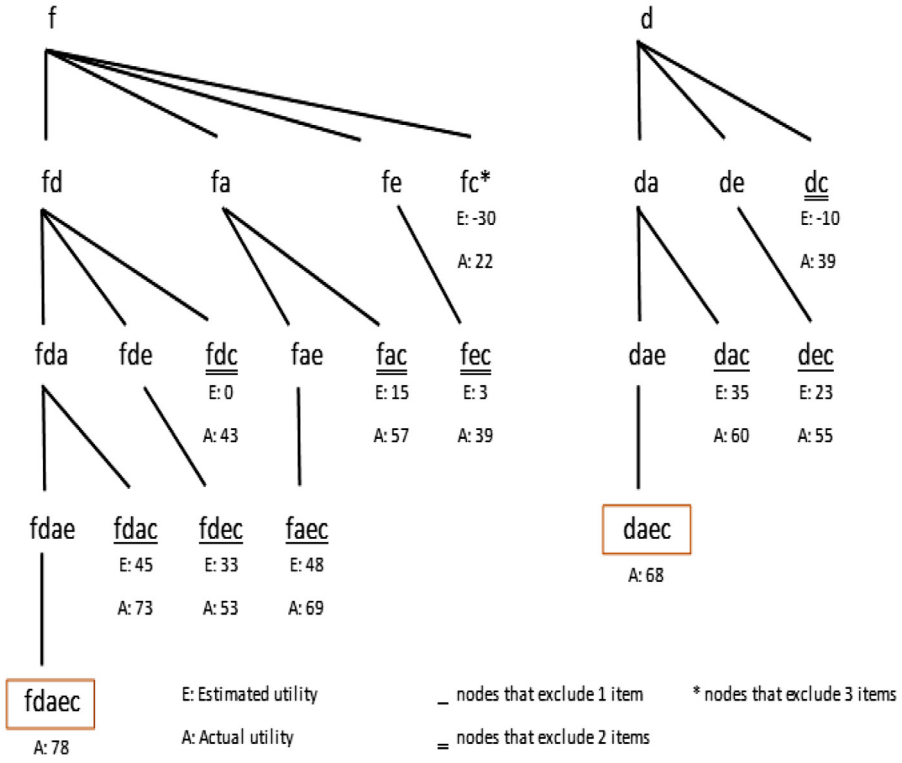

Fig. 3. Illustration of itemset subset generation and ULB estimation from LIU.

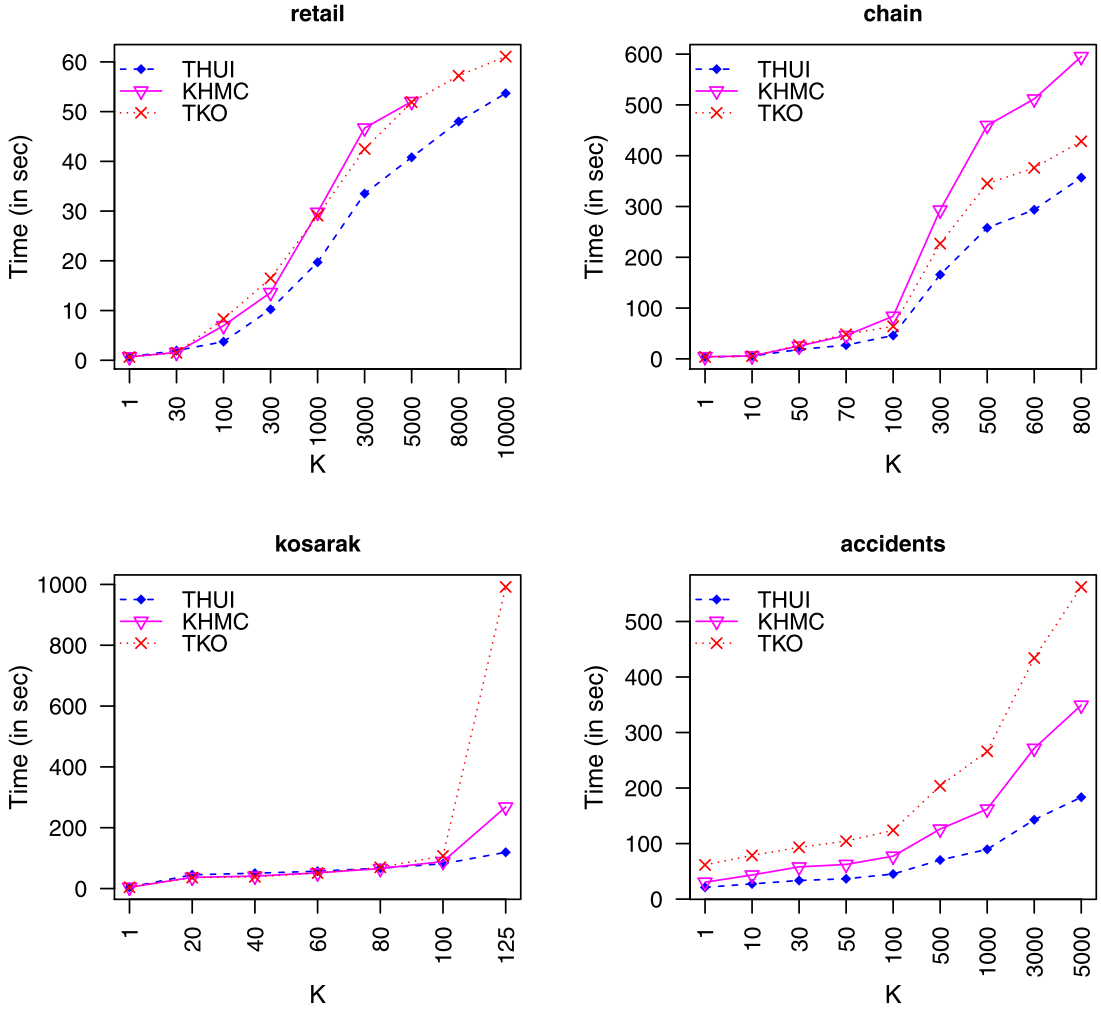

Fig. 4. Runtime performance analysis on sparse datasets.

