Nowadays, the environments of smart systems for Industry 4.0 and Internet of Things are experiencing fast industrial upgrading. Big data technologies such as design making, event detection, and classification are developed to help manufacturing organizations to achieve smart systems. By applying data analysis, the potential values of rich data can be maximized, which will help manufacturing organizations to finish another round of upgrading. In this article, we propose two new algorithms with respect to big data analysis, namely UFC\(_{\text{gen}}\) and UFC\(_{\text{fast}}\). Both algorithms are designed to collect three types of patterns to help people determine the market positions for different product combinations. We compare these algorithms on various types of datasets, both real and synthetic. The experimental results show that both algorithms can successfully achieve pattern classification by utilizing three different types of interesting patterns from all candidate patterns based on user-specified thresholds of utility and frequency. Furthermore, the list-based UFC\(_{\text{fast}}\) algorithm outperforms the levelwise-based UFC\(_{\text{gen}}\) algorithm in terms of both execution time and memory consumption.

CCS Concepts: • Information Systems → Data mining; • Applied computing → Business intelligence;

Additional Key Words and Phrases: Big data, smart manufacturing, pattern classification, utility measure

ACM Reference format:
Qi Lin, Wensheng Gan, Yongdong Wu, Jiahui Chen, and Chien-Ming Chen. 2022. Smart System: Joint Utility and Frequency for Pattern Classification. ACM Trans. Manage. Inf. Syst. 13, 4, Article 43 (December 2022), 24 pages.
https://doi.org/10.1145/3531480

Qi Lin and Wensheng Gan contribute equally to the paper.
This research was supported in part by the National Natural Science Foundation of China (Nos. 62002136, 62272196, and 61902079), Guangzhou Basic and Applied Basic Research Foundation (Nos. 202102020277 and 202102020928), Guangdong Basic and Applied Basic Research Foundation (Nos. 2022A1515011861 and 2019B1515120010), Guangdong Key R&D Plan2020 (No. 2020B0101090002), and National Key R&D Plan2020 (No. 2020YFB1005600).
Authors’ addresses: Q. Lin, W. Gan and Y. Wu, Jinan University, Guangzhou City, Guangdong Province, 510632, China; emails: qlin0910@gmail.com, wsgan001@gmail.com, wuyd007@qq.com; J. Chen, Guangdong University of Technology, Guangzhou City, Guangdong Province, 510006, China; email: csjhchen@gmail.com; C.-M. Chen (corresponding author), Shandong University of Science and Technology, Qingdao City, Shandong Province, 266510, China; email: chienmingchen@ieee.org.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2022 Association for Computing Machinery.
2158-656X/2022/12-ART43 $15.00
https://doi.org/10.1145/3531480

ACM Transactions on Management Information Systems, Vol. 13, No. 4, Article 43. Publication date: December 2022.
1 INTRODUCTION

In today’s manufacturing environment, massive raw data are collected from shops and online e-commerce platforms or through sensors and electronic equipment. These data can be related to user behaviors, transaction records, the characteristics of products, production lines, and so on. The size of the data is increasing in the manufacturing industries [11, 12], and this has led to a very important issue: how to discover the potential knowledge from the high-volume databases. During the past few years, big data analysis has attached great importance to different fields, especially smart manufacturing. In big data analysis, knowledge discovery in databases (KDD) [10] aims to extract useful patterns from data, and data mining [10, 18] is the key step in KDD, since it provides the algorithm or model to efficiently discover the useful patterns and knowledge.

Association rule mining (ARM) [4, 27] and pattern classification are two of the most attractive and studied fields in data mining. ARM has been very popular in the past two decades. Scientists have intended to determine the correlation and association between various items, thereby predicting future trends or developing feasible strategies. To describe the frequency of a pattern and the reliability of a rule, the concepts of support and confidence are adopted to record such information in a database. For ARM, the first step is to filter data, a common and traditional method is to formulate rules by utilizing algorithms in high-frequency itemset mining (HFIM) [26, 27, 35]. During this step, high-frequency itemsets (HFIs) are discovered, and the other itemsets that do not meet the minimum frequency threshold are eliminated. The second step is to determine the hidden rules or patterns in HFIM and choose the rules with higher confidence. Finally, these interesting rules are applied in the real world to help design appropriate strategies for different tasks. In the past, a number of ARM approaches have been proposed, such as Apriori [25], FP-growth [26], H-Mine [39], and Eclat [48].

Based on ARM, pattern classification and classification based on associations [2, 38, 42, 43] have been studied to label items or build classifiers to predict future trends. It makes use of the rule discovery process in ARM by extracting efficient rules that can precisely generalize the training databases. Nowadays, pattern classification has been applied in various world applications such as commercial prediction, finance analysis, and phishing website detection [1]. For example, algorithms of pattern classification extract classifiers containing association rules of high confidence to put different labels on websites, thereby identifying which websites can be viewed as phishing websites. As ARM normally adopts frequency as the sole standard to extract valuable knowledge, every item is assigned to the same utility (value), and therefore frequency becomes the only precondition to obtain the association rules. It is noticeable that such a strategy ignores the variety of importance among items, and strategies developed to address this may not improve the sales of some items. To overcome this drawback, utility-driven pattern mining [21] has been proposed.

In smart manufacturing, high-utility patterns are always welcomed by merchants, because they simply create higher profits. High-utility pattern mining (HUPM) [8, 29, 34] assigns different utilities to items and considers both occurred quantity (internal utility) and utility (external utility). In recent years, a series of alternative approaches have been proposed, including IHUP [5], HUI-Miner [33], UMEpi [17], UP-growth [45], TKU [44], and so on.

In the actual process of manufacturing, both frequency and utility are the essential indicators for determining the position of a product combination. On the one hand, in HFIM, high-frequency itemsets are discovered, and they are often related to the ARM. A number of association rules are conducted from these frequent itemsets. However, if a majority of these itemsets have low utility, then it might be a waste of resource unless these itemsets have a high frequency so that new strategies can be applied. On the other hand, we do not expect similar situation in HUPM. In manufacturing, if many itemsets with high utility being discovered are quite infrequent, then the
discovered rules may be less valuable. It is clear that one of the major challenges in pattern mining for manufacturing is how to better locate each pattern’s position and design more specific and valuable strategies for bundling sale. Therefore, if frequency and utility can be taken into account in big data analysis, then we can better evaluate each combination of different products and then obtain a more accurate position of each combination and thus develop more effective and specific bundling sale strategies. In this study, we introduce a new model for pattern classification. We formulate this model as joint utility and frequency for pattern classification and also develop two effective algorithms.

Based on the properties of frequency and utility, there are four types of patterns in pattern classification, including (i) High Frequency and High Utility Itemset (HFHUI), (ii) High Frequency and Low Utility Itemset (HFLUI), (iii) Low Frequency and High Utility Itemset (LFHUI), and (iv) Low Frequency and Low Utility Itemset (LFLUI). Since LFLUI is meaningless and numerous, our proposed model will focus on the interesting three types, i.e., HFHUI, HFLUI, and LFHUI. The key contributions of this article are summarized as follows:

- In data analysis, only a few studies consider both frequency and utility as two common standards to discover interesting patterns. In this study, two algorithms, namely UFC\textsubscript{gen} and UFC\textsubscript{fast}, have been developed to help better locate the market locations for different product combinations in real-world applications, such as smart manufacturing. Both algorithms can efficiently collect three different types of patterns.
- Based on utility and frequency, several powerful strategies have been developed to prune the search space. Different strategies are applied in UFC\textsubscript{gen} and UFC\textsubscript{fast} to ensure the reliability of pattern classification.
- A special novel structure named frequency-utility-list (FU-list) has been developed in the efficient UFC\textsubscript{fast} algorithm. Such a structure stores information about transaction identifier, frequency, and remaining utility of patterns.
- Extensive experiments on real and synthetic datasets show that the UFC\textsubscript{gen} and UFC\textsubscript{fast} algorithms can be applied to various sizes of datasets, and UFC\textsubscript{fast} has remarkable performance over UFC\textsubscript{gen}.

Note that some key concepts and an initial algorithm were presented in a preliminary version [32] of this article. The remainder of this extended version is organized as follows. The relevant literature and necessary background of association rule mining, associative classification, and utility-driven pattern mining are respectively reviewed in Section 2. Some descriptions of associative classification and the statement of the studied problem are provided and discussed in Section 3. The detailed process of level-based algorithm are presented in Section 4. Then, in Section 5, we provide a supplementary algorithm that has better performance in solving the pattern classification problem. Extensive experimental results are presented in Section 6. Finally, Section 7 concludes the article and highlights several future studies.

2 RELATED WORK

In this section, we review the related works that have been studied in big data analysis, including data mining in manufacturing, utility-driven pattern mining, and pattern classification.

