Journal of Network Intelligence ©2025 ISSN 2414-8105 (Online)
Taiwan Ubiquitous Information Volume **10**, Number **1**, February **2025**

THL: Fast Targeted High Utility Itemset Mining within Length Consideration

Peiming Xu[1], Yixin Jiang[1], Xiaoyun Kuang[1], Yu Luo[2], Jiahui Chen[2,*]

[1] Guangdong Provincial Key Laboratory of Power System Network Security
CSG, Guangzhou 510000, China.
xupm@csg.cn, jiangyx@csg.cn, kuangxy@csg.cn
[2] School of Computer Science and Technology
Guangdong University of Technology, Guangzhou 510006, China
yuluo@gdut.edu.cn, csjhchen@gmail.com

*Corresponding author: Jiahui Chen

Received August 29, 2024, revised October 30, 2024, accepted December 17, 2024.

Abstract.
High utility itemset mining is a research topic of the pattern mining domain, which has been widely applied in reality during the past decade. Although most high utility itemset mining algorithms show significant execution performance regarding runtime and memory usage, numerous results are useless for users. Targeted high utility itemset mining can reduce the number of high utility itemsets and improve the efficiency of pattern analysis. However, the targeted high utility itemset mining task aims to find targeted high utility itemsets with as many items as possible. Users are not that interested in long results in actual scenarios. Therefore, in this article, we first formulate the problem and then address this issue by proposing a new algorithm named THL for mining targeted high utility itemset within length constraints, based on utility list structure. Furthermore, to improve the performance of the novel algorithm, THL adopts several efficiency pruning strategies, including length and utility aspects. Finally, we conduct an extensive experimental evaluation to compare the SOTA algorithms with THL on several datasets.
Keywords: targeted items, high utility itemset, data mining, length constraint

1. **Introduction.** With the rapid development of information technology and database management systems, extracting potential information, patterns, and knowledge from massive data is an important task in data mining. Data mining technology involves numerous computer science principles and techniques, such as applied mathematics, statistics, machine learning, and database systems. The application of data mining technology can help people to extract hidden patterns, rules, and associations from numerous data, and help to reveal the valuable information behind the data. Data mining technology can help enterprises more accurately understand market demand, target customer groups and competitors, develop more effective marketing strategies, and enhance market competitiveness. For example, data mining technology is used to analyze users 'interests and hobbies, achieve personalized recommendations of goods, forecast sales, and optimize supply chain management to improve users' shopping experience and increase sales. Secondly, data mining technology can build predictive models to help predict the probability of future events' occurrence, provide effective decision support for decision-makers, reduce risks, and optimize decision effects. For example, banks and financial institutions can use data mining technology to detect fraud, assess credit risks, predict stock market trends, and help risk management and investment decisions. As a research topic in data mining, association rule mining (ARM) [1] aims to discover the correlation between different frequent itemsets.

ARM algorithm considers the factors, including support and confidence, to find frequent and high confidence patterns. Among them, based on the setting of the support threshold, the ARM algorithm can mine frequent itemsets and rare itemsets, so researchers have proposed frequent pattern mining (FPM) [2] and rare pattern mining (RPM) [3]. FPM can find all itemsets or other types of subpatterns in the transaction database that occur at least as often as a user-specified minimum threshold. Contrary to FPM, RPM focuses on finding unusual or rare patterns in the data, which is important for identifying abnormal events, potential risks, or novel trends. After more than two decades of research, FPM and RPM have become the aggregated topics and important research tasks in data mining domain, including frequent itemset mining [4], frequent sequential pattern mining [5], and rare itemset mining [6], etc.

However, only using frequency to measure user behavior can no longer meet the needs of today's e-commerce development. FPM suffers from the unavoidable drawback that it assumes that all items/itemsets have the same utility value (e.g., profit, weight, or risk). For example, in daily life, the purchasing behavior of consumers often presents diversified characteristics. Take supermarket sales as an example. As a daily consumable, the sales of paper towels are usually far higher than that of high-end wine. However, although the sales of high-end wine are not high, its unique brewing process, scarce raw materials, and deep cultural heritage make it far more expensive and profitable than paper towels. In the field of Frequent Pattern mining (FPM), fine wine may be wrongly perceived as uninteresting to consumers if we analyze items based on the single factor of purchase frequency. But in fact, high-end wine plays an important role in improving the brand image of supermarkets, attracting high-end consumers, and increasing overall profits. Therefore, in the commodity's analysis, we need to adopt a more comprehensive and accurate method to evaluate and develop more precise and effective marketing strategies to maximize profits.

To address the issue of frequent pattern mining, utility pattern mining (UPM) [7] is proposed. Compared with FPM, UPM considers the problem from a more comprehensive perspective. It considers both the internal utility (such as sales volume) and external utility (such as unit price) of an item and obtains high utility itemsets from the transaction database. What's more, there is still a shortage in traditional utility pattern mining, which focuses on discovering the combinations of itemsets with actual value and utility to the business but ignores the hidden internal relationship and interdependence between itemsets. Taking online shopping on e-commerce platforms as an example, when consumers buy a computer, they often expect the background system to intelligently recommend highly relevant accessories, such as mice, keyboards, etc., rather than simply pushing those products with high utility value but not strong relevance, such as mobile phones or TVs. Neglecting this internal relationship may lead to the deviation between the recommendation results and the actual needs of users, and then affect the maximization of sales effect. Therefore, how to deeply mine and utilize the internal relationship between these items to make the recommendation more accurate and more in line with user expectations has become a key issue to be solved. Recently, Gan *et al.* [8] proposed a high utility itemset mining algorithm based on strong association, which aims to mine the premise of high-profit goods and explore the relationship between goods. Based on utility mining, this mining mode uses a prior knowledge method named Kulc to detect whether the mined itemsets have high specific relevance attributes. The strong correlation and high utility itemset mining algorithm can be effectively applied to different e-commerce platforms. The background system can better analyze users' purchasing habits and behaviors, accurately recommend relevant products, optimize business strategies, and effectively improve the competitiveness of e-commerce platforms and user loyalty.

On the other hand, with the increasing diversification and personalization of consumer needs, user-centered marketing strategies based on users' interests and needs have become mainstream. Taking e-commerce platform sales as an example, when users buy goods, merchants often tend to bundle high-profit products, such as technology products, to maximize profits. However, from the user's point of view, even if there is a certain internal relationship between technology products, users are more eager to get recommendations for products with strong goals. For example, suppose the user needs a computer for a certain time. In that case, the product that should be recommended to the user must be the computer as the primary product, and the mouse, headset, etc., are recommended as ancillary products. In the meanwhile, products that are not directly related to the needs of computers, such as mobile phones and TVs, should be avoided to ensure the accuracy and effectiveness of the recommendation. Therefore, a goal-oriented information mining method based on user needs or specific goals is another feasible method. Recently, Miao *et al.* [9] applied the goal attribute to the field of high utility itemset mining for the first time, aiming to mine the combinations of itemsets that satisfy both high utility and strong goals. At the same time, the concept of goal high utility itemsets mining algorithm is proposed, and the algorithm realizes the purpose of quickly querying the required goal itemsets by building the goal decision tree. Target high utility itemset mining can be a good way to meet the needs of users and increase the sales or profits of merchants. However, in some specific fields, such as potential credit risk detection, high-end luxury recommendation, abnormal data detection, etc., unique or abnormal patterns hidden behind the data often contain great value and insight. Therefore, it is important to consider the rare property in the field of high utility itemset mining. In the high-end luxury scene, for example, limited edition jewelry, limited edition art, or special holiday commemorative goods are often favored by certain consumers and collectors due to their scarcity and uniqueness. Due to infrequent sales, cognitive bias, or demand dispersion, enterprises tend to pay more attention to commodities and models with large sales volume and wide audiences, while ignoring those rare and unique commodities may bring new innovation opportunities and competitive advantages. Therefore, to comprehensively consider various factors, Zhang et al. [10] proposed a goal-based high utility rare itemset mining framework. The framework considers three factors simultaneously: goal, rarity, and utility, and introduces the concept of goal high utility rare itemsets. Target high utility rare itemset mining can help decision-makers better understand rare patterns and associations in data and support decision-making, product optimization, and marketing strategy formulation.