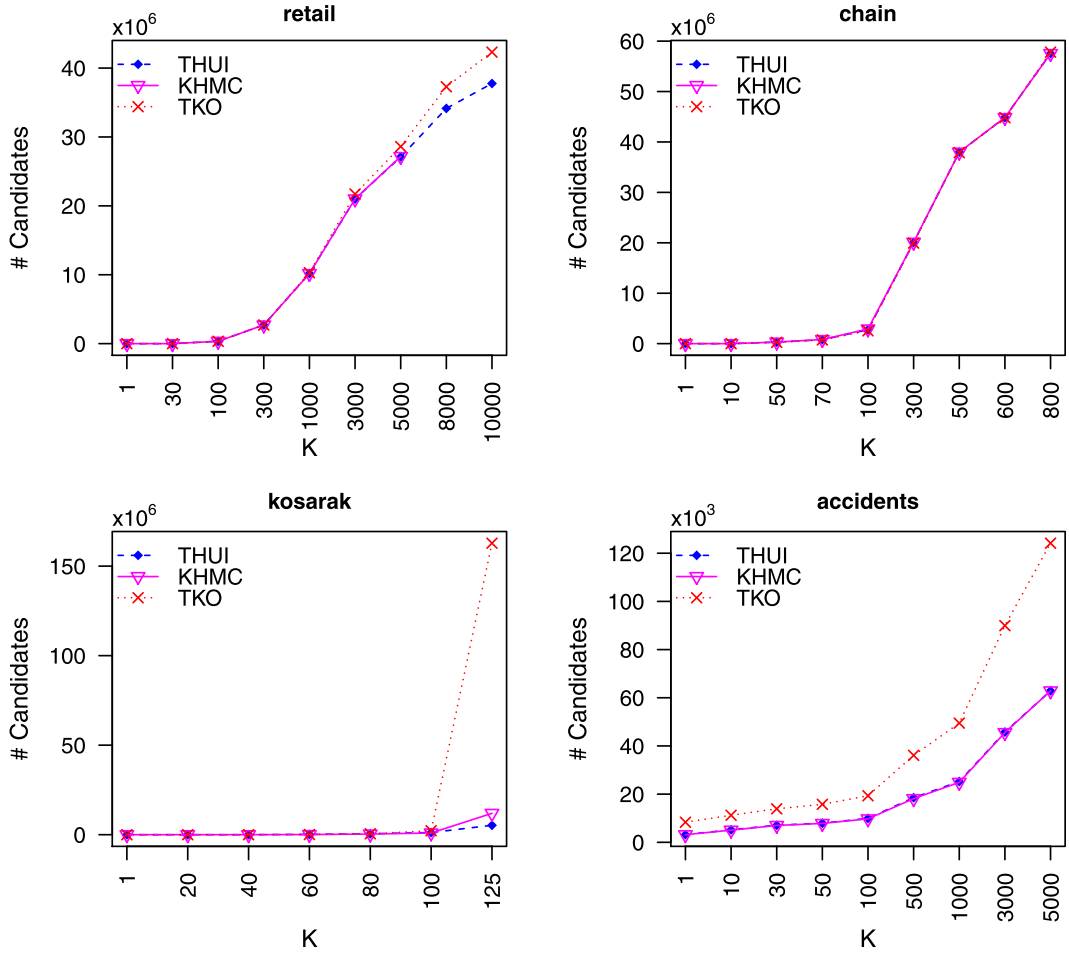

Fig. 5. Number of candidates generated on sparse datasets.

estimate the lower bound utility value of an itemset Y using its superset $X \cup Y$ and the itemset X. □

For example, let us assume $Y = \{faec\}$ and $X = \{d\}$. Now, $ULB(Y) = U(X \cup Y) - U(X) = U(fdaec) - U(d)$. The LIU matrix holds the utility value of the contiguous sequence {*fdaec*} as 78 (refer to Fig. 1). The utility value of $U(d)$ can be verified as 30 from Table 6. Therefore, $ULB(faec) = U(fdaec) - U(d) = 78 - 30 = 48$, which is less than its actual utility value of 69 (refer to Table 6).

Definition 17. Let *LIULB*(*x, y, q*) denote the lower bound estimate for an itemset with all items in q removed from the sequence (*x..y*), where $q \subset I$. More specifically, *LIULB*(*x, y, q*) indicate the lower bound estimate for an itemset $Y = \{(x..y) - q\}$ i.e. $ULB(Y = \{(x..y) - q\})$.

For example, *LIULB*(*f, c, d*) indicates the lower bound estimate for the itemset {fdaec}-{d}={faec} i.e. *ULB*(*faec*).

Definition 18. Let us denote the set of all estimated utilities of items as *PQ_LB_LIU* and define as:

$$PQ\_LB\_LIU = \{LIULB(x, y, q) \mid (x..y) \in LIU \text{ and } q \subset I \text{ and } |q| \leq 3\} \quad (11)$$

The proposed method generates itemset subsets from entries in the LIU matrix and estimates their utilities by applying the utility lower bound property (i.e. property 3). The total number of itemset subsets generated, however, are limited by setting the size of q as 3. This is a heuristic as very large number of itemset subsets can get generated from each contiguous sequence (x.y) in the LIU matrix. Besides, the lower bound estimates degrade as the size of q is increased. The degradation in lower bound estimates can be attributed to approximation applied to compute $U(X)$ as summation of individual item utilities in property 3.

Let us illustrate the itemset subset generation process from LIU matrix for two specific cases. These cases are also pictorially represented in Fig. 3. For example, LIU(f,c) = LIU((f..c)) = 78 represent the actually utility value of the contiguous itemset {*f, d, a, e, c*}. By removing one item from (f..c), while retaining the first and last item, we can generate subsets *fdac, fdec*, and *faec*. These itemset subsets are shown in Fig. 3 along with their estimated lower bound and actual utility values. The lower bound estimates for these itemset subsets are computed using property 3. For example, $ULB(fdac) = LIU(f, c) - U(e) = 78 - 33 = 45$. Similarly, one can remove two items from (f..c), while retaining the first and last item, and generate subsets *fdc, fac*, and *fec*. These subsets are double-underlined in Fig. 3 along with their estimated utility values. For example, *ULB*(*fac*) is computed as $LIU(f, c) - U(d) - U(e) = 78 - 30 - 33 = 15$. Lastly, one can generate the subset *fc* by removing 3 items from the LIU sequence (f..c). Fig. 3 also provides another illustration for itemset subset generation from LIU sequence (d.c) by removing 1 and 2 items. We ensure that no two itemset sequence in LIU generates a lower bound estimate for the same subset itemset. This is done by fixing the start and end items and removing only the intermediate items in any given sequence. Consider the illustration in Fig. 3, the subset itemsets for which lower bound estimates are made for the itemset {*fdaec*} will always include the item *f* as the start item and *c* as the end or last item. Therefore, no

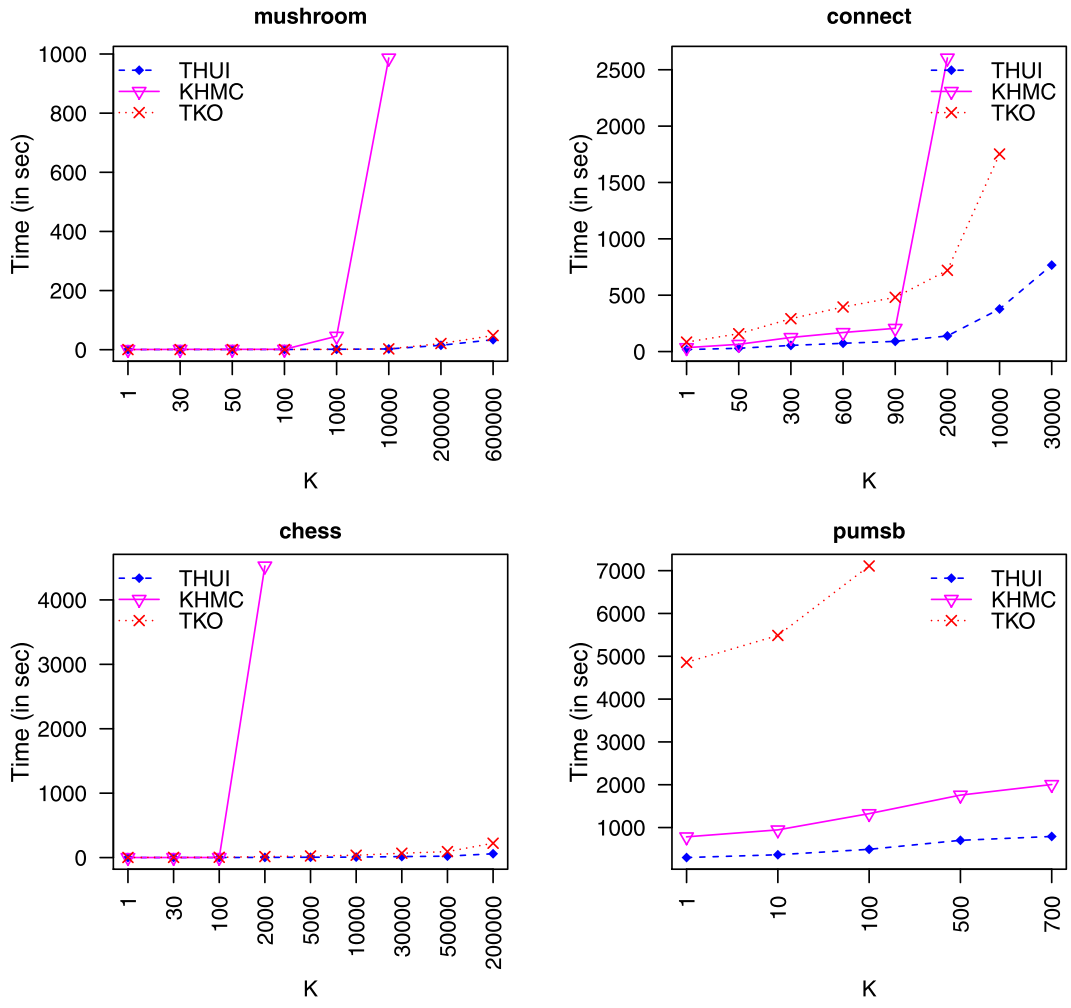

Fig. 6. Runtime performance analysis on dense datasets.

duplicate item sequences are generated during lower bound estimation. The detailed set of steps involved in itemset subset generation, utility computation and lower bound estimation are described as part of the threshold raising strategy in Algorithm 2.