2.1 Data Mining in Manufacturing

The general goal of data mining is to find the potential relationship between items and hidden patterns and to predict the future trend. In smart manufacturing, the algorithm for data mining can help diagnose fault information, discover uncovering relationship among products, and design complex strategies. With the extensive use of machines and a large production of items, data
mining in manufacturing usually plays an increasingly important role. In the past two decades, a number of approaches have been developed and applied in the real process of manufacturing. For instance, Fan et al. [13] proposed an algorithm that can automatically discover knowledge from production databases for fault diagnosis, and finally it can optimize device performance with a given target. In 2017, Nakata et al. [36] proposed a system that can identify the cause of failure from wafer failure map patterns and manufacturing histories. Then, by applying big data analysis at different stages, the system can help engineers to support their work.

ARM [3, 7] was first proposed to analyze basket data and is nowadays widely applied in different aspects. In smart manufacturing industries, ARM can be used to find the association from products and induce a number of rules that can help predict user behavior. In general, there are two main steps in association rule mining. The first step is to discover frequent itemsets by calculating the occurrence of each possible combination in the datasets. And in the next step, it simply generates and selects rules from the itemset by acknowledging the confidence of each rule. In the past, approaches such as Apriori [4] and FP-growth [26] were applied to obtain the HFIs. In these approaches, Apriori, proposed by Agrawal and Srikant [4] in 1994, is the most classic algorithm. According to the frequency downward closure property, if a 1-itemset’s frequency cannot meet the minimum frequency (min_freq) threshold, then all of its supersets cannot be a high-frequency itemset. Apriori adopts this property to discover the HFIs using a generate-and-test search. Although it can avoid a brute-force manner, it usually needs to scan the database multiple times. Therefore, it usually requires high consumption of execution time and a large search space. To overcome these shortages, the FP-growth algorithm [26] only requires twice scans of the database by utilizing a tree structure, called FP-tree, that extends its branches by using the prefix. Compared with Apriori, the FP-tree algorithm has less time and space complexity. To date, a large number of HFIM and ARM algorithms have been proposed in different domains [18, 35, 47, 48].

2.2 Utility-driven Pattern Mining

In the field of ARM, a number of rules with high support and confidence can be discovered by applying different existing algorithms, but the items included in the rules may not be the best target for the company. Generally, the profit of products is emphasized in smart manufacturing, people are eager to realize which product combination can bring more profits. HUPM [16, 20, 21] can well solve the problem, since for HUPM in manufacturing organizations, utility can often be replaced by profit. As the frequency downward closure property was first developed in HFIM, to maintain the similar property in HUPM, the transaction-weighted utility downward closure property was first applied in a two-phase model that proposed by Liu et al. [34]. In the first phase, it scans the database level-by-level to obtain a set of candidate patterns whose TWU satisfies the minimum utility threshold. In the second phase, the database is scanned again to compute the specific utility for each candidate. Therefore, all the actual high-utility itemsets (HUIs) are found in the second phase. To date, a series of utility-driven mining approaches have been proposed, as reviewed in Reference [21], including IHUP [5] for processing dynamic data, UP-growth [45] for processing static data, utility mining in uncertain data [15, 30], FHN [28] for processing transaction data containing both positive and negative utility values, UMEpi [17] for discovering high-utility episodes, up-to-date high-utility pattern mining [14, 31], TKU [44], and TKUS [49] for top-k mining, HUPM in dynamic profit databases [37] or noisy databases [6], and utility mining on sequence data [23, 24, 50]. In addition, based on the new utility-occupancy concept, the HUOPOm [22] and UHUOPM [9] algorithms are proposed. Some research tried to conduct a linear combination of frequency and utility as a weighted model, and thus more information about the patterns can be included [19, 46]. Similarly, Shao et al. [41] proposed an algorithm to mine a combined pattern with high utility and frequency. Such a pattern considers the utility in generated associative rules,
Table 1. Example Database

| tid | Transaction          | Utility |
|-----|----------------------|---------|
| $T_1$ | (A, 1), (B, 2), (C, 1) | $13$    |
| $T_2$ | (A, 2), (B, 3), (F, 2) | $23$    |
| $T_3$ | (B, 2), (D, 2), (E, 2) | $16$    |
| $T_4$ | (C, 2), (D, 1), (F, 1), (G, 3) | $10$    |
| $T_5$ | (B, 1), (C, 2), (F, 2), (G, 1) | $12$    |

Table 2. Utility Table

| Item | A | B | C | D | E | F | G |
|------|---|---|---|---|---|---|---|
| Utility ($) | 5 | 3 | 2 | 1 | 4 | 2 | 1 |

which aims to discover rules containing HUIs. In the past, a fast utility frequent mining algorithm (FUFM) [40] was introduced.

2.3 Pattern Classification

In smart manufacturing, pattern classification aims to automatically classify each pattern (a single product or a product combination) as to help decision makers to locate market position of each pattern and design more suitable strategies. To be different from general pattern mining applying single goal, pattern classification often consider multiple goals. For example, some research tried to classify patterns according to frequency and utility together. In the past, some researchers tried conducting a linear combination of frequency and utility as a weighted model, and thus more information about the patterns can be included. Lin et al. [29] introduced the concept of discriminative high-utility pattern with strong frequency affinity. Similarly, Shao et al. [41] proposed an algorithm to mine a combined pattern with high utility and frequency. Such a pattern considers the utility in generated associative rules, aiming to discover rules containing high-utility patterns. Meanwhile, in the past, an FUFM [40] was also introduced that conducted two approaches about FRM and HUPM in two different phases to categorize the pattern. The jointly utility-frequency approach discussed in the following sections utilizes both pattern classification and utility-driven pattern mining. Besides, the two designed algorithms are quite different. The first one adopts a two-phase manner, and the second one utilizes a vertical data structure containing transaction number, frequency, and remaining utility. Both algorithms aim to collect three different kinds of patterns based on frequency and utility.

3 PRELIMINARIES

3.1 Definitions

In this subsection, the preliminaries and definitions of key terms related to pattern classification and utility-driven mining are presented. A transaction database with the utility table as a running example is given in Tables 1 and 2, respectively.

Suppose there is a finite set of distinct items $I = \{I_1, I_2, \ldots, I_m\}$, which is stored in a transaction database $D = \{T_1, T_2, \ldots, T_n\}$. Each transaction contains a subset of $I$ and a unique identified number, which can be abbreviated as $tid$. Each item in a transaction has a positive value $q(I_t, T_i)$, called its internal utility, which can also be interpreted as the occurrence of an item in the transaction. A set of definitions and properties is given as follows.

**Definition 3.1 ($U(X)$, Utility of an Itemset).** The external utility of an item is associated with the utility table and can be considered as $v(I_t)$. And the external utility of an itemset $X$ is simply
computed by \( \sum_{I_i \in X} v(I_i) \). \( u(I_i, T_i) \) measures the total utility of an item in a transaction, which is calculated by \( q(I_i, T_i) \times v(I_i) \). The utility of an itemset \( X \) in a transaction is calculated as \( (\min(q(x_j, T_i))) \times v(X) \) where \( x_1, \ldots, x_j, \ldots, x_n \in X \). Thus, the utility of an itemset is the sum of utility of this itemset in all transactions, i.e., \( \sum_{x_j \in T_i} (\min(q(x_j, T_i))) \times (\sum_{x \in X} v(x_j)) \).

For instance, in Table 1, the external utility of \( A \) is \$5, and the utility of \( A \) in transaction \( T_1 \) is denoted by \( u(A, T_1) = $5 \times 2 = $10 \). Therefore, the utility of itemset \( \{A, B\} \) in \( T_1 \) is \( ($5 + $6) \times 1 = $11 \), and the utility of itemset \( AB \) is computed as \( U(AB) = U(AB, T_1) + U(AB, T_2) = ($5 + $3) \times 1 + ($5 + $3) \times 2 = $24 \). The transaction-weighted utility of itemset \( X \) is denoted as \( TWU(X) \) [44]. It is the sum of transaction utility of all the transactions containing \( X \), which is defined as \( TWU(X) = \sum_{T_i \in D} TU(T_i) \), in which \( TU(T_i) = \sum_{x \in T_i \land x \in X} u(x, T_i) \).

**Definition 3.2 (S(X), Support of an Itemset).** The support (or frequency) of an itemset \( X \) is associated with the internal utility of an item, which is defined as \( S(X) = \sum_{X \subseteq T_i} q(X, T_i) = \sum_{X \subseteq T_i} \min(x_j, T_i) \) where \( x_1, \ldots, x_j, \ldots, x_n \in X \).

