Applied Intelligence (2023) 53:29363–29377
https://doi.org/10.1007/s10489-023-04791-2

Fast privacy-preserving utility mining algorithm based on utility-list dictionary

Chunyong Yin[1] · **Ying Li**[1]

Accepted: 12 June 2023 / Published online: 27 October 2023

Abstract
Privacy preserving utility mining (PPUM) aims to solve the problem of sensitive information leakage in utility pattern mining. In recent years, researchers have proposed algorithms to solve the privacy-preserving problem. However, these algorithms have high side effects, long sanitization time, and computational complexity. Although the FPUTT algorithm reduces the number of database scans, tree construction and traversal still take much time. The paper proposes a fast utility-list dictionary algorithm (FULD). The utility-list dictionary consists of all sensitive items. Through dictionary lookup, sensitive items can be found and modified. In addition, the novel concepts of SINS and tns are proposed to reduce the side effects of the algorithm. In this paper, the experiments show that the FULD algorithm has good performance, such as running time and side effects. The running time of the FULD is 15–20 times shorter than the FPUTT algorithm. It performs well both on sparse and dense datasets.

Keywords Sensitive information · Utility mining · Privacy preserving · Utility-list dictionary · Utility lists

1 Introduction

Data mining [1] aims to extract useful information from structured or unstructured data and structure it for people to use. In recent years, data mining [2–5] can meet practical applications in various fields [1, 6–8]. Although these techniques are popular, these algorithms ignore the utility of itemsets, such as importance, interest, satisfaction, or risk. Generally speaking, the implicit information of utility is common in life. Traditional data mining algorithms often may not be able to dig out information that is truly important to users. Therefore, the High Utility Pattern Mining (HUPM) [9] algorithm emerged, which can set the utility of an item according to its importance.

The utility is a subjective measure that reflects the human desire to some extent. In Wikipedia, utility refers to a user's level of satisfaction with a good or service. In [10], the authors summarized several measures. They are objective measures, subjective measures, and semantics-based measures [10–12]. Objective measures [12, 13] are measured according to statistics and have strict standards, such as itemset utility in utility mining. Subjective and semantic-based measures [9, 14] are determined based on the user's background knowledge. They are suitable for experienced users. Due to the different background knowledge of users, the semantic-based methods are varied. In utility mining, people use utility to measure the importance of an item.

High-utility pattern mining (HUPM) can mine important information. But it has a flaw in privacy preserving. Therefore, scholars have proposed Privacy Preserving Utility Mining (PPUM). Research in this area is still in its infancy. The methods of privacy preserving are differential privacy, encryption technology, and blockchain. Differential privacy [15] can ensure a balance between privacy and efficiency. Encryption technology [16] encodes original data into ciphertext to ensure data privacy in storage and transmission. The blockchain [17] can monitor abnormal behaviors of devices or servers. But its throughput is limited and scalability is poor as well. However, these techniques are not applicable in PPUM, which would result in a severe loss of utility.

✉ Chunyong Yin
yinchunyong@hotmail.com

Ying Li
lucindaxyy@hotmail.com

[1] School of Computer and Software, Nanjing University of Information Science & Technology, Ningliu Road, Nanjing 210044, State, China

To protect sensitive information in utility mining, researchers adopt methods of reducing internal utility, transaction insertion, or deletion. Yeh et al. [18] first proposed a privacy-preserving strategy in this field. The algorithm removes some sensitive itemsets so that the cleaned database no longer contains sensitive information. Gan et al. [19] proposed MSU-MAU and MSU-MIU, which adopted a projection mechanism to speed up the sanitization process of the original database. However, these algorithms take a long time to run on large datasets. In addition to the above methods, Yun et al. [20] constructed a tree structure to store item information, reducing the number of database scans. Nevertheless, the algorithm has large side effects and high computational complexity. Li et al. [21] proposed an exact algorithm based on integer linear programming. Although this algorithm can obtain an exact solution, it has high computational complexity and takes a long time. Therefore, to solve these problems, this paper proposes the FULD algorithm. The main contributions of this paper are summarized as follows:

- In past studies on privacy preserving utility mining, researchers used tree structure, heuristic algorithm, etc. to speed up the sanitization process. For the first time, this paper proposes a novel utility-list dictionary structure to solve these problems. The UTLDic proposed in this paper consists of sensitive items and UTLists. This structure is completely different from the utility-list structure in high-utility pattern mining. It can reduce the number of utility calculations and search for sensitive information without repeatedly scanning the database.
- To reduce the side effects of the algorithm, this paper proposes the concepts of SINS and tns. SINS reflects the number of occurrences of sensitive items in non-sensitive high-utility itemsets. Tns represents the relationship between the transaction and non-sensitive high-utility itemsets. Through these new concepts, the performance of the algorithm can be further optimized.
- It can be seen from the experiments that the performance of the FULD is affected by many factors, including the number of sensitive itemsets, the density of the dataset, the minimum utility threshold, the size of the dataset, etc.
- The experiments are performed on multiple datasets and compared with other algorithms. The experiments show that the FULD algorithm performs well on sparse and dense datasets. In addition, the FULD algorithm is 15–20 times shorter than the FPUTT algorithm.

The rest of the paper is organized as follows. In Section 2, we review related work on HUPM and PPUM. The details of PPUM background knowledge are described in Section 3. In Section 4, we introduce the FULD algorithm. In Section 5, we conducted extensive comparative experiments on four datasets. In Section 6, we analyze the complexity of the FULD algorithm. Finally, we conclude this paper and future research directions in Section 7.

2 Related work

High-utility pattern mining can obtain important information from massive data. Privacy preserving utility mining addresses privacy issues in utility mining. To better understand, the related work is presented in this section.

2.1 High-utility pattern mining

With the increasing number of IoT devices, it is hard to mine information in a short time. Therefore, Lin et al. [22] proposed the MCUI-Miner algorithm. The algorithm combines the GA and the MapReduce model to speed up the search process. Gan [23] et al. summarized the utility pattern mining algorithms. The advantages and disadvantages of the algorithm are analyzed by classifying. The author has reviewed some open-source software and datasets. Lin et al. [24] proposed the DFM-Miner model. The model can mine closed high-utility itemset in parallel and distributed environments. The model has better performance in dealing with large databases with less memory usage. Lin et al. [25] used a clustering model to group transactions with high correlation. The closed high-utility itemsets can be obtained by a compact GA model in a short time. The TGA algorithm has a good performance in terms of accuracy and the number of recognition modules.

To mine information from the database, many utility pattern mining algorithms are proposed by scholars. In [26], the EHMIN algorithm can consider not only the positive unit profits in the project but also the negative unit profits. The algorithm mainly focused on using the negative unit profits of the project to mine useful patterns. At the same time, the algorithm performed well in all indicators. Lee et al. [27] improved the traditional erasable pattern mining by introducing the list structure, adding the quantity and price attributes of items to estimate the profit, and extracting the maximized erasable pattern according to the user's preference. Ryu et al. [28] proposed the mining method HUOMI, which can be used to process dynamically generated data flows, and at the same time efficiently and quickly analyze massive data on an ever-increasing database. Lee et al. [27] proposed an efficient algorithm for mining high-average utility patterns based on sliding windows with the list structure, which was used to mine the latest valuable patterns in data streams.

To store frequent itemsets, the FP-Growth [5] algorithm is proposed. It compresses the information in the database and reduces the number of database scans. Inspired by this algorithm, Hu et al. [29] proposed a frequent itemset mining algorithm HYP to identify high-utility combinations. The

algorithm reduces data mining to an optimization problem and obtains approximate solutions through High-Yield Partition Trees. Similar to the FP-tree structure, Lin et al. [30] presented the HUP-growth algorithm. HUP tree can not only store mining information but also reduce database scanning time. But different from the traditional FP-tree, each node was attached with an array to keep the quantities of its prefix items in the path.

To obtain higher efficiency than tree-based utility pattern mining, new data structures are proposed to mine itemsets, such as HUP-Miner [31], EFIM [32] and d2HUP [33]. Kim et al. [34] proposed an improvement based on the high-efficiency algorithm of the sliding window method. It mined high utility patterns by continuously inserting damped flow data into the latest data. Baek et al. [35] proposed the Approximate High Utility Pattern Mining (AHUPM) algorithm. The algorithm defined the utility tolerance factor to calculate the ranges of trustworthy utilities for patterns, and thus extracted the high-utility patterns defined as the approximate high utility patterns from the database.

2.2 Privacy preserving utility mining

The target of HUPM is to mine high-utility itemsets (HUIs) and discover interesting information. But utility pattern mining can reveal sensitive information. Accordingly, HUPM also has the problem of sensitive information leakage. People often need to hide these sensitive high-utility itemsets (SHUIs) in real scenarios. Therefore, PPUM has become the focus of scholars' research in recent years.