To sum up, the existing algorithms for mining high utility itemsets and target high utility itemsets still have the following problems:

1. Efficiency: how to avoid the problem that the existing algorithms generate numerous irrelevant candidates in the mining process.
2. Scalability: the existing algorithms have the problems of slow running efficiency and large memory consumption.
3. Effectiveness: How to design a more efficient pruning strategy to improve the performance of the algorithm.

We notice the length factor in the targeted itemset mining task is a good method to tackle these, and then we propose a new utility list-based targeted itemset mining algorithm considering length constraints. Especially, we also consider users are usually clear about how many items they need at least but often have trouble in at most. Therefore, we assume the minimal length is the number of user-specified targeted items, which reduces the operation complexity of the algorithm to a certain degree. By these considerations, this work advances research through the following contributions:

1. We analyze the current situation and existing problems and propose a new utility list-based targeted itemset mining algorithm named THL, which considers length constraints. In particular, we construct a new data structure called the co-occurrence structure, which holds the pruning information and can effectively narrow the search space.

2. We propose several effective pruning strategies that take into account the length constraints and users' understanding of the number of items and give analysis to show that THL can support significantly improved performance of targeted mining algorithms.
3. We conduct several experiments on significant transaction datasets to assess the performance of THL and the state-of-the-art algorithms THUIM and TIRUP in terms of Runtime, Memory, Candidate generations, and scalability. More precisely, the runtime consumption of THL is up to 40 times faster than that of THUIM, and the memory usage of THL is up to 4 and 10 times smaller than that of THUIM on the BMSPOS dataset. The runtime consumption of THL is about 4 times faster than that of TIRUP, while the memory consumption of TIRUP and variant THL algorithms are similar on the Kosarak dataset.

The remain of this article is structured as follows: Section 2 presents the related work, providing context and background. Section 3 provides problem definitions and establishes essential preliminaries. Section 4 provides an in-depth examination of the THL algorithm, outlining its core structures, strategies, and how they function together. Section 5 is devoted to experiments and result analysis, where we present compelling evidence for THL's efficiency and scalability. Finally, Section 6 concludes with a recap of the research contributions and outlines future directions for further extending this work.

2. **Related work.**

2.1. **High utility itemset mining.**

A utility-based approach is a measure that considers not only the statistical aspects of the raw data but also the utility of the mined patterns. Since Shen *et al.* [11] first introduced the utility concept from the financial domain to the data mining algorithm, high utility itemset mining is a relatively mature and well-researched topic after decades. Yao *et al.* [12] applied the mathematical property of utility constraints to find high utility itemsets, and then proposed a generic framework for solving utility itemset mining tasks [13]. However, the biggest issue of their work is wasting so much execution time to filter out real interesting itemsets, because the utility measure is neither anti-monotonic nor monotonic. Therefore, a "two-phase" algorithm [14] first designed a significant utility upper-bound, called transaction-weighted utilization (simplified as *TWU*). The *TWU* of an itemset is defined as the sum of transaction utility such that the transaction contains the itemset, where the transaction utility is equal to the sum of contained items. If *TWU* of an itemset is less than a user-defined minimum utility threshold, the utility values of itemset and its super-itemsets must be less than the threshold too. Besides, the "two-phase" means the algorithm utilizes a kind of "generation then test" paradigm. In the first phase, the algorithm adopts *TWU* upper-bound to filter out potential high utility itemsets such that utility values are no less than the threshold. Then, in the second phase, the algorithm recalculates the real utility of these potential high utility itemsets and outputs a complete set. The algorithm repeats the above steps until there is no new itemset generated.

Nevertheless, a significant shortcoming of the "two-phase" algorithm is the number of candidate generations is huge, which is intolerable for users. Then, more efficient "one-phase" algorithms were proposed [15, 16, 17, 18, 19, 20, 21, 22]. HUI-Miner [15] is a famous utility list-based mining algorithm. A utility list of an itemset is a set of triples, and each triad contains necessary utility information about the itemset in a transaction. The algorithm constructs a utility list of a high-level itemset according to interactive utility lists of two low-level itemsets. The advantages of the utility list structure are located in two aspects: 1) only scanning the database twice; and 2) the generalization is better than previous data structures (e.g., tree). Nevertheless, these algorithms may face a "combinatorial explosion" of high-level itemsets since the number of itemsets may be very large, especially since the database is dense. A pseudo-projection-based approach called EFIM [23] was proposed. With the assistance of transaction merging and database projection technologies, EFIM greatly compresses the size of the database in memory. Furthermore, the EFIM algorithm proposed two efficient utility upper-bounds named subtree utility and local utility to prune search space. After that, to meet the requirements of various domains, several EFIM extension algorithms have been proposed, which allow taking into account constraints and more complex data types [24, 25, 26, 27].

2.2. **targeted itemset mining.** However, for the itemset mining task, the redundancy issue of the amount of candidate generations is not fully addressed in the literature. The reason is that these itemset mining algorithms lack user interaction and demand. To reduce irrelevant itemsets discovered, the targeted itemset mining task was proposed [28]. The Itemset Tree [29] was the first data structure designed for discovering targeted itemsets. It converts essential information from a transaction database into a tree structure, with leaf nodes representing transaction records. To identify a complete set of targeted itemsets for a query task, it checks nodes from the top downward. After that, there are a lot of works of literature studied for improving the performance of matching targeted itemsets by various modified trees [30, 31, 32]. Recently, Shabtay *et al.* [33] modified the classic FP-Growth algorithm for solving multiple querying issues.

The targeted itemset mining task is likely a searching process. It serves many applications well because avoiding unnecessary itemset generations. However, the above-mentioned targeted itemset mining algorithms only consider the frequency metric. The frequency of an itemset is not a sufficient indicator of interestingness because it only reflects the number of transactions in the database that contain the itemset. In a utility-oriented itemset mining task, each item has a unit utility (e.g., unit profit) and can appear more than once in each transaction or event (e.g., purchase quantity). TD-FVAUFM [34] is a simple approach that considers both frequency and utility metrics for addressing prediction tasks on medical databases. Then, Miao *et al.* [9] proposed to identify high utility item sets that correspond to the query parameters. It is a tree-based algorithm designed to create a lexicographic tree (referred to as TP-Tree) for storing all high utility itemsets. Initially, these high utility itemsets are identified by other high utility itemset mining algorithms. Subsequently, the TP-Tree is built, and the target search process commences. During the mining process, two threshold constraints on utility are established to ensure the retrieval of the desired results. Besides, three effective pruning strategies, based on the transaction-weighted utilization and the remaining utility of itemsets, respectively, are estimates that play a significant role in reducing the search space for potential target high utility itemsets. In addition, to meet the requirements of various domains, several targeted sequential pattern mining extensions have been proposed [35, 36, 37, 38], which consider various constraints and more complex data types.

3. **Preliminaries and problem definition.** Assuming a finite set $I = \{x_1, x_2, \ldots, x_n\}$ consists of n distinct items, and an itemset X is a subset of I, where l-itemset contains l different items. For instance, an item is the 1-itemset. In this paper, we also utilize utility to show users' interestingness of each item or itemset, whereby the utility of an item ($U(x_i, T_j)$) in a transaction T_j is the product of its internal utility ($q(x_i, T_j)$) and external utility ($p(x_i)$). In addition, the utility of an itemset X is the summation of utilities of consisted items, i.e., $U(X) = \sum_{x_i \in X \wedge T_j \in \mathcal{D}} U(x_i, T_j)$. Furthermore, as shown in Table 1, a transaction is assigned with a unique ID, call *Tid*. Then, a transaction contains finite distinct items and their corresponding internal utilities. A simple transaction database is composed of several transactions. Besides, in this paper, we assume external utilities of items a, b, c, d, e, and f are \$1, \$3, \$5, \$2, \$7, and \$9, respectively.

