Discovering Top-k Profitable Patterns for Smart Manufacturing

Shicheng Wan
Guangdong University of Technology
Guangzhou, China
scwan1998@gmail.com

Jiahui Chen
Guangdong University of Technology
Guangzhou, China
csjhchen@gmail.com

Peifeng Zhang
Guangdong University of Technology
Guangzhou, China
paddeyzhang@gmail.com

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

Tianlong Gu
Jinan University
Guangzhou, China
gutianlong@jnu.edu.cn

ABSTRACT

In the past, many studies were developed to discover useful knowledge from rich data for decision making in wide applications in Internet of Things and Web of Things, such as smart manufacturing. Utility-driven pattern mining (UPM) technology is famous in knowledge discovering domain. However, one of the biggest issues of UPM is the setting of a suitable minimum utility threshold (*minUtil*). The higher *minUtil* is, the fewer interesting patterns are obtained. Conversely, the lower *minUtil* is, the more useless patterns are discovered. In this paper, we propose a solution for discovering top-k profitable patterns with average-utility measure, which can be applied to manufacturing. The average-utility of a pattern, w.r.t its corresponding length, can be used to fairly measure the pattern. The proposed new upper-bounds on average-utility are tighter than previous upper-bounds. Moreover, based on these upper-bounds, the novel PPT algorithm utilizes merging and projection techniques to greatly reduce the search space. By adopting several threshold raising strategies, the PPT algorithm can discover correct top-k patterns in a short time. We also implemented the efficiency and effectiveness of the algorithm on real and synthetic datasets. The experimental results reveal that the algorithm not only get a complete set of top-k interesting patterns, but also works better than the state-of-the-art algorithm in terms of runtime, memory consumption and scalability. Especially, the proposed algorithm performs very well on dense datasets.

CCS CONCEPTS

• **Computing methodologies** → **Data mining**; • **Networks** → Web of Things.

KEYWORDS

Web of Things, smart manufacturing, data analytics, top-k mining, average-utility.

*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/10.1145/3487553.3524706

ACM Reference Format:
Shicheng Wan, Jiahui Chen, Peifeng Zhang, Wensheng Gan, and Tianlong Gu. 2022. Discovering Top-k Profitable Patterns for Smart Manufacturing. 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/10.1145/3487553.3524706

1 INTRODUCTION

The rapid developing of the information technologies has promoted the combination of Internet, Internet of Things (IoT) [1, 23] and Web of Things (WoT) [43] in the past decades. In manufacturing industry, massive raw data (i.e., moistness, temperature, pressure, etc.) will be generated and collected in every day. How to take advantage of these information and then provide a better service for customers is a vital and interesting task. Li *et al.* [22] figured out that applying "Big Data" techniques in manufacturing can greatly improve the service process. Meanwhile, frequent itemset mining (FIM) [9] technique has been extensively studied in how to efficiently and effectively mine interesting and useful patterns from massive data. FIM technologies mainly focus on the frequency feature of different itemsets. A frequent itemset is the itemset whose frequency is no less than the user-specified minimum support threshold. However, one of the well-known problems of FIM technique is that they cannot reveal the importance (e.g. risk [10], weight [15, 26] and profit [30, 44]) of items in an itemset. For example, hundreds of bread cannot earn a higher revenue than a diamond ring. The reason is the unit profit of bread and diamond ring are different in real life. What's more, they will be regard as identical in FIM. Thus a new branch of data mining named high-utility itemset mining (HUIM) [2, 3, 11, 24] has attracted a lot of attention.

HUIM technologies usually use internal utility (i.e., quantity) and external utility (i.e., unit profit) to evaluate the importance (i.e., utility) of an itemset. The utility of an interesting itemset (i.e., high-utility itemset, HUI) is higher than or equal to the user-specified minimal utility threshold. Obliviously, HUIs can represent more meaningful and useful patterns than that of frequent itemsets. Therefore, many recommendation applications like user behavior analysis [35], website click-stream analysis [6], and marketing analysis [4] adopt HUIM technologies to enhance service quality. Through the timely interaction of information (i.e., HUIs) in industrial chain, HUIM technologies can strengthen the supply chain management and optimize the allocation of distributed manufacturing resources. However, the utility value of an itemset will increase while the length of the itemset increases. This causes the long itemsets may have advantage over these short itemsets in terms of utility [33]. For example, the utility of the product combination "desktop computer", "mouse" and other computer accessories usually is larger than that of a single product "laptop". Hence, it is not suitable for mining HUIs with different lengths by the same threshold. Fortunately, average-utility measurement [17] can reveal the utility effect of combining several items accurately. The "average" means the utility of a pattern should be divided by its length. Hence high average-utility mining (HAUIM) [18, 19, 34] technologies can more objectively assess the utility of itemsets with different lengths.