The utility of an item is equal to the sum of its internal utility multiplied by its external utility. Generally speaking, the methods for reducing the utility of an item can be summarized as follows: 1) reduce the internal utility of the item directly, 2) change the external utility of the item. However, external utility is generally fixed and will not change in the real world.

Therefore, it is more reasonable to reduce the internal utility of the item. Yeh et al. [18] proposed HHUIF and MSICF algorithms, which not only hide sensitive information but also minimize the side effects of the algorithms. Yun et al. [20] introduced a new tree structure to expedite the perturbation process and reduce the search space of PPUM. Lin et al. [19] proposed MSU-MAU and MSU-MIU algorithms to minimize side effects caused by hiding sensitive itemsets. In [36], SIF-IDF is a greedy algorithm capable of removing sensitive information in transactions. The algorithm drew on the concept of term frequency and inverse document frequency (TF-IDF) in text mining. It can hide the appropriate items after calculating the similarity between the items in the transaction and the desired sensitive itemsets.

In addition to the above, Li et al. [21] proposed a new algorithm based on integer linear programming (ILP) to reduce side effects generated in the sanitization process, which described the hiding of sensitive itemsets process as a problem that satisfies constraints. Jangra et al. [37] proposed two heuristic-based algorithms to hide SHUIs. The algorithm designs two methods for sensitive itemsets and improves data utility by minimizing missing costs. In [38], the Privacy-Preserving and Security Mining Framework (PPSF) is proposed. The framework offers some PPUM algorithms, such as HHUIF, MSICF, etc. The users can execute the privacy-preserving algorithm through the operation interface. Wu et al. [39] used the dynamic minimum threshold technique to hide sensitive information. In addition, the algorithm combined with GA can minimize side effects. Compared with traditional algorithms, the algorithm can hide more sensitive information. To further reduce side effects, Lin et al. [40] proposed an ant colony optimization approach. It hid sensitive information through transaction deletion. Each ant in the population was denoted as a group of possible deletion transactions. Compared with the genetic algorithm, this approach has better performance.

However, it is an important task to hide HUIs and extend these algorithms to handle sequential data. In [41], the author first proposed the HHUSP and MSPCF algorithms to hide all sensitive itemsets. In addition, Huynh et al. [42] proposed HHUSP-A and HHUSP-D algorithms. These algorithms reduce algorithm execution time and loss cost by ascending or descending order, thereby improving the performance of HHUSP.

3 Background knowledge

To understand privacy preserving utility mining (PPUM), the preliminary knowledge and the problem of PPUM are introduced briefly in this section.

3.1 Definition

The database is denoted as $D=\{T_1, T_2, \cdots, T_n\}$, which consists of n transactions. A transaction $T_k(1 \le k \le n)$ is made up of a limited set of items. Let $I=\{i_1, i_2, \cdots, i_m\}$ be m different items and T_k is a subset of I. The r-itemset means that it includes r items, such as 2-itemset $I=\{A, C\}$.

Definition 1 The internal utility of $i_v(1 \le v \le m)$ in T_n, denoted as $q(i_v, T_n)$, means the quantity of i_v in T_n. For example in Table 1, $q(C, T_2)=2$.

Definition 2 The external utility of i_v, denoted as $p(i_v)$, represents importance, profit, number of page link clicks, etc. For example in Table 2, $p(C)=4$.

Table 1 A Transaction Database

TID	AA	BB	CC	DD	EE	tu
T_1	0	0	7	1	1	41
T_2	1	0	2	0	2	31
T_3	0	6	4	3	7	149
T_4	0	5	3	9	0	121
T_5	3	0	10	3	0	85
T_6	0	0	5	0	9	83
T_7	6	0	9	2	5	137
T_8	1	6	2	5	3	134

Definition 3 The utility of i_v in T_n, denoted as $u(i_v, T_n)$, is as follows:

$$u(i_v, T_n) = q(i_v, T_n) \times p(i_v) \tag{1}$$

For instance in Table 1, $u(C, T_2) = q(C, T_2) \times p(C) = 2 \times 4 = 8$.

Definition 4 The utility of itemset X in $T_n (X \subseteq T_n)$, denoted as $u(X, T_n)$, is represented as

$$u(X, T_n) = \sum_{i_v \in X} u(i_v, T_n) \tag{2}$$

For example in Table 1, $u(AC, T_2) = u(A, T_2) + u(C, T_2) = 1 \times 9 + 2 \times 4 = 17$.

Definition 5 The utility of itemset X in database D, denoted as $u(X)$, is shown below:

$$u(X) = \sum_{i_v \in X, X \subseteq T_n} u(X, T_n) \tag{3}$$

For instance in Table 1, $u(AC) = u(AC, T_2) + u(AC, T_5) + u(AC, T_7) + u(AC, T_8) = 1 \times 9 + 2 \times 4 + 3 \times 9 + 10 \times 4 + 6 \times 9 + 9 \times 4 + 1 \times 9 + 2 \times 4 = 191$.

Definition 6 The utility of T_n in database D, denoted as $tu(T_n)$, is represented as follows:

$$tu(T_n) = \sum_{i_v T_n} u(i_v, T_n) \tag{4}$$

Table 2 An External Utility Table

Item	External Utility
AA	9
BB	11
CC	4
DD	6
EE	7

For example in Table 1, $tu(T_2) = 1 \times 9 + 2 \times 4 + 2 \times 7 = 31$.

Definition 7 The total utility of database D, denoted as TU, is as follows:

$$TU = \sum_{T_n \in D} tu(T_n) \tag{5}$$

For example in Table 1, $TU = 41 + 31 + 149 + 121 + 85 + 83 + 137 + 134 = 781$.

Definition 8 The importance of itemsets is assessed using minimum utility threshold. The value of minimum utility threshold, denoted as $\delta = c (0 \le c \le TU)$, is a constant specified by an expert.

Definition 9 The itemset X is considered as a high-utility itemset if and only if the utility of itemset X is not less than δ, denoted as $HUI = \{X \subseteq I, u(X) \ge \delta\}$.

3.2 Problem statement

Given a database D, it contains high-utility itemsets, denoted as HUIs. Let $S = \{S_1, S_2, \ldots, S_n\}$ be a set of sensitive high-utility itemsets and $NS (= HUIs - S)$ be a set of non-sensitive high-utility itemsets. The purpose of PPUM is to hide all sensitive high-utility itemsets S. The main way to solve this problem is to reduce the utility value of $S_k (1 \le k \le n)$ and make the non-sensitive high-utility itemsets (NS) as discoverable as possible. The database after sanitization is denoted as D'. $HUIs$ is mined from D', denoted as H'. Generally, the process of PPUM algorithm is as follows: 1) All HUIs are found by utility pattern mining algorithms; 2) Identify all sensitive high-utility itemsets; 3) All sensitive high-utility itemsets are hidden by using privacy preserving algorithm and side effects of the algorithm are minimized. The performance of PPUM is measured as follows.

Definition 10 Hiding failure (HF) represents sensitive high-utility itemsets S that still are found in D' after sanitization. HF is calculated by using Eq. 6.

$$HF = \frac{|S \cap H'|}{|H|} \tag{6}$$

Definition 11 Missing cost *(MC)* represents missing non-sensitive high-utility itemsets, which is defined as:

$$MC = \frac{|NS - H'|}{|NS|} \tag{7}$$

Definition 12 Artificial cost (*AC*) represents that the itemsets becomes HUIs after sanitization, which are not HUIs before sanitization. *AC* is defined as:

$$AC = \frac{|H' - H|}{|H|} \tag{8}$$

Definition 13 Hiding Cost *(HidingCost):* Inspired by evolutionary algorithms [43], the paper proposes a novel concept of *HidingCost* with multiple thresholds to measure side effects, which is defined as:

$$HidingCost = w_1 \times HF + w_2 \times MC + w_3 \times AC \tag{9}$$

where *HF* is hiding failure, *M*C is missing cost, and *AC* is artificial cost (see section 4.2). w_1, w_2, w_3 are the relative weights of each side effect defined by the user and $w_1+w_2+w_3=1$. When the hidden rate of sensitive itemsets is high, w_1 is a small value or even 0. The user specifies the values of w_2 and w_3 according to the relative importance of *MC* and *AC*, respectively.

4 The proposed algorithm

To hide sensitive utility itemsets, many previous algorithms propose different methods to reduce utility values. Although the FPUTT algorithm reduces the number of database scans by constructing a tree, the construction and maintenance of its tree still require a lot of overhead. In this section, a utilitylist dictionary structure is proposed to hide sensitive information. The entire process of the FULD algorithm is shown in Fig. 1.