Algorithm 2 RaiseThreshold_LB_LIU.

Input: *LIU*, *PQ_LIU*, *PQ_LB_LIU*, δ

```
1:  for each entry x, y ∈ LIU do
2:      startItem = succ(x)
3:      endItem = pred(y)
4:      for each a ∈ (startItem..endItem) do
5:          utilLB = LIU(x,y) - U(a)
6:          if utilLB ≥ δ then add utilLB to PQ_LB_LIU endif
7:          for each (b = succ(a)) ∈ (startItem.endItem) do
8:              utilLB = LIU(x,y) - U(a) - U(b)
9:              if utilLB ≥ δ then add utilLB to PQ_LB_LIU endif
10:             for each (c = succ(b)) ∈ (startItem.endItem) do
11:                 utilLB = LIU(x,y) - U(a) - U(b) - U(c)
12:                 if utilLB ≥ δ then add utilLB to PQ_LB_LIU endif
13:             end for
14:         end for
15:     end for
16: end for
17: if |PQ_ALL = {PQ_LIU ∪ PQ_LB_LIU}| ≥ K then
18:     δ = Kth highest value in PQ_ALL //raise minimum utility
19: end if
```

Property 4. *LIU-Lower Bound (LIU-LB) Let PQ_ALL* $= \{PQ\_LIU \cup PQ\_LB\_LIU\}$. *If* $|PQ\_ALL| \geq K$, *then* δ *can be raised to the Kth highest value in PQ_ALL.*

Proof. This property is an extension of the LIU-E property. For each contiguous item sequence in *LIU*, an estimate of lower bound for several subsets are generated and maintained in *PQ_LB_LIU*. Each of these subset itemsets are part of the overall HUIs at the current δ value. Therefore, raising the δ value to the Kth highest utility value in PQ_ALL does not impact the correctness of the results. The resulting δ value will be less than or equal to the optimal δ_F. □

It is to be noted that actual itemset information is not stored during the lower bound estimation (refer to Algorithm 2, steps 6, 9 and 12). Only the estimated lower bound utility values are stored in *PQ_LB_LIU* for raising the minimum utility threshold value. Furthermore, a priority queue structure is used to maintain only the top-k utility values.

4.3. THUI algorithm

The THUI algorithm is a one-phase utility list based approach for mining top-k HUIs. There are three distinct stages in the algorithm. In stage 1, the database *D* is scanned to compute *TWU* of all 1-itemsets. Subsequently, the RIU threshold raising strategy is applied to raise δ from the initial value of zero.

In the second stage, the algorithm performs the following steps: (a) filter non-promising items based on TWU-Prune prop-

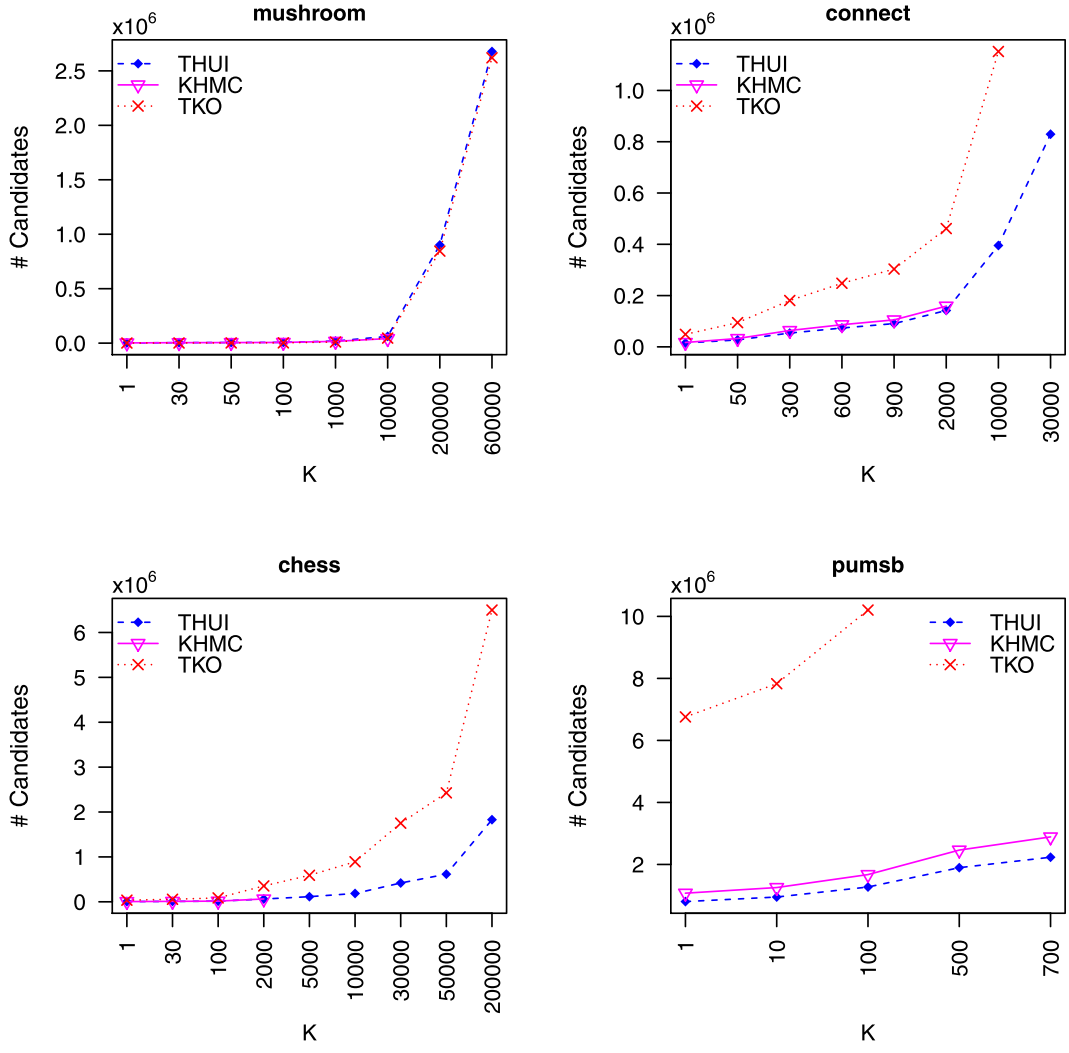

Fig. 7. Number of candidates generated on dense datasets.

Algorithm 3 THUI algorithm.

Input: D, K **Output:** *top-kHUI*

1: Scan D and Compute TWU for all 1-itemsets
2: Apply RIU threshold raising strategy to raise δ
3: **for** each $T_s \in D$ **do**
4: $newT = \{x \mid TWU(x) >= \delta \ \ \forall x \in T_s\}$ *//TWU-Prune*
5: Sort $newT$ as per the ordering heuristic (Definition 11)
6: **for** each $x_i \in T_s$ in reverse order **do**
7: UpdateLIU(i, T_s) *//Algorithm 1*
8: **end for**
9: Construct UL for each transaction (Liu & Qu, 2012)
10: **end for**
11: Compute PQ_LIU from LIU *//refer section 4.2.1*
12: Apply LIU-Exact threshold raising strategy *//property 1*
13: RaiseThreshold_LB_LIU(LIU, PQ_LIU, {}, δ) *//refer Algorithm 2*
14: top-kHUI = Explore-Search-Tree($\emptyset$, ULs, δ)

Algorithm 4 Explore-Search-Tree.

Input: R the itemset prefix, ULs, δ
Output: *top-kHUIs* with prefix R

1: **for** each utility list position i in ULs **do**
2: $X = ULs[i]$
3: **if** $U(X) \geq \delta$ **then**
4: $top\text{-}kHUI \leftarrow \{top\text{-}kHUI \ \cup \ X\}$
5: δ = $min\{U(X) | X \in top\text{-}kHUI\}$ *//RUC strategy*
6: **end if**
7: **if** $U(X) + RU(X) \geq \delta$ **then**
8: $exULs$ = ConstructUL(X, ULs, $i+1$, δ) (Liu & Qu, 2012)
9: $R = \{R \cup X\}$ *//update prefix with extension*
10: Explore-Search-Tree(R, $exULs$, δ)
11: **end if**
12: **end for**

erty (Liu et al., 2005), (b) order items in transactions based on the ordering heuristic (refer to Definition 11), (c) create an LIU matrix and update the entries with utility values (Algorithm 1), and (d) construct a Utility List (UL) (Liu & Qu, 2012).