For instance, in Table 1, the TWU of item \( A \) is computed by \( TWU(A) = TU(T_1) + TU(T_2) = $13 + $23 = $36 \). The frequency of itemset \( C \) is calculated by \( q(C) = q(C, T_1) + q(C, T_3) + q(C, T_3) = 1 + 2 + 2 = 5 \).

**Definition 3.3 (High-utility Itemset and High-frequency Itemset).** An itemset is called an HUI if its utility in a database is higher than or equal to a specified utility threshold, denoted as \( \min_{util} \). Similarly, an itemset is called an HFI if its support is higher than or equal to a specified frequency threshold, denoted as \( \min_{freq} \).

Table 1 shows a transaction database containing five transactions, where each letter represents a specific item, and the corresponding number represents the quantity in the corresponding transaction. The utility table is shown in Table 2. If \( \min_{util} \) is \$15, and \( \min_{freq} \) is 3, then a complete set of HUIs should be \{A\}: \$15, \{B\}: \$24, \{E\}: \$16, \{AB\}: \$24, \{BF\}: \$15, \{BDE\}: \$16 and a complete set of HFIs is \{A\}: 3, \{B\}: 8, \{C\}: 5, \{D\}: 3, \{E\}: 4, \{F\}: 3, \{G\}: 4, \{AB\}: 3, \{BF\}: 4, \{FG\}: 3.

### 3.2 Problem Statement

With the significant progress in data mining and knowledge discovery, tremendous data mining algorithms have been applied in many fields such as business. With appropriate applications in marketing, salesmen can better develop their own strategies to obtain more profit.

Our study mainly focuses on pattern classification by applying the indicators of frequency and utility. Both of them are very essential in developing strategies from a transaction database, because they are the fundamental information of an item or itemset. However, it is widely seen that utility or frequency is used as the sole measure in many typical applications, such as shopping basket analysis. Therefore, the following situations can be seen: What if an item (or itemset) of high frequency has a quite low utility (low profit in business) in a transaction database? What if an item (or itemset) of high utility only occurs for a few times in a transaction database?

Both situations mentioned above indicates two types of itemsets, high-frequency itemsets utility and low-frequency itemsets with high utility. At the same time, there are also itemsets that can bring high profit and have high occurrences. Naturally, in most cases people are preferable to high-frequency itemsets with high utility. However, the first two types of itemsets can also have positive impact, with less importance than high-frequency itemsets with high utility. Therefore, these three types of itemsets actually play different important roles in strategies developing and rule designing.
To better discover the potential of all patterns, an acceptable solution is to classify patterns according to these two indicators simultaneously. Given the threshold of frequency and utility, there are four types of patterns [46].

**Definition 3.4.** By taking frequency and utility together into account, there are four types of patterns: HFHUI; HFLUI; LFHUI; and LFLUI. These four patterns can be illustrated in Figure 1.

However, among these four categories, LFLUI actually have little contribution to rule discovery and strategies designing. Therefore, the goal of our proposed algorithms is to collect the other three types of itemsets from the given database. Naturally, if the utility and frequency of an itemset can satisfy both thresholds, then it belongs to HFHUI. If the itemset utility satisfies the threshold but the frequency does not, then it is HFLUI. Similarly, LFHUI and LFLUI are sorted in the same way. Recall that in Example 1, we denote the minimum frequency threshold is 3 and the minimum utility threshold is $15. In such case, we can obtain HFHUI: \{A\}: $15, 3 (the first one represents utility, the second one represents frequency), \{B\}: $24, 4, \{E\}: $16, 4, \{AB\}: $24, 3, \{BF\}: $15, 4; HFLUI: \{C\}: $6, 3, \{D\}: $3, 3, \{F\}: $6, 3, \{FG\}: $9, 3; LFHUI: \{BDE\}: $16, 2.

Generally, the application of our proposed algorithm in smart manufacturing can be split into the following steps: (1) read the transaction records from the database; (2) extract the necessary information from the records, i.e., the id of each good and its frequency and utility in each transaction; (3) apply UFC\textsubscript{gen} or UFC\textsubscript{fast} on these records; (4) obtain the classification results of three types of patterns; and, finally, (5) proceed advanced analysis and design new strategies according to the classification results. These processes are shown in Figure 2.

### 4 UFC\textsubscript{gen} Algorithm

In this section, we introduce an algorithm called UFC\textsubscript{gen}. The key idea is that we collect three types of itemsets based on the their utility and frequency in two phases. Details of this are presented below.

In the first phase, by utilizing the transaction-weighted utility downward closure property [34] and frequency downward closure property, the UFC\textsubscript{gen} algorithm can efficiently generate new candidates and put them into the set of final candidates. In the second phase, an extra scan of the whole transaction database is needed to calculate the real utility of each itemset in the final candidates set.

#### 4.1 Phase I

In the first scan of the database, we record the TWU and frequency of each 1-itemset and then generate new candidates by applying an operation called connection operation level by level until there are no new candidates generated. Now we introduce some significant properties and definitions that are applied in Phase I.
Property 1 (TWU Downward Closure Property [34]). In a given database $D$, for any itemset $X \in D$, if $TWU(X)$ is less than a user-specified min_util, then all supersets of $X$ cannot be an HUI. If $X \subseteq X'$, then $min\_util > TWU(X) \geq TWU(X')$.

For example, assume that $min\_util$ is $20$, then in Table 1 we can obtain $TWU$ of the itemset $E$, which is exactly $16$. Because $16 < 20$, by Property 1, we can conclude that $E$ cannot be an HUI.

Property 2 (Frequency Downward Closure Property [4]). In a given database $D$, for any itemset $X$, if $X$ is a low-frequency itemset, then for any superset $X'$ of $X$, $X'$ cannot be a high-frequency itemset.

For example, suppose that $min\_fre$ is $4$, then by Tables 1 and 2 we obtain the support of the itemset $AB$, which is computed by $q(AB, T_1) + q(AB, T_2) = 1 + 2 = 3$. Since $3 < 4$, then by Property 2, we can conclude that any superset of $AB$ such as $ABC$ cannot be a HFI.

Property 3. In a given database, We denote any high transaction-weighted utility itemset in $D$ as HTWUI. Then with the same user-specified min_util, if an itemset is an HUI, then it must be a HTWUI.

Proof. Let $D$ be the database and $W_i$ be the collection of all itemsets in transaction $T_i$; for a given itemset $X$, we define $X' = W \setminus X$, and then for any itemset $X \subseteq HUIS$, we have $min\_util \leq U(X) = \sum_{X \in T_i \land T_i \in D} U(X, T_i) \leq \sum_{X' \in T_i \land T_i \in D} (U(X, T_i) + U(X', T_i)) = \sum_{X' \in W_i \land W_i \in D} TU(W_i) = TWU(X)$. Therefore, $X$ is also a HTWUI. □

For example, we still assume that $min\_util$ is $20$, from Tables 1 and 2, we obtain the utility of itemset $B$, which is computed by $u(B, T_1) + u(B, T_2) + u(B, T_3) + u(B, T_5) = 3 \times (2 + 3 + 2 + 1) = 24$. Because $24 > 20$, then by Property 3, we can conclude that $B$ must be a HTWUI.

Definition 4.1 (Operation of Connection). We define a set as $C_k$ if for every itemset $I \subseteq C_k$, $I$ is an itemset containing $k$ items. We denote an operation as $\oplus$. For each $I_1, I_2 \subseteq C_k$, if the condition satisfied that for $i \in 1, 2, \ldots, k - 1$, $I_1[i] = I_2[i], I_1[k] \neq I_2[k]$, then $I_1 \oplus I_2 = I_3$, where $I_3 = \{I_1[1], I_1[2], \ldots, I_1[k], I_2[k]\}$.

Through the operation of connection, we can generate new candidates level by level and any itemset in level $k$ would be a $k$-itemset. From Table 1, we can obtain each item’s TWU and frequency in Table 3. Now suppose $min\_util$ and $min\_fre$ is $30$ and $4$, respectively, and then by
Table 3. Characteristics of Candidate Sets

| Itemset | TWU | Frequency |
|---------|-----|-----------|
| A       | $36 | 3         |
| B       | $52 | 8         |
| C       | $35 | 5         |
| D       | $26 | 3         |
| E       | $16 | 2         |
| F       | $22 | 3         |
| G       | $16 | 4         |

scanning Table 3, we have to omit item D, item E, and item F, since the either of their frequency or utility satisfy the thresholds. Hence they cannot engage in connection operation. Hence, in the next level, we will only obtain itemset like AB, AC, AG, and so on, connected by A, B, C, and G. In phase I, we continue this operation until no candidate sets can take part in the connection operation to enter the next level.