4.1 Initial utility-list dictionary

As the first step of the FULD algorithm, the paper constructs a utility-list dictionary that consists of sensitive items and UTLists. When data holders share data, the third party can find important information through data mining. However, sensitive information may cause certain losses to the data holder if obtained by an untrusted third party. Therefore, the paper proposes the utility-list dictionary (UTLDic) to store sensitive high-utility information. To reduce the times of database scans and the number of item utilities calculated during the cleanup, the information of sensitive items is stored in UTLDic. In addition, the paper also

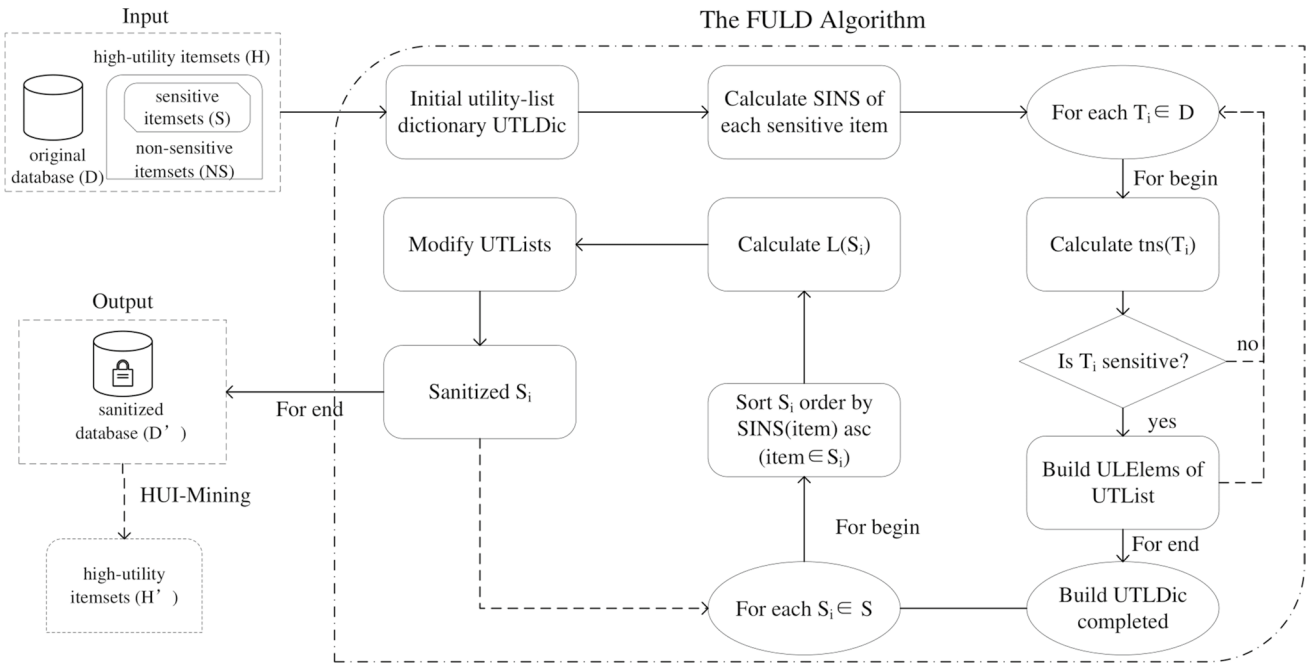


Fig. 1 The overall process of FULD

Table 3 HUIs Table

Itemset	Utility	Itemset	Utility	Itemset	Utility	Itemset	Utility
ACDE	205	ACD	234	BE	202	BDE	250
BCDE	274	BCE	226	BD	289	BCD	325
BC	223	CDE	266	CE	305	CD	278

proposes the concepts of *SINS* and *tns* to select victim item, which can reduce the side effects of the algorithm. Compared with traditional privacy preserving utility mining algorithms, FULD can hide sensitive itemsets more quickly and reduce side effects. The definitions of FULD are as follows:

Definition 14 The number of times each sensitive item appears in the non-sensitive itemsets is denoted as SINS(i_v). For example in Table 3, it is assumed that {ACD} and {BC} are sensitive high-utility itemsets. The set of sensitive items is {*A, B, C, D*}. According to Table 3, there are only one non-sensitive high-utility itemsets {ACDE}, which contains sensitive item A. Therefore, $SINS(A)=1$. Similarly, $SINS(B)=6$.

Definition 15 The relationship between transaction T_n and non-sensitive itemsets, denoted as $tns(T_n)$, is shown below:

$$tns(T_n) = \frac{1}{1 + NSI\_num} \tag{10}$$

where NSI_num indicates the number of non-sensitive itemsets contained in T_n. When a transaction contains more non-sensitive items, the *tns* is smaller. On the contrary, the *tns* is larger. Taking Table 1 as an example, T_5 contains one non-sensitive high-utility itemsets {*CD*}. Therefore, $tns(T_5)=1/(1+1)=0.5$.

Definition 16 Utility List Elem (ULElem) ULElem is a sensitive item element. It contains three fields: TID, tns (see Definition 15) and item-utility, where TID is the unique identifier of the transaction and item-utility is the utility of a sensitive item. The ULElem is shown in Fig. 2.

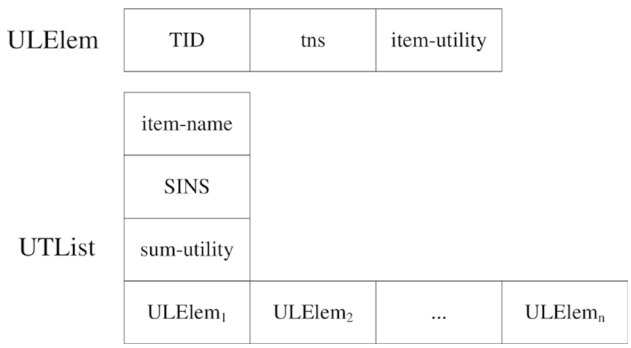


Fig. 2 ULElem and UTlist

Definition 17 Utility list (UTList) UTList contains all the information of the sensitive item $S_i(S_i \in S)$ in database D. It consists of four parts: item-name, SINS (see Definition 14), sum-utility, and ULElems, where item-name is the item name, and sum-utility is the total utility of the sensitive item in database D. The UTList is shown in Fig. 2.

Definition 18 Utility-list dictionary structure (UTLDic) Utility Lists make up the UTLDic structure. The keys of the UTLDic are a set of sensitive items. The UTLDic is shown in Fig. 3.

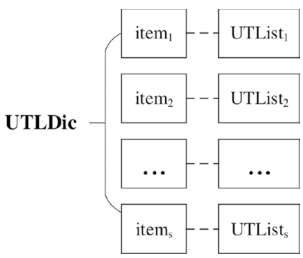
Definition 19 Let ∩ be the common ULElems between two UTLists, where the ULElems have the same TID. Let $L(X)$ be a set, which is formulated as follows:

$$L = UTList_1 \cap UTList_2 \cap \cdots \cap UTList_n \tag{11}$$

For example in Fig. 4, $L(BC)=UTList_2 \cap UTList_3=\{3,4\}$. $L(ACD)=UTList_1 \cap UTList_3 \cap UTList_4=\{5,7,8\}$.