After performing all of the steps outlined above, the LIU threshold raising strategy is applied to raise the δ value (steps 11 and 12 of Algorithm 3). Subsequently, the LIU-LB threshold raising strategy is applied (step 13 of Algorithms 3 and 2).

In the third and final stage of THUI algorithm, the itemset search tree is explored to mine top-k HUIs. The overall steps of the exploration process are outlined in Algorithm 4. The tree exploration process is similar to other utility list based methods in the literature (Fournier-Viger et al., 2014b; Liu & Qu, 2012). The key difference is that the THUI method generates only the top-k HUIs

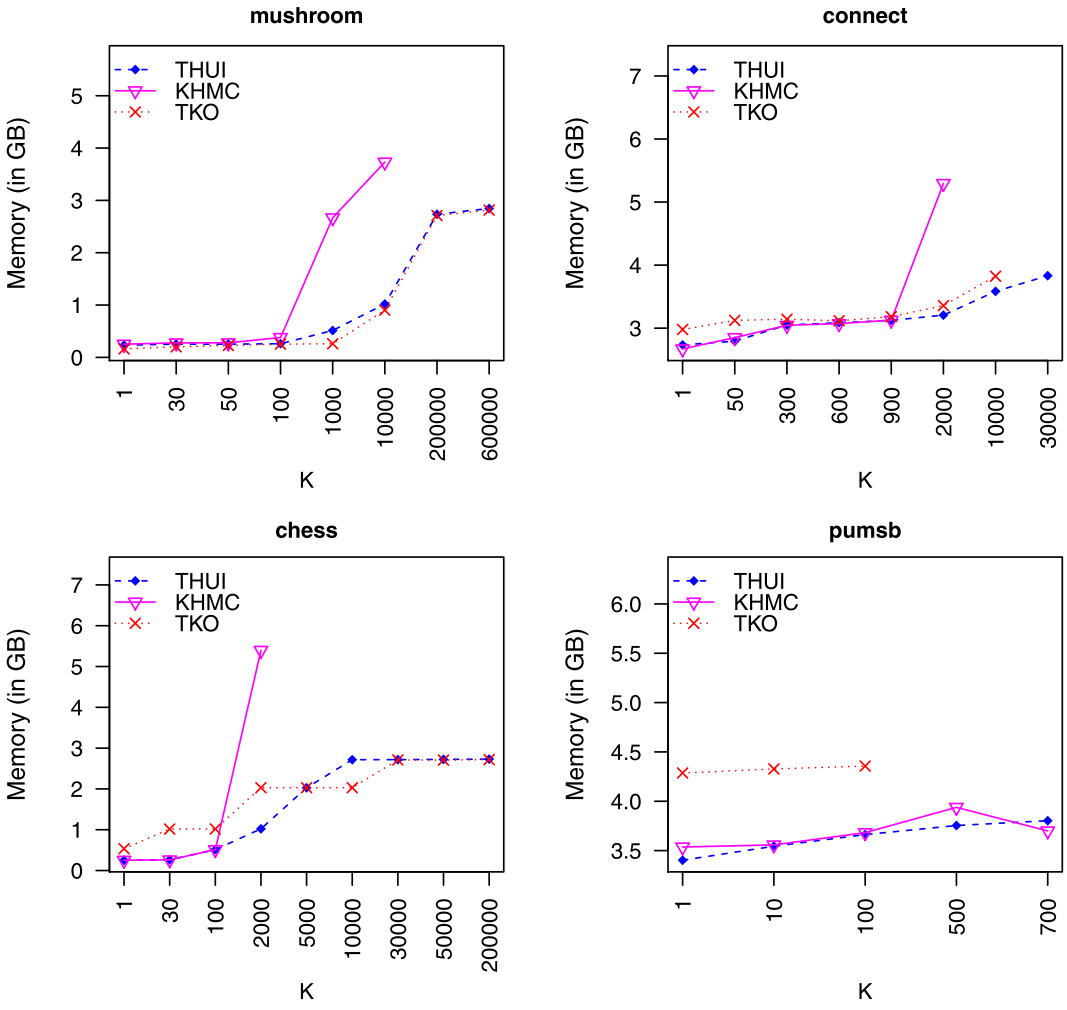

Fig. 8. Memory consumption performance on dense datasets.

by dynamically raising the δ value in each iteration. The δ value is raised using the RUC strategy (step 5 of Algorithm 4). The complete pseudo-code of the THUI algorithm can be referred in Algorithm 3.

THUI algorithm makes use of the following pruning properties: TWU-Prune (Liu et al., 2005), U-Prune (Krishnamoorthy, 2017; Liu & Qu, 2012), and LA-Prune (Krishnamoorthy, 2015). These pruning properties are applied at different stages of the THUI algorithm. The TWU-Prune is applied during early stages to remove unpromising 1-itemsets (step 4 of Algorithm 3). The U-Prune strategy is applied during itemset tree exploration (step 3 of Algorithm 4). The LA-Prune is applied as part of the tree exploration process (step 8 of Algorithm 4).

4.3.1. An illustrative example

We illustrate the THUI algorithm in a step-by-step manner using the sample database D shown in Table 3. Assume that we are interested in mining top-6 HUIs (i.e. K=6). The algorithm first scans D and computes *TWU* values. The computed *TWU* values are given in Table 5. Subsequent application of RIU threshold raising strategy raises the δ value from zero to 14. For the running example, there are seven 1-itemsets {*g, b, f, d, a, e, c*} with utility values {7, 22, 14, 30, 45, 33, 19}. As the 6th highest utility value is 14, the δ value is raised from 0 to 14 (RIU threshold raising strategy).

THUI algorithm then performs second scan of the database, orders the transaction entries and updates LIU values. The ordered transaction entries are presented in Table 6. The updated LIU matrix for the running example is given in Fig. 1. A set of utility list for each item is also constructed during the second scan of the database.

After the second database scan is completed, the LIU-E threshold raising strategy is applied to raise the δ value (step 12 of Algorithm 3). Fig. 2 depicts the LIU itemset details as well as the utility values, in ranked order. Applying LIU-E strategy, the 6th highest utility value obtained for the running example is 67. Hence, the δ value is raised to 67 from the earlier value of 14. If one were to apply the CUD strategy (Duong et al., 2016) in lieu of LIU based strategy, the resulting δ value for the example will be 47. The application of COV threshold raising strategy (Duong et al., 2016) can increase δ further to 49. This is substantially lower than the LIU based strategy, and a similar pattern has been observed for most of our experimental results for the dense datasets.

Subsequently, in step 13 of Algorithm 3, the subset itemset generation and lower bound estimation method is invoked. The step 1 of Algorithm 2 starts with an entry in LIU. Let us consider, for example, the LIU itemset sequence *fdaec*. In steps 2 and 3, the start and end items are respectively set as *d* and *e*. This is done to avoid duplicate itemset subset generation as described earlier. Then steps 4, 7 and 10 apply the 1-item, 2-item and 3-item exclusion heuristic to generate itemset subsets. For example, from the LIU itemset sequence *fdaec*, the itemset subsets *fdac, fdec, faec, fac* and so on (refer to Fig. 3) are generated. The utility lower bounds for the item susbets are computed in steps 5, 8 and 11. If the generated utility lower bound estimates are greater than the δ value, the values are pushed to the priority queue. At the end of the iteration (step