Using the properties and definitions above, we develop our pruning strategies to efficiently reduce search spaces. The strategies are presented the followings:

- **Strategy 1.** If the TWU of current itemset is less than the utility threshold and the frequency of it is less than the frequency threshold, then current itemset cannot appear in phase II, which indicates the current itemset is automatically a LFLUI.

- **Strategy 2.** If the transaction-weighted utility of current itemset is higher than or equal to the utility threshold or the frequency of it is higher than or equal to the frequency threshold, we add the itemset to phase II. Then we apply the connection operation to all such itemsets to generate new candidates.

In Phase I, the initial inputs are $C_1$, $\text{min\_util}$, and $\text{min\_fre}$. We use $\text{cur}$ to denote the sets of current itemsets with the proposition of TWU and frequency, denoted as $\text{twu}$ and $\text{fre}$, respectively. When the length of $\text{cur}$ is zero, the algorithm automatically terminates. And the set $\text{cand}$ is used to represent the final candidate itemsets to be classified into three different types of itemsets. To be noticed, the null of a set means there is no itemset in the set. In lines 3–8, we apply our strategies in this algorithm. If the proposition of current itemset in $\text{cur}$ does not satisfy either threshold, then we remove it from $\text{cur}$. In lines 10–19, we perform the connection operation on all itemsets in $\text{cur}$. Based on these strategies, all the itemsets that satisfy the condition would be added to the final candidates $\text{cand}$, which will be classified in phase II. Details of the procedure in phase I are represented in Algorithm 1.

### 4.2 Phase II

An additional scan of the transaction database is required. The extra scan is designed to calculate the real utility of each candidate set. There are no pruning strategies in phase II. Its purpose is to collect three types of itemsets according the user-specified thresholds. Eventually, $\text{HHFUI}$, $\text{HFLUI}$, and $\text{LFHUI}$ are presented as the output. In Phase II, $\text{util}$ and $\text{fre}$ are denoted as the real utility and frequency of each itemset respectively in the set $\text{cand}$. Details of phase II are given in Algorithm 2.

### 5 UFC$_{\text{fast}}$ ALGORITHM

In this section, a more efficient algorithm called UFC$_{\text{fast}}$ is supplemented to solve the pattern classification problem. First, we briefly introduce a special data structure called utility-list [33]. The basic definitions are presented in the follows.
**ALGORITHM 1:** Phase I of UFC_gen

**Input:** $C_1$: the set of all 1-itemsets; $\text{min}_\text{util}$, the threshold of utility; $\text{min}_\text{fre}$, the threshold of frequency.

**Output:** $\text{cand}$: the set of final candidates.

1. initialize $\text{cur} \leftarrow C_1$, $\text{cand} \leftarrow \text{null}$;
2. while $\text{length}(\text{cur}) > 0$ do
   3. for each itemset $e$ in $\text{cur}$ do
      4. if $e.\text{twu} \geq \text{min}_\text{util}$ or $e.\text{fre} \geq \text{min}_\text{fre}$ then
         5. add $e$ to $\text{cand}$;
      6. else
         7. remove $e$ from $\text{cur}$;
      8. end
   9. end
10. $\text{tmp} \leftarrow \text{null}$;
11. for each itemset $I_1$ in $\text{cur}$ do
   12. $k \leftarrow \text{length}(I_1)$;
   13. for each itemset $I_2$ after $I_1$ in $\text{cur}$ do
      14. if the first $(k-1)$ items in $I_1$ and $I_2$ are equal then
         15. $I_3 \leftarrow \{I_1[1], I_1[2], \ldots, I_1[k], I_2[k]\}$;
         16. add $I_3$ to $\text{cand}$;
         17. add $I_3$ to $\text{tmp}$;
      18. end
   19. end
   20. clear $\text{cur}$;
21. $\text{cur} \leftarrow \text{tmp}$;
22. end
23. return $\text{cand}$

### 5.1 Frequency-Utility-List

**Definition 5.1 (Revised Transaction [33]).** (1) A transaction is considered revised if the transaction is sorted in ascending order according to the TWU; (2) all the itemsets whose TWU is less than the specified utility threshold is deleted from the transaction.

**Definition 5.2 (Remaining Utility [33]).** In a database $D$, given a revised transaction and an item, the remaining utility of an itemset represents the sum of utility for items that are sorted after the current itemset. Let $\text{rutil}(X)$ [28, 33] denote the remaining utility of an itemset $X$ in a given database $D$, which is computed by $\sum_{X \subseteq T_1 \land T_1 \subseteq D \sum_{x_j > X} u(x_j, T_1)}$. Here $> \text{ means the position of } x_j \text{ is after } X \text{ in each revised transaction.}$

For instance, by assuming the threshold of utility is $30$, we can rearrange Table 1 according to the definition of revised transaction. The new table can be seen in Table 4. Considering the itemset $A$ in Table 4, the remaining utility of $C$ in $T_1$ can be computed as $\text{rutil}(C, T_1) = u(A, T_1) + u(B, T_1) = 5 \times 1 + 3 \times 2 = 11$. For a list that stores (i) the transaction number, (ii) the utility of an itemset in a transaction, and (iii) its remaining utility, we call such a list a utility-list [33]. This structure is very useful, because it contains significant information, which helps avoid generating unpromising candidates. The utility-list [33] stores the transaction number, utility in a transaction, and the
**ALGORITHM 2: Phase II of UFC**

Input: _cand_: the final candidates after the _Phase I_; _min_util_: the threshold of utility; _min_fre_: the threshold of utility.

Output: _HFHUI_: high frequency and high utility itemsets; _HFLUI_: high frequency and low utility itemsets; _LFHUI_: low frequency and high utility itemsets.

1. initialize _HFHUI_ $\leftarrow$ null, _HFLUI_ $\leftarrow$ null, _LFHUI_ $\leftarrow$ null;
2. scan the database again to obtain the real utility of each itemset;
3. for each pattern _elem_ in _cand_ do
   4. if _elem.util_ $\geq$ _min_util_ and _elem.fre_ $\geq$ _min_fre_ then
      5. add _elem_ to _HFHUI_;
   6. else if _elem.util_ $<$ _min_util_ and _elem.fre_ $\geq$ _min_fre_ then
      7. add _elem_ to _HFLUI_;
   8. else
      9. add _elem_ to _LFHUI_;
   10. end
4. end
5. return _HFHUI_, _HFLUI_, _LFHUI_

---

**Table 4. Example Database (Revised Transaction)**

| tid | Transaction   | Utility |
|-----|---------------|---------|
| $T_1$ | (C, 1), (A, 1), (B, 2) | $13$ |
| $T_2$ | (A, 2), (B, 3) | $19$ |
| $T_3$ | (B, 2) | $6$ |
| $T_4$ | (G, 3), (C, 2) | $7$ |
| $T_5$ | (G, 1), (C, 2), (B, 1) | $8$ |

remaining utility. Such a structure is very useful, because it contains very significant information and can be easily extended to a new itemset and thus avoid generating new candidates.

To solve the pattern classification problem, we slightly change the utility-list by substituting the second column into internal frequency of the item in each transaction, that is, the frequency of the item is recorded at each _tid_. For the above example, suppose the external utility of item _a_ is $2$, then the list of _a_ should be transformed into the list illustrated in Figure 3. We present it as a new data structure, called the frequency-utility-list or simply denote it as _FU-list_.

**Definition 5.3 (FU-list).** For a given database, the _FU-list_ contains a set of tuples, and each tuple has three fields: _tid_, _fre_, and _rutil_. The _fre_ is the occurrence of an itemset in the current transaction.

To obtain the set of _FU-lists_, we have to change the definition of a revised transaction, and now instead of deleting those itemsets with low TWU, we remove itemsets with low TWU and low frequency. Constructing the _FU-lists_ for one-itemsets requires two scans of the database. In the first scan, we calculate each one-itemset’s TWU and frequency to transform each transaction into a revised transaction. Then in the second scan, we construct _FU-lists_ from these new transactions.

### 5.2 Frequency-Utility-List of Multiple Itemsets

Building the _FU-list_ of an _k_-itemset is similar to the operation of connection mentioned in the _UFC_gen_ algorithm, where the intersection of transactions is not null and the first _k_-1 terms in two
different \( k \)-itemsets are equivalent. We now introduce a new operation called 1-extension, which is used to construct a new FU-list of \((k+1)\)-itemset from two different \( k \)-itemsets with the same prefix.

Definition 5.4 (1-extension [33]). For two different \( k \)-itemsets having the same first \((k-1)\)-items, denote them as \( ex \) and \( ey \), where \( e \) denotes the first \((k-1)\)-items. Assume in the revised transaction the item \( y \) is after item \( x \), then by traversing their FU-list, if a shared transaction is found, then we create a new entry by adding the information of shared \( tid \), the smaller frequency of \( ex \) and \( ey \) in this transaction, and the remaining utility of \( ey \). Then we add the entry to the FU-list of \( exy \). We keep the 1-extension operation until no more shared transactions are found.