According to Definitions 16–18, the paper proposed the utility-list dictionary. For example in Table 1, the construct of UTLDic is shown in Figs. 4 and 5. In addition, the UTLDic is faster to be built and sanitized than FPUTT-tree. In [20], the author used the Header Table to point to the node of the FPUTT-tree. During perturbing the FPUTT-tree, it is necessary to traverse the tree many times to find the maximum sensitive high-utility item. To solve this problem, the paper proposes the utility-list dictionary structure. During the sanitization, it is no longer necessary to traverse the tree multiple times. The FULD algorithm only needs to look up UTLDic to find the sensitive items in the database D. The UTList consists of item-name, SINS, sum-utility, and ULElems. Then UTLists and sensitive items make up UTLDic. Let {*ACD*} and {*BC*} be sensitive itemsets. The set of sensitive items is {*A*, *B*, *C*, *D*}, which makes up the keys of

Fig. 3 The utility-list Dictionary



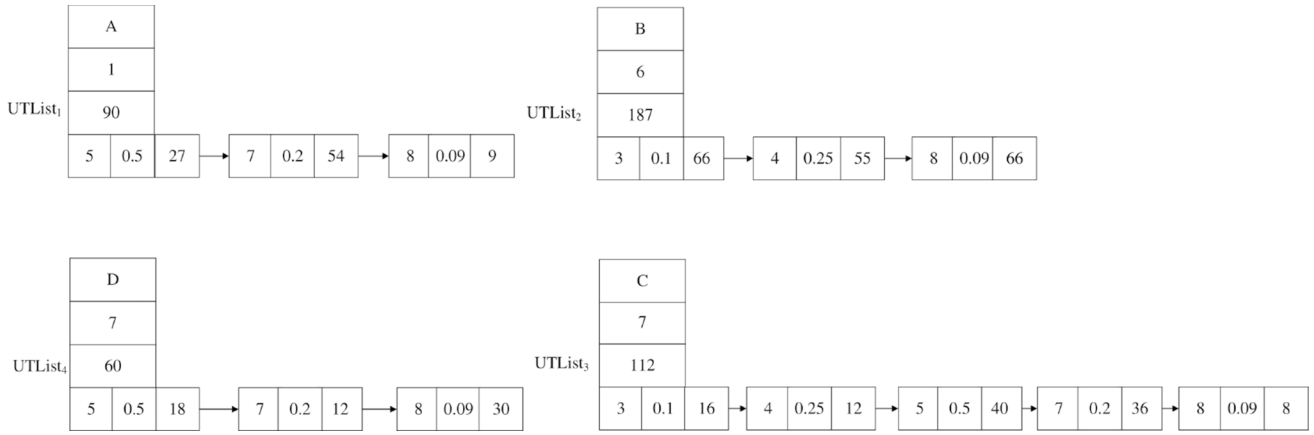


Fig. 4 An example of the UTlists

UTLDic. Taking Table 1 as an example, the construct of UTLDic is shown in Figs. 4 and 5. For example, sensitive item A in Fig. 5 points to $UTList_1$ in Fig. 4. According to Definition 14, $SINS(A) = 1$. Because sensitive item A of $\{ACD\}$ is in T_5, T_7, and T_8, the ULElems of $UTList_1$ are (5, 0.5, 27), (7, 0.2, 54), and (8, 0.09, 9). Among them, $tns(ULElem.TID)$ is calculated according to Definition 15. $sum\_utility = 27 + 54 + 9 = 90$. The detail of the construct utility-list dictionary is shown in Algorithm 1.

In Algorithm 1, firstly the union of all sensitive high-utility itemsets S is obtained and stored in *SItem* (lines 1–4). Then the items of *SItem* are deduplicated (line 5). Secondly, traverse the sensitive items *SItem* and initialize the UTLDic information (lines 6–13). Then, calculate $SINS(item)$ for each key of UTLDic (line 11). Thirdly, the database D is scanned (line 14). According to Eq. 10, $tns(T_i)$ is calculated. The set of sensitive items SI is got from T_i. It constitutes the sensitive high-utility itemsets (line 16). Finally, traverse SI

Algorithm 1 Construct UTLDic Algorithm

Require: the database D, the sensitive high-utiltiy itemsets S, the non-sensitive high-utility itemsets NS
Ensure: the utility-list dictionary $UTLDic$

1: $SItem = \{\}$
2: **for** each $S_i \in S$ **do**
3: $SItem = SItem \cup S_i$
4: **end for**
5: $SItem = \text{set}(SItem)$
6: $UTLDic = \phi$
7: **for** each $item \in SItem$ **do**
8: create a new node $UTList$
9: $UTList.item\_name = item$, $UTList.sum\_utility = 0$
10: $UTList.ULElems = \phi$, $UTList.SINS = 0$
11: calculate $SINS(item)$ according to Definition 14
12: $UTLDic[item] = UTList$
13: **end for**
14: **for** each $T_i \in DB$ **do**
15: calculate $tns(T_i)$ using Eq. c
16: get sensitive items SI of T_i
17: **for** each $item \in SI$ **do**
18: create a new ULElem node ULE
19: $ULE.TID$=the TID of T_i, $ULE.utility = q(item, T_i) \times p(item)$
20: $ULE.tns = tns(T_i)$
21: $UTLDic[item].ULElems.append(ULE)$
22: $UTLDic[item].sum\_utility \mathrel{+}= ULE.utility$
23: **end for**
24: **end for**
25: **return** $UTLDic$

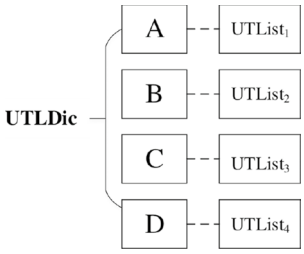


Fig. 5 An example of the UTLDic

and construct UTList of *UTLDic* (lines 17–23). ULElem is created according to *TID*, $u(item, T_i)$ and $tns(T_i)$ (lines 18–20). The ULElem is added to UTList and *sum_utility* of UTList is updated (lines 21–22). The entire UTLDic construction process is over. Taking Table 1 as an example, the construction results of UTLDic are shown in Figs. 4 and 5.

4.2 Hiding sensitive high-utility itemsets

This subsection describes hiding sensitive high-utility itemsets by the UTLDic in detail. The principle of sanitization is to keep the utility below δ by reducing the internal utility of the sensitive item. The sensitive itemsets are sorted in descending order of utility. The victim ULElems are modified according to the concepts of *SINS* and *tns* proposed in the paper.

During the sanitization, the UTLDic can be accessed to find the victim ULElem. In the previous works, the database was scanned repeatedly. To solve this problem, [20] proposed FPUTT-tree to reduce the multiple scans of the database during the perturbation. Regardless, there are some flaws in FPUTT-tree. The FPUTT-tree was accessed repeatedly to find the maximum item. It wasted much time and caused high side effects. Besides, the item's utility is calculated repeatedly in the previous work. The FULD algorithm is proposed to solve these problems. During the sanitization, the algorithm does not need to access all sensitive items repeatedly. Apart from that, the item's utility does not need to be recalculated. The novel concepts of *SINS* and *tns* proposed in the paper reduce side effects of FULD.

In Algorithm 2, the sensitive itemsets are sorted in descending order according to the utility (line 1). Firstly, S is traversed (line 2). According to $SINS(item)$, the sensitive items of S_i are sorted in ascending order (line 3). The TIDs of transactions that contain S_i are got according to definition 19 (line 4). Secondly, the target utility that needs to be reduced is calculated (line 5). The victim ULElems are modified until *targetUtil* is not greater than 0 (lines 6–22). During the process of modifying victim ULElems, the sensitive items of S are accessed (line 7). When accessing a sensitive item, the ULElems of UTlist are sorted in descending order according to *tns*. When the *tns* values are the same, they are sorted in ascending

Algorithm 2 Hide Sensitive High-utility Itemsets Algorithm

Require: $UTLDic$, min_util, the sensitive high-utility itemsets S, the database D
Ensure: the sanitized $UTLDic$

1: sort S in descending order of $u(S_i)$ $(S_i \in S)$
2: **for** each $S_i \in S$ **do**
3: sort S_i in ascending order of $SINS(item)$ $(item \in S_i)$
4: calculate $l = L(S_i)$ according to Definition 19
5: $targetUtil$=$u(S_i)$-$min\_util + 1$
6: **while** $targetUtil > 0$ **do**
7: **for** each $item \in S_i$ **do**
8: $UTlist = UTLDic[item].UTList$
9: sort $ULElems$ of $UTlist$ order by tns desc, $utility$ asc
10: **for** each $elem \in UTlist$ and $elem \in l$ and $targetUtil > 0$ **do**
11: **if** $elem.utility \leq targetUtil$ **then**
12: $targetUtil- = elem.utility$
13: $elem.utility$=0
14: **else**
15: $count = q(item, elem.TID) - \lceil \frac{targetUtil}{p(item)} \rceil$
16: $elem.utility = count \times p(item)$
17: $targetUtil = 0$
18: **end if**
19: update $UTLDic[item].sum\_utility$
20: **end for**
21: **end for**
22: **end while**
23: **end for**

order of utility (lines 8–9). Then visit the ULElem of *UTlist* in turn (line 10). If the ULElem contained in the transactions whose TIDs are in $L(S_i)$, the ULElem is modified according to the formula (lines 11–18). Finally, the UTLDic is updated (line 19).

