# Mining with Rarity for Web Intelligence

Yijie Gui
Jinan University
Guangzhou, China
y.j.gui123@gmail.com

Wensheng Gan*
Jinan University
Guangzhou, China
wsgan001@gmail.com

Yao Chen
Jinan University
Guangzhou, China
csyaochen@gmail.com

Yongdong Wu
Jinan University
Guangzhou, China
wuyd007@qq.com

## ABSTRACT

Mining with rarity makes sense to take advantage of data mining for Web intelligence. In some scenarios, the rare patterns are meaningful in data intelligent systems. Interesting pattern discovery plays an important role in real-world applications. In this field, a great deal of work has been done. In general, a high-utility pattern may include frequent items and also rare items. Rare pattern discovery emerges gradually and helps policy-makers making related marketing strategies. However, the existing Apriori-like methods for discovering high-utility rare itemsets (HURIs) are not efficient. In this paper, we address the problem of mining with rarity and propose an efficient algorithm, named HURI-Miner, which uses the data structure called revised utility-list to find HURIs from a transaction database. Furthermore, we utilize several powerful pruning strategies to prune the search space and save the computational complexity. In the process of rare pattern mining, the HURIs are directly generated without the generate-and-test method. Finally, a series of experimental results show that this proposed method has superior effectiveness and efficiency.

## CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; **Data analytics**.

## KEYWORDS

Web of Things, artificial intelligence, data analytics, rarity.

**ACM Reference Format:**
Yijie Gui, Wensheng Gan, Yao Chen, and Yongdong Wu. 2022. Mining with Rarity for Web Intelligence. In *Companion Proceedings of the Web Conference 2022 (WWW '22 Companion), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3487553.3524708

*Corresponding author, also with Pazhou Lab, Guangzhou, 510330, China


*WWW '22 Companion, April 25–29, 2022, Virtual Event, Lyon, France*

ACM ISBN 978-1-4503-9130-6/22/04…$15.00
https://doi.org/10.1145/3487553.3524708

## 1 INTRODUCTION

Data mining [1, 12, 38, 39] is a key process of knowledge discovery from data (KDD) which has multi-stages. The process of mining useful information from data includes dataset selection, data preprocessing, data transformation, data mining, and data analysis. In the field of data mining, researchers are often interested in association analysis, such as association rule mining (ARM) [1]. Since the Apriori algorithm [1] was proposed, many algorithms for mining frequent patterns have been developed [2, 18, 19]. In frequent pattern mining (FPM), most studies rarely pick out rare itemsets that are usually filtered. However, the rare patterns may more interesting than the frequent ones in some real-life applications. For instance, in the field of biology, a rare event can be a harmful reaction to a drug. In the field of cyber security, the occurrence of some rare behaviors in the network can be regarded as a threat to network security. Once network administrators discover these dangerous behaviors, they can take effective measures to prevent attackers from intruding. In the field of telecommunication, if the data is modified by the attacker in the process of transmission, we can find out the modified data by mining rare events. Obviously, in some scenarios, the rare patterns are also meaningful in data intelligent systems. Therefore, the concept of rare pattern mining (RPM) [5, 20, 34] is introduced. It is important to note that rare pattern is different from frequent one. The algorithms for RPM is, of course, different from that of FPM. In other words, the data structures and mining strategies designed for FPM can not be directly applied to deal with discovering rare patterns. Up until now, RPM attracts many attentions in both academia and industry due to its wide applications. How to develop effective and efficient algorithms for RPM is quite useful for various domains, e.g., decision-making, but this is quite challenging.

Association rule mining has become an active research area during the past years. However, for ARM and RPM, every object/item has the same value/weight. Those algorithms only care about the co-occurrence (aka frequency) of the itemsets, but not the profit they can generate by themselves. To solve the problem, Chan *et al.* [6] proposed a new computing framework called utility-driven data mining (aka utility mining) which aims at mining high-utility patterns from databases. In utility mining [11, 14, 15], high-utility itemset mining (HUIM) is a subfield, and it refers to the mining of itemsets with high utility such as high profit from a transaction database. Effective HUIM is very important in the field of data

mining and plays a key role in different particular fields. Subsequently, a massive number of related algorithms for utility mining [26, 35, 40] have been proposed. For example, mining recent high-utility itemsets (HUIs) from temporal or steam data [25, 33], HUIM over uncertain data [23, 24], and extracting correlated HUIs [10, 13] have been extensively studied. In these studies, the discovered patterns not only include the itemset of high frequency, but also have the itemset of low frequency. However, as the highlight in RPM, rarity of interesting patterns is also useful and makes sense.

In the past, a few studies have been conducted and shown great passion in mining high-utility rare patterns [17, 30]. A high-utility rare itemset (HURI) is defined as support value of an itemset is no greater than a given maximum support threshold and its utility is greater than a given minimum utility threshold. There has also been plenty of work on high-utility rare itemset mining (HURIM). They are typically used in basket data analysis, cross-marketing, catalog design, or other real-world applications. HURIM is helpful for decision-makers to make the business strategies. For example, a customer segmentation algorithm, named CSHURI [31], can find out the customers who buy high-profit rare goods and classify them according to certain criteria. What's more, in the field of network intrusion detection, we can rank different types of intrusions according to their risk level. Note that the utility can be refers to various meaning, i.e., profit, risk, weight, and others. The higher the intrusion danger level is, the higher its utility value is. Therefore, we can identify the intrusions which have high risk and and then make adjustments of strategies. Some remarkable attempts to address HURIM have recently been made, while their efficiency is bad, especially when dealing with large-scale database or database containing many district items and long transactions. Most of the HURIM algorithms usually have two phases. However, because the number of rare itemsets increases rapidly with the increase of maximum support threshold, resulting in longer execution time, these two-phase methods are not efficient.

In summary, mining with rarity makes sense to take advantage of data mining techniques for Web intelligence. To address the above limitation, in this paper, we mainly focus on the improvement of efficiency of HURIM algorithm. In contrast to existing algorithms, our work is fast and scale-well. The HURI-Miner algorithm applies several different pruning strategies to prune the search space. In summary, our work is interactive and efficient (both time and memory) while discovering rare HUIs.

Our main technical contributions are fourfold:

- We introduce the formal description of pattern rarity and HURIM, and provide theories to help us find HURIs effectively. The proposed algorithm takes both the utility factor and the frequency factor into account. It can more make sense since the mining with rarity.
- We propose a revised utility-list structure to directly generate HURIs, which avoids producing redundant candidates and improves the mining efficiency.
- HURI-Miner utilizes the minimum support threshold *min_sup*, the maximum threshold *max_sup*, and the minimum utility threshold *min_util* to discover HURIs simultaneously. Specially, it uses *min_sup* to prune the branch whose support is less than *min_sup* and save the execution time. In addition, some other pruning strategies are adopted in HURI-Miner to improve the performance.
- We conduct an extensive experiments on different datasets and compare HURI-Miner with the state-of-the-art algorithm to show the effectiveness and efficiency of HURI-Miner.

The rest of this paper is organized as follows. The related work is stated in Section 2. Some preliminaries and the formal description of HURIM are described in Section 3. Our algorithm for mining HURIs is demonstrated in Section 4. We conduct some experiments and present the detailed results in Section 5. Conclusion is given in Section 6.