Nevertheless, compared with FIM and HUIM technologies, HAUIM is more complicated because the anti-monotonic property in FIM and HUIM cannot be directly adopted in average-utility. How to determine the super-itemsets of a high average-utility itemset (HAUI for short) are HAUIs or not is a difficult challenge. To solve this issue, Hong *et al.* [17] proposed average-utility upper-bound (abbreviated as *auub*) which has the downward closure property (anti-monotonic). It uses maximum utilities of itemset in transactions to ensure the calculation result be higher than or equal to the real average-utility of the itemset. In addition, another inevitable limitation of FIM, HUIM and HAUIM technologies is how to find a suitable threshold. In most cases, users have to do experiments many times to get an appropriate threshold value. This is disturbing and wasting time. Therefore, a solution of this problem is mining top-k frequent itemsets [21], top-k HUIs [5, 16], and top-k HAUIs [39], respectively. The parameter k represents the number of interesting itemsets a user requires. What's more, there are several challenges in top-k HAUIM: 1) due to the threshold is initialized as 0 or 1, how to raise the threshold value quickly and accurately is a difficult task; 2) the pruning strategies of HUIM cannot be directly utilized in HAUIM because of average concept; and 3) the traditional *auub* is so loose that massive unpromising itemsets generating. To author's best knowledge, there is still lack of studies about top-k HAUIM technologies. For brevity, there is only the TKAU algorithm proposed by Wu *et al.* [39]. Though they proposed several novel upper-bounds which are tighter than that of traditional HAUIM algorithms, it will generate massive list structures when the number of distinct items increases. More importantly, their redundant join operation may slow down the mining process.

In view of this, we propose a novel algorithm for discovering top-k **P**rofitable **P**atterns with average-u**T**ility (PPT for short), and this algorithm can be applied into various applications in WoT [43], such as smart manufacturing. It is an efficient pattern-growth method which uses depth-first mechanism to search the top-k HAUIs. The major contributions of our study are summarized as follows:

- Two effective technologies, called database projection and transaction merging, are adopted to reduce the search space when the algorithm mines top-k HAUIs.
- Two novel upper-bounds named local average-utility and maximum average-utility are proposed, and their corresponding pruning strategies are utilized to improve the performance of the novel algorithm.
- Two strategies (RAU and RUC) are utilized to raise the minimum average-utility threshold effectively. These strategies can ensure the accuracy of final results and help cutting low average-utility itemsets in search space effectively.
- In order to make comparison between PPT and TKAU, we did various experiments on several real and synthetic datasets. The experimental results show that PPT is more efficient than TKAU in terms of execution time and memory usage.

The remaining content of this paper is organized as follows: related works on traditional and top-k high average-utility mining are introduced in Section 2. Section 3 introduces some basic preliminaries and the problem statement of top-k HAUM. Furthermore, the pruning strategies and pseudo-code of the PPT algorithm are proposed in Section 4. The experimental results and analysis are presented in Section 5. Finally, Section 6 presents conclusion and future work.

2 RELATED WORK

In this section, we discuss the relationship between Web of Things and data mining technique, and then briefly review some studies about top-k itemset mining technologies with average-utility factor.