An example: In Table 3, we assume that $\{ACD\}$ and $\{BC\}$ are sensitive high-utility itemsets, where $u(ACD)$ is 234 and $u(BC)$ is 223. These sensitive itemsets are sorted in descending order of the utility as follows: $\{ACD\}$ and $\{BC\}$. $\{ACD\}$ is first visited. Then $\{ACD\}$ is sorted in ascending order of $SINS(item)$ as follows: $\{ADC\}$. In the paper, the minimum utility threshold δ is set to 200. According to Definition 19, the TIDs of transactions containing $\{ACD\}$ is $L(ACD)=\{5,7,8\}$. The total utility that needs to be modified is $u(ACD,T_5)+u(ACD,T_7)+u(ACD,T_8)-\delta=234-200+1=35$. Secondly, A is visited. According to Fig. 5, $UTList_1$ is got. The ULElems of $UTList_1$ are sorted in descending order according to *tns*. When the *tns* values are the same, they are sorted in ascending order of utility. Then in order, ULElems are visited in turn. Because $27<35$, the utility of ULElem (5,0.5,27) is changed to 0 and $sum_u tility$ of $UTList_1$ is modified to $90-27=63$. *targetUtil* is equal to $35-27=8$. Next, the ULElem (7,0.2,54) is accessed. Since $8<54$, the utility of this ULElem is updated to $(6-\lceil 8/9\rceil)\times 9=45$. The value of *targetUtil* is 0. The operation of $\{BC\}$ is similar to the above Fig. 6. The final sanitization result is shown in Fig. 7.

4.3 The FULD Algorithm

This subsection introduces the entire process of the FULD algorithm. Compared with previous related work, the algorithm does not need to visit and scan the database multiple times when perturbing the original database. As described in the algorithm 2, the algorithm only needs to access the utility-list dictionary to modify the item's utility. In the FULD algorithm, the original database is accessed to build the UTLDic. Then traverse the sensitive items of sensitive itemsets for sanitization according to Section 4.2. Finally, the algorithm generates the sanitized database DB'. The FULD algorithm is shown in Algorithm 3.

5 Experiments

The experiments contain four datasets. The characteristics of the datasets are shown in Table 4. These datasets can be obtained from SPMF (http://www.philippe-fournier-viger.com/spmf/). The *Density* indicates the sparseness of the dataset. The *AvgLen* signifies the number of transactions. The calculation formula is as follows:

$$Density = \frac{AvgLen}{Itemcount} \tag{12}$$

Besides experiments on different datasets, the paper also compared previous algorithms, such as: HHUIF and MSICF [18], MSU-MAU and MSU-MIU [19], FPUTT [20]. In the following experiment, we used different threshold δ and various percentages of sensitive information for comparative experiments. The sensitive itemsets in this experiment were randomly selected from HUI. The algorithm used to mine the high utility patterns is ULB-Miner [44]. Since the experimental datasets do not contain the internal and external utility, we adopted the uniform distribution proposed by Ge et al. [45] to assign an internal utility value to each item in the transaction. Furthermore, we used the Gaussian distribution proposed by Tassa et al. [46] to generate the external utility corresponding to each unique item. The experiments were conducted under Windows 10 with Intel Core i7 CPU and 16 GB of RAM.

Besides experiments on different datasets, this paper also compared previous algorithms, such as HHUIF and MSICF, MSU-MAU and MSU-MIU, and FPUTT. In the

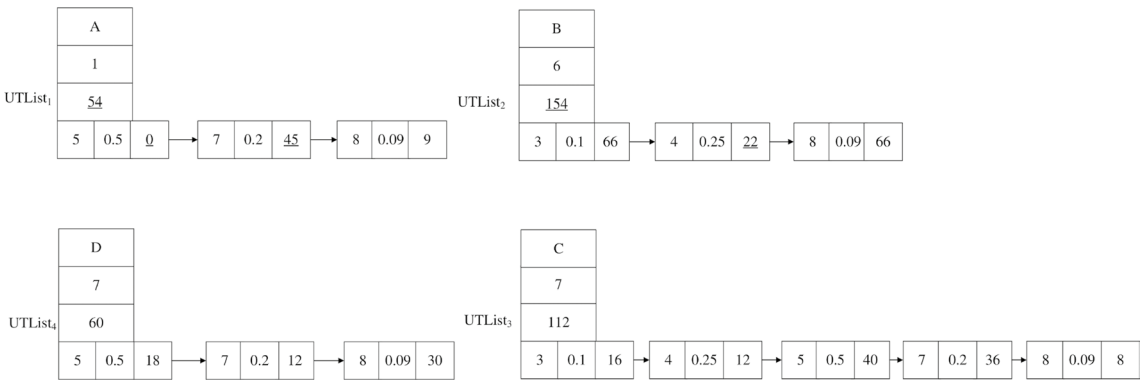


Fig. 6 An example of the sanitized UTLDic'

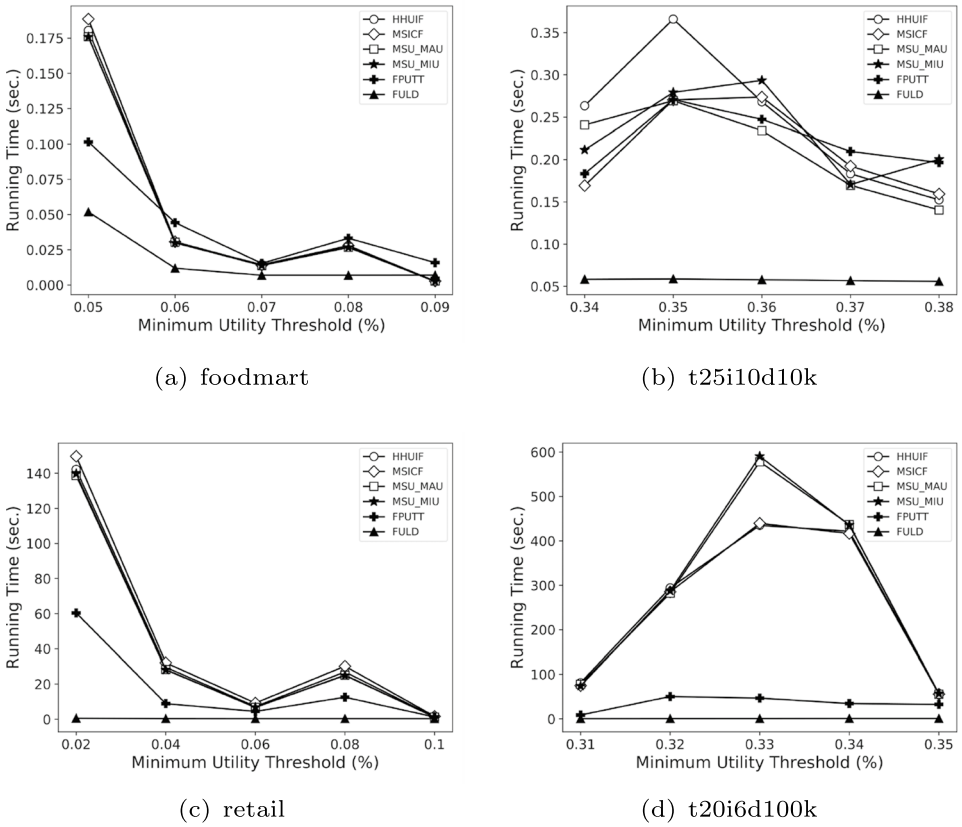


Fig. 7 Comparison of running time under different sensitive information percentages

following experiments, we used different threshold δ and various percentages of sensitive information for comparative experiments. The sensitive itemsets in this experiment were randomly selected from HUI. The algorithm used to mine the high utility patterns is ULB-Miner. Since the experimental datasets do not contain the internal and external utility, we adopted the uniform distribution proposed by Ge et al. to assign an internal utility value to each item in the transaction. Furthermore, We used the Gaussian distribution proposed by Tassa et al. to generate the external utility corresponding to each unique item. The experiments were conducted under Windows 10 with Intel Core i7 CPU and 16 GB of RAM.

5.1 Runtime analysis

This subsection introduces the comparison of the operating effects of the six algorithms under different datasets, different minimum utility thresholds, and different percentages of sensitive information, as shown in Figs. 7 and 8.

In high-utility pattern mining, sensitive high-utility itemsets are only a small part of high-utility itemsets. The FULD only stores ULElems, where the sensitive items are located, not all transactions containing sensitive items. The FULD compresses and stores relevant sensitive information into ULDic. The ULDic proposed in the paper is mainly stored in memory, and the space overhead for the compressed sensitive information is small.