Definition 1 (Transaction-weighted utilization [14]). The utility of a transaction T_j is denoted as $U(T_j) = \sum_{x_i \in T_j} U(x_i) = \sum_{x_i \in T_j} u(x_i) \times u(x_i, T_j)$. Similarly, the transaction-weighted utilization

Table 1. A simple transaction database

Tid	**Transaction**
T_1	$(a, 2), (b, 4), (f, 1)$
T_2	$(c, 5), (d, 3), (e, 2)$
T_3	$(b, 1), (d, 4), (e, 4)$ $(f, 1)$
T_4	$(a, 6), (f, 2)$
T_5	$(b, 4), (c, 2), (e, 2)$
T_6	$(a, 3), (b, 2), (c, 2), (d, 6), (e, 2), (f, 1)$

of an item is defined as $TWU(x_i) = \sum_{x_i \in T_j \wedge T_j \in \mathcal{D}} U(T_j)$. Given an itemset X, $U(X) \leq TWU(X)$ always holds, and thus *TWU* is a utility upper-bound in search space.

Table 2. The transaction-weighted utilization of items

Item	a	b	c	d	e	f
TWU	\$101	\$161	\$183	\$147	\$183	\$149

Definition 2 (High utility itemset)**.** Considering a user-predefined minimum utility threshold γ (simplified as *minUtil*), if the utility of an itemset is higher than or equal to *minUtil*, that is $U(X) \geq \gamma$, we suppose the itemset is a high utility itemset (abbreviated as HUI).

Definition 3 (Matching)**.** Considering two distinct itemsets X and Y, if $X \subseteq Y$, then we suppose Y matches with X or X is matched by Y; otherwise, we assume X and Y mismatch.

Definition 4 (targeted high utility itemset)**.** Assuming a user-specified set of items $\mathcal{T} = \{x_1, \ldots, x_k\}$, where $1 \leq k \leq n$ and $x_k \in I$, if a HUI X contains all items of $\mathcal{T}$ (i.e., $\mathcal{T} \subseteq X$), X is a targeted high utility itemset (abbreviated as THUI).

Problem statement: Let *minUtil*, *minLength*, *maxLength*, and $\mathcal{T}$ be parameters set by user. The problem of targeted high utility itemset mining with length constraints task is to find a complete set of high utility itemsets matching all targeted items and containing at least *minLength* items, and at most *maxLength* items.

4. The Proposed Algorithm.

Since Liu *et al.* [15] first proposed the utility list structure, the utility list has been extensively studied in the past decade. It has been well-recognized that the size of lists is orders of magnitude smaller than the original database. The novel algorithm also modified the utility list to discover targeted high utility itemsets respecting length constraints. Herein, we take some introduce as follows:

4.1. Utility list structure.

Definition 5 (The global order)**.** In fact, the targeted high utility itemset mining algorithm can also be regarded as a depth-first search approach. The search space of the new algorithm is a targeted set-enumeration tree. Thus, all items are sorted by *TWU*-ascending order. Besides, if two itemsets have the same *TWU*, we take alphabetical order as supplementary. For instance, considering Table 2, the global order of items is $a \prec d \prec f \prec b \prec c \prec e$.

Definition 6 (Extending items)**.** After sorting items with the global order, given an item x_i, the items after x_i are called extending items and defined as $E(x_i) = \{x_j \mid x_i \prec x_j \wedge x_j \in I\}$.

Definition 7 (Remaining items)**.** A transaction is called a revised transaction if its elements (i.e., items) are sorted by *TWU*-ascending order. Taking the transaction T_1 as an example, the revised transaction is "(a, 2), (f, 1), (b, 4)". Furthermore, given an itemset X and a revised transaction T_j with $X \subseteq T_j$, the set of all the items after X in T_j is named remaining items and defined as $re(X, T_j) = \{x_j \mid x_i \prec x_j, \forall x_i \in X \wedge x_j \in T_j\}$. Hence, the sum utility of these remaining items of X in a revised transaction T_j is denoted as $reu(X, T_j) = \sum_{x_i \in re(X,T_j)} U(x_i, T_j)$. Furthermore, the remaining utility of an itemset in the database $\mathcal{D}$ is denoted as $reu(X) = \sum_{T_j \in \mathcal{D}} reu(X, T_j)$.

Definition 8 (Utility list of 1-itemset)**.** In the first database scan, *TWU* of items are calculated. Then, during the second database scan, the algorithm constructs utility lists for 1-itemsets. As shown in Fig. 1, the utility list is composed of several triples, and each tuple contains three elements <*Tid*, *Iutil*, *Rutil*>. The first element indicates a transaction T_j containing the 1-itemset; the *Iutil* records the utility of 1-itemset in T_j, that is $U(x_i, T_j)$; the last field *Rutil* is the remaining utility of 1-itemset in T_j, i.e., $reu(X, T_j)$. In conclusion, the utility list structure stores the key information for pruning X and its super-itemsets. Because the sum of *Iutil* is the utility of X in the database, and the sum of *Rutil* reveals the maximum utility value that super-itemsets of X can reach.

{ *a* }		
Tid	***Iutil***	***Rutil***
T_1	\$4	\$21
T_4	\$6	\$18
T_6	\$3	\$49

{ *d* }		
Tid	***Iutil***	***Rutil***
T_2	\$6	\$39
T_3	\$8	\$40
T_6	\$12	\$37

{ *f* }		
Tid	***Iutil***	***Rutil***
T_1	\$9	\$12
T_3	\$9	\$31
T_4	\$18	0
T_6	\$9	\$28

{ *b* }		
Tid	***Iutil***	***Rutil***
T_1	\$12	0
T_3	\$3	\$28
T_5	\$12	\$24
T_6	\$4	\$24

{ *c* }		
Tid	***Iutil***	***Rutil***
T_2	\$25	\$14
T_5	\$10	\$14
T_6	\$10	\$14

{ *e* }		
Tid	***Iutil***	***Rutil***
T_2	\$14	0
T_3	\$28	0
T_5	\$14	0
T_6	\$14	0

FIGURE 1. The utility list of 1-itemsets.

Definition 9 (Utility list of l-itemset)**.** Without scanning the database, the utility list of l-itemset ($l \geq 2$) can be constructed by the intersection of utility lists of two ($l - 1$)-itemsets, as shown in Fig. 2. The algorithm identifies the common transactions by comparing the *Tid*s in the two utility lists and then constructs a new one. It only needs ($m + n$) comparisons at most if the lengths of two utility lists are m and n, respectively. Furthermore, considering the common prefix of two itemsets Px and Py ($x \prec y$), the *Iutil* of the new itemset Pxy has computed the utility of the prefix $\{P\}$ twice. Hence, the correct *Iutil* is denoted as *Iutil*(Pxy) = *Iutil*(Px) + *Iutil*(Py) - *Iutil*($\{P\}$). In addition, the *Rutil* of Pxy is equal to *Rutil*(Py).

4.2. **Estimated utility co-occurrence structure.** As previous content was introduced, the utility list structure stores vital information about an itemset and reduces the database scanning times. However, an unavoidable issue is that computing the utility list of an itemset according

{ *d, e* }		
Tid	*Iutil*	*Rutil*
T_2	$20	0
T_3	$36	0
T_6	$26	0

{ *a, f* }		
Tid	*Iutil*	*Rutil*
T_1	$13	$12
T_4	$24	0
T_6	$12	$28

{ *b, c* }		
Tid	*Iutil*	*Rutil*
T_5	$22	$14
T_6	$14	$14

● ● ●

Figure 2. The utility list of l-itemsets ($l = 2$).

to the join operation is very costly. Herein, the new algorithm adopts a new data structure [16] to prune the low utility of 2-itemsets in the search space.

Definition 10 (EUCS)**.** The estimated utility co-occurrence structure (*EUCS*) consists of a set of triples of the form $(x, y, z) \in I \times I \times \mathbb{R}$. The z is indicated as $TWU(\{a, b\})$ and $z \neq 0$. As shown in Fig. 3, the *EUCS* can be regarded as a triangular matrix. If the database is more sparse (i.e., few items co-occurs with other items), the better pruning effect of *EUCS*.

item	***a***	***b***	***c***	***d***	***e***	***f***
a	0	$75	$52	$52	$52	$76
b		0	$88	$100	$136	$123
c			0	$97	$133	$52
d				0	$145	$100
e					0	$100

Figure 3. The estimated utility co-occurrence structure.