Suppose there are two FU-lists created for two new items \( x \) and \( y \), whose TWU values are $50 and $100, respectively. It can be found that their common \( tid \)s are \{5, 9\}. For the frequency of itemset \{xy\} in \( T_5 \) and \( T_9 \), we choose the smaller frequency of item \( x \) and item \( y \). Therefore, in \( T_5 \), the frequency of \{xy\} should be 1 and in \( T_9 \) it should be 2. As for the remaining utility of \{xy\}, because the TWU of \( y \) is larger than \( x \), then in the revised transaction \( y \) should be after \( b \), and thus we have \( rutil(xy, T_5) = rutil(y, T_5) = $5 \) and \( rutil(xy, T_9) = rutil(y, T_9) = $10 \). Therefore, the FU-list of \{xy\} is built. The process is shown in Figure 4.

Property 4 (Remaining Utility Downward Closure Property). The property is similar with Property (1). If the sum of an itemset’s utility and the remaining utility of the itemset is less than the utility threshold, then any 1-extension of the itemset \( X \) cannot be an HUI, i.e., if \( X \subseteq X' \) and \( U(X) + rutil(X) < \text{min}_\text{util} \), then \( X' \) cannot be an HUI.

Proof. In the database \( D \), for an itemset \( X \), let \( X' \) be a 1-extension of \( X \), and define \( D_{X'} \) as the set of revised transactions containing \( X' \) and \( D_X \) as the set of revised transactions containing \( X \). Since \( X' \) is a 1-extension of \( X \), then \( D_{X'} \subseteq D_X \). Furthermore, every item after itemset \( X' \) must be
**ALGORITHM 3: Extend procedure**

**Input:** \( e \): an item; \( ex.FU \): the FU-list of itemset \( ex \); \( ey.FU \): the FU-list of itemset \( ey \).

**Output:** \( exy.FU \): the FU-list of itemset \( exy \).

1. initialize \( exy.FU ← null \);
2. for each entry \( a ∈ ex.FU \) do
   3. if \( ∃ a.tid = b.tid ∧ b ∈ ey.FU \) then
      4. \( fre ← min(a.fre, b.fre) \);
      5. create a new entry \( elem ← [a.tid, fre, b.rutil] \);
      6. \( exy.FU ← exy.FU ∪ elem \);
   end
   8. end
9. return \( exy.FU \)

Included in the set of items after itemset \( X \). Therefore, we can obtain \( \text{min}\_\text{util} > U(X) + rutil(X) = \sum_{T_i ∈ D_X} (u(X, T_i) + rutil(X, T_i)) \geq \sum_{T_j ∈ D_{X'}} (u(X', T_j)) = U(X') \). \( □ \)

In the extended procedure, we take \( e, ex.FU, \) and \( ey.FU \) as input, where \( e \) is an item (can be empty) and \( y \) is the item after the item \( x \) in the revised transactions. The remaining utility of an itemset in each transaction is denoted by \( rutil \). In lines 2 and 3, we collect transactions where \( ex \) and \( ey \) both exist. In lines 3–5, for the construction of new FU-list \( exy.FU \), for each satisfying transaction, we create a new entry for them. Then we select the smaller frequency of \( ex \) and \( ey \) as the frequency in the entry. Furthermore, the remaining utility of \( ey \) is chosen as the new remaining utility of \( exy \). Finally, we append the new entry to the FU-list of \( exy \). Details of the extended procedure for the 1-extension of \( ex \) are presented in Algorithm 3.

### 5.3 Pruning Strategies in UFC\(_{\text{fast}}\)

The pruning strategies are slightly different from the strategies in UFC\(_{\text{gen}}\) because of the special structure of the FU-list. There is an extra remaining utility, which can be used to derive a new but similar pruning strategy. In the FU-list, the frequency is already denoted in the second column; thus, the itemset utility can be easily obtained by acquiring the external utility of the itemset through the corresponding utility table, and thus we can immediately identify its category. Now we introduce a lemma, which is used as one of the pruning strategies in the UFC\(_{\text{fast}}\) algorithm.

**Lemma 5.5.** Given the FU-list of an itemset \( X \) with its external utility, if the product of its frequency and external utility (the utility of the itemset) plus its remaining utility is less than the given minimum threshold, then any extension \( X' \) of \( X \) is not an HUI.

Taking the FU-list of \( x \) as an instance, suppose its external utility is \$1 and the utility threshold is \$35. And its frequency multiples external utility plus its remaining utility is equal to \( u(x, T_2) + (x, T_3) + (x, T_9) = (2 \times \$1 + \$6) + (2 \times \$1 + \$7) + (2 \times \$1 + \$12) = \$31 < \$35 \). Then any extension of \( x \) cannot be a HUI. With Lemma 1 and Property 1 and Property 2 mentioned in UFC\(_{\text{gen}}\) algorithm, the pruning strategies are developed as the follows:

**Strategy 3.** In the first scan of the database, we select those 1-itemsets whose transaction-weighted utility are higher than or equal to the utility threshold or those whose frequency is higher than or equal to the frequency threshold. If neither of the thresholds are achieved, then the 1-itemset and its extension must be a LFLUI, and thus we simply remove it from the current transaction.

For example, suppose the \text{min}\_\text{util} is \$25 and \text{min}\_\text{fre} is 5. According to Tables 1 and 2, we obtain the TWU of itemset \( F \) and frequency of it are computed by \( U(T_4) + U(T_5) = \$10 + \$12 = \$22 \) and
ALGORITHM 4: UFC\textsubscript{fast} algorithm

```
\textbf{Input:} e: an item, initially empty; FUs: the set of FU-lists of itemset e’s 1-extensions; min\_util: the threshold of utility; min\_fre: the threshold of frequency.
\textbf{Output:} HHFUI: high frequency and high utility itemsets; HFLUI: high frequency and low utility itemsets; LFHUI: low frequency and high utility itemsets.

1. initialize HHFUI \leftarrow null, HFLUI \leftarrow null, LFHUI \leftarrow null;
2. for each FU-list \( t_1 \) in FUs do
3. \hspace{1em} x \leftarrow the itemset corresponding to \( t_1 \);
4. \hspace{1em} classify itemset \( x \);
5. \hspace{1em} if \( x.\text{fre} \times x.\text{util} + x.\text{rutil} \geq \text{min\_util or } x.\text{fre} \geq \text{min\_fre} \) then
6. \hspace{2em} exFUs \leftarrow null;
7. \hspace{2em} for FU-list \( t_2 \) after \( t_1 \) in FUs do
8. \hspace{3em} y \leftarrow the itemset corresponding to \( t_2 \);
9. \hspace{3em} exFUs \leftarrow exFUs \cup \text{Extend}(e, x, y);
10. \hspace{2em} end
11. \hspace{1em} end
12. \hspace{1em} call UFC\textsubscript{fast}(x, exFUs, min\_util, min\_fre);
13. end
```

\( q(F, T_4) + q(F, T_5) = 1 + 3 = 4 \), respectively. Because \$22 < \$25 and 4 < 5, then itemset \( F \) would be a LFLUI, and we should delete it.

\textbf{Strategy 4}. For each itemset, we can immediately obtain its real frequency and utility through its FU-list and utility table in the database, and thus the itemset can be classified immediately.

For example, as shown in Figure 3, the FU-list of item \( a \), by assuming the external utility of \( a \) is \$2, we can obtain the frequency and utility of \( a \): 5, \$10. If \( \text{min\_fre} \) and \( \text{min\_util} \) are 4 and \$15, respectively, then \( a \) should be a HFLUI.

\textbf{Strategy 5}. If the current itemset’s real frequency is higher than or equal to the frequency threshold or the sum of its utility plus and remaining utility is higher than the utility threshold, then we can apply 1-extension to the current itemset with other itemsets after it.

For example, we assume that \( \text{min\_util} \) is \$30 and \( \text{min\_fre} \) is 3. According to Tables 2 and 4, we obtain the sum of itemset \( G \)’s utility and its remaining utility is computed by \( U(G) + \text{rutil}(G) = (u(G, T_4) + u(G, T_5)) + (\text{rutil}(G, T_4) + \text{rutil}(G, T_5)) = 1 \times (3 + 1) + (2 \times 2 + 2 \times 2 + 3 \times 1) = 15. \)

The support of \( G \) is computed by \( q(F, T_4) + q(F, T_5) = 3 + 1 = 4 \). Because \$15 < \$30 and 4 > 3, we should create an 1-extension of \( G \), including itemset \( GC, GA, \) and \( GB \).