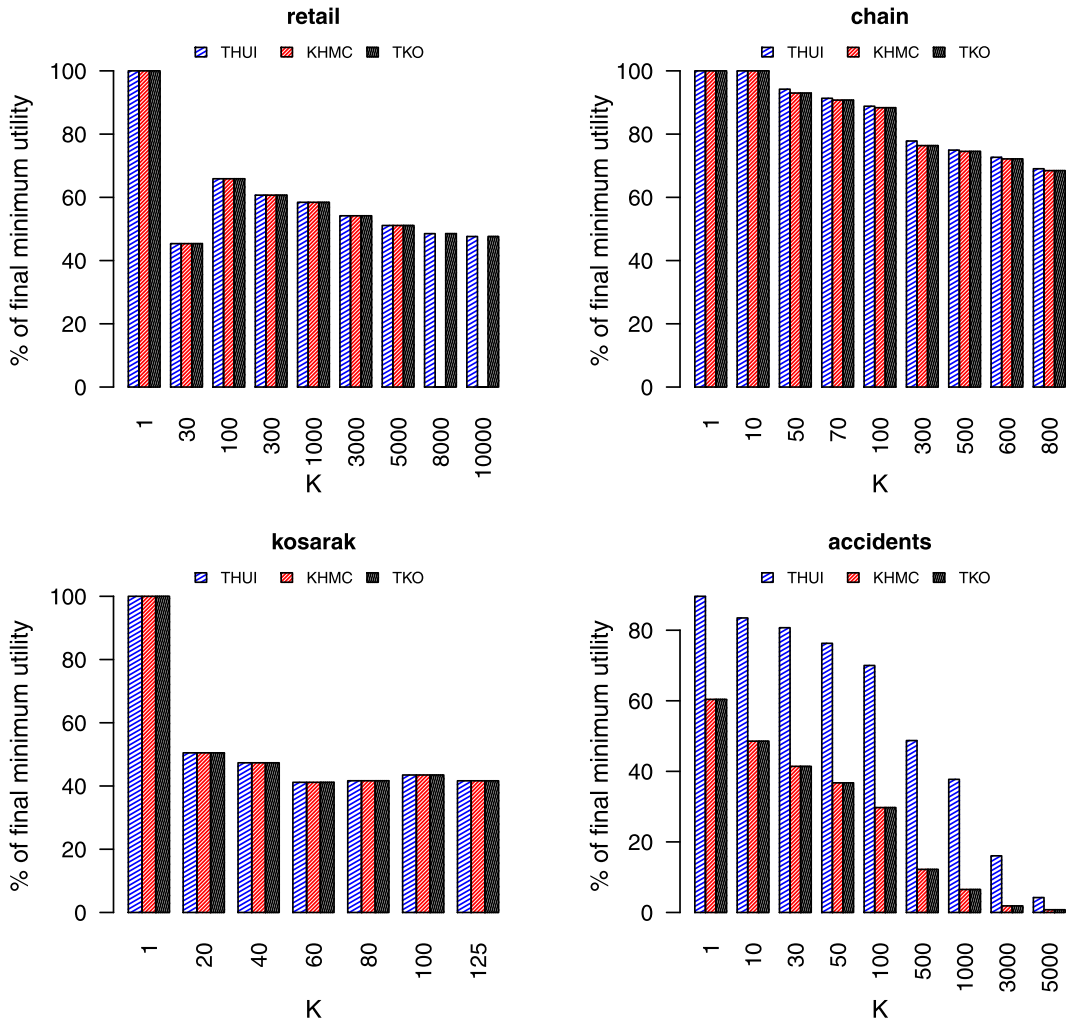

Fig. 9. Effectiveness of threshold raising strategies on sparse datasets.

16 of Algorithm 2), the priority queue maintains the set of subset itemsets (from LIU) and their estimated utility values. The LIU-LB property (property 4) is then applied in step 17 of Algorithm 2 to raise the utility threshold value. For the running example, the δ value does not change from 67 as the value is already very close to the optimal δ value of 68.

In the final growth stage, the THUI algorithm explores the itemset search tree to mine the top-k HUIs. The items are ordered as per the ordering heuristic and are processed in the same order. The key steps of the exploration are similar to the HUI-Miner (Liu & Qu, 2012) algorithm. The δ value is increased during the itemset search space exploration using the RUC strategy (step 5 of Algorithm 4).

The final set of generated top-k HUIs are: $top\text{-}kHUI = \{aec : 80, fdaec : 78, fdae : 73, fdac : 73, faec : 69, daec : 68\}$. The optimal $\delta_F = 68$ and is almost close to the δ value (67) determined by the LIU based strategy at the beginning of the growth stage of the THUI algorithm.

5. Experimental results

We implemented all of the algorithms used in the experiments by extending the open-source data mining library (Fournier-Viger, Gomariz, Soltani, Lam, & Gueniche, 2014a). All our experiments were performed on a Dell workstation having Intel Xeon 3.7GHz processor with 64GB of main memory, 8GB java heap size, and running a Linux OS.

Table 7
Dataset characteristics.

Dataset	#Trans	#Items (I)	AvgLen(L)	Density%
chain	1,112,949	46,086	7.3	0.0158
kosarak	990,002	41,270	8.1	0.0196
retail	88,162	16,470	10.3	0.0625
pumsb	49,046	2113	74	3.5021
accidents	340,183	468	33.8	7.2222
mushroom	8124	119	23	19.3277
connect	67,557	129	43	33.3333
chess	3196	75	37	49.3333

5.1. Experimental design

We evaluated the performance of THUI on eight benchmark sparse and dense datasets. All the datasets, except chain, were downloaded from the SPMF data mining library (Fournier-Viger et al., 2014a). The chain dataset was downloaded from NU-MineBench 2.0 (Pisharath et al., 2005). The characteristics of all the datasets used in the experiments are provided in Table 7. Chain, kosarak, retail datasets are highly sparse datasets commonly used in the literature. The accidents dataset is a large dataset with moderately long transactions. The pumsb, mushroom, connect and chess datasets are highly dense in nature. The external utilities of items were generated between 0.01 and 10 using a log-normal distribution. The internal utilities of items were randomly generated

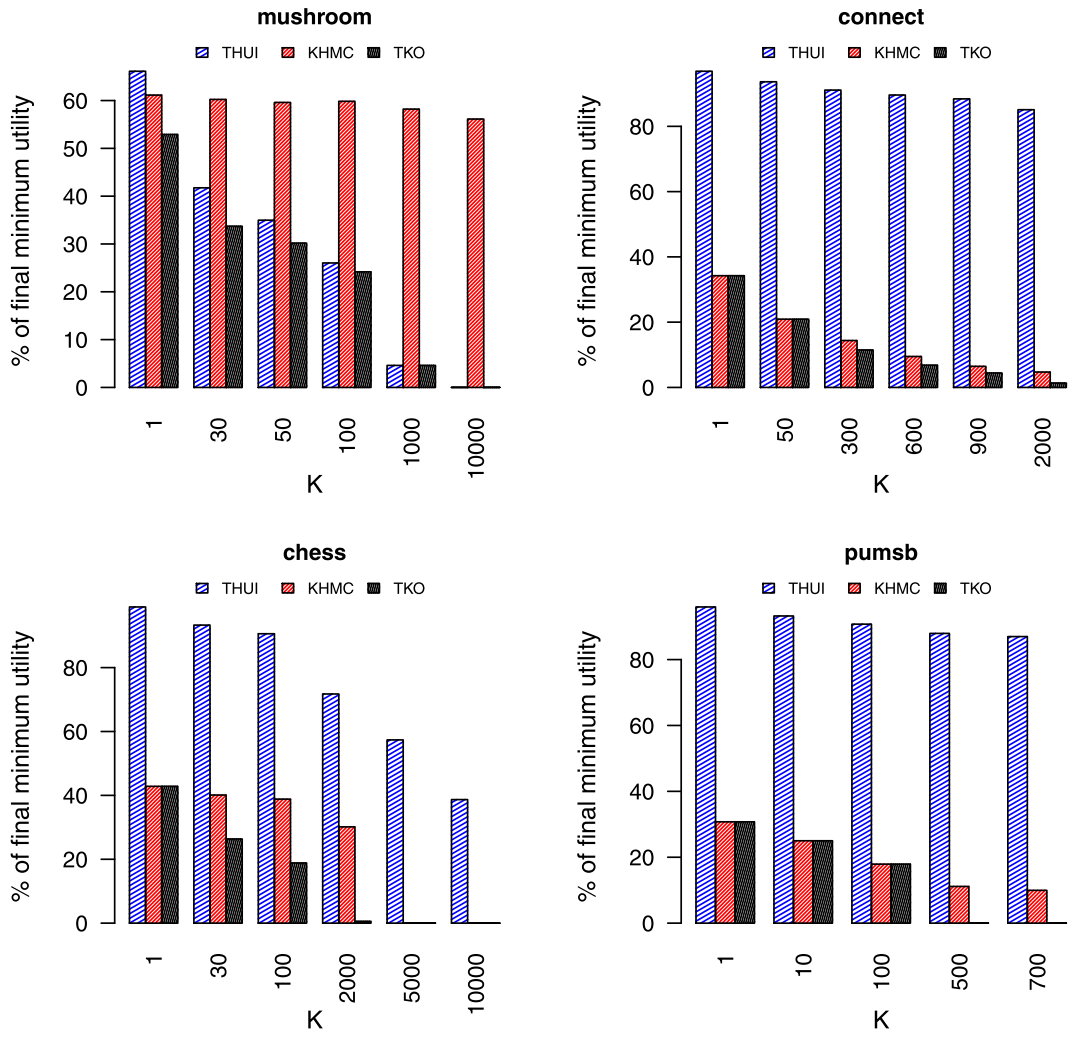

Fig. 10. Effectiveness of threshold raising strategies on dense datasets.