4.3. Pruning strategies.

In this subsection, we first discuss the length parameters and the size of a set of targeted items. Considering two user-specified thresholds (*minLength* and *maxLength*) and a set of targeted items $\mathcal{T}$, the size of the set of targeted items ($|\mathcal{T}|$) is usually the minimal number for user expect. It is reasonable to take $|\mathcal{T}|$ as *minLength* because users always know how many items they want at least. Most of the time, users are unsure how many items they need at most. For instance, in a retail store, customers will buy a bottle of water if they are thirsty, but if they are hungry, there is a lot of food to choose from. Therefore, we give some settings in this paper as follows:

Definition 11 (targeted items constraints)**.** We take the size of the set of targeted items ($|\mathcal{T}|$) as *minLength* threshold, and *maxLength* threshold must be greater than $|\mathcal{T}|$. Besides, if a transaction does not contain all targeted items, there is no need to continue mine.

Strategy 1. During the first database scanning, if the length of a transaction is less than *minLength*, the algorithm will mark and then remove this transaction; if a transaction does not contain all targeted items, the algorithm will remove this transaction too.

A naive method for utilizing length constraint is checking the length of the current itemset, if the number of its contained items is equal to *maxLength*, then the algorithm does not extend the itemset in the next iteration. Nonetheless, this method does not consider the utilities of itemsets. The novel algorithm focuses on mining HUIs that meet conditions. How to effectively reduce upper-bounds on the utilities of itemsets is vital for pruning the search space. Herein, we adopt some pruning technologies from the study [17].

Definition 12 (Revised transaction-weighted utilization)**.** Considering the length constraints, the largest utilities of itemset X in a transaction T_j consists of a set of top *maxLength* largest utilities of items. It is defined as $L(X, T_j) = \{U(x_1, T_j), \ldots, U(x_k, T_j)\}$, where $|L(X, T_j)| = maxLength$ and $k \leq n$. Then, the revised transaction-weighted utilization of an itemset X is defined as $RTWU(X) = \sum_{X \subseteq T_j \wedge \mathcal{T} \subseteq X} \sum L(X, T_j)$.

Similarly, within the length constraints, compared to the original *TWU*, $U(X) \leq RTWU(X) \leq TWU(X)$ always holds, where X is an itemset. Thus, we assume all 1-itemsets are sorted by *RTWU*-ascending order. Besides, if two itemsets have the same *RTWU*, we take alphabetical order as supplementary in this paper. In addition, we suppose a transaction is a revised transaction if its items are sorted by the global order in this paper. Then a modified database $\mathcal{D}'$ consists of revised transactions.

Strategy 2. Let X be an itemset, if $RTWU(X)$ is less than the user-defined minimal threshold γ, X and its super-itemsets cannot be THUIs respecting the *maxLength* constraint.

After utilizing the revised transcation-weighted utilization pruning strategy, the set of rest items is renamed I'. Then, the novel algorithm adopts *EUCS* structure to prune low utility 2-itemsets and all its transitive extensions. Furthermore, to improve the efficiency of *EUCS* structure, the new algorithm replaces original $z = TWU(\{x, y\})$ as $z = RTWU(\{x, y\})$, and the size of I' is far less than I because of targeted items constraints and revised transaction-weighted utilization pruning strategy.

Strategy 3. Given an *EUCS* structure, if there is tuple (x, y, z) in *EUCS* where $z < \gamma$, then itemset *xy* and all its super-itemsets cannot be THUIs. There is no need to construct a utility list of *xy* either.

Definition 13 (Revised remaining utility)**.** In a transaction T_j, the maximum number of items that extend an itemset X is defined as $|maxExtend(X, T_j)| = maxLength$ - $|X|$, where $|X|$ represents the length of X. Thus, the largest utilities in T_j w.r.t a matched itemset X is the set of top $|maxExtend(X, T_j)|$ largest values of remaining items. It is denoted as $maxExtend(X, T_j) = \{U(x_1, T_j), \ldots, U(x_k, T_j)\}$, where $x_k \in re(X, T_j)$ and $k \leq n$. Furthermore, the revised remaining utility of a matched itemset X in a transaction T_j is defined as $rreu(X, T_j) = \sum maxExtend(X, T_j)$. In addition, the revised remaining utility of X in the database $\mathcal{D}$ is denoted as $rreu(X) = \sum_{T_j \in \mathcal{D}} rreu(X, T_j)$.

Similarly, within the length constraints, compared to the original *reu*, $rreu(X, T_j) \leq reu(X, T_j)$ always holds.

Strategy 4. Given a matched itemset X, if the summation of $U(X)$ and $rreu(X)$ is less than the user-specified minimal threshold γ, then X and its super-itemsets are not THUIs respecting the *maxLength* constraint.

Definition 14 (Revised utility list)**.** The revised utility list of a matched itemset X in the database $\mathcal{D}$ also consists of a set of triples. The difference with the original utility list is that the third element (*Rutil*) is replaced by the set of revised remaining utilities (*Rrutil*), that is *maxExtend*(X, T_j). Similarly, the sum of *Iutil* (*Iutil*(X)) is the utility of X in the database, and the sum of *Rutil* (*Rrutil*(X)) reveals the maximum utility value that matched super-itemsets of X can reach.

Strategy 5. Considering a matched itemset X, if the summation of *Iutil*(X) and *Rrutil*(X) is less than the user-specified minimal threshold γ, and thus X and its super-itemsets are not THUIs respecting the *maxLength* constraint.

As previous content was introduced, the utility list of a new itemset is constructed by two different utility lists with a common prefix itemset. Let *RUL*(*Px*) and *RUL*(*Py*) be two different revised utility lists, their super-itemset *Pxy* is occurring in transactions such that *Px* and *Py* appear together. Actually, the utility upper-bound (*Iutil*(X) + *Rrutil*(X)) is still loose [39].

Strategy 6. Let *Px* and *Py* are distinct itemset, and $x \prec y$, if $\sum_{T_j \in RUL(Px)}(Iutil(Px, T_j) + Rrutil(Px, T_j))$ - $\sum_{T_j \in RUL(Px) \wedge Py \notin T_j}(Iutil(Px, T_j) + Rrutil(Px, T_j)) < \gamma$ holds, all extensions of *Pxy* cannot be THUIs. There is no need to construct the revised utility list of *Pxy*.

Algorithm 1: The THL algorithm

```
Input: $\mathcal{D}$: a transaction database; $\mathcal{T}$: a set of targeted items; $\gamma$: a user-specified
       minimum utility threshold; maxLength: user expects the maximum number of
       items; minLength: user expects the minimum number of items.
Output: THUIs: a complete set of targeted high utility itemsets respecting the length
        constraints.
1  set minLength $\leftarrow |\mathcal{T}|$;
2  for each transaction $T_j \in \mathcal{D}$ do
3      remove transactions such that $T_j <$ minLength;
4      remove transactions such that $T_j$ mismatch $\mathcal{T}$;
5      compute RTWU of all 1-itemsets;
6  end
7  set $I' \leftarrow$ 1-itemsets $X$ such that $RTWU(X) \geq \gamma$;
8  Let $\prec$ be the global order of RTWU-ascending on $I'$ and $\mathcal{T}$;
9  set $\mathcal{D}' \leftarrow$ as modified database;
10 for each transaction $T_j \in \mathcal{D}'$ do
11     $RUL(X) \leftarrow$ build revised utility lists of all 1-itemsets $X \in I'$;
12     $EUCS \leftarrow$ build the EUCS structure;
13 end
14 call Miner($\emptyset$, $RUL(I')$, $\mathcal{T}$, $\gamma$, minLength, maxLength, EUCS);
15 return THUIs
```

4.4. **The THL algorithm.** We introduce the proposed THL algorithm in this subsection. The **Algorithm** 1 takes five parameters ($\mathcal{D}$, $\mathcal{T}$, γ, *minLength*, and *maxLength*) as input, and finally output a complete set of targeted high utility itemsets respecting the length constraints. In line 1, the algorithm first sets the length of targeted items as *minLength*. Then, during the first database scanning, the algorithm marks and removes some transactions that do not meet targeted item constraints (lines 3 and 4). The *RTWU* of all 1-itemsets are calculated in the meanwhile (line 5). Then, in line 7, those 1-itemsets cannot be high utility itemsets will be pruned in advance. According to the global order (i.e., *RTWU*-ascending order), the algorithm resorts 1-itemsets, targeted items, and transactions (lines 8 and 9). Mining processing of the algorithm is more efficient after sorting items in the database and targeted items. During the second database scanning (lines 10–13), the algorithm builds the revised utility lists of 1-itemsets ($X \in I'$)

and constructs *EUCS* structure. In line 14, the algorithm utilizes a depth-first approach (cf. **Algorithm** 2) to discover a complete set of targeted high utility itemsets respecting the length constraints.