5.4 Proposed UFC\textsubscript{fast} Algorithm

After twice scans of the database, we build the initial FU-list for all the one-itemsets. The operation of 1-extension starts from the set of original FU-lists. At each time, we first obtain the current itemset’s frequency and utility and sort it according to the given thresholds. Then, by applying Strategy 5, we extend the current itemset with other itemsets. The UFC\textsubscript{fast} algorithm terminates when there are no more candidates that can be generated. Details of the algorithm are presented in Algorithm 4.

In this algorithm, an item \( e \) and the set of FU-lists \( FUs \) are used as the input. At first, we scan every FU-list of \( t_1 \) in \( FUs \), and the FU-lists of 1-itemsets are constructed for those whose transaction-weighted utility or frequency satisfy the minimum threshold; thus, we can immediately sort the current itemset \( x \) to corresponding category through its FU-list. This is exactly the process of Strategy 3 and Strategy 4. However, in lines 5–10, an empty set of FU-lists \( \text{exFUs} \) is created, and
the Strategy 5 is applied. If the sum of utility and remaining utility for \( x \) or its frequency meets the minimum value, then we apply the extended algorithm to it and all the itemset \( y \) after it to create new FU-lists. Once a new FU-list is created, it is added to the exFUs. Finally, the algorithm is called recursively, and it terminates when there is no more itemset that can be classified.

According to Property 2 and Property 4, the completeness and correctness of the proposed UFC\(_{\text{fast}}\) algorithm is satisfied; hence, it can ensure that each itemset is sorted into the corresponding category. Furthermore, all the itemsets can be applied to the 1-extensions operation if the condition is satisfied, and hence the algorithm can guarantee that all the possible itemsets would be discovered.

6 EXPERIMENTS

We performed extensive experiments to evaluate the UFC\(_{\text{gen}}\) and UFC\(_{\text{fast}}\) algorithms. Because the model is the first model that collects three types of itemsets according to frequency and utility, we can only compare these two algorithms on different datasets, in terms of execution time, memory consumption, and scalability.

Both algorithms are implemented in the Java language. All experiments were conducted in a personal computer with 4 GB of RAM, running the 64-bit Microsoft Windows 10 operating system. In the experiments, the execution time, memory consumption, classification results, and scalability analysis are evaluated, respectively. To visualize the impact of thresholds on other factors, \( \text{min'util} \) and \( \text{min'fre} \) are represented as percentage in all the experiments. For instance, if the sum of utility in a database is $1,000 and the minimum utility threshold is $150, then \( \text{min'util} \) should be 15%, which is calculated as \( \frac{\text{150}}{1000} \times 100\% \).

In the experiment, unlike the general experiments for HFIM and HUPM, the parameters of thresholds are set in the form of combinations, which means that both frequency and utility are taken into account in the same experiment. In each experiment, we choose different thresholds of frequency and vary the threshold of utility. For each dataset, the parameters of both thresholds are all set at a very low level (less than 1%), because in earlier testings we found that when the thresholds were set at a relative high level, then many itemsets could not meet the condition, which could cause difficulties in comparison for the proposed algorithms. From our previous testings, it is proved that an increase of 5% for each point is a relative appropriate setting to show the general trend for the proposed algorithms.

6.1 Experiment Datasets

Both synthetic and real datasets (foodmart, retail, T40I10D100K, and yoochooseBuys) were used in the experiments. The resource link for the synthetic datasets and real dataset are http://www.philippe-fournier-viger.com/spmf and recsys.acm.org/recsys15/challenge/. For the dataset foodmart, it has 21,556 rows of transactions, while having around 1,559 items contained. For datasets retail and T40I10D100K, the quantities of their transactions are much larger. At the same time, there are only averagely 4 items and 10.3 items for most of the transactions from foodmart and retail, separately, but for dataset T40I10D100K, it includes more than 70 items in each row of transactions. For all datasets, both internal utility and external utility are assigned to each item. The detailed characteristics of all datasets are listed in Table 5, where \( |D| \) represents the number of transactions in the database, and \( |I| \) indicates the number of items in the database.

6.2 Execution Time

The running time of UFC\(_{\text{gen}}\) and UFC\(_{\text{fast}}\) algorithms on synthetic datasets are illustrated in Figure 5. A task would terminate if its running time exceeded 10,000 s. By setting a group of thresholds for both frequency and utility, we obtain the execution time of the proposed algorithms at each point. It is likely that the lower the threshold is, the more computation time is
Table 5. Characteristics of Test Datasets

| Dataset          | #|D| | #|I| | AvgLen |
|------------------|--------------|---|---|---|---|---|
| retail           | 88,162       | 16,470 | 10.3 |
| foodmart         | 21,556       | 1,559 | 4.0  |
| T40I10D100K      | 100,000      | 942   | 39.6 |
| yoochooseBuys    | 507,746      | 19,102 | 2.3  |

required for both algorithms. The execution time increases, because there are more itemsets satisfying the requirements. For instance, on dataset retail, when min_fre and min_util are both 0.15%, the running times for UFC_gen and UFC_fast are 210 seconds and 6.5 seconds, respectively. However, when the thresholds both become 0.1%, their execution time became 217 seconds and 23 seconds, respectively.

It is noticeable that UFC_fast always requires less time than that on UFC_gen. This is mainly because UFC_gen keeps generating new candidates level by level, during which a large amount of time is consumed in getting new candidates by applying the connection operation. In particular, when the number of items is extremely large, the computation time for combining current itemsets can be very slow. When the number of itemsets becomes smaller, the UFC_gen algorithm can spend less time on connecting the old itemsets to generate new ones, thereby reducing the time difference between both algorithms. For example, from Figure 5(b), it can be seen that when applying the dataset foodmart, the difference of running time between UFC_fast and UFC_gen is much smaller than that on dataset retail. The ranges for both proposed algorithms are within 5 seconds, and the largest time consumed is 2.6 seconds and 4.5 seconds, respectively, when the thresholds of frequency and utility are 0.1% and 0.05%, separately.

Figure 5(c) shows the running time on dataset (T40I10D100K). Because the average length of the transaction database is large, for the UFC_gen, it has to deal with an enormous number of itemsets when setting the same thresholds as in dataset retail; in that case, the execution time is remarkably large, which means it has to be terminated. By contrast, for UFC_fast, it finishes the classification tasks longer than on dataset retail and foodmart, where the most time consumed is 1,635 seconds when the thresholds are both 1%. The overall trend of execution time for UFC_fast on it is relatively smooth.

Overall, the proposed UFC_fast has better performance than UFC_gen on computation time under various thresholds of utility and frequency on each dataset. This is particularly evident when the database contains a large number of items and many rows of transactions. However, from the figures it can be seen that the execution time of both algorithms does not always reduce linearly as the thresholds increase, but the fluctuations are often acceptable.

6.3 Memory Consumption

Figure 6 shows the peak memory consumption of both algorithms on real and synthetic datasets. It can be seen that UFC_fast always consumes much less memory than UFC_gen. This is mainly because UFC_fast does not generate candidates, while UFC_gen always generates new candidates from previous candidates, satisfying conditions each time.

Generally, the trends of memory consumption for both algorithms on the datasets are not linear. The peak memory of UFC_gen fluctuates significantly on datasets retail and foodmart. When applying UFC_fast to synthetic datasets, the memory consumption for it tends to be more stable. For instance, on dataset retail, with the increase of one of the thresholds, the memory consumption by applying UFC_fast is maintained at approximately 200 MB, reaching the highest (271.78 MB) when
Fig. 5. Execution time w.r.t. different combination of thresholds of frequency and utility.

the thresholds of frequency and utility are 0.3% and 0.25%, respectively. By contrast, the figure for \textit{UFC}_{gen} experiences some significant fluctuations when situated in different thresholds. When the utility threshold is 0.05%, its memory consumption of it is dramatically higher than that in other situations. While increasing the utility threshold, the memory consumed drops noticeably, but it does not maintain the declining trend and increases in other situations. However, large fluctuations can be seen in \textit{UFC}_{fast} on T40I10D100K.

It can be concluded that the \textit{UFC}_{gen} algorithm may not fit in datasets containing a high quantity of items or transactions of a long average length. This means that there is likely to be a remarkable
memory consumption on storage for new itemsets generated by connecting the current candidate sets. For instance, when we apply \(\text{UFC}_{\text{gen}}\) on dataset T40I10D100K, the memory consumption is so large that exceeds the stack capacity of JVM, when the threshold of utility is 0.1%. However, for the \(\text{UFC}_{\text{fast}}\) algorithm, it stores information about all itemsets using the FU-list. At each time it directly classifies the current itemset using the pruning strategies without generating new candidates. Therefore, its memory consumption of it on different datasets always has a much better performance than that of \(\text{UFC}_{\text{gen}}\). It only requires a little extra memory to store the remaining utility of each itemset. Overall, as in the experiment of testing execution time, \(\text{UFC}_{\text{fast}}\) outperforms \(\text{UFC}_{\text{gen}}\) dramatically, and the main reason for this is that the former does not generate new candidates, while the latter does.