As shown in Fig. 7, the running time of algorithms shows a downward trend as the minimum utility threshold increases. The reason is that as the minimum utility threshold increases, the number of HUIs will decrease. This issue leads to a decrease in the percentage of HUIs, and the running time of the algorithms decreases. Since the sensitive itemsets are randomly selected, the number

Algorithm 3 The FULD Algorighm

Require:the original database DB, the sensitive high-utility itemsets S, δ
Ensure:the sanitized database DB'

1: call Construct UTLDic Algorithm
2: call Hide Sensitive High-utility Itemsets Algorithm
3: generate the sanitized database DB'

Table 4 Experimental datasets

Dataset	|Dataset|	|Item|	AvgLen	Density
foodmart	4141	1559	4.42	0.28%
t25i10d10k	9976	929	24.77	2.67%
retail	88,162	16,470	1030.00	6.25%
t20i6d100k	99,922	893	24.77	2.77%

of transactions having sensitive high-utility itemsets may differ. The more transactions contain sensitive itemsets, the longer the sanitization takes. For example, the t20i6d100k dataset in Fig. 7 still does not affect the overall downward trend of algorithms running time as the minimum utility threshold increases. When the datasets and the minimum utility threshold are the same, the FULD algorithm proposed in this paper is significantly better than the other five algorithms. The running time of the FULD is 15–20 times shorter than the FPUTT. In PPUM, we should minimize the side effects and consider the running time. When the datasets are large, the running time is tens or even hundreds of times as long as compared to smaller datasets. If it consumes much time because of minimum side effects, it is also not advisable for researchers. To solve the issue, we propose the FULD algorithm. In Fig. 7, as the sensitive information percentage increases, the running time of different algorithms also increases. Because the greater the number of SHUIs, the more time it takes to perturb the database. When the dataset and the sensitive information percentage are the same, the performance of the FULD algorithm is more significant.

Whether it is Fig. 7 or Fig. 8, the lower the number of transactions in the dataset, the less the algorithm runs, such as the foodmart and t25i10d10k datasets. Compared with the other two datasets, they have only a few thousand transactions, so the running time is relatively short. According to the dataset density in Table 4, the experiments show that no matter whether the dataset is dense or sparse, the FULD algorithm has good scalability and the operating effect is better than the other five algorithms.

5.2 Side effects

In this subsection, we introduce the experimental results of all algorithms in terms of side effects. The side effects discussed in this article are measured using Hiding Cost. It includes hiding failure, missing cost, and artificial cost (see section 3.2).

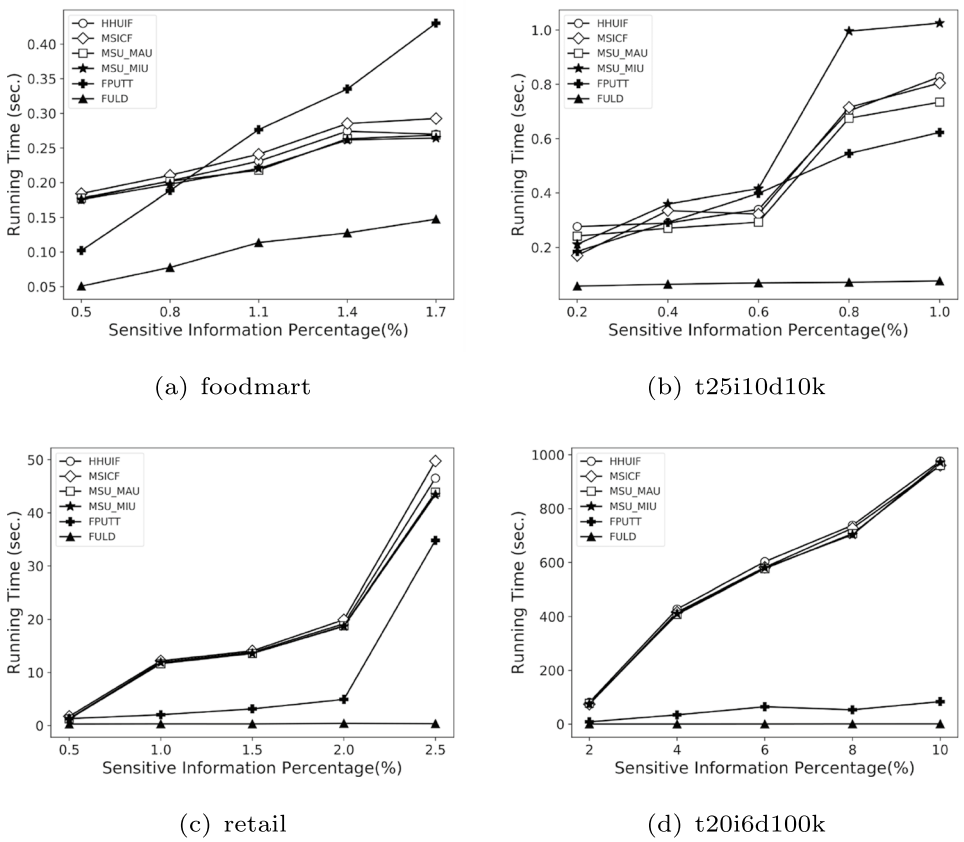


Fig. 8 Comparison of running time under different minimum utility thresholds

The *HidingCost* applies to all situations. When the algorithms will not lead to hiding failure, w_1 can be set to 0 or a smaller value. The values of w_2 and w_3 are set similarly. In the experiments, w_2 and w_3 are set to 0.4 and 0.1 for all algorithms. Since hiding failure is more important, w_1 is set to 0.5.

During the sanitization process, FULD preferentially selects the sensitive item with the smaller SINS(item) for modification. The smaller $SINS(item)$ is, the less the item appears in non-sensitive high-utility itemsets. This can reduce the impact on non-sensitive high-utility itemsets (NS). When the victim ULElem is selected, the ULElem with larger $tns(T_i)$ is preferred. Because the larger the $tns(T_i)$ is, the less NS the T_i contains. In other words, when the item is modified, the impact on NS is less. To reduce the impact on other HUIs, ULElem with less utility is preferred.

The paper compares the hiding cost of six algorithms under different conditions. Figures 9 and 10 show the experimental results. Figure 9 shows that the side effects of the FULD are relatively small compared to other algorithms under different minimum utility thresholds. When the minimum utility threshold increases, the side effects of all algorithms will decrease. In Fig. 10, FULD still has better performance in terms of side effects. As the percentage of sensitive information increases, the hiding costs increase. The more sensitive information needs to be hidden, the side effects are higher. In conclusion, the FULD algorithm not only has a shorter running time but also produces fewer side effects.

6 Complexity analysis

In this section, the time and space complexity of the FPUTT and FULD algorithms is performed for the theoretical comparison. The relevant parameters are as follows:

- N_s is the number of sensitive high-utility itemsets;
- DB_{scan} is the number of items that need to be accessed when scanning the database;
- $NR(S_p)$ is the number of repetitions required to hide a certain sensitive itemset S_p;
- $TC(k, S_p)$ is the time required to update the utility of S_p in the kth scanning the database;
- $TREE_{scan}$ is the number of nodes visited during one tree traversal.
- $UTLIST_{scan}$ is the number of ULElem visited during one UTList traversal.