Algorithm 2: The Miner procedure

```
Input: $P$: the common prefix itemset; $RUL(P)$: a set of extensions of $P$; $\mathcal{T}$: a set of
       targeted items; $\gamma$: a user-specified minimum utility threshold; $maxLength$: user
       expects the maximum number of items; $minLength$: user expects the minimum
       number of items; $EUCS$: the $EUCS$.
Output: $THUIs$: a complete set of targeted high utility itemsets respecting the length
        constraints.
for itemset $Px \in RUL(P)$ do
    set $currentIndex \leftarrow$ the position index of $x$ in $\mathcal{T}$, $originalOrder \leftarrow 0$, and
     $patternOrder \leftarrow 0$;
    if $currentIndex < minLength$ then
        $originalOrder \leftarrow$ the position index of $x$ in $I'$;
        $patternOrder \leftarrow$ the index $\mathcal{T}[currentIndex]$ in $I'$;
        if $originalOrder == patternOrder$ then
            $currentIndex \leftarrow currentIndex + 1$;
        end
    end
    if $currentIndex < minLength$ and $patternOrder < originalOrder$ then
        break;
    end
    if $Iutil(Px) \geq \gamma$ and $minLength \leq |Px| \leq maxLength$ then
        THUIs $\leftarrow Px$;
    end
    if $Iutil(Px) + Rrutil(Px) \geq \gamma$ then
        $RUL(Px) \leftarrow \emptyset$;
        for itemset $Py \in RUL(P)$ where $x \prec y$ do
            if $\exists(x, y, z) \in EUCS$ where $z \geq \gamma$ then
                $Pxy \leftarrow Px \cup Py$;
                if $Iutil(Pxy) + Rrutil(Pxy) \geq \gamma$ then
                    $RUL(Pxy) \leftarrow$ Construct($RUL(P), RUL(Px), RUL(Py)$);
                end
                $RUL(Px) \leftarrow RUL(Px) \cup RUL(Pxy)$;
            end
        end
        if $|Pxy| < maxLength$ then
            Miner($Px, RUL(Px), \mathcal{T}, \gamma, maxLength, minLength$);
        end
    end
end
```

The Miner procedure (cf. **Algorithm** 2) is a recursive function and is also the major process of the novel algorithm. The procedure takes seven parameters as input in the first iteration like a common prefix itemset P, a set of extensions of P $RUL(P)$, a set of targeted items $\mathcal{T}$, a user-specified minimum utility threshold γ, two user-defined length thresholds *minLength* and *maxLength*, and an estimated utility co-occurrence structure *EUCS*. Especially, the algorithm calls the Miner procedure at the first round, the common prefix itemset is null due to the processing objects being 1-itemsets. *RUL*(*P*) is a set of revised utility lists of 1-itemsets. For each itemset *Px* of the revised utility list *P*, the procedure initializes a series of position indexes of items. Then it checks item matching during each recursive execution (lines 3–9). The procedure gets the index of matching items (i.e., *originalOrder* and *patternOrder*). Then, if a current item x matches an element of $\mathcal{T}$ (line 6), the procedure will continue to check the next item in *Px* matches elements of $\mathcal{T}$ or not (line 7). Next, if *patternOrder* is less than *originalOrder*, the traversed itemset mismatch with $\mathcal{T}$, and the current recursion will terminate (lines 10–12). Then, if the summation of *Iutil* values of *RUL*(*Px*) is higher than or equal to γ, and *Px* respects the length constraints, *Px* is a THUI (lines 13–15).

After that, the procedure adopts several efficient utility-pruning strategies to improve the performance of the proposed algorithm. The procedure calculates the maximum utility value the current itemset *Px* can reach. If the maximum utility value is no less than γ, the procedure will take the next mining step (line 16). Otherwise, *Px* is not a THUI, and thus the procedure will check another itemset in *RUL*(*P*). In line 17, the procedure initializes an empty revised utility list of *Px*. Then, taking another extension itemset *Py* of *P* such that $x \prec y$ (line 18). If *RTWU*(*xy*) is less than γ, *xy* and its super-itemsets cannot be THUIs (line 19). Then, in line 21, the procedure checks the maximum utility value of super-itemset of *Pxy* whether greater than or equal to γ. If the condition is true, the procedure calls the Construct function (cf. **Algorithm** 3) to build the revised utility list of *Pxy* (line 22). Furthermore, if |*Pxy*| is less than *maxLength*, a recursive call to the Miner procedure with *Pxy* is done to compute its utility and explore its extensions (lines 27–29).

Algorithm 3: The Construct procedure

```
Input: RUL(P): the revised utility list of itemset P; RUL(Px): the revised utility list of
       itemset Px; RUL(Py): the revised utility list of itemset Py
Output: RUL(Pxy): the revised utility list of itemset Pxy
1  set RUL(Pxy) ← $\emptyset$;
2  for tuple $Ex \in RUL(Px)$ do
3      if $\exists Ey \in RUL(Py)$ and $Ex.Tid == Ey.Tid$ then
4          if RUL(P) is not empty then
5              use binary search to find tuple $E \in RUL(P)$ such that $E.Tid == Ey.Tid$;
6              $Exy = <Ex.Tid, Iutil(Ex) + Iutil(Ey) - Iutil(E), Rrutil(Ey)>$;
7          else
8              $Exy = <Ex.Tid, Iutil(Ex) + Iutil(Ey), Rrutil(Ey)>$;
9          end
10         add Exy to RUL(Pxy);
11     end
12 end
13 return RUL(Pxy)
```

The Construct procedure (cf. **Algorithm** 3) operates as follows. The novel algorithm adopts an insertion method to reduce the database scan times. The input parameters of the Construct procedure are revised utility lists of three distinct itemsets, i.e., *P*, *Px*, and *Py*, where *P* is a prefix of *Px* and *Py*. In line 1, the procedure first builds an empty revised utility list of the super-itemset *Pxy*. Then, the procedure traverses each tuple *Ex* of the revised utility list of *Px* (line 2). If the revised utility list of *Py* exists a tuple *Ey* has the same *Tid* with that of *Ex*, the procedure then takes the following steps to construct a new tuple of *RUL*(*Pxy*). If itemset *P* is not null, the *Iutil* of the new tuple *Exy* has to reduce the utility value of prefix itemset because of double counting (line 6); otherwise, the *Iutil* of *Exy* is the sum *Iutil* values of *Ex* and *Ey* (line 8). Besides, the *Rrutil* of *Exy* is the same as that of *Ey*. However, there is a key step that should be pointed out. The novel algorithm initiates with 1-itemsets and subsequently expands the search space through a recursive process, incrementally adding one item at a time to the itemsets. If the sum of the length of *Pxy* and its extension (*maxExtend*(*Exy*)) is greater than *maxLength*, the newly added item should be removed. Finally, the Construct procedure outputs a newly revised utility list of the super-itemset *Pxy*.