## 2 RELATED WORK

### 2.1 Traditional HUIM algorithms

With technical development of the Web of Things (WoT) [4, 37], many attempts of data intelligence are often used in basket data analysis, cross-marketing, catalog design and or other applications [8, 29]. In the past decades, many algorithms have been proposed to discover high-utility itemsets. The Two-Phase algorithm [27] firstly proposes the concept of transaction-weighted utilization (twu) and the twu-based downward closure property. Especially, Ahmed *et al.* [3] proposed three tree structures to perform incremental and interactive mining effectively. After that, Tseng *et al.* [35] proposed the UP-Growth algorithm which uses a special data structure named UP-tree to discover high-utility itemsets. The candidate itemsets can be generated effectively only from two scans of the database which decreases the execution time greatly. As the first one-phase HUIM algorithm, HUI-Miner [26] uses a novel structure, called utility-list, to store utility information about an itemset. By avoiding the generation of numerous candidates, HUI-Miner generates HUIs directly. Subsequently, various algorithms based on utility-list have been proposed. For instance, the FHM algorithm [9] uses an estimated utility co-occurrence pruning strategy to reduce the number of join operations. Considering the negative unit profits, the FHN algorithm [21] was proposed. Then, Zida *et al.* [40] proposed the EFIM algorithm that uses database projection and transaction merging technique to save the execution time. As a result, EFIM is about two to three orders of magnitude faster than the previous HUIM algorithms. Moreover, another advantage of these algorithms is their low memory consumption. In general, those one-phase algorithms are more efficient than two-phase ones. Other advances about utility mining have been reviewed in several literature [11, 14, 15]

### 2.2 Frequency-based HUIM algorithms

The concepts of discriminative high-utility pattern [22] and high utility-occupancy pattern [7, 16] were introduced previously. Some researchers pay attention to the interesting itemsets which not only have high utility but also have low frequency (high-utility rare itemsets, HURIs). In the process of high-utility rare itemset mining, a large number of algorithms have been proposed. Pillai and Vyas [30] proposed an algorithm to mine HURIs by using the concept of Apriori-inverse. Apriori-inverse is used to generate the rare itemsets firstly and it can be used to find those rare itemsets in HURIs, which are of high utility as well. The Apriori-inverse algorithm for generation of rare itemsets is extended to find HURIs.

Subsequently, the CSHURI algorithm [31] (Customer Segmentation using HURI) was proposed, and is a modified version of HURI. It finds customers who purchase high profitable rare itemsets and accordingly classify the customers based on some criteria. Goyal *et al.* [17] proposed UP-Rare Growth algorithm, which uses the UP-Tree data structure to find the rare itemsets with high utility from the transaction database. The algorithm does not directly find the rare itemsets, it deals with the frequency and utility of the itemset at the same time. Some helpful strategies are proposed to avoid searching the useless branches of the tree. Nowadays, due to the wide application of uncertain data, mining itemsets on uncertain databases has been paid more and more attention. For example, in the process of telecommunication, we adopt the mean of encryption or adding noise to ensure the security of the transmitted data. These processed data are uncertain, and how to discover them correctly and effectively is a hot field of research. Ninoria *et al.* [28] proposed the HURIU algorithm for mining HURIs over uncertain database. This innovative approach uses the concept of Apriori-inverse over uncertain databases and gives a new extension of the HURI algorithm. Then, a new two-phase algorithm [32] was proposed. The first step is to compute transaction utility of each transaction, transaction-weighted utility (twu), and item's utility in turn and apply minimum utility threshold on twu to prune some branches. If an item has less value than this threshold, it will be discarded. The second step is to apply minimum support threshold. The itemset having less value will be extracted as a rare pattern. Finally, the HURIs are obtained from each transaction.

## 3 PRELIMINARIES

In this section, we describe some definitions about high-utility rare itemset mining and briefly introduce the revised new structure.

### 3.1 Basic preliminaries

Let $I = \{i_1, i_2, i_3, \ldots, i_m\}$ be a set of district items. Let $\mathcal{D}$ be a transaction table along with an utility table of each item. Each item has an external utility $p_\ell$, $1 \le \ell \le m$ in the utility table. A subset $X \subseteq I$ is called an itemset, if $X$ contains $k$ distinct items $\{i_1, i_2, i_3, \ldots, i_k\}$, where $i_\ell \in I$, $1 \le \ell \le k$. The transaction database $\mathcal{D} = \{t_1, t_2, t_3, \ldots, t_n\}$, contains a set of $n$ transactions, and each transaction in the database is associated with a unique identifier $t_{id}$. A $k$-itemset is a collection of $k$ items (e.g., $\{bread, milk, jam\}$, $k = 3$). Fraction of transactions that contain an itemset is called *support* (e.g., in a set of 8 transactions, it contains 3 itemsets $\{bread, milk, jam\}$, thus its support is $\frac{3}{8}$). In every transaction, each item $i_\ell$, $1 \le \ell \le m$ has a non-negative quantity $q(i_\ell, t_{id})$, which represents the occurred quantity (e.g., purchased quantity of products) and is known as internal utility of the item $i_\ell$.

*Definition 3.1.* In the utility table, the external utility of item $i$ is denoted as $eu(i)$. The internal utility of item $i$ in the transaction $t_{id}$ is denoted as $iu(i, t_{id})$, which is the count value of item i in the transaction. The real utility of item $i$ in the transaction $t_{id}$ is denoted as $u(i, t_{id})$, where $u(i, t_{id}) = eu(i) \times iu(i, t_{id})$.

*Definition 3.2.* The utility of an itemset $X$ contained in the transaction $t_{id}$, denoted as $u(X, t_{id})$, is defined by the sum of utility of all items of $X$ in $t_{id}$, where $u(X, t_{id}) = \sum_{i_\ell \in X \wedge X \subseteq t_{id}} u(i_\ell,\ t_{id})$. The utility of an itemset $X$ in $\mathcal{D}$, denoted as $u(X)$, is the sum of the utilities of $X$ in all the transactions containing $X$ in $\mathcal{D}$, where $u(X) = \sum_{X \subseteq t_{id} \wedge t_{id} \in \mathcal{D}} u(X,\ t_{id}) = \sum_{X \subseteq t_{id} \wedge t_{id} \in \mathcal{D}} \sum_{i_\ell \in X} u(i_\ell,\ t_{id})$.

**Table 1: Example database**

(a) Original transaction database

| $T_{id}$ | Transaction | Count |
|---|---|---|
| $t_1$ | $\{a, b, d\}$ | $\{1, 3, 2\}$ |
| $t_2$ | $\{c, e, f, g\}$ | $\{3, 1, 2, 4\}$ |
| $t_3$ | $\{a, f, g\}$ | $\{1, 4, 2\}$ |
| $t_4$ | $\{b, c, e\}$ | $\{1, 2, 1\}$ |
| $t_5$ | $\{a, c, e, f, g\}$ | $\{4, 1, 2, 1, 2\}$ |
| $t_6$ | $\{a, e, f\}$ | $\{3, 4, 2\}$ |
| $t_7$ | $\{b, c, d, g\}$ | $\{1, 2, 3, 1\}$ |

(b) Utility table

| *Item* | *a* | *b* | *c* | *d* | *e* | *f* | *g* |
|---|---|---|---|---|---|---|---|
| *Utility* | \$2 | \$1 | \$2 | \$1 | \$5 | \$4 | \$3 |

For example, in Table 1, $eu(e) = \$5$, $iu(e, t_5) = 2$, then $u(e, t_5) = eu(e) \times iu(e, t_5) = \$5 \times 2 = \$10$. $u(\{bd\}, t_1) = u(b, t_1) + u(d, t_1) = 3 \times \$1 + 2 \times \$1 = \$5$, $u(\{bd\}) = u(\{bd\}, t_1) + u(\{bd\}, t_7) = \$5 + \$4 = \$9$, $u(a) = u(a, t_1) + u(a, t_3) + u(a, t_5) + u(a, t_6) = 1 \times \$2 + 1 \times \$2 + 4 \times \$2 + 3 \times \$2 = \$18$, and $u(\{ae\}) = u(\{ae\}, t_5) + u(\{ae\}, t_6) = \$18 + \$26 = \$44$.