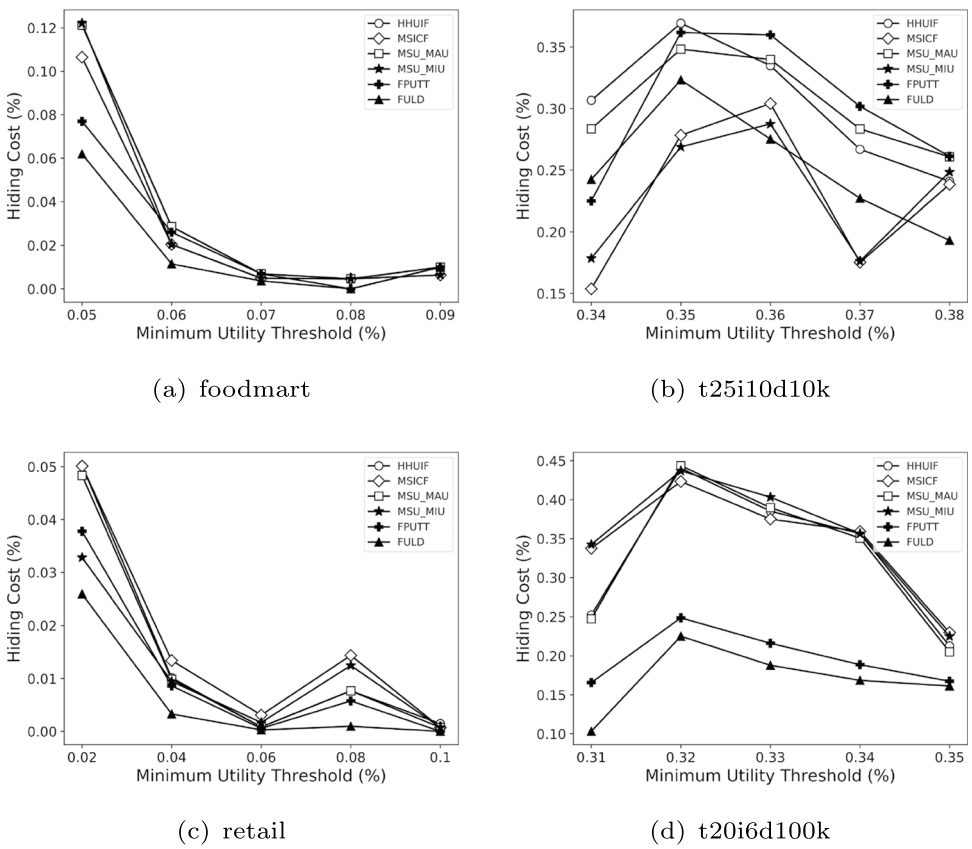


Fig. 9 Comparison of Hiding Cost under different minimum utility thresholds

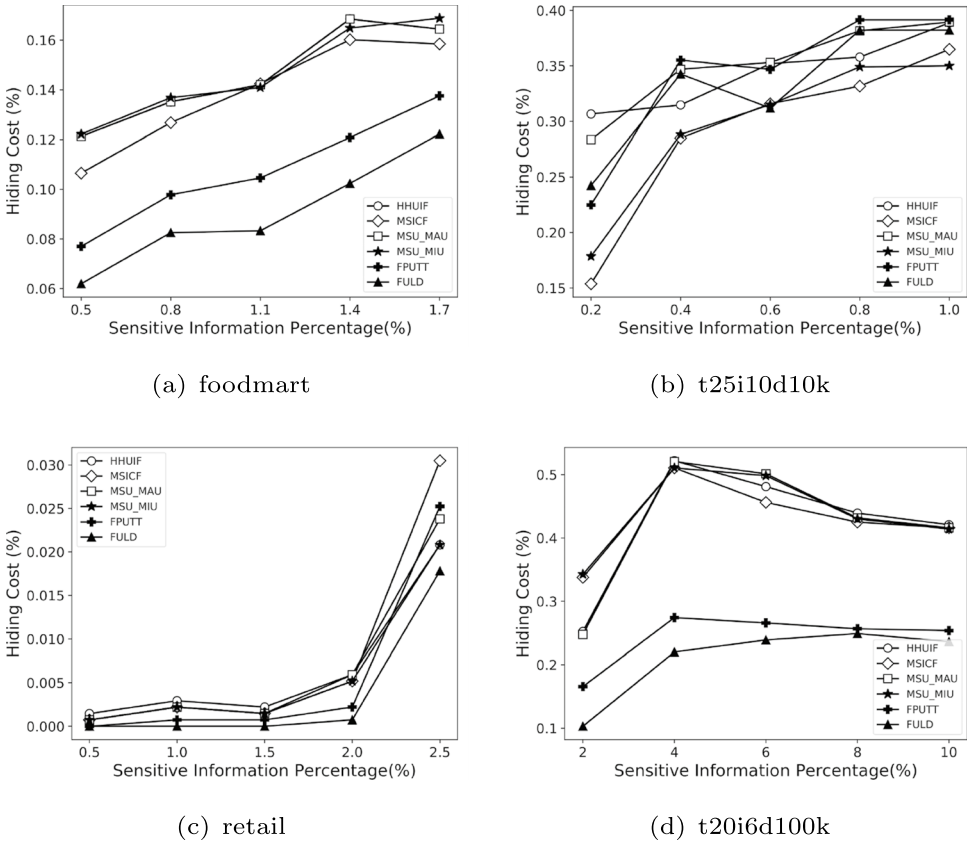


Fig. 10 Comparison of Hiding Cost under different sensitive information percentages

Based on the pseudo-code analysis of the FPUTT [20] and FULD algorithms, we have drawn the following time complexity formula:

$$\text{FPUTT} \rightarrow 3 * DB_{scan} + \sum_{p=1}^{N_S} \sum_{k=1}^{NR(S_p)} \{TREE_{scan} + TC(k, S_p)\},$$
$$\text{FULD} \rightarrow 2 * DB_{scan} + \sum_{r=1}^{N_S} \sum_{k=1}^{NR(S_r)} \{UTLIST_{scan} + TC(k, S_r)\}.$$

The FULD algorithm constructs the UTLDic with one database scan (DB_{scan}) and access the UTLDic $\sum_{r=1}^{N_S} \sum_{k=1}^{NR(S_r)} \{UTLIST_{scan}\}$. Then the sensitive items in the UTLDic are updated $\left(\sum_{r=1}^{N_S} \sum_{k=1}^{NR(S_r)} \{TC(k, S_r)\}\right)$. However, the FPUTT algorithm requires two database scans ($2 * DB_{scan}$) to build the tree and traverse the tree $\left(\sum_{p=1}^{N_S} \sum_{k=1}^{NR(S_p)} \{TREE_{scan}\}\right)$. Note that $\exists NR(S_r) \neq NR(S_p)$ and $UTLIST_{scan}$ $TREE_{scan}$. According to the paper [20], the worst $TREE_{scan}$ is calculated as follows:

$$TREE_{scan} = N_n \times \frac{\sum_{j=1}^{N_n} LN(n_j)}{N_n} \times \frac{\sum_{p=1}^{N_s} LS(n_p)}{N_s T_s} \tag{13}$$

N_n is the number of nodes in the tree. T_s is the number of transactions with multiple sensitive itemsets. In each sensitive itemset S_p of the SI-table, the length of the TID set is denoted as $LS(S_p)$. $LN(n_j)$ is the length of the TID set contained in each node of the tree. N_s is the number of sensitive itemsets. Let N_e be the number of the ULElems in the UTList. The worst $UTLIST_{scan}$ is calculated as follows:

$$UTLIST_{scan} = N_e \tag{14}$$

Obviously, $UTLIST_{scan}$ is less than $TREE_{scan}$. According to the above analysis, the time complexity of the FPUTT is: $O(DB_{scan} + N_s \times NR(S_p) \times TREE_{scan})$. The time complexity of the FULD is: $O(DB_{scan} + N_s \times NR(S_r) \times UTLIST_{scan})$. Therefore, the FULD algorithm has better performance in running time. Let N_{si} be the length of the sensitive item set, and LV be the length of the TID set of the node in the FPUTT-tree. The space complexity of the FPUTT and FULD algorithm are as follows: $FPUTT = O(N_{si} + N_n \times LN_{avg})$, and $FULD = O(N_{si} + N_{si} \times N_e)$.

7 Conclusion

With the improvement of people's privacy awareness, the potential privacy breach caused by HUPM has attracted the attention of researchers. In short, PPUM is hiding sensitive high-utility itemsets. When companies share data, the hidden sensitive information can not be mined by third parties. In the paper, we propose the FULD

algorithm, which attempts to minimize side effects while preserving privacy. We use the utility-list dictionary to improve the sanitization process. Through extensive experimental analysis and comparison, our algorithm has outstanding performance in respect of running time and side effects. In future studies, researchers should pay attention to that reduce the side effects and running time of PPUM. On the one hand, the tree-based strategy can improve the sanitization process. It is an important research direction that design a novel pruning strategy to reduce the search space. On the other hand, although the exact algorithm based on integer linear programming can find the optimal solution, it takes a long time. How to combine the heuristic algorithm to find the optimal solution while taking into account the running time is also a new research direction.

Acknowledgments The authors are sincerely grateful to the editor and anonymous reviewers for their insightful and constructive comments, which helped us improve this work greatly.

Author Contributions All authors contributed to the study conception and design. Material preparation, data collection and analysis were performed by Chunyong Yin and Ying Li. The first draft of the manuscript was written by Ying Li and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Funding This research was funded by the National Natural Science Foundation of China (61772282). It was also supported by the Postgraduate Research & Practice Innovation Program of Jiangsu Province (KYCX21 1008).

Declarations

Conflict of interest The authors declare that they have no known competing financial interest or personal relationship that could have appeared to influence the work reported in this paper.