Table 3. The experimental datasets

Dataset	**#Trans**	**#Items**	**#Avg**	**Type**
BMSPOS	59,601	497	4.8	sparse, short transactions
Foodmart	4,141	1,559	4.4	sparse, short transactions
Accidents	340,183	468	33.8	dense, long transactions
Kosarak	990,000	41,720	8.1	sparse, short transactions
Mushroom	8,124	119	23	dense, long transactions

Table 4. The candidate generation of algorithms on Kosarak

γ	260M	280M	300M	320M	340M	360M	380M	400M
THUIM	1,338,461	688,514	406,774	237,142	137,778	47,539	15,737	9,567
TIRUP	1,208	955	724	527	324	162	100	79
THL_3	57	50	44	41	32	30	27	26
THL_5	312	240	198	179	150	135	118	98
THL_{full}	669	539	429	323	257	219	198	177

5. **Experimental Study.** We conduct several experiments on five significant transaction datasets to assess the performance of THL. All experiments were performed on a computer with 64-bit Core i5 processor 2.5 GHz running Windows 10 and 32 GB RAM. Considering there is no targeted high utility itemset mining algorithm respecting length constraints yet, we choose two state-of-the-art algorithms (THUIM [40] and TIRUP [27]) as experimental benchmarks. In particular, the TIRUP algorithm also adopts the utility list-based structure to discover a complete set of targeted high utility itemsets respecting support constraints. Therefore, we removed support constraints and modified TIRUP mines THUIs in the same as THUIM does. This paper uses symbols "K" and "M" to replace thousands and millions. Besides, this paper assumes the minimum length as the number of targeted items. We thus use a numeric subscript of the novel algorithm to represent the maximum length setting. For instance, THL_{10} means *maxLength* is 10 on experiment. Furthermore, to test the performance of the new algorithm without length constraints, we assume the THL_{full} algorithm discovers the number of THUIs as much as that of THUIM and TIRUP. All experimental algorithms were implemented in Java language, and runtime and memory measurements were done using the Java API. If an experiment executes over 36,000 seconds, we suppose the algorithm is terminated and cannot discover the right results.

5.1. **Experimental datasets.** The experimental datasets can be freely downloaded from the SPMF[1] open-source data mining library. As shown in Table 3, **#Trans** shows how many transactions the dataset has, **#Items** reveals the number of distinct items, **#Avg** represents the average length of a transaction of the dataset, and **Type** describes some features of datasets. The

[1]https://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php

BMSPOS dataset contains clickstream data from an e-commerce. The Foodmart dataset of customer transactions from a retail store was obtained and transformed from SQL-Server 2000. The Accidents dataset is anonymized traffic accident data. Kosarak consists of transactions from click-stream data from a Hungarian news portal. The Mushroom dataset includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms. Four datasets (BMSPOS, Foodmart, Accidents, and Kosarak) were used to evaluate the execution efficiency of tested algorithms, and Mushroom was used to experiment scalability of the THL algorithm.

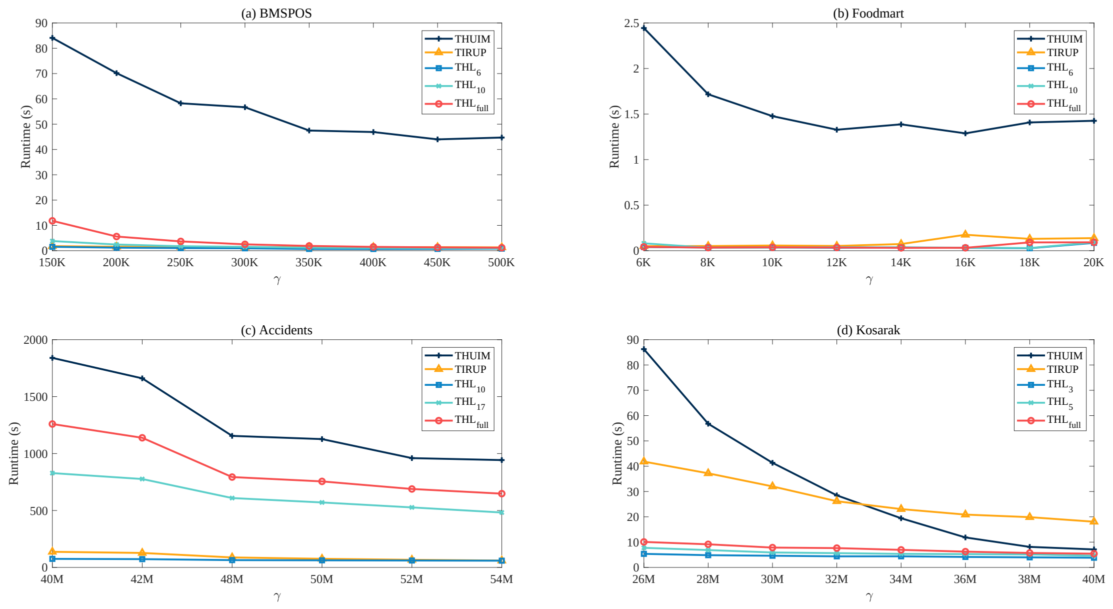

Figure 4. The runtime consumption of experimental algorithms on four datasets.

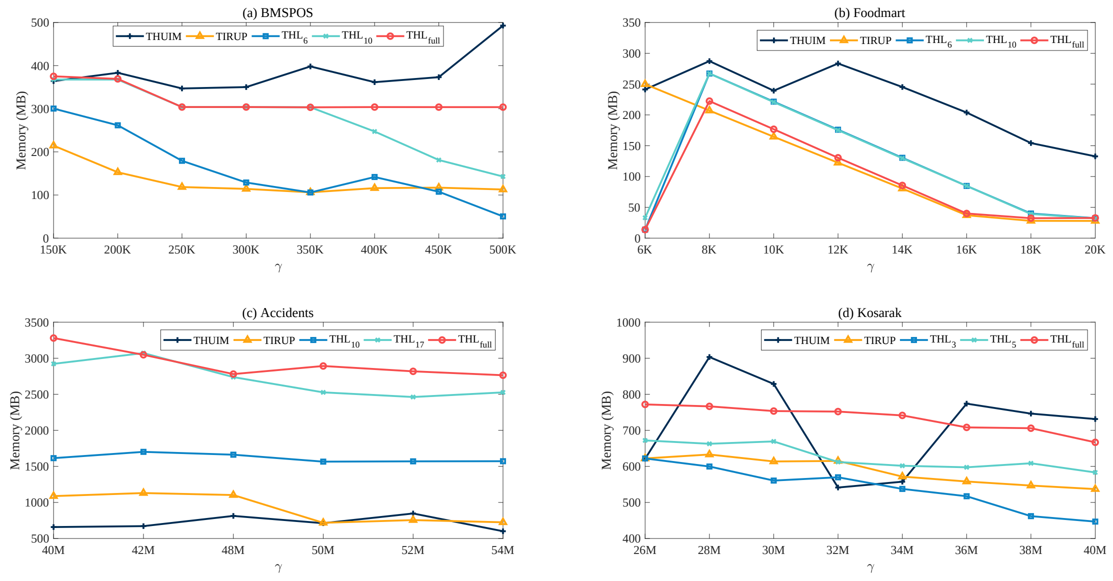

Figure 5. The memory usage of tested algorithms on four datasets.

5.2. **Efficiency analysis.** It can be first observed that *maxLength* can effectively speed up the mining process in Fig. 4. The runtime consumption continuously declines while the minimal utility threshold γ increases. The THUIM algorithm performs the worst on most experimental datasets. On the BMSPOS dataset, the runtime consumption of THUIM is up to 40 times greater than that of TIRUP and THL when γ = 500K. On the Kosarak dataset, the runtime consumption of Kosarak is up to 2-4 times greater than that of THL when γ = 500K. Compared to other variant THL algorithms, the THL$_{full}$ algorithm still performs well except on the Accidents dataset. The runtime consumption of tested THL algorithms gradually decreases while *maxLength* reduces. With the same γ setting as displayed in Fig. 4, the memory usage of tested algorithms is shown in Fig. 5. It can be noticed that the memory consumption of THUIM is also much greater in most cases. For example, considering γ = 500K on the BMSPOS dataset, the memory usage of THUIM is up to 4 and 10 times greater than that of TIRUP and THL$_6$, respectively. The memory consumption of TIRUP and variant THL algorithms are similar on Foodmart in most cases. Then, the memory cost of THL and TIRUP steadily decreases as the minimal utility threshold increases on Kosarak.