Fig. 6. Memory consumption w.r.t different thresholds of frequency and utility.
Table 6. Classification Results on Tested Datasets

| Dataset          | min_util | HFHUI | HFLUI | LFHUI |
|------------------|----------|-------|-------|-------|
| Foodmart (0.07%) |          |       |       |       |
| 0.070%           | 224      | 200   | 400   |       |
| 0.075%           | 211      | 213   | 333   |       |
| 0.080%           | 187      | 237   | 273   |       |
| 0.085%           | 170      | 254   | 214   |       |
| Retail (0.1%)    |          |       |       |       |
| 0.10%            | 97       | 22    | 293   |       |
| 0.15%            | 79       | 40    | 107   |       |
| 0.20%            | 66       | 53    | 47    |       |
| 0.25%            | 54       | 65    | 24    |       |
| T10I4D100K (0.1%)|          |       |       |       |
| 0.10%            | 243      | 150   | 23,347|       |
| 0.15%            | 188      | 205   | 3,086 |       |
| 0.20%            | 138      | 255   | 361   |       |
| 0.25%            | 105      | 288   | 59    |       |

6.4 Classification Results

By selecting different thresholds for frequency and utility, there are various classification results. Both algorithms obtained the same classification results for the datasets. Table 6 compares the number of HFHUI, HFLUI, and LFHUI for the tested datasets, with fixed frequency threshold and varying utility thresholds. The chosen frequency threshold from each dataset is denoted in the racket of the first column in the table.

For dataset retail, the number of HFHUI only decreases from 97 to 54 when the threshold increased, while the quantity of LFHUI decreases dramatically from 293 to 24. Similarly, it can be seen that the number of LFHUI on dataset T40I10D100K is 23,347 when min_util is 0.1%, which is significantly large. However, the quantity drops rapidly to 3,086 when the threshold rises to 0.15%. As for dataset foodmart, it can be seen that the number of HFLUI accounts for the majority of total candidates in all situations, which indicates that a large number of itemsets in the dataset tends to have the preposition of higher frequency and lower utility. With an increase in the threshold of utility, the numbers of HFLUI and HFHUI both decline, and the quantity of LFHUI increases significantly.

6.5 Scalability Analysis

We use two datasets (foodmart and retail) to perform the scalability analysis of the proposed algorithms. We fix the thresholds of utility and frequency to 0.1%, while increasing the number of transactions each time. With the increase of the number of transactions, the number of candidates could see a growing trend, because some candidates may not be included in those new additive transactions. Figures 7 and 8 illustrate the execution time and memory consumption of UFC\textsubscript{fast} and UFC\textsubscript{gen} on dataset foodmart and retail.

Both proposed algorithms have excellent scalability with regard to execution time. On foodmart, we find that the execution time of both algorithms increases almost linearly, although the time for UFC\textsubscript{gen} is higher than UFC\textsubscript{fast} throughout the period. However, this gap is not noticeable. For dataset retail, the increment of transactions is 10,000 each time. Changes in UFC\textsubscript{gen} become larger when applying different sizes of total transactions, spending 69.637 seconds when the size is 2,000, while it becomes 187.717 seconds when the size is 8,000. However, the time spent by UFC\textsubscript{fast} seems to have a linear trend. The time grows smoothly and slowly, and in each situation, the time consumed is within 25 seconds, which is much less than the time spent by UFC\textsubscript{gen}.
However, in Figure 8, it shows the memory consumption of UFC\textsubscript{fast} and UFC\textsubscript{gen} on both datasets, with the same division of transactions as before. For UFC\textsubscript{fast}, it tends to keep unchanged or grows steadily, while for UFC\textsubscript{gen}, there is often a significant growth on it. For example, when running both algorithms on foodmart, the memory consumption of UFC\textsubscript{fast} increases steadily as a linear trend, standing at 49.9 MB when the number of transactions is 20,000, with the figure for that being 114.64 MB when the quantity of transactions is 80,000. By contrast, when applying UFC\textsubscript{gen}, the memory consumption is 1006.88 MB when the transaction number is 20,000. However, the number becomes 2,173 MB when the quantity of transactions is 80,000.

### 6.6 Case Study on Market Analysis

A case study is also evaluated on both proposed algorithms. The real-life dataset is yoochooseBuys, which is a collection of sequences of click events. It represents an activity period of six months for a big e-commerce business in Europe selling all kinds of products, such as garden tools, toys, clothes, and electronics. There are 507,746 rows of transactions and 19,102 items in total. Each transaction records the quantity and the price of each item. The frequency threshold is set to range from 0.1% to 0.25%, with the threshold of utility starting from 0.05% to 0.3% after testing different parameters. Table 7 shows the detailed classification results.

Figures 9 and 10 showed the performances of both algorithms with regard to execution time and memory consumption, respectively. For the running time, both algorithms have excellent performances on it, presenting a general smooth trend. When the UFC\textsubscript{fast} is applied, it only takes around 5 seconds when the utility threshold is 0.05%, with four different frequency thresholds. As for other combinations of utility and frequency, the running time remains at the level under 3 seconds, ranging from 1.180 to 2.731 seconds. However, the UFC\textsubscript{gen} cost more than 20 seconds when the utility threshold is 0.5%. However, the time drops sharply when the threshold of utility is transformed into 1%. It shares a similar trend with that for UFC\textsubscript{fast}, and the gap between them
Table 7. Classification Results on YoochooseBuys

| Dataset       | min_util | # HFHUI | # HFLUI | # LFHUI |
|---------------|----------|---------|---------|---------|
| yoochooseBuys | 0.05%    | 148     | 68      | 348     |
|               | 0.1%     | 86      | 130     | 127     |
|               | 0.15%    | 62      | 154     | 70      |
|               | 0.20%    | 50      | 166     | 40      |
|               | 0.25%    | 42      | 174     | 31      |
|               | 0.30%    | 28      | 188     | 21      |

Fig. 9. Execution time on yoochooseBuys.

Fig. 10. Memory consumption on yoochooseBuys.
is getting smaller. Overall, dramatic differences between both algorithms do not appear on this business dataset from the real world, while both of the proposed methods behave well on it.

7 CONCLUSION AND FUTURE STUDIES

In this article, we studied the problem of pattern classification for smart systems. We used the original properties of both frequency and utility and developed two algorithms, UFC_{gen} and UFC_{fast}, which were used to efficiently collect three types of patterns, HFHUIs, HFLUIs, and LFHUIs. For the proposed levelwise-based UFC_{gen} algorithm, it adopted a level-based structure, and a number of new candidates were generated each time when a connection operation was applied. As for UFC_{fast}, it utilized a list-based structure and can reduce the computational time, since the database was only scanned twice. Compared with UFC_{fast}, UFC_{gen} may not be suitable enough, since it required the generation of a large number of candidates and multiple scans for the database during the mining process. Experimental results on real and synthetic datasets showed that both UFC_{gen} and UFC_{fast} were effective for pattern classification in smart systems. UFC_{fast} had a more significant performance when dealing with different types of datasets under various parameters in terms of both execution time and memory consumption. In practice, UFC_{fast} was more suitable for processing large-scale datasets.

For future studies, we plan to perform research to further improve the performance of both proposed algorithms. Currently, there is a relatively small number of research efforts being conducted on pattern classification, and the method of how to better combine the properties of frequency and utility still requires further discovery. Moreover, studies of pattern classification based on different models in smart systems, such as smart manufacturing, will be performed in the future.