*Definition 3.3.* Let *min_util* be a value defined by the user. If the utility of the itemset $X$ in $\mathcal{D}$ is no less than *min_util*, this itemset $X$ is called a high-utility itemset (HUI). Note that *min_util* is called the minimum utility threshold. High-utility itemset mining (HUIM) is to find all the high-utility itemsets in a transaction database $\mathcal{D}$.

*Definition 3.4.* The support of the itemset $X$ in a transaction is denoted as $sup(X, t_{id})$, where $sup(X, t_{id}) = min(_{i_\ell \in X} q(i_\ell, t_{id}))$. The support of the itemset $X$ in the database is denoted as $sup(X)$ which is sum of the support of $X$ in every transaction. If the support of the itemset $X$, that is $sup(X)$, is more than *min_sup* and less than *max_sup*, we define the itemset as rare itemset.

For example, $sup(\{f\}, t_2) = min(q(\{f\}, t_2)) = 2$, $sup(\{fe\}, t_2) = min(q(\{f\}, t_2), q(\{e\}, t_2)) = min(2, 1) = 1$, $sup(\{f\}) = sup(\{f\}, t_2) + sup(\{f\}, t_3) + sup(\{f\}, t_5) + sup(\{f\}, t_6) = 2 + 4 + 1 + 2 = 9$.

Property 1. *If the support value of the itemset $X$ is less than the minimum support threshold min_sup, then the support value of its superset $X'$ is also less than the minimum support threshold min_sup. Therefore, neither $X$ nor $X'$ is a rare itemset we defined and further research is not required.*

Property 1 is also called SUP pruning strategy. For example, we set *min_sup* = 0 and *max_sup* = 4. Because the value of $sup(\{f\})$ is 9 which is more than *max_sup*, the itemset $\{f\}$ is not the rare itemset.

*Definition 3.5.* An itemset $X$ is called a high-utility rare itemset (HURI) if $twu(X)$ is no less than a given user-defined minimum threshold denoted by *min_util* as well as $sup(X)$ is more than *min_sup* and less than *max_sup*. The process of miming all high-utility rare itemsets from a given transaction database $\mathcal{D}$ is called high-utility rare itemset mining (HURIM).

**Problem statement**: Given a transaction database $\mathcal{D}$, the minimum utility threshold *min_util*, the minimum support threshold *min_sup* and the maximum support threshold *max_sup*, the addressed problem of mining with rarity in this paper can be formulated as to discover all high-utility rare itemsets (HURIs) from transaction database $\mathcal{D}$. Mining with rarity makes sense to take advantage of data mining for Web intelligence, with both utility and rarity factors.

For example, we set *min_util* = \$40, *min_sup* = 0, and *max_sup* = 4. The value of $twu(\{f\})$ is \$119 which is no less than *min_util* and the value of $sup(\{f\})$ is 9 which is more than *max_sup*. Therefore, the itemset $\{f\}$ is not the high-utility rare itemset.

From Definition 3.5, it can be seen that traditional HUIM problem has no relationship with support. Besides, the utility constraint does not satisfy the downward closure property in frequent pattern mining. The downward closure property is that if the itemset $X$ is frequent, any of its supersets is also frequent; if itemset $X$ is rare, then its superset is also rare. For example, let the minimum utility threshold *min_util* = \$40, $u(a)$ = \$18 < \$40, and $u(\{ae\})$ = \$44 > \$40. Therefore, this downward closure property is not suitable for utility mining. Previous studies [14, 36] has shown that the utility constraint is neither monotonic nor anti-monotonic.

*Definition 3.6.* The utility of the transaction $t_{id}$, denoted as $tu(t_{id})$, is defined by the sum of the utilities of all the items in $t_{id}$, where $tu(t_{id}) = \sum_{i_\ell \in t_{id}} u(i_\ell, t_{id})$. Similarly, the utility of database $\mathcal{D}$, denoted as $tu(\mathcal{D})$, can be defined by the sum of the utilities of all the transactions in $\mathcal{D}$. The transaction-weighted utility (twu) of itemset $X$ in $\mathcal{D}$ is denoted as $twu(X)$ and it represents the sum of the utilities of $X$ of all the transactions which contain $X$ in $\mathcal{D}$, where $twu(X) = \sum_{t_{id} \in \mathcal{D} \wedge X \subseteq t_{id}} tu(t_{id})$ [27].

For example, in Table 1, $tu(t_1) = u(a, t_1) + u(b, t_1) + u(d, t_1)$ = \$2 + \$3 + \$2 = \$7, and the total utility of $\mathcal{D}$ is \$147. $twu(a) = tu(t_1) + tu(t_3) + tu(t_5) + tu(t_6)$ = \$7 + \$24 + \$30 + \$34 = \$95, and the transaction-weighted utility of all 1-itemsets are shown in Table 2.

**Table 2: Transaction-weighted utility**

(a) The *twu*-lexicographical order

| *Item* | *a* | *b* | *c* | *d* | *e* | *f* | *g* |
|---|---|---|---|---|---|---|---|
| *twu* | \$95 | \$28 | \$82 | \$18 | \$105 | \$119 | \$96 |

(b) The *twu*-descending order

| *Item* | *f* | *e* | *g* | *a* | *c* | *b* | *d* |
|---|---|---|---|---|---|---|---|
| *twu* | \$119 | \$105 | \$96 | \$95 | \$82 | \$28 | \$18 |

Property 2. *The transaction-weighted utility satisfies the downward closure property. It means if $twu(X)$ is less than the min_util which is given by users, all supersets of $X$ are not high utility itemset (HUI).*

If we suppose the *min_util* is equal to \$40, according to Property 2, itemsets $\{b\}$ and $\{d\}$ are not HUIs and all supersets of them are not HUIs. Property 2 is called the *twu* pruning strategy which was first proposed in the Two-Phase algorithm [27] to prune the search space. *EUCS* (Estimated Utility Co-occurrence Structure) [9]

| **Item** | ***a*** | ***b*** | ***c*** | ***d*** | ***e*** | ***f*** |
|---|---|---|---|---|---|---|
| ***b*** | \$7 | | | | | |
| ***c*** | \$30 | \$21 | | | | |
| ***d*** | \$7 | \$18 | \$11 | | | |
| ***e*** | \$64 | \$10 | \$71 | \$0 | | |
| ***f*** | \$88 | \$0 | \$61 | \$0 | \$95 | |
| ***g*** | \$54 | \$11 | \$72 | \$11 | \$61 | \$85 |

**Figure 1: EUCS.**

is defined as a set of triples of the form $(a, b, c) \in I \times I \times R$. A triple $(a, b, c)$ indicates that $twu(\{a, b\}) = c$.