5.3. **Candidate generations comparison.** Table 4 indicates the number of candidate generations of experimental algorithms. The number of candidates continuously decreases as the minimal utility threshold increases. Due to the poor performance of THUIM, the account of candidates of THUIM is far greater than that of other algorithms. For instance, when γ is 260M, the quantity of generated candidates of THUIM is up to 3 and 5 orders of magnitude higher than that of TIRUP and THL respectively. While γ becomes smaller, the gap of generated candidates between THL (e.g., THL$_5$) and TIRUP becomes larger. Compared to variants of THL, the value of *maxLength* has a great impact on the number of candidate generations. For example, when γ is equal to 400M, the candidate quantity of THL$_{full}$ is nearly twice that of THL$_5$, but the accounts of THUIs they output are equal (Fig. 6). We also present the THUIs discovered of tested algorithms on four datasets in Fig. 6. The generated quantities of THUIM, TIRUP, and THL$_{full}$ are always equal. Then, it is clear that the less *maxLength* is, the fewer final results discovered. On Accidents dataset, the THUIs of THL$_{10}$ are less than 300 w.r.t various utility thresholds, and thus it is hard to be seen in the figure.

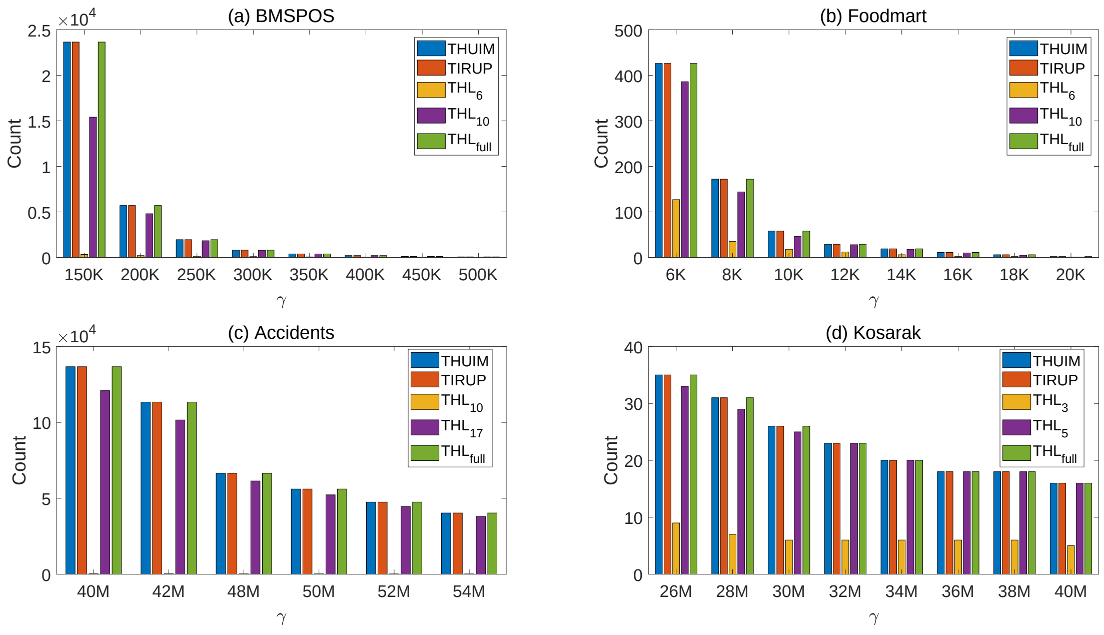

FIGURE 6. The THUI generations of algorithms on four datasets.

5.4. **Scalability test.** This paper takes Mushroom as an experimental dataset to evaluate the scalability of THL w.r.t length constraints. In Fig. 7, we take 1000, 3000, 5000, 7000, and 8000 transactions from Mushroom. The targeted items are $\{34, 37, 38\}$ and γ = 24K. Scalability experiments are carried out to evaluate four aspects: runtime and memory consumption, candidate, and THUI generations. The runtime usage increases linearly as the data size increases (as shown in Fig. 7(a)). The value of *maxLength* certainly influences the performance of THL. For instance, the memory consumption of THL_9 is around 5 times less than that of THL_{full} while the data size is 8K. In summary, THL is quite an efficient algorithm, which can quickly and effectively discover a complete set of targeted high utility itemsets respecting length constraints.

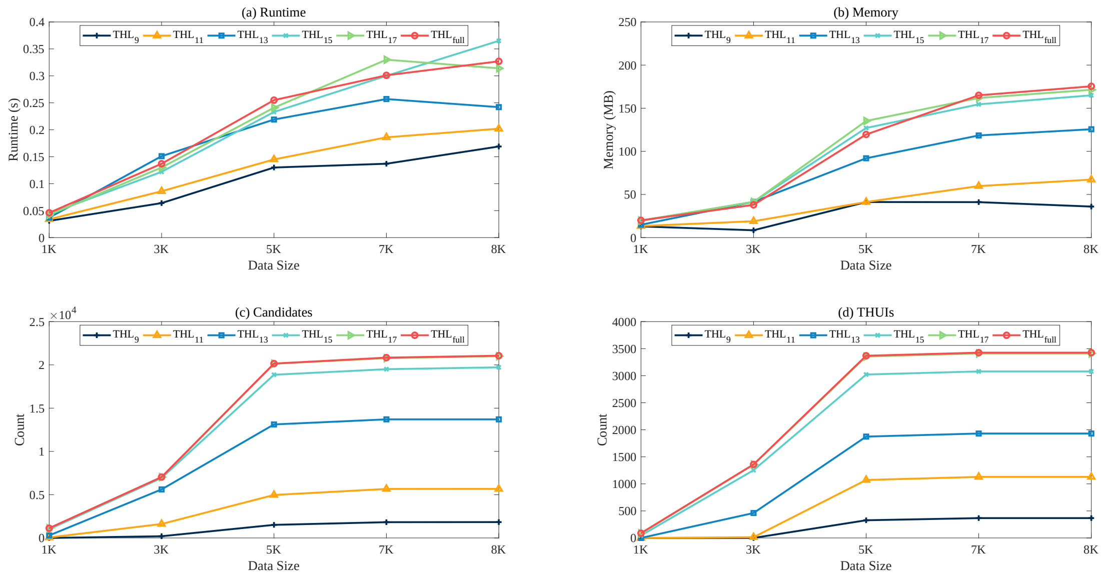

FIGURE 7. The scalability test of the THL algorithm w.r.t different *maxLength*s.

6. Conclusion.

In this paper, we notice current works of literature ignore the length factor in the targeted itemset mining task, and then propose a new utility list-based targeted itemset mining algorithm respecting length constraints. Especially, we also consider users are usually clear about how many items they need at least but often have trouble in at most. Therefore, we assume the minimal length is the number of user-specified targeted items, which reduces the operation complexity of the algorithm to a certain degree. In the future, we try to optimize the matching mechanism and applied in other domains, such as user sequences behavior analysis, privacy protection, targeted recommendation system, and intelligent search.

Acknowledgment. This research was supported by the Guangdong Provincial Key Laboratory of Power System Network Security (No.GPKLPSNS-2022-KF-04).