References

1. Chen M-S, Han J, Philip SY (1996) Data mining: an overview from a database perspective. IEEE Trans Knowl Data Eng 8(6):866–883
2. Gan W, Lin JC-W, Fournier-Viger P, Chao H-C, Yu PS (2019) A survey of parallel sequential pattern mining. ACM Transac Knowl Disc Data (TKDD) 13(3):1–34
3. Mannila H, Toivonen H, Verkamo IA (1997) Discovery of frequent episodes in event sequences. Data Min Knowl Disc 1(3):259–289
4. Agrawal R, Imieliński T, Swami A (1993) Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD international conference on Management of data, p. 207–216
5. Han J, Pei J, Yin Y, Mao R (2004) Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data Min Knowl Disc 8(1):53–87
6. Fournier-Viger P, Lin JC-W, Kiran RU, Koh YS, Thomas R (2017) A survey of sequential pattern mining. Data Sci Patt Recog 1(1):54–77
7. Koh YS, Ravana SD (2016) Unsupervised rare pattern mining: a survey. ACM Transac Knowl Disc Data (TKDD) 10(4):1–29
8. Fournier-Viger P, Lin JC-W, Vo B, Chi TT, Zhang J, Le HB (2017) A survey of itemset mining. Wiley Interdisciplinary Reviews. Data Min Knowl Disc 7(4):e1207
9. Yao H, Hamilton HJ, Geng L (2006) A unified framework for utility-based measures for mining itemsets. In: Proc. of ACM SIGKDD 2nd Workshop on Utility-Based Data Mining, pages 28–37. Citeseer
10. Geng L, Hamilton HJ (2006) Interestingness measures for data mining: A survey. ACM Comput Surv (CSUR) 38(3):9–es
11. Tan P-N, Kumar V, Srivastava J (2004) Selecting the right objective measure for association analysis. Inf Syst 29(4):293–313
12. McGarry K (2005) A survey of interestingness measures for knowledge discovery. Knowl Eng Rev 20(1):39–61
13. Hilderman RJ, Hamilton HJ (2003) Measuring the interestingness of discovered knowledge: A principled approach. Int Data Analy 7(4):347–382
14. Silberschatz A, Tuzhilin A (1995) On subjective measures of interestingness in knowledge discovery. In: KDD, volume 95, pp. 275–281
15. Dwork C (2006) Differential privacy. In: International Colloquium on Automata, Languages, and Programming, pp. 1–12, Springer
16. Gentry C (2009) Fully homomorphic encryption using ideal lattices. In: Proceedings of the forty-first annual ACM symposium on Theory of computing, pp. 169–178
17. Weng J, Weng J, Zhang J, Li M, Zhang Y, Luo W (2019) Deepchain: Auditable and privacy-preserving deep learning with blockchain-based incentive. IEEE Transactions on Dependable and Secure Computing
18. Yeh J-S, Hsu P-C (2010) Hhuif and msicf: Novel algorithms for privacy preserving utility mining. Expert Syst Appl 37(7):4779–4786
19. Lin JC-W, Gan W, Fournier-Viger P, Hong T-P, Tseng VS (2016) Fast algorithms for mining high-utility itemsets with various discount strategies. Adv Eng Inform 30(2):109–126
20. Yun U, Kim J (2015) A fast perturbation algorithm using tree structure for privacy preserving utility mining. Expert Syst Appl 42(3):1149–1165
21. Li S, Nankun M, Le J, Liao X (2019) A novel algorithm for privacy preserving utility mining based on integer linear programming. Eng Appl Artif Intell 81:300–312
22. Lin JC-W, Djenouri Y, Srivastava G, Fournier-Viger P (2022) Efficient evolutionary computation model of closed high-utility itemset mining. Appl Intell, p. 1–13
23. Gan W, Lin JC-W, Fournier-Viger P, Chao H-C, Tseng VS, Philip SY (2021) A survey of utility-oriented pattern mining. IEEE Trans Knowl Data Eng 33(4):1306–1327
24. Lin JC-W, Djenouri Y, Srivastava G (2021) Efficient closed high-utility pattern fusion model in large-scale databases. Inform Fusion 76:122–132
25. Lin JC-W, Djenouri Y, Srivastava G, Yun U, Fournier-Viger P (2021) A predictive ga-based model for closed high-utility itemset mining. Appl Soft Comput, 108:107422
26. Kim H, Ryu T, Lee C, Kim H, Yoon E, Vo B, Lin JC-W, Yun U (2022) Ehmin: Efficient approach of list based high-utility pattern mining with negative unit profits. Expert Syst Appl, 209:118214
27. Lee C, Baek Y, Ryu T, Kim H, Kim H, Lin JC-W, Vo B, Yun U (2022) An efficient approach for mining maximized erasable utility patterns. Inf Sci 609:1288–1308
28. Ryu T, Yun U, Lee C, Lin JC-W, Pedrycz W (2022) Occupancy-based utility pattern mining in dynamic environments of intelligent systems. Int J Intell Syst 37(9):5477–5507

29. Jianying H, Mojsilovic A (2007) High-utility pattern mining: A method for discovery of high-utility item sets. Pattern Recogn 40(11):3317–3324
30. Lin C-W, Hong T-P, Wen-Hsiang L (2011) An effective tree structure for mining high utility itemsets. Expert Syst Appl 38(6):7419–7424
31. Krishnamoorthy S (2015) Pruning strategies for mining high utility itemsets. Expert Syst Appl 42(5):2371–2381
32. Zida S, Fournier-Viger P, Lin JC-W, Cheng-Wei W, Tseng VS (2015) Efim: a highly efficient algorithm for high-utility itemset mining. In: Mexican international conference on artificial intelligence, pp. 530–546. Springer
33. Liu J, Wang K, Fung BCM (2012) Direct discovery of high utility itemsets without candidate generation. In: 2012 IEEE 12th international conference on data mining, pages 984–989. IEEE
34. Kim H, Yun U, Baek Y, Kim H, Nam H, Lin JC-W, Fournier-Viger P (2021) Damped sliding based utility oriented pattern mining over stream data. Knowl-Based Syst, 213, p. 106653
35. Baek Y, Yun U, Kim H, Kim J, Vo B, Truong T, Deng Z-H (2021) Approximate high utility itemset mining in noisy environments. Knowl-Based Syst, 212:106596
36. Hong T-P, Lin C-W, Yang K-T, Wang S-L (2013) Using tf-idf to hide sensitive itemsets. Appl Intell, 38:502–510
37. Jangra S, Toshniwal D (2022) Efficient algorithms for victim item selection in privacy-preserving utility mining. Futur Gener Comput Syst, 128, pp. 219–234
38. Lin JC-W, Fournier-Viger P, Wu L, Gan W, Djenouri Y, Zhang J (2018) Ppsf: An open-source privacy-preserving and security mining framework. In: 2018 IEEE International Conference on Data Mining Workshops (ICDMW), pp. 1459–1463
39. Jimmy Ming-Tai W, Srivastava G, Jolfaei A, Pirouz M, Lin JC-W (2021) Security and privacy in shared hitlcps using a ga-based multiple-threshold sanitization model. IEEE Transactions on Emerging Topics in Comput Intell
40. Lin JC-W, Srivastava G, Zhang Y, Djenouri Y, Aloqaily M (2020) Privacy-preserving multiobjective sanitization model in 6g iot environments. IEEE Internet Things J 8(7):5340–5349
41. Dinh T, Quang MN, Le B (2015) A novel approach for hiding high utility sequential patterns. In: Proceedings of the Sixth International Symposium on Information and Communication Technology, pp. 121–128
42. Quang MN, Huynh U, Dinh T, Le NH, Le B (2016) An approach to decrease execution time and difference for hiding high utility sequential patterns. In: International Symposium on Integrated Uncertainty in Knowledge Modelling and Decision Making, pp. 435–446. Springer
43. Lin JC-W, Liu Q, Fournier-Viger P, Hong T-P, Voznak M, Zhan J (2016) A sanitization approach for hiding sensitive itemsets based on particle swarm optimization. Eng Appl Artif Intell, 53:1–18
44. Duong Q-H, Fournier-Viger P, Ramampiaro H, Nørvåg K, Dam T-L (2018) Efficient high utility itemset mining using buffered utility-lists. Appl Intell 48(7):1859–1877
45. Ge Z, Song Z, Ding SX, Huang B (2017) Data mining and analytics in the process industry: The role of machine learning. Ieee Access, 5, pp. 20590–20616
46. Tassa T (2013) Secure mining of association rules in horizontally distributed databases. IEEE Trans Knowl Data Eng 26(4):970–983

Chunyong Yin received the B.S. degree from Shandong University of Technology, China in 1998. He received the M.S. and Ph.D. degrees in computer science from Guizhou University, China, in 2005 and 2008, respectively. He was a post-doctoral research associate at the University of New Brunswick, Canada, in 2011 and 2012. He is currently a Professor and Dean with the Nanjing University of Information Science and Technology, China. He has authored or coauthored more than twenty journal and conference papers. His current research interests include privacy preserving and sensor networking, machine learning and network security.

Ying Li received his bachelor degree in Huaiyin Institute of Technology, China. Now she is studying for her master degree in Nanjing University of Information Science and Technology, China. Her current research interests are privacy protection and network security.