2.1 Web of things and data mining

Internet of Things (IoT) [1, 23] and Web of Things (WoT) [43] have been more and more popular to collect sensing data and build intelligent services and applications."Cloud Manufacturing" (abbreviated as CMfg) is a new manufacturing paradigm developed from existing cloud computing, IoT, and virtualization technologies [29, 41]. Due to the on-demand manufacturing service, CMfg system aims to provide products with high utility value, low cost, and global manufacturing services. Forecasting results and then automatically achieve dynamic production is one of the highlights of CMfg systems [40]. Given a simplified example that a manufacturer has offered a group of computers to retailers, manufacture can easily obtain the sale volume of computer and adjust shipments in time. This means retails no longer need to purchase equipment and other resources, by consulting through the public platform to buy and lease manufacturing capacity. However, an accurate sales analysis rather than a massive sales record can give manufacturers a lot of help in making decision on whether they should prepare for production in advance. Besides, enormous amounts of raw data are generated in manufacturing industry [7]. How to take advantage of these information is a valuable issue in WoT and data mining. Fortunately, data mining technique, such as interesting pattern discovery from database [12, 14, 25], aims to discover knowledge in big data, which effectively offers data analysis and prediction services. Thus, along with the previous content, applying UPM algorithms, especially high average-utility itemset mining (HAUIM for short) technology, into Web of Things is feasible.

2.2 Top-k high average-utility itemset mining

So far, many utility-driven mining algorithms [11, 13, 16] have been developed. As mentioned above, most of them have to scan database repeatedly. This case will wast a lot of runtime and memory. Since Hong *et al.* [17] firstly developed high average-utility concept, a multitude of investigators have hastened to improve HAUI technologies [18, 19, 27, 34, 38, 42]. However, a slight increase or decrease of the minimum average-utility threshold will cause a significance variation of the number of itemsets visited [37]. As we describe in previous section, how to obtain a suitable minimum utility (*minUtil*) threshold is a disturbing problem. In traditional top-k high-utility itemset mining (HUIM) domain, all approaches use parameter k instead of *minUtil* to obtain the desired itemsets. In top-k HUIM algorithms, the *minUtil* will be initial as 0 or 1. Thus, the *minUtil* raising strategy plays a key role during the mining process. For example, a list-based algorithm called TKO [36] used PE-matrix structure to raise *minUtil*. Moreover, the utility of promising itemsets during mining process are also considered as new *minUtil* values. This useful method has been applied in the following top-k HUIM technologies [5, 8, 16].

At the same time, inspired by traditional top-k HUIM, Wu *et al.* [39] first proposed a novel depth-first mechanism technology named TKAU to solve the threshold setting issue of traditional HAUIM. TKAU relies on a new list structure called AUO-List to compactly store itemset information. An efficient pruning strategy based on local maximum average-utility upper-bound is used to reduce the number of unpromising itemsets before constructing their AUO-Lists. Another utilized pruning strategy (named EMUP) helps the algorithm perform well in sparse dataset. Meanwhile, TKAU also adopts several threshold raising strategies like RIU, CAD and EPBF. Especially, the CAD is a revision of CUD strategy employed by the KHMC algorithm [8]. It can raise the current threshold by the average-utility of 2-itemsets. However, generating the lists for itemsets requires a significant amount of memory, since massive lists need to be maintained in memory while searching a long branch. To author's best knowledge, since the year 2018, there are no more studies dedicated to developing more advancing algorithms about mining top-k HAUIs.

3 PRELIMINARIES

Some basic definitions of our novel algorithm adopted are shown in Table 1, and other properties will be introduced in this section. Readers can refer to the studies [39, 45] for more details. In addition, Table 2 is a simple quantitative transaction database as running example in this paper. It is consisted of eight transactions with five distinct items {A, B, C, D, E}, and the corresponding external utility of each item is defined as \$2, \$4, \$2, \$3, and \$6, respectively.

Table 1: Summary of definitions

Symbol	Description
I	A finite set of n items, $I = \{x_1, x_2, \ldots, x_n\}$.
X	An itemset $X = \{x_1, x_2, \ldots, x_i\}$, where $1 \leq i \leq n$.
T_j	A subset of I, and each $x_i \in T_j$ is distinct.
$\mathcal{D}$	A transaction database, $\mathcal{D} = \{T_1, T_2, \ldots, T_j\}$.
δ	A changing minimum average-utility threshold (initial as 0 or 1).
$eu(x_i)$	Each item $x_i \in I$ has an external utility (e.g., unit profit).
$iu(x_i, T_j)$	Each item x_i in T_j has an internal utility (e.g., quantity).
$u(X, T_j)$	The utility of X in T_j where $u(X, T_j) = \sum_{x_i \in X}(eu(x_i) \times iu(x_i, T_j))$.
$u(X)$	The overall utility of itemset X in $\mathcal{D}$ where $u(X) = \sum_{X \subseteq T_j \wedge T_j \subseteq \mathcal{D}} u(X, T_j)$.
$tu(T_j)$	The utility of T_j where $tu(T_j) = \sum_{x_i \in T_j}(eu(x_i) \times iu(x_i, T_j))$.
$tmu(X, T_j)$	The maximum utility of itemset X in transaction T_j, where $x_i \in T_j$ and $X \subseteq T_j$.