Property 3. *If there is a tuple $(x, y, c)$ in EUCS in which $c <$ min_util, then Pxy and all its supersets are not high-utility itemsets .*

For example, if we suppose *min_util* = \$40, according to Property 3, $twu(\{a, b\})$ = \$7 < \$40. The itemsets $\{ab\}$ is not a high-utility itemset, and all its supersets are low-utility itemsets. The *EUCS* structure related to this database is shown in Figure 1. Property 3 is also called EUCP strategy [9], which can be used to reduce the search space.

*Definition 3.7.* If all the items whose transaction-weighted utilities are less than a given *min_util* are deleted from a transaction and the remaining items are sorted in the *twu*-descending order, we call such transaction as "revised transaction".

After the first scan, the transaction-weighted utilities of all items are calculated. If the *twu* of an item is less than the *min_util*, then this item will be discarded according to the Property 2. Thus, this operation can prune the search space. Then we sort the qualified items in *twu*-descending order. For instance, if we set the *min_util* as \$40, after the first scan of the database in Table 1, items *b* and *d* are not considered in the latter steps. The remaining items are sorted as: $f \prec e \prec g \prec a \prec c$. After scanning the database in Table 1, we can obtain the revised transactions, as shown in Table 3.

**Table 3: Revised database**

| $T_{id}$ | **(Item, Utility)** | ***TU*** |
|---|---|---|
| $t_1$ | $(a, \$2)$ | \$2 |
| $t_2$ | $(f, \$8), (e, \$5), (g, \$12), (c, \$6)$ | \$31 |
| $t_3$ | $(f, \$16), (g, \$6), (a, \$2)$ | \$24 |
| $t_4$ | $(e, \$5), (c, \$4)$ | \$9 |
| $t_5$ | $(f, \$4), (e, \$10), (g, \$6), (a, \$8), (c, \$2)$ | \$30 |
| $t_6$ | $(f, \$8), (e, \$20), (a, \$6)$ | \$34 |
| $t_7$ | $(g, \$3), (c, \$4)$ | \$7 |

*Definition 3.8.* If a given itemset $X$ and a transaction $t_{id}$ have the inclusion relationship $X \subseteq t_{id}$, the remaining items after $X$ in $t_{id}$ is denoted as $t_{id}/X$. The remaining utility of itemset $X$ in the transaction $t_{id}$ is the sum of the utilities of all the items in $t_{id}/X$ in $t_{id}$ which is denoted as $ru(X, t_{id})$, where $ru(X, t_{id}) = \sum_{i_\ell \in t_{id}/X} u(i_\ell,\ t_{id})$. The remaining utility of itemset $X$ in database $\mathcal{D}$ is denoted as $ru(X)$, where $ru(X) = \sum_{X \subseteq t_{id} \in \mathcal{D}} ru(X, t_{id})$

For example, in Table 3, $ru(\{fe\}, t_2)$ = \$12 + \$6 = \$18, $ru(\{fgc\}, t_2)$ = 0. $ru(\{fe\}) = ru(\{fe\}, t_2) + ru(\{fe\}, t_5) + ru(\{fe\}, t_6)$ = \$18 + \$16 + \$6 = \$40.

Property 4. *If given itemsets $X$ and $Y$ have the relationship $\sum_{\forall t_{id} \in \mathcal{D}} u(X, t_{id}) + ru(X, t_{id}) - \sum_{\forall t_{id} \in \mathcal{D}, X \subseteq t_{id} and Y \nsubseteq t_{id}} u(X, t_{id}) + ru(X, t_{id}) < min\_util$, then $\forall Y \subseteq Y'$ and $\forall X \subseteq X', X'Y' \notin HUI$.*

The HUP-Miner algorithm [30] applies this property to prune the search space. Property 4 is also called LA-prune strategy which provides a tighter upper bound on utility for any itemset $X$ that contains $Y$, which is applied when building the utility-list of $XY$.

### 3.2 The utility-list structure

Many two-phase algorithms directly perform on the given database and generate HUIs in two steps which make execution time more longer. However, the HUI-Miner algorithm [26] uses a special data structure named utility-list to store the utility information about itemsets while mining HUIs. In HUI-Miner, each itemset has a utility-list. This algorithm constructs the initial utility-lists by two scans of the database. Each itemset has a utility-list which is consisted of three parts: *tid*, *iutil*, and *rutil*. Part one is *tid* which indicates the transaction containing the itemset. Part two is *iutil* which is the utility of the itemset in $t_{id}$. Part three is the remaining utility of the itemset in $t_{id}$. Without scanning the database again, the 2-itemsets utility-list $\{XY\}$ can be constructed from the intersection of utility-list $\{x\}$ and utility-list $\{y\}$. We can identify the same transactions by comparing the *tids* in the two utility-lists.

Similarly, the utility-list of 3-itemset $\{xyz\}$ can be constructed from the intersection of utility-lists of $\{xy\}$ and $\{xz\}$. There is a little different in calculating the *iutil* of itemset $\{xyz\}$. Let us take itemsets $\{fe\}$ and $\{fg\}$ for an example. To construct the utility-list of $\{feg\}$, we can find itemset $\{feg\}$ appears in $t_2$ and $t_5$ in the database. If we just plus the *iutil* of $\{fe\}$ and the *iutil* of $\{fg\}$ directly in $t_2$, the *iutil* of $\{feg\}$ is \$33 rather than \$29. This is because the sum of the utilities of both $\{fe\}$ and $\{fg\}$ in $t_2$ contain the utility of $\{f\}$ twofold. Therefore, to calculate the *iutil* of $\{i_1, i_2, i_3, \ldots, i_m\}$ in $t_{id}$, we give the formula: $u(\{i_1, i_2, i_3, \ldots, i_m\}, t_{id}) = u(\{i_1, i_2, i_3, \ldots, i_{m-1}\}, t_{id}) + u(\{i_1, i_2, i_3, \ldots, i_{m-1}\}, t_{id}) - u(\{i_1, i_2, i_3, \ldots, i_{m-2}\}, t_{id})$. Given the utility-list of itemset $X$, if the sum value of all the *iutil* and *rutil* in the utility-list is less than *min_util*, any extension associated with $X$ as a prefix is not the high-utility itemset.

## 4 THE HURI-MINER ALGORITHM

### 4.1 The revised utility-list structure

HUI-Miner can discover high-utility itemsets efficiently, but it can not classify them according to a user's preference. For a shop keeper, he or she wants to find high-profit itemsets as well as rare itemsets to make related marketing strategies. To address this limitation w.r.t. main-memory bottleneck, we propose a new revised utility-list structure.

Each itemset has a revised utility-list which contains four parts: *tid*, *iutil*, *rutil* and *sup*. The definitions of the former three parts are same as described above. Part four is the support value of the itemset. Algorithm 1 shows how to construct the revised utility-list of itemset *Pxy*. It takes *P.RUL*, *Px.RUL*, *Py.RUL* as input. Respectively, *P.RUL*, *Px.RUL*, *Py.RUL* are the revised utility-lists of the itemset *P*, *Px*, *Py*. If the prefix *P.RUL* is empty, the revised utility-list of a 2-itemset is constructed. If the prefix *P.RUL* is not empty, the revised utility-list of a $k$-itemset ($k \geq 2$) is constructed. During the construction of the revised utility-list, the calculations of *tid*, *iutil*, and *rutil* are the same as above. For the newly added *sup*, according to the above definitions, take the minimum of the support values of the two itemsets as the value of *sup* in the newly built utility-list.