in the range of 1 to 10. This is in line with past experimental studies in the literature (Ahmed et al., 2009; Liu & Qu, 2012; Liu et al., 2005). We conduct rigorous experimental evaluation on wide variety of these datasets to demonstrate the utility of our ideas.

The base version of the KHMC (Duong et al., 2016) algorithm was shared to us by the authors. We completed the KHMC implementation as per the descriptions provided in the paper. We implemented two versions of KHMC, one with transitive extension property (TEP) enabled and another without TEP. It is to be noted that TEP requires a pure depth-first itemset tree exploration. However, a standard utility list based approach perform an iterative depth-first itemset tree exploration. Our evaluation of the two KHMC implementations revealed that KHMC without TEP produces far lesser number of candidates and performs significantly better. Hence, we used the KHMC without TEP in all our experimental evaluations.

We also implemented TKO (Tseng et al., 2016) and THUI algorithms in Java. All three algorithms (KHMC, TKO and THUI) were used for rigorous experimental evaluation of the proposed ideas for top-k HUI mining.

5.2. Performance analysis on sparse datasets

In the first set of experiments, we analyzed the performance of each of the algorithms on sparse benchmark datasets. The THUI algorithm was found to perform better on almost all of the sparse datasets studied (Fig. 4). On kosarak and accidents datasets, the performance of KHMC and TKO was found to degrade considerably at higher values of K. We also observed that KHMC runs out of memory and fails to execute at very high values of *K*. This can be attributed to the use of the expensive coverage based threshold raising strategy that evaluates utilities of supersets based on the coverage of individual items.

The details of the number of candidates generated by each of these methods are presented in Fig. 5. The use of the new LIU based strategy was found to be very effective in the case of kosarak and accident datasets that generate substantially lower number of candidates at higher values of K. On other sparse datasets, the performance on all of the methods in terms of candidate sizes are almost similar.

5.3. Performance analysis on dense datasets

In the next set of experiments, we examined the performance of each of the algorithms on dense datasets. The results of our experiments are presented in Figs. 6 and 7. The runtime performance results for the proposed THUI algorithm was found to be significantly better. On mushroom, connect and chess datasets, the THUI algorithm showed remarkable performance (one to three orders of magnitude improvement) compared to the most recent KHMC algorithm. On mushroom dataset, the KHMC algorithm was found to run out of memory for higher values of K (greater than 10000). The poor performance of KHMC algorithm on these datasets can be attributed to the use of expensive coverage based threshold rais-

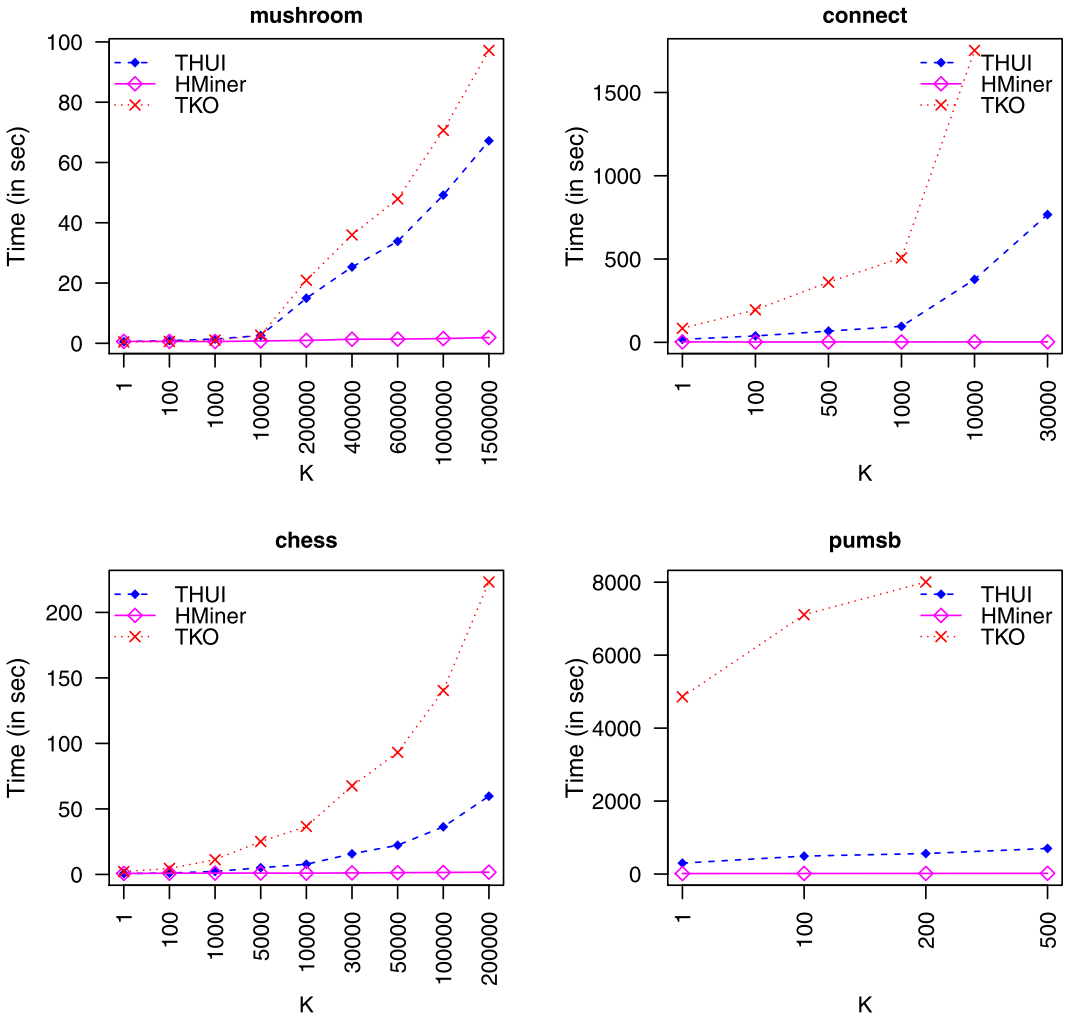

Fig. 11. Comparative evaluation of THUI and TKO against HMiner.

ing strategy. The TKO algorithm, on the other hand, shows comparable performance (though 1.5 to 3 times slower) on mushroom and chess datasets. It is to be noted that mushroom and chess are much smaller datasets. On the more dense and large connect and pumsb datasets, the performance of THUI algorithm was significantly better. On pumsb dataset the performance of TKO algorithm was an order of magnitude slower compared to the proposed THUI algorithm. Overall, the performance of THUI algorithm on dense datasets show very promising results and demonstrates the utility of the proposed data structure (LIU) and threshold raising strategies (LIU-E and LIU-LB).

The memory consumption analysis of each of the algorithms are shown in Fig. 8. The proposed algorithm was found to consume lesser memory compared to the state-of-the-art methods, especially at higher values of K. Overall, the memory consumption requirements of KHMC for most of the dense datasets are quite significant. This can be attributed to the use of expensive coverage based threshold raising strategy in KHMC.