REFERENCES

[1] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *Proceedings of the 20th International Conference Very Large Data Bases*, vol. 1215. Santiago, 1994, pp. 487–499.
[2] P. Fournier-Viger, J. C.-W. Lin, B. Vo, T. T. Chi, J. Zhang, and H. B. Le, "A survey of itemset mining," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 7, no. 4, p. e1207, 2017.
[3] Y. S. Koh and S. D. Ravana, "Unsupervised rare pattern mining: a survey," *ACM Transactions on Knowledge Discovery from Data*, vol. 10, no. 4, pp. 1–29, 2016.
[4] G. Grahne and J. Zhu, "Fast algorithms for frequent itemset mining using fp-trees," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 10, pp. 1347–1362, 2005.
[5] J. Wang, J. Han, and C. Li, "Frequent closed sequence mining without candidate maintenance," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 8, pp. 1042–1056, 2007.
[6] M. Adda, L. Wu, and Y. Feng, "Rare itemset mining," in *Proceedings of the 6th International Conference on Machine Learning and Applications*. IEEE, 2007, pp. 73–80.

[7] W. Gan, J. C.-W. Lin, P. Fournier-Viger, H.-C. Chao, V. S. Tseng, and P. S. Yu, "A survey of utility-oriented pattern mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 4, pp. 1306–1327, 2019.

[8] W. Gan, J. C.-W. Lin, P. Fournier-Viger, H.-C. Chao, and H. Fujita, "Extracting non-redundant correlated purchase behaviors by utility measure," *Knowledge-Based Systems*, vol. 143, pp. 30–41, 2018.

[9] J. Miao, S. Wan, W. Gan, J. Sun, and J. Chen, "Targeted high-utility itemset querying," *IEEE Transactions on Artificial Intelligence*, vol. 4, no. 4, pp. 871–883, 2022.

[10] P. Zhang, J. Chen, S. Wan, and W. Gan, "Targeted mining of rare high-utility patterns," in *IEEE International Conference on Big Data*. IEEE, 2022, pp. 6271–6280.

[11] Y.-D. Shen, Z. Zhang, and Q. Yang, "Objective-oriented utility-based association mining," in *Proceedings of the IEEE International Conference on Data Mining*. IEEE, 2002, pp. 426–433.

[12] H. Yao, H. J. Hamilton, and C. J. Butz, "A foundational approach to mining itemset utilities from databases," in *Proceedings of the SIAM International Conference on Data Mining*. SIAM, 2004, pp. 482–486.

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

[14] Y. Liu, W.-k. Liao, and A. Choudhary, "A fast high utility itemsets mining algorithm," in *Proceedings of the 1st international workshop on Utility-based data mining*, 2005, pp. 90–99.

[15] M. Liu and J. Qu, "Mining high utility itemsets without candidate generation," in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, 2012, pp. 55–64.

[16] P. Fournier-Viger, C.-W. Wu, S. Zida, and V. S. Tseng, "FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning," in *Foundations of Intelligent Systems: 21st International Symposium*. Springer, 2014, pp. 83–92.

[17] P. Fournier-Viger, J. C.-W. Lin, Q.-H. Duong, and T.-L. Dam, "FHM+: Faster high-utility itemset mining using length upper-bound reduction," in *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer, 2016, pp. 115–127.

[18] S. Wan, W. Gan, X. Guo, J. Chen, and U. Yun, "FUIM: Fuzzy utility itemset mining," *arXiv preprint arXiv:2111.00307*, 2021.

[19] P. Wu, X. Niu, P. Fournier-Viger, C. Huang, and B. Wang, "UBP-Miner: An efficient bit based high utility itemset mining algorithm," *Knowledge-Based Systems*, p. 108865, 2022.

[20] S. Wan, J. Chen, Z. Qi, W. Gan, and L. Tang, "Fast RFM model for customer segmentation," in *Companion Proceedings of the Web Conference*, 2022, pp. 965–972.

[21] S. Wan, Z. Ye, W. Gan, and J. Chen, "Temporal fuzzy utility maximization with remaining measure," *arXiv preprint arXiv:2208.12439*, 2022.

[22] Z. Cheng, W. Fang, W. Shen, J. C.-W. Lin, and B. Yuan, "An efficient utility-list based high-utility itemset mining algorithm," *Applied Intelligence*, vol. 53, no. 6, pp. 6992–7006, 2023.

[23] S. Zida, P. Fournier-Viger, J. C.-W. Lin, C.-W. Wu, and V. S. Tseng, "EFIM: a fast and memory efficient algorithm for high-utility itemset mining," *Knowledge and Information Systems*, vol. 51, no. 2, pp. 595–625, 2017.

[24] J. M.-T. Wu, J. C.-W. Lin, and A. Tamrakar, "High-utility itemset mining with effective pruning strategies," *ACM Transactions on Knowledge Discovery from Data*, vol. 13, no. 6, pp. 1–22, 2019.

[25] S. Wan, J. Chen, P. Zhang, W. Gan, and T. Gu, "Discovering top-*k* profitable patterns for smart manufacturing," in *Companion Proceedings of the Web Conference*, 2022, pp. 956–964.

[26] S. Wan, J. Deng, W. Gan, J. Chen, and P. S. Yu, "Fast mining RFM patterns for behavioral analytics," in *IEEE 9th International Conference on Data Science and Advanced Analytics*. IEEE, 2022, pp. 1–10.

[27] Y. Xu, C. Peng, J. Chen, W. Gan, and S. Wan, "Mining rare utility patterns within target items," in *Proceedings of the IEEE International Conference on Big Data*. IEEE, 2023, pp. 6015–6024.

[28] G. Huang, W. Gan, and P. S. Yu, "TaSPM: Targeted sequential pattern mining," *ACM Transactions on Knowledge Discovery from Data*, vol. 18, no. 5, pp. 1–18, 2024.

[29] M. Kubat, A. Hafez, V. V. Raghavan, J. R. Lekkala, and W. K. Chen, "Itemset trees for targeted association querying," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 6, pp. 1522–1534, 2003.

[30] Y. Li and M. Kubat, "Searching for high-support itemsets in itemset trees," *Intelligent Data Analysis*, vol. 10, no. 2, pp. 105–120, 2006.

[31] J. Lavergne, R. Benton, and V. V. Raghavan, "Min-max itemset trees for dense and categorical datasets," in *Foundations of Intelligent Systems: 20th International Symposium*. Springer, 2012, pp. 51–60.

[32] J. Lewis, R. G. Benton, D. Bourrie, and J. Lavergne, "Enhancing itemset tree rules and performance," in *Proceedings of the 7th IEEE International Conference on Big Data*. IEEE, 2019, pp. 1143–1150.

[33] L. Shabtay, P. Fournier-Viger, R. Yaari, and I. Dattner, "A guided FP-Growth algorithm for mining multitude-targeted item-sets and class association rules in imbalanced data," *Information Sciences*, vol. 553, pp. 353–375, 2021.

[34] G. Arumugam and V. Vijayakumar, "Design of query-driven system for time-utility based data mining on medical data," in *Proceedings of the 10th Knowledge Management in Organizations: 10th International Conference*. Springer, 2015, pp. 664–682.

[35] C. Zhang, Q. Dai, Z. Du, W. Gan, J. Weng, and S. Yu, Philip, "TUSQ: Targeted high-utility sequence querying," *IEEE Transactions on Big Data*, vol. 9, no. 2, pp. 512–527, 2022.

[36] W. Gan, G. Huang, J. Weng, T. Gu, and P. S. Yu, "Towards target sequential rules," *arXiv preprint arXiv:2206.04728*, 2022.

[37] K. Hu, W. Gan, S. Huang, H. Peng, and P. Fournier-Viger, "Targeted mining of contiguous sequential patterns," *Information Sciences*, vol. 653, p. 119791, 2024.

[38] J. Zhu, X. Chen, W. Gan, Z. Chen, and S. Yu, Philip, "Targeted mining precise-positioning episode rules," *IEEE Transactions on Emerging Topics in Computational Intelligence*, pp. 1–14, 2024.

[39] S. Krishnamoorthy, "Pruning strategies for mining high utility itemsets," *Expert Systems with Applications*, vol. 42, no. 5, pp. 2371–2381, 2015.

[40] J. Miao, S. Wan, W. Gan, J. Sun, and J. Chen, "Targeted high-utility itemset querying," *IEEE Transactions on Artificial Intelligence*, vol. 4, no. 4, pp. 871–883, 2022.