Take itemset $\{f\}$ and itemset $\{e\}$ for an example. To construct the revised utility-list of $\{fe\}$, it can find that the itemset $\{f\}$ appears in $t_2$, $t_3$, $t_5$, and $t_6$ of the database, and the itemset $\{e\}$ appears in $t_2$, $t_4$, $t_5$, and $t_6$ of the database. More importantly, the two revised utility-lists of itemset $\{f\}$ and itemset $\{e\}$ have the same transaction $t_2$, $t_5$, and $t_6$. Thus, we can plus the value of *iutil* in the two revised utility-lists of itemset $\{f\}$ and itemset $\{e\}$ in transaction $t_2$, $t_5$, and $t_6$ as the value of *iutil* of the revised utility-list of $\{fe\}$. At this time, the value of *iutil* in transaction $t_2$, $t_5$, and $t_6$ is \$13, \$14, and \$28, respectively. Take *rutil* in the revised utility-list of $\{e\}$ as *rutil* in the revised utility-list of $\{fe\}$. Then, the value of *iutil* in transaction $t_2$, $t_5$, and $t_6$ is \$18, \$16, and \$6, respectively. Take the minimum value of *sup* of each transaction in itemset $\{f\}$ and itemset $\{e\}$ as the value of *sup* of the revised utility-list of $\{fe\}$. In this case, the value of *sup* in $t_2$, $t_5$, and $t_6$ is \$1, \$1, and \$2, respectively.

**Algorithm 1:** The *Construct* procedure

**Input:** *P.RUL*; *Px.RUL*; *Py.RUL*.
**Output:** *Pxy.RUL*, the revised utility-list of itemset *Pxy*.

```
1  Pxy.RUL = null;
2  set Utility = SUM(X.iutil) + SUM(X.rutil);
3  set Sup = SUM(X.sup);
4  for each element Ex ∈ Px.RUL do
5      if ∃Ey ∈ Py.RUL and Ex.tid == Ey.tid then
6          if P.RUL is not empty then
7              search such element E ∈ P.RUL that
8              Ex.tid == Ey.tid;
9              Exy = <Ex.tid, Ex.iutil + Ey.iutil - E.iutil, Ey.rutil,
                 min(Ex.sup, Ey.sup)>;
10         end
11         if P.RUL is empty then
12             Exy = <Ex.tid, Ex.iutil + Ey.iutil, Ey.rutil,
                 min(Ex.sup, Ey, sup)>;
13         end
14         append Exy to Pxy.RUL;
15     end
16     else
17         Utility = Utility - Ex.iutil - Ex.rutil;
18         Sup = Sup - Ex.sup;
19         if Utility < min_util or Sup < min_sup then
20             return null;
21         end
22     end
23 end
24 return Pxy.RUL
```

In Lines 16–22 of the Algorithm 1, it uses the LA-prune strategy to prune the itemsets that do not meet the conditions. In addition to the utility value of the itemset for the screening conditions, it also

adds the support value. The new revised utility-lists of 1-itemsets and 2-itemsets are given in Table 4 and Table 5, respectively.

**Table 4: Revised utility-lists of 1-itemsets**

$\{f\}$

| *tid* | *iutil* | *rutil* | *sup* |
|---|---|---|---|
| $t_2$ | \$8 | \$23 | \$2 |
| $t_3$ | \$16 | \$8 | \$4 |
| $t_5$ | \$4 | \$26 | \$1 |
| $t_6$ | \$8 | \$26 | \$2 |

$\{e\}$

| *tid* | *iutil* | *rutil* | *sup* |
|---|---|---|---|
| $t_2$ | \$5 | \$18 | \$1 |
| $t_4$ | \$5 | \$4 | \$1 |
| $t_5$ | \$10 | \$16 | \$2 |
| $t_6$ | \$20 | \$6 | \$4 |

$\{g\}$

| *tid* | *iutil* | *rutil* | *sup* |
|---|---|---|---|
| $t_2$ | \$12 | \$6 | \$4 |
| $t_3$ | \$6 | \$2 | \$2 |
| $t_5$ | \$6 | \$10 | \$2 |
| $t_7$ | \$3 | \$4 | \$1 |

$\{a\}$

| *tid* | *iutil* | *rutil* | *sup* |
|---|---|---|---|
| $t_1$ | \$2 | 0 | \$1 |
| $t_3$ | \$2 | 0 | \$1 |
| $t_5$ | \$8 | \$2 | \$4 |
| $t_6$ | \$6 | 0 | \$3 |

$\{c\}$

| *tid* | *iutil* | *rutil* | *sup* |
|---|---|---|---|
| $t_2$ | \$6 | 0 | \$3 |
| $t_4$ | \$4 | 0 | \$2 |
| $t_5$ | \$2 | 0 | \$1 |
| $t_7$ | \$4 | 0 | \$2 |

**Table 5: Revised utility-lists of 2-itemsets**

$\{fe\}$

| *tid* | *iutil* | *rutil* | *sup* |
|---|---|---|---|
| $t_2$ | \$13 | \$18 | \$1 |
| $t_5$ | \$14 | \$16 | \$1 |
| $t_6$ | \$28 | \$6 | \$2 |

$\{fg\}$

| *tid* | *iutil* | *rutil* | *sup* |
|---|---|---|---|
| $t_2$ | \$20 | \$6 | \$2 |
| $t_3$ | \$22 | \$2 | \$2 |
| $t_5$ | \$10 | \$10 | \$1 |

$\{fa\}$

| *tid* | *iutil* | *rutil* | *sup* |
|---|---|---|---|
| $t_3$ | \$18 | 0 | \$1 |
| $t_5$ | \$12 | \$2 | \$1 |
| $t_6$ | \$14 | 0 | \$2 |

$\{fc\}$

| *tid* | *iutil* | *rutil* | *sup* |
|---|---|---|---|
| $t_2$ | \$14 | 0 | \$2 |
| $t_5$ | \$6 | 0 | \$1 |

## 4.2 HURI-Miner with pruning strategies

After scanning the database $\mathcal{D}$ at the first time, we can get the TWU values of all items. If the utility of an item is less than *min_util*, it does not consider it in the next steps. Then, it sorts the remaining items according to their transaction-weighted utilities. After that, initial revised utility-lists that is *RULs* can be constructed during the scanning the database $\mathcal{D}$ at the second time. The HURI-Miner algorithm takes *P.RUL* which is the revised utility-list of itemset *P* and is initially empty, *RULs* which is the revised utility-lists of all *P*'s 1-extensions, a minimum utility threshold *min_util*, a minimum support threshold *min_sup*, a maximum support threshold *max_sup* as input and finally outputs all the HURIs with *P* as prefix. The steps of the algorithm are shown in Algorithm 2. For each revised utility-list *X* in *RULs*, if the sum value of *X.iutil* is no less than *min_util* and the sum value of *X.sup* is no less than *min_sup*, it intersects *X* and each revised utility-list *Y* after *X* in *RULs* to find and output the itemset whose support is no less than *min_sup* and no more than *max_sup*. According to aforementioned properties, only when the sum of all the *iutil* and *rutil* in *X* is no less than *min_util* and the sum value of *X.sup* is no less than *min_sup* should it be processed further. Finally, the revised utility-lists of all the 1-extensions of itemset *Px* is recursively processed.