5.4. Effectiveness of threshold raising strategy

We also assessed the effectiveness of the proposed threshold raising strategies on each of the datasets. For this experiment, we examined the δ value produced before the growth stage (i.e. δ_I). The ratio of the value of δ_I to the optimal final δ_F was computed. The computed results are reported in Figs. 9 and 10. The results for the sparse datasets reveal that the THUI algorithm produces competitive performance on most of the datasets. In the case of accidents dataset, the performance of THUI was found to be superior. This can be attributed to the moderately long nature of the accidents dataset.

The performance results for the dense datasets, as expected, was far superior compared to the state-of-the-art methods. This clearly demonstrates the usefulness of the key ideas proposed in this paper. In the case of mushroom dataset, the performance of KHMC was found to be better due to the use of coverage based threshold raising strategy. However, applying the coverage based threshold raising strategy is quite expensive. The KHMC algorithm takes enormous amount of time and runs out of memory at higher values of K. This is clearly evident from our earlier experimental analysis results in Figs. 6 and 8.

5.5. Comparison with other optimal methods

We examined the performance of THUI and TKO against an optimal method that uses the δ_F value directly to mine the top-k HUIs. We used the HMiner (Krishnamoorthy, 2017) algorithm for the analysis. The method uses a compact utility list data structure and was shown to perform better than other state-of-the-art methods.

The results of our analysis are shown in Fig. 11. We performed comparative evaluation against TKO method that was found to be

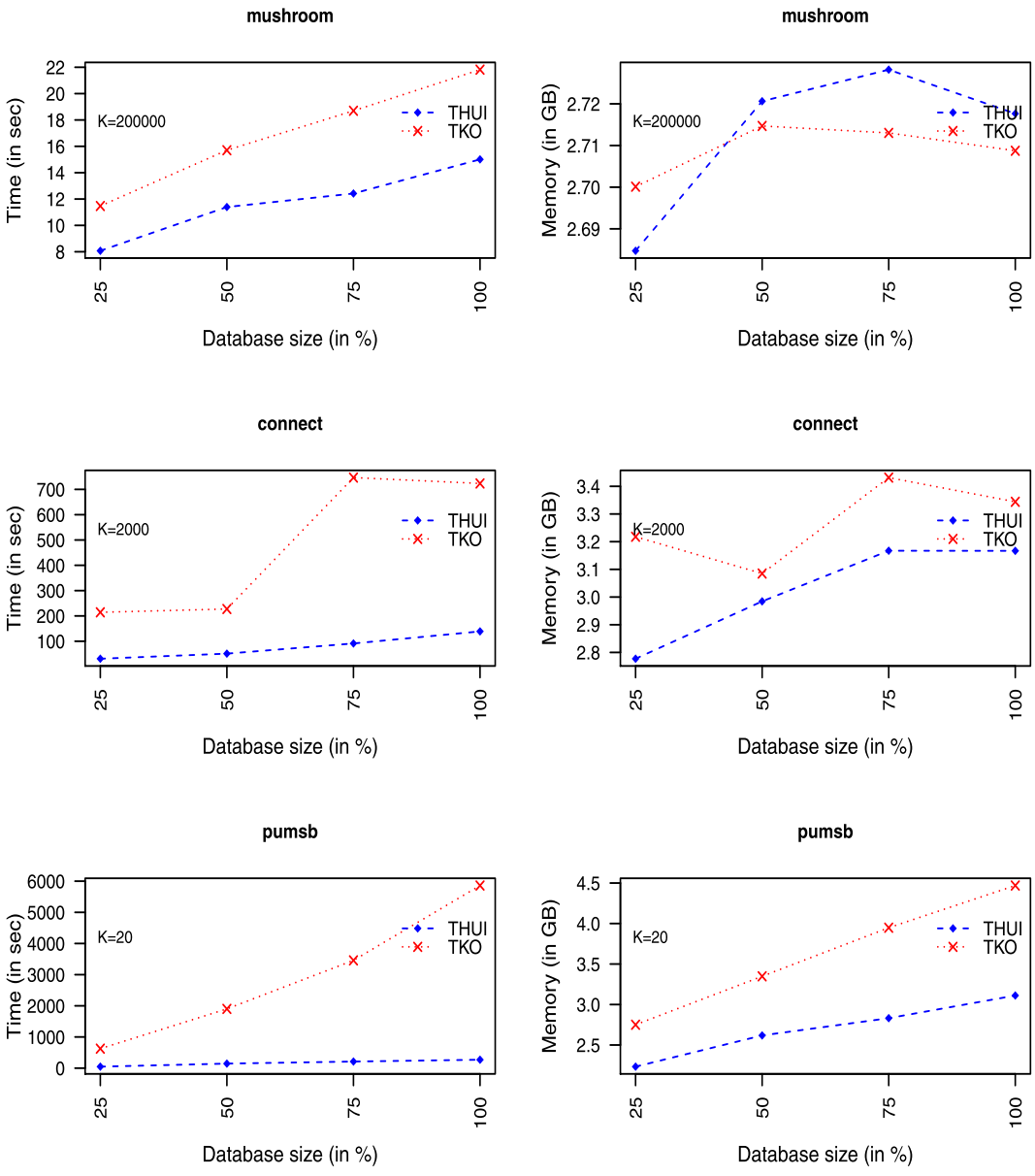

Fig. 12. Scalability experiments.

superior compared to KHMC on most of the dense datasets studied (refer to Section 5.3). HMiner method that directly uses δ_F delivered the best results, as expected, in all of the cases. THUI method was found to deliver close to optimal performance on most of the dense datasets studied.

5.6. Analysis of leaf size

One of the key ideas proposed in this paper is the use of the LIU data structure for effectively capturing the utilities of long itemsets. Theoretically, the space required for the LIU structure can be verified as $O(m^2)$, given the matrix structure illustration in Fig. 1. In practice, the size of this structure is unlikely to be large. In order to validate our intuition, we conducted experiments to assess the typical size of LIU for each of the datasets for a relatively large value of K. The results of our experiment are presented in Table 8. The columns 3, 4 of the table show the square root of the size of the LIU and the average length of transactions in the original database (L). One can clearly observe that the space requirements of LIU are extremely small. The space requirement can be approximated to $O((cL)^2)$, where is c is a constant. The constant factor is higher for datasets like kosarak and chain that are highly sparse and contains very large number of items (refer to Table 7 for dataset characteristics).

Table 8
Analysis of LIU sizes.

Dataset	K	$\sqrt{\lvert LIU \rvert}$	AvgLen
retail	10K	38.00	10.30
kosarak	300	141.40	8.10
chain	800	63.80	7.30
accidents	5K	27.71	33.80
mushroom	1500K	13.86	23.00
chess	200K	27.78	37.00
connect	200K	26.63	43.00
pumsb	50K	47.37	74.00

5.7. Scalability experiments

In the final set of experiments, we studied the impact of changes in the database size on overall algorithm performance. We varied the database sizes in increments of 25% and studied the runtime and memory consumption performance of THUI and TKO algorithms. The results of our experiments are presented in Fig. 12. We restrict our comparative evaluation to TKO algorithm as KHMC algorithm show poor results both on runtime and memory consumption performance on highly dense and large datasets.

The proposed method was observed to scale very well on all of the dense and large datasets studied. On a relatively smaller and sparser mushroom dataset, the memory requirement was found to be marginally higher.

5.8. Discussion

From the foregoing discussions, it is quite evident that the proposed ideas are quite useful for mining top-k high utility itemsets. The threshold raising strategy was observed to be very effective for most of the datasets studied. In this section, we discuss the implications of our algorithm design choices and present a few key insights based on our experimental analysis.

Choice of K (exact vs flexible): There are slightly varying definitions of top-k HUI mining in the literature. The actual choice of the nature of K is largely dependent on the application requirements. We utilized the exact K like the most recent KHMC method (Duong et al., 2016). From the observed results, the differences were found to be marginal for the two scenarios. Therefore, the choice of the nature of K has a limited impact on the overall findings made in this paper.

Choice of utility list (normal vs compact): We chose to use the normal utility list (Liu & Qu, 2012) in all our experimentation. Future work can consider the more recent compact utility list (Krishnamoorthy, 2017) data structure to further improve the performance of top-k HUI mining.

Number of exclusions for LB estimation: We adopted a heuristic strategy to create additional itemset subsets for a given leaf itemset. The lower bound utility values were then estimated for the generated itemsets. Theoretically, one can generate 2^{p-2} itemset subsets for a given p-itemset. However, we relied on a simple heuristic scheme to exclude items in the range of 1 to 3 to generate itemset subsets. This choice is driven by the trade-offs involved between the efficiency of computations and the effectiveness of lower bound estimates. It is to be noted that generating subsets with large number of item exclusions can result in poor lower bound estimates.