REFERENCES

[1] Neda Abdelhamid, Aladdin Ayesh, and Fadi Thabtah. 2014. Phishing detection based associative classification data mining. Expert Syst. Appl. 41, 13 (2014), 5948–5959.
[2] Neda Abdelhamid and Fadi Thabtah. 2014. Associative classification approaches: Review and comparison. J. Inf. Knowl. Manage. 13, 3 (2014), 1450027.
[3] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining association rules between sets of items in large databases. In ACM SIGMOD Record, Vol. 22. ACM, 207–216.
[4] Rakesh Agrawal, Ramakrishnan Srikant, et al. 1994. Fast algorithms for mining association rules. In Proceedings of the 20th International Conference on Very Large Data Bases. 487–499.
[5] 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 Trans. Knowl. Data Eng. 21, 12 (2009), 1708–1721.
[6] Yoonji Baek, Unil Yun, Heonho Kim, Jongseong Kim, Bay Vo, Tin Truong, and Zhi-Hong Deng. 2021. Approximate high utility itemset mining in noisy environments. Knowl.-Bas. Syst. 212 (2021), 106596.
[7] Sergey Brin, Rajeev Motwani, and Craig Silverstein. 1997. Beyond market baskets: Generalizing association rules to correlations. In ACM SIGMOD Conference. ACM, 265–276.
[8] Raymond Chan, Qiang Yang, and Yi Dong Shen. 2003. Mining high utility itemsets. In Proceedings of the 3rd IEEE International Conference on Data Mining. IEEE, 19–26.
[9] Chien-Ming Chen, Lili Chen, Wensheng Gan, Lina Qiu, and Weiping Ding. 2021. Discovering high utility-occupancy patterns from uncertain data. Inf. Sci. 546 (2021), 1208–1229.
[10] Ming Syan Chen, Jiawei Han, and Philip S. Yu. 1996. Data mining: An overview from a database perspective. IEEE Trans. Knowl. Data Eng. 8, 6 (1996), 866–883.
[11] Alok Kumar Choudhary, Jenny A. Harding, and Manoj Kumar Tiwari. 2009. Data mining in manufacturing: A review based on the kind of knowledge. J. Intell. Manufact. 20, 5 (2009), 501–521.
[12] Alican Dogan and Derya Birant. 2020. Machine learning and data mining in manufacturing. Expert Syst. Appl. (2020), 114060.
[13] Chih-Min Fan, Ruey-Shan Guo, Argon Chen, Kuo-Ching Hsu, and Chih-Shih Wei. 2001. Data mining and fault diagnosis based on wafer acceptance test data and in-line manufacturing data. In Proceedings of the IEEE International Symposium on Semiconductor Manufacturing. IEEE, 171–174.
[14] Wensheng Gan, Jerry Chun-Wei Lin, Han-Chieh Chao, Philippe Fournier-Viger, Xuan Wang, and Philip S. Yu. 2020. Utility-driven mining of trend information for intelligent system. *ACM Trans. Manage. Inf. Syst.* 11, 3 (2020), 1–28.

[15] Wensheng Gan, Jerry Chun-Wei Lin, Han-Chieh Chao, Athanasios V. Vasilakos, and Philip S. Yu. 2020. Utility-driven data analytics on uncertain data. *IEEE Syst. J.* 14, 3 (2020), 4442–4453.

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

[17] Wensheng Gan, Jerry Chun-Wei Lin, Han-Chieh Chao, and Philip S. Yu. 2019. Utility-driven mining of high utility episodes. In *Proceedings of the IEEE International Conference on Big Data*. IEEE, 2644–2653.

[18] Wensheng Gan, Jerry Chun-Wei Lin, Han-Chieh Chao, and Justin Zhan. 2017. Data mining in distributed environment: A survey. *Data Min. Knowl. Discov.* 7, 6 (2017), e1216.

[19] 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. *Knowl.-Bas. Syst.* 143 (2018), 30–41.

[20] Wensheng Gan, Jerry Chun-Wei Lin, Philippe Fournier-Viger, Han-Chieh Chao, and Hamido Fujita. 2018. A survey of incremental high-utility itemset mining. *Data Min. Knowl. Discov.* 8, 2 (2018), e1242.

[21] 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 Trans. Knowl. Data Eng.* 33, 4 (2021), 1306–1327.

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

[23] Wensheng Gan, Jerry Chun-Wei Lin, Jiexiong Zhang, Han-Chieh Chao, Hamido Fujita, and Philip S. Yu. 2020. ProUM: Projection-based utility mining on sequence data. *Inf. Sci.* 513 (2020), 222–240.

[24] Wensheng Gan, Jerry Chun-Wei Lin, Jiexiong Zhang, Philippe Fournier-Viger, Han-Chieh Chao, and Philip S. Yu. 2021. Fast utility mining on sequence data. *IEEE Trans. Cybernet.* 51, 2 (2021), 487–500.

[25] Jiawei Han, Micheline Kamber, and Jian Pei. 2011. Data mining concepts and techniques (3rd ed.). Morgan Kaufmann, ISBN 978-0123814791.

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

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

[28] 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. *Knowl. Inf. Syst.* 51, 3 (2017), 873–909.

[29] 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. *Knowl.-Bas. Syst.* 96 (2016), 171–187.

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

[31] Qi Lin, Wensheng Gan, Yongdong Wu, Jiachun Chen, and Chien-Ming Chen. 2021. Joint utility and frequency for pattern classification. In *Proceedings of the IEEE International Conference on Big Data*. IEEE, 5524–5533.

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

[33] Ying Liu, Wei keng Liao, and Alok Choudhary. 2005. A two-phase algorithm for fast discovery of high utility itemsets. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 689–695.

[34] José María Luna, Philippe Fournier-Viger, and Sebastián Ventura. 2019. Frequent itemset mining: A 25 years review. *Data Min. Knowl. Discov.* 9, 6 (2019), e1329.

[35] Kouta Nakata, Ryohsei Orihara, Yoshiaki Mizuoka, and Kentaro Takagi. 2017. A comprehensive big-data-based monitoring system for yield enhancement in semiconductor manufacturing. *IEEE Trans. Semicond. Manufact.* 30, 4 (2017), 339–344.

[36] Loan T. T. Nguyen, Phuc Nguyen, Trinh D. D. Nguyen, Bay Vo, Philippe Fournier-Viger, and Vincent S. Tseng. 2019. Mining high-utility itemsets in dynamic profit databases. *Knowl.-Bas. Syst.* 175 (2019), 130–144.

[37] Loan T. T. Nguyen, Bay Vo, Tzung-Pei Hong, and Hoang Chi Thanh. 2012. Classification based on association rules: A lattice-based approach. *Expert Syst. Appl.* 39, 13 (2012), 11357–11366.

[38] Jian Pei, Jiawei Han, Hongjun Lu, Shojiro Nishio, Shiwei Tang, and Dongqing Yang. 2001. H-Mine: Hyper-structure mining of frequent patterns in large databases. In *Proceedings of the IEEE International Conference on Data Mining*. IEEE, 441–448.

[39] S. Shankar, Nishanth Babu, T. Purusothaman, and S. Janythi. 2009. A fast algorithm for mining high utility itemsets. In *Proceedings of the IEEE International Advance Computing Conference*. IEEE, 1459–1464.
[41] Jingyu Shao, Junfu Yin, Wei Liu, and Longbing Cao. 2015. Mining actionable combined patterns of high utility and frequency. In Proceedings of the IEEE International Conference on Data Science and Advanced Analytics. IEEE, 1–10.

[42] Fadi Abdeljaber Thabtah. 2007. A review of associative classification mining. Knowl. Eng. Rev. 22, 1 (2007), 37–65.

[43] Fadi A. Thabtah, Peter Cowling, and Yonghong Peng. 2004. MMAC: A new multi-class, multi-label associative classification approach. In Proceedings of the 4th IEEE International Conference on Data Mining. IEEE, 217–224.

[44] Vincent S. Tseng, Cheng-Wei Wu, Philippe Fournier-Viger, and Philip S. Yu. 2015. Efficient algorithms for mining top-k high utility itemsets. IEEE Trans. Knowl. Data Eng. 28, 1 (2015), 54–67.

[45] Vincent S. Tseng, Cheng Wei Wu, Bai En Shie, and Philip S. Yu. 2010. UP-Growth: An efficient algorithm for high utility itemset mining. In Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 253–262.

[46] Jing Wang, Ying Liu, Lin Zhou, Yong Shi, and Xingquan Zhu. 2007. Pushing frequency constraint to utility mining model. In Proceedings of the International Conference on Computational Science. Springer, 685–692.

[47] Yan Xie and Philip S. Yu. 2010. Max-clique: A top-down graph-based approach to frequent pattern mining. In Proceedings of the IEEE International Conference on Data Mining. IEEE, 1139–1144.

[48] Mohammed Javeed Zaki. 2000. Scalable algorithms for association mining. IEEE Trans. Knowl. Data Eng. 12, 3 (2000), 372–390.

[49] Chunkai Zhang, Zilin Du, Wensheng Gan, and Philip S. Yu. 2021. TKUS: Mining top-k high utility sequential patterns. Inf. Sci. 570 (2021), 342–359.

[50] Chunkai Zhang, Zilin Du, Yuting Yang, Wensheng Gan, and Philip S. Yu. 2021. On-shelf utility mining of sequence data. ACM Trans. Knowl. Discov. Data 16, 2 (2021), 1–31.

Received 17 August 2021; revised 22 March 2022; accepted 12 April 2022