**Algorithm 2:** The HURI-Miner algorithm

**Input:** *P.RUL*, *RULs*, *min_util*, m*in_sup*, *max_sup*.
**Output:** all the HURIs with *P* as prefix.

```
1  for each revised utility-list X in RULs do
2      if SUM(X.iutil) ≥ min_util and SUM(X.sup) ≥ min_sup then
3          if SUM(X.sup) ≤ max_sup then
4              output the extension of X;
5          end
6      end
7      if (SUM(X.iutil) + SUM(X.rutil)) ≥ min_util and SUM(X.sup)
         ≥ min_sup then
8          exULs = null;
9          for each revised utility-list Y after X in RULs do
10             if ∃twu({x, y}) ∈ EUCS and twu({x, y}) ≥
                 min_util then
11                 exULs = exULs + Construct (P.RUL, X, Y);
12             end
13         end
14         call HURI-Miner (X, exULs, min_util, min_sup,
             max_sup);
15     end
16 end
```

For many pattern discovering algorithms, main memory is the critical resource. It leads to many memory-consuming algorithms. When scanning the processed database, it needs to count something, e.g., occurrences and utilities of patterns. Therefore, how to solve main-memory bottleneck is critical. To prune the search space, HURI-Miner applies the following four strategies:

***The TWU pruning strategy***. If *twu(X)* is less than the minimum utility threshold *min_util*, any extension with *X* as prefix is not processed further and the algorithm proceeds with the next item. Otherwise, the itemset *X* is processed further.

***The SUP pruning strategy***. If the sum value of *X.sup* in revised utility-list of itemset *X* is less than the minimum support threshold *min_sup*, then itemset *X* is not the rare itemset and will not be processed further.

***The EUCP strategy***. The low-utility itemset *Pxy* and all its transitive extensions are directly eliminated in the EUCS structure without constructing their utility-list. The pruning condition is that if there is a tuple $(x, y, c)$ in which $c < min\_util$ in EUCS, then *Pxy* and all its supersets are the low-utility itemsets, which do not need to be explored.

***The LA-prune strategy.*** If the value of the total utility of $Px$ minusing the value of $Px.iutil$ and $Px.rutil$ in the transaction is less than $min\_util$, the itemset $Px$ is pruned and will not be processed in the algorithm. Similarly, if the value of $Px.sup$ minusing $Px.sup$ in the transaction is less than $min\_sup$, the itemset $Px$ is not a rare itemset and does not continue to be processed by the algorithm.

## 5 PERFORMANCE EVALUATION

To the best of our knowledge, few studies for interesting pattern discovery consider both utility and rarity factors in rich data. In order to evaluate the performance of the HURI-Miner algorithm, we conducted a large number of experiments on different datasets and compared HURI-Miner with UP-Rare Growth [17], which is the most advanced algorithm for mining high-utility rare itemsets. The experiments were conducted on a personal computer with an i5-7200U core, 2.50GHz GPU, 5GB of free RAM, and a 64-bit Windows 10 operating system. All the algorithms in the experiments are implemented in Java language. In the following subsections, experimental results are discussed and analyzed in detail.

### 5.1 Datasets

Experiments are conducted on two real-life datasets and two synthetic datasets which are shown in Table 6. Characteristics of these datasets include: 1) the name of the dataset, 2) the number of transactions, 3) the number of items, 4) the average length (AvgLen) of transactions, and 5) the maximum length (MaxLen) of transactions. Foodmart is a real-life dataset, which is derived from an anonymous chain store, and provides 21,556 transactions and 1560 items. Retail is also a real-life dataset consisting of transactions from an anonymous retail store in Belgium, containing 90,000 transactions approximately. T10I4D100K and T5I2N2KD100K are synthetic datasets generated by the benchmark IBM data generator and are often used to evaluate the scalability of algorithms. At this point, the external utility of each item is randomly generated between 1 and 1000 using log-normal distribution, and the internal utility of each item is also randomly generated within a range of 1 to 5.

**Table 6: Characteristics of different datasets**

| Dataset | #Trans | #Items | AvgLen | MaxLen |
|---|---|---|---|---|
| Foodmart | 21,556 | 1,560 | 10.3 | 76 |
| Retail | 88,162 | 16,470 | 4 | 11 |
| T5I2N2KD100K | 100,000 | 2,000 | 7.5 | 15 |
| T10I4D100K | 100,000 | 1,000 | 10.1 | 29 |

### 5.2 Runtime analysis

Under different minimum utility thresholds ($min\_util$) and support thresholds ($min\_sup$ and $max\_sup$), the running time of our proposed HURI-Miner algorithm on four datasets Foodmart, Retail, T5I2N2KD100K and T10I4D100K is compared with that of the most advanced UP-Rare Growth algorithm. The running time is recorded by the TIME command, which includes input time, CPU time, and output time.

Firstly, we fix two support thresholds. In the HURI-Miner algorithm, the minimum support threshold $min\_sup$ is 3, the maximum

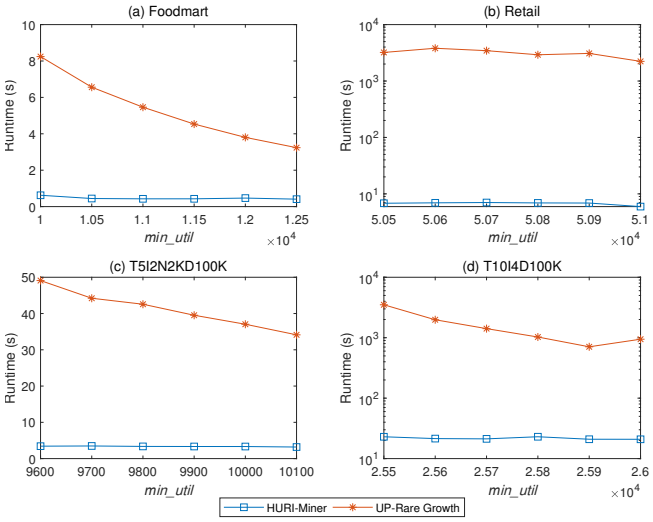


**Figure 2: Runtime w.r.t *min_util*.**

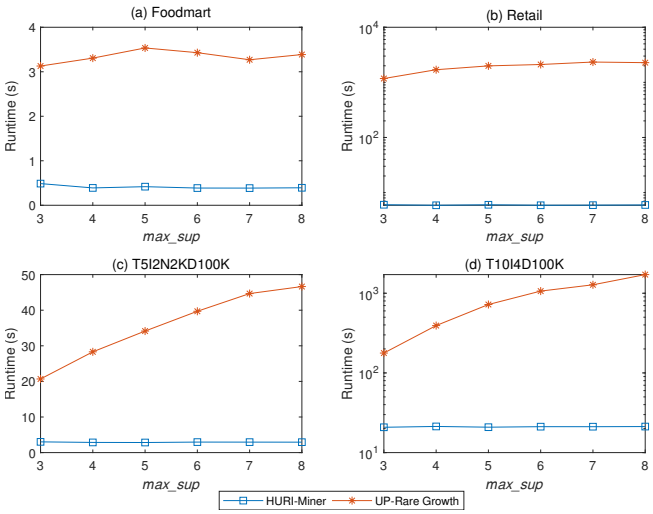


**Figure 3: Runtime w.r.t *max_sup*.**

support threshold $max\_sup$ is 5. In the UP-Rare Growth algorithm, the maximum support threshold $max\_sup$ is 5. The running time of the two algorithms on the four datasets is studied under different minimum utility thresholds $min\_util$ and the experimental results are shown in Figure 2. It can be clearly seen that for the four datasets, with the increase of the minimum utility threshold $min\_util$, the running time of UP-Rare Growth is gradually shortened, while the running time of HURI-Miner is relatively stable and does not change a lot. However, for each stage of each dataset, the proposed HURI-Miner takes much less time than the UP-Rare Growth algorithm to generate the HURIs. This is because UP-Rare Growth generates HURIs in two stages and candidate itemsets are

generated in the process of generating HURIs. HURI-Miner can directly generate the HURIs without generating candidate itemsets.