Overall, our work makes useful contributions to the HUI mining literature. The core ideas presented in this paper are found to be extremely effective for mining top-k HUIs, especially for highly dense datasets.

6. Conclusion

We presented a THUI algorithm for efficiently mining top-k HUIs. The presented method used a utility list data structure and followed a one-phase approach to mine top-k HUIs. The key novelty of the approach is the use of Leaf Itemset Utility (LIU) data structure, with extremely small memory footprint, for storing utility information. Two new threshold raising strategies were introduced, namely LIU-E and LIU-LB. Both of these strategies were found to be quite effective in raising the minimum utility threshold value during early stages of mining. Our rigorous experimental evaluation reveal that the proposed THUI algorithm delivers one to three orders of magnitude runtime improvement over the state-of-the-art methods, especially on large, dense and long average transaction length datasets. The memory requirements are also observed to be quite low.

Future work can consider further optimization in LIU data structures. For example, the mushroom datasets had very large number of non-contiguous patterns and LIU structure was found to be less effective in raising the threshold value. It might be useful to consider extensions to LIU structure to improve the performance of top-k HUI mining on such datasets. The EUCS (Fournier-Viger et al., 2014b) and LIU data structures can be visualized as operating at two extreme ends of the spectrum. While the former captures utilities of every pair of items, the latter captures utilities of long contiguous sequence of items. Future work can consider materializing more promising non-contiguous itemsets to improve the performance of top-k HUI mining. The proposed ideas can also be adapted to several top-k mining variants such as on-shelf utility mining, data stream mining and sequential pattern mining.

References

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. In *Proceedings of the 20th international conference on very large databases, VLDB* (pp. 487–499).

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

Cheung, Y.-L., & Fu, A. W.-C. (2004). Mining frequent itemsets without support threshold: With and without item constraints. *IEEE Transactions on Knowledge and Data Engineering, 16*(9), 1052–1069.

Dam, T.-L., Li, K., Fournier-Viger, P., & Duong, Q.-H. (2017). An efficient algorithm for mining top-k on-shelf high utility itemsets. *Knowledge and Information Systems*, 1–35.

Dawar, S., Sharma, V., & Goyal, V. (2017). Mining top-k high-utility itemsets from a data stream under sliding window model. *Applied Intelligence*, 1–16.

Duong, Q.-H., Liao, B., Fournier-Viger, P., & Dam, T.-L. (2016). An efficient algorithm for mining the top-k high utility itemsets, using novel threshold raising and pruning strategies. *Knowledge-Based Systems, 104*, 106–122.

Fournier-Viger, P., Gomariz, A., Soltani, A., Lam, H., & Gueniche, T. (2014a). SPMF: Open-source data mining platform. http://www.philippe-fournier-viger.com/spmf.

Fournier-Viger, P., Lin, J. C.-W., Kiran, R. U., Koh, Y. S., & Thomas, R. (2017). A survey of sequential pattern mining. *Data Science and Pattern Recognition, 1*(1), 54–77.

Fournier-Viger, P., Wu, C.-W., Zida, S., & Tseng, V. S. (2014b). FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning. In *International symposium on methodologies for intelligent systems* (pp. 83–92).

Fournier-Viger, P., & Zida, S. (2015). FOSHU: Faster on-shelf high utility itemset mining–with or without negative unit profit. In *Proceedings of the 30th annual acm symposium on applied computing* (pp. 857–864).

Gan, W., Lin, J. C., Fournier-Viger, P., Chao, H., Tseng, V. S., & Yu, P. S. (2018). A survey of utility-oriented pattern mining. *CoRR*. abs/1805.10511.

Krishnamoorthy, S. (2015). Pruning strategies for mining high utility itemsets. *Expert Systems with Applications, 42*(5), 2371–2381.

Krishnamoorthy, S. (2017). HMiner: Efficiently mining high utility itemsets. *Expert Systems with Applications, 90*, 168–183.

Krishnamoorthy, S. (2018). A comparative study of top-k high utility itemset mining methods, 2018. https://arxiv.org/abs/1809.00792.

Lan, G.-C., Hong, T.-P., & Tseng, V. S. (2011). Discovery of high utility itemsets from on-shelf time periods of products. *Expert Systems with Applications, 38*(5), 5851–5857.

Lin, J. C.-W., Fournier-Viger, P., & Gan, W. (2016a). FHN: An efficient algorithm for mining high-utility itemsets with negative unit profits. *Knowledge-Based Systems, 111*, 283–298.

Lin, J. C.-W., Gan, W., Fournier-Viger, P., Hong, T.-P., & Zhan, J. (2016b). Efficient mining of high-utility itemsets using multiple minimum utility thresholds. *Knowledge-Based Systems, 113*, 100–115.

Lin, J. C.-W., Gan, W., Hong, T.-P., & Tseng, V. S. (2015). Efficient algorithms for mining up-to-date high-utility patterns. *Advanced Engineering Informatics, 29*(3), 648–661.

Lin, J. C.-W., Ren, S., Fournier-Viger, P., Hong, T.-P., Su, J.-H., & Vo, B. (2017). A fast algorithm for mining high average-utility itemsets. *Applied Intelligence*, 1–16.

Liu, J., Wang, K., & Fung, B. (2012). Direct discovery of high utility itemsets without candidate generation. In *Proceedings of the IEEE 12th international conference on data mining* (pp. 984–989).

Liu, M., & Qu, J. (2012). Mining high utility itemsets without candidate generation. In *Proceedings of the 21st ACM international conference on information and knowledge management* (pp. 55–64). doi:10.1145/2396761.2396773.

Liu, Y., Liao, W.-K., & Choudhary, A. (2005). A two-phase algorithm for fast discovery of high utility itemsets. In T. Ho, D. Cheung, & H. Liu (Eds.), *Advances in knowledge discovery and data mining*. In *Lecture Notes in Computer Science: 3518* (pp. 689–695).

Pisharath, J., Liu, Y., Liao, W.-K., Choudhary, A., Memik, G., & Parhi, J. (2005). NU-MineBench 2.0. *Tech. Rep*. Department of Electrical and Computer Engineering, Northwestern University,.

Quang, T. M., Oyanagi, S., & Yamazaki, K. (2006). Exminer: An efficient algorithm for mining top-k frequent patterns. In *International conference on advanced data mining and applications* (pp. 436–447). Springer.

Ryang, H., & Yun, U. (2015). Top-k high utility pattern mining with effective threshold raising strategies. *Knowledge-Based Systems, 76*, 109–126.

Ryang, H., & Yun, U. (2016). High utility pattern mining over data streams with sliding window technique. *Expert Systems with Applications, 57*, 214–231.

Salam, A., & Khayal, M. S. H. (2012). Mining top-k frequent patterns without minimum support threshold. *Knowledge and information systems, 30*(1), 57–86.

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

Tseng, V. S., Wu, C.-W., Fournier-Viger, P., & Philip, S. Y. (2016). Efficient algorithms for mining top-k high utility itemsets. *IEEE Transactions on Knowledge and Data Engineering, 28*(1), 54–67.

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

Wu, C. W., Shie, B.-E., Tseng, V. S., & Yu, P. S. (2012). Mining top-k high utility itemsets. In *Proceedings of the 18th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 78–86). ACM.

Yao, H., & Hamilton, H. J. (2006). Mining itemset utilities from transaction databases. *Data & Knowledge Engineering, 59*(3), 603–626.

Yin, J., Zheng, Z., Cao, L., Song, Y., & Wei, W. (2013). Efficiently mining top-k high utility sequential patterns. In *Ieee 13th international conference on data mining ICDM* (pp. 1259–1264). IEEE.

Zida, S., Fournier-Viger, P., Lin, J. C.-W., Wu, C.-W., & Tseng, V. S. (2017). EFIM: A fast and memory efficient algorithm for high-utility itemset mining. *Knowledge and Information Systems, 51*(2), 595–625.

Zihayat, M., & An, A. (2014). Mining top-k high utility patterns over data streams. *Information Sciences, 285*, 138–161.