Then, we fix the minimum utility threshold *min_util* and minimum support threshold *min_sup* and study the running time of the two algorithms on four datasets under different maximum support thresholds *max_sup*. The experimental results are shown in Figure 3. It can be clearly seen that for the four datasets, with the increase of the maximum support threshold *max_sup*, the running time of UP-Rare Growth gradually increases, while the running time of HURI-Miner is relatively stable. However, for each stage of each dataset, the proposed HURI-Miner algorithm takes less time than UP-Rare Growth to generate the HURIs. This is because HURI-Miner adopts EUCP, SUP, and LA-prune pruning strategies. For example, according to the SUP strategy, some itemsets are pruned through the minimum support threshold *max_sup*, which shortens the running time of the algorithm to some extent. At the same time, the *min_sup* is added to make the algorithm more flexible.

## 5.3 Pattern analysis

We evaluate the number of the HURIs generated by the two compared algorithms. The HURI-Miner algorithm's support for an item in a transaction is defined as the number of the item appearing in the transaction, while the UP-Rare Growth algorithm's support for an item in a transaction is defined as 1, namely the number of the transaction. Since HURI-Miner is different from UP-Rare Growth in the process of processing the support of itemsets, the number of HURIs generated by the two algorithms under the same threshold is also different. The experimental results of each dataset under different *min_util* and fixed support thresholds and different *max_sup* and fixed utility thresholds are shown in Figure 4 and Figure 5, respectively.

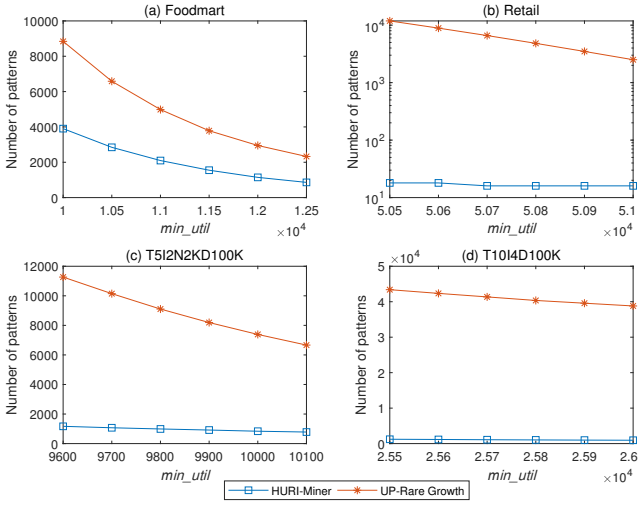


**Figure 4: Patterns analysis w.r.t *min_util*.**

For example, for the Foodmart dataset in Figure 4, we first fix the support threshold. In the HURI-Miner algorithm, we set *min_sup* = 3, *max_sup* = 5. In the UP-Rare Growth algorithm, we set *max_sup* = 5. It is found that with the increase of the minimum utility threshold *min_util*, the number of HURIs generated by both algorithms decreases. However, for Retail, T5I2N2KD100K and T10I4D100K, with the increase of *min_util*, the number of HURIs generated by HURI-Miner is stable.

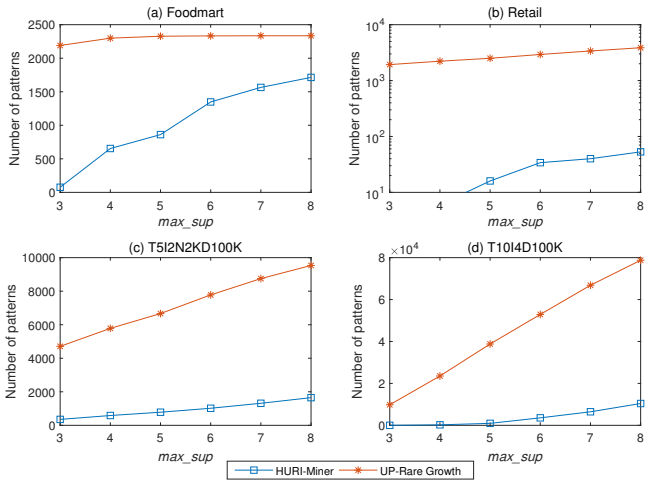


**Figure 5: Patterns analysis w.r.t *max_sup*.**

For the Foodmart dataset in Figure 5, we fix the minimum utility threshold *min_util*. In the HURI-Miner algorithm and the UP-Rare Growth algorithm, we set *min_util* = 12500. It is found that with the increase of the maximum support threshold *max_sup*, the number of HURIs generated by both algorithms increases. The situation is also the same for the other three datasets.

## 6 CONCLUSION

In this paper, we focus on the efficiency of mining high-utility rare itemsets (HURIs) and first propose an algorithm which takes both utility and frequency into account and produces HURIs directly. Former algorithms produce HURIs in two steps. However, the execution time of these algorithms is long and these algorithms are not efficient. Therefore, we propose the fast HURI-Miner algorithm using the revised utility-list structure to avoid the generation of candidates and produce HURIs directly. We also use four pruning strategies to prune unpromising itemsets early during scanning of the database to reduce the search space and improve the temporal and spatial performance of the proposed algorithm. At the same time, extensive experiments show our algorithm outperforms the state-of-the-art algorithm to discover HURIs.

## 7 ACKNOWLEDGMENT

This research was supported in part by the National Natural Science Foundation of China (Grant Nos. 62002136 and 61932011), Guangzhou Basic and Applied Basic Research Foundation (Grant No. 202102020277), Guangdong Basic and Applied Basic Research Foundation (Grant No. 2019B1515120010), Guangdong Key R&D Plan2020 (Grant No. 2020B0101090002), and National Key R&D Plan2020 (Grant No. 2020YFB1005600).

# REFERENCES


[1] Rakesh Agarwal, Ramakrishnan Srikant, et al. 1994. Fast algorithms for mining association rules. In *The 20th VLDB Conference*. 487–499.

[2] Charu C Aggarwal, Yan Li, Jianyong Wang, and Jing Wang. 2009. Frequent pattern mining with uncertain data. In *The 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 29–38.

[3] Chowdhury Farhan Ahmed, Syed Khairuzzaman Tanbeer, Byeong-Soo Jeong, and Young-Koo Lee. 2009. Efficient tree structures for high utility pattern mining in incremental databases. *IEEE Transactions on Knowledge and Data Engineering* 21, 12 (2009), 1708–1721.

[4] Luigi Atzori, Antonio Iera, and Giacomo Morabito. 2010. The internet of things: A survey. *Computer Networks* 54, 15 (2010), 2787–2805.

[5] Anindita Borah and Bhabesh Nath. 2019. Rare pattern mining: challenges and future perspectives. *Complex & Intelligent Systems* 5, 1 (2019), 1–23.

[6] Raymond Chan, Qiang Yang, and Yi-Dong Shen. 2003. Mining high utility itemsets. In *Third IEEE International Conference on Data Mining*. IEEE Computer Society, 19–19.

[7] Chien-Ming Chen, Lili Chen, Wensheng Gan, Lina Qiu, and Weiping Ding. 2021. Discovering high utility-occupancy patterns from uncertain data. *Information Sciences* 546 (2021), 1208–1229.

[8] Hsinchun Chen, Roger HL Chiang, and Veda C Storey. 2012. Business intelligence and analytics: From big data to big impact. *MIS Quarterly* (2012), 1165–1188.

[9] Philippe Fournier-Viger, Cheng-Wei Wu, Souleymane Zida, and Vincent S Tseng. 2014. FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning. In *International Symposium on Methodologies for Intelligent Systems*. Springer, 83–92.

[10] Wensheng Gan, Jerry Chun-Wei Lin, Han-Chieh Chao, Hamido Fujita, and Philip S Yu. 2019. Correlated utility-based pattern mining. *Information Sciences* 504 (2019), 470–486.

[11] Wensheng Gan, Jerry Chun-Wei Lin, Han-Chieh Chao, Shyue-Liang Wang, and Philip S Yu. 2018. Privacy preserving utility mining: a survey. In *IEEE International Conference on Big Data*. IEEE, 2617–2626.

[12] Wensheng Gan, Jerry Chun-Wei Lin, Han-Chieh Chao, and Justin Zhan. 2017. Data mining in distributed environment: a survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 7, 6 (2017), e1216.

[13] Wensheng Gan, Jerry Chun-Wei Lin, Philippe Fournier-Viger, Han-Chieh Chao, and Hamido Fujita. 2018. Extracting non-redundant correlated purchase behaviors by utility measure. *Knowledge-Based Systems* 143 (2018), 30–41.

[14] Wensheng Gan, Jerry Chun-Wei Lin, Philippe Fournier-Viger, Han-Chieh Chao, Tzung-Pei Hong, and Hamido Fujita. 2018. A survey of incremental high-utility itemset mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8, 2 (2018), e1242.

[15] Wensheng Gan, Jerry Chun-Wei Lin, Philippe Fournier-Viger, Han-Chieh Chao, Vincent S Tseng, and Philip S Yu. 2021. A survey of utility-oriented pattern mining. *IEEE Transactions on Knowledge and Data Engineering* 33, 4 (2021), 1306–1327.

[16] Wensheng Gan, Jerry Chun-Wei Lin, Philippe Fournier-Viger, Han-Chieh Chao, and Philip S Yu. 2020. HUOPM: High-utility occupancy pattern mining. *IEEE Transactions on Cybernetics* 50, 3 (2020), 1195–1208.

[17] Vikram Goyal, Siddharth Dawar, and Ashish Sureka. 2015. High utility rare itemset mining over transaction databases. In *International Workshop on Databases in Networked Information Systems*. Springer, 27–40.

[18] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. 2007. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery* 15, 1 (2007), 55–86.

[19] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. 2004. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery* 8, 1 (2004), 53–87.

[20] Yun Sing Koh and Sri Devi Ravana. 2016. Unsupervised rare pattern mining: a survey. *ACM Transactions on Knowledge Discovery from Data* 10, 4 (2016), 1–29.

[21] Jerry Chun-Wei Lin, Philippe Fournier-Viger, and Wensheng Gan. 2016. FHN: An efficient algorithm for mining high-utility itemsets with negative unit profits. *Knowledge-Based Systems* 111 (2016), 283–298.

[22] Jerry Chun-Wei Lin, Wensheng Gan, Philippe Fournier-Viger, Tzung-Pei Hong, and Han-Chieh Chao. 2017. FDHUP: Fast algorithm for mining discriminative high utility patterns. *Knowledge and Information Systems* 51, 3 (2017), 873–909.

[23] Jerry Chun-Wei Lin, Wensheng Gan, Philippe Fournier-Viger, Tzung-Pei Hong, and Vincent S Tseng. 2016. Efficient algorithms for mining high-utility itemsets in uncertain databases. *Knowledge-Based Systems* 96 (2016), 171–187.

[24] Jerry Chun-Wei Lin, Wensheng Gan, Philippe Fournier-Viger, Tzung-Pei Hong, and Vincent S Tseng. 2017. Efficiently mining uncertain high-utility itemsets. *Soft Computing* 21, 11 (2017), 2801–2820.

[25] Jerry Chun-Wei Lin, Wensheng Gan, Tzung-Pei Hong, and Vincent S Tseng. 2015. Efficient algorithms for mining up-to-date high-utility patterns. *Advanced Engineering Informatics* 29, 3 (2015), 648–661.

[26] Mengchi Liu and Junfeng Qu. 2012. Mining high utility itemsets without candidate generation. In *The 21st ACM International Conference on Information and Knowledge Management*. 55–64.

[27] Ying Liu, Wei-Keng Liao, and Alok Choudhary. 2005. A fast high utility itemsets mining algorithm. In *Proceedings of the 1st international workshop on Utility-based data mining*. 90–99.

[28] S Zanzote Ninoria and SS Thakur. 2019. An efficient algorithm for mining high utility rare itemsets over uncertain databases. *International Journal of Computer Engineering and Technology* 10, 2 (2019).

[29] Daniel E O'Leary. 2013. Artificial intelligence and big data. *IEEE Intelligent Systems* 28, 2 (2013), 96–99.

[30] Jyothi Pillai and OP Vyas. 2011. High utility rare item set mining (HURI): an approach for extracting high utility rare item sets. *Journal on Future Engineering and Technology* 7, 1 (2011), 1.

[31] Jyothi Pillai and OP Vyas. 2012. CSHURI-modified HURI algorithm for customer segmentation and transaction profitability. *arXiv preprint, arXiv:1205.1609* (2012).

[32] Sunidhi Shrivastava and Punit Kumar Johari. 2017. Privacy preservation of infrequent itemsets mining using GA approach. In *Recent Developments in Intelligent Computing, Communication and Devices*. Springer, 97–104.

[33] Wei Song, Caiyu Fang, and Wensheng Gan. 2021. TopUMS: Top-$k$ utility mining in stream data. In *International Conference on Data Mining Workshops*. IEEE, 615–622.

[34] Laszlo Szathmary, Amedeo Napoli, and Petko Valtchev. 2007. Towards rare itemset mining. In *19th IEEE International Conference on Tools with Artificial Intelligence*. IEEE, 305–312.

[35] Vincent S Tseng, Bai-En Shie, Cheng-Wei Wu, and Philip S Yu. 2012. Efficient algorithms for mining high utility itemsets from transactional databases. *IEEE Transactions on Knowledge and Data Engineering* 25, 8 (2012), 1772–1786.

[36] Hong Yao, Howard J Hamilton, and Cory J Butz. 2004. A foundational approach to mining itemset utilities from databases. In *The SIAM International Conference on Data Mining*. SIAM, 482–486.

[37] Deze Zeng, Song Guo, and Zixue Cheng. 2011. The web of things: A survey. *Journal of Communication* 6, 6 (2011), 424–438.

[38] Chunkai Zhang, Zilin Du, Wensheng Gan, and Philip S Yu. 2021. TKUS: Mining top-$k$ high utility sequential patterns. *Information Sciences* 570 (2021), 342–359.

[39] Chunkai Zhang, Zilin Du, Yuting Yang, Wensheng Gan, and Philip S Yu. 2021. On-shelf utility mining of sequence data. *ACM Transactions on Knowledge Discovery from Data* 16, 2 (2021), 1–31.

[40] Souleymane Zida, Philippe Fournier-Viger, Jerry Chun-Wei Lin, Cheng-Wei Wu, and Vincent S Tseng. 2017. EFIM: a fast and memory efficient algorithm for high-utility itemset mining. *Knowledge and Information Systems* 51, 2 (2017), 595–625.