Table 2: A sample quantitative database

TID	Transaction (item, quantity)
T_1	(B, 2), (C, 2), (E, 1)
T_2	(B, 4), (D, 2)
T_3	(A, 1), (C, 3), (E, 1)
T_4	(A, 2), (B, 2), (D, 1), (E, 3)
T_5	(B, 2), (C, 8)
T_6	(A, 6), (B, 5), (C, 4)
T_7	(B, 4), (C, 4), (D, 2), (E, 1)
T_8	(B, 2), (C, 3)

Definition 3.1. (**The average-utility of an itemset**) The average-utility of X in T_j is $au(X, T_j) = \frac{u(X, T_j)}{|X|}$, where $|X|$ represents the size or length of X. Similarly, the average-utility of X in $\mathcal{D}$ is denoted as $au(X) = \sum_{T_j \subseteq \mathcal{D}} au(X, T_j)$.

Definition 3.2. (**High average-utility itemset**) If $au(X)$ is no less than the current δ, we suppose that X is a high average-utility itemset (HAUI for short). Otherwise, X is a low average-utility itemset which is not interested.

Definition 3.3. (**Top-k high average-utility itemsets**) In this paper, the threshold δ will be initialized as 0 or 1. Thus, k is the only parameter specified by user. The set of k HAUIs with the highest utilities in $\mathcal{D}$ are defined as TopHAUIs.

Definition 3.4. The average-utility upper-bound (abbreviated as *auub*) of an itemset is $auub(X) = \sum_{X \subseteq T_j \wedge T_j \subseteq D} tmu(X, T_j)$ [20].

If $auub(X)$ is no less than the current δ, we suppose that X is a high average-utility upper-bound itemset (HAUUBI for short). This represents X is a potential HAUI or candidate which needs to check its real average-utility. Similar to the transaction-weighted utilization (TWU) [28], *auub* can be used to overestimate an itemset where $auub(X) \geq au(X)$. Furthermore, the *auub* is also downward closed: if an itemset is not HAUUBI, all its supersets cannot be HAUUBIs. Note that the proof detail has been provided in previous studies [17, 20, 39].

Problem statement: Given a transaction database $\mathcal{D}$ and a user desired number of HAUIs (aka k), then the problem of top-k high average-utility itemset mining (simplified as top-k HAUIM) is discovering the rank k highest average-utility itemsets in $\mathcal{D}$.

4 THE PPT ALGORITHM

In this section, we present the PPT algorithm for mining a complete set of TopHAUIs. Two efficient techniques, called database projection and transaction merging, are introduced in Subsection 4.1. Subsection 4.2 introduces the effective pruning strategies, and Subsection 4.3 introduces how to calculate some upper-bounds in linear time and space. Moreover, Subsection 4.4 presents the threshold raising strategies. At last, Subsection 4.5 shows the pseudo-code of the PPT algorithm.

4.1 Projection and merging technologies

Our novel algorithm is a depth-first method. The search space can be represented as a set-enumeration tree [32] whose root is empty. In this tree, from up to down, the first level nodes are given distinct items (which belongs to I), the second level nodes are 2-itemsets (means the size of the itemset is 2), and so on. To avoid generating the same nodes repeatedly, a global order $\prec$ is denoted as the ascending order of *auub* values of items. For example, according to the Table 2, $auub(A)$ = \$44, $auub(B)$ = \$102, $auub(C)$ = \$74, $auub(D)$ = \$50, and $auub(E)$ = \$48 respectively. Hence, we can obtain an order "$A \prec E \prec D \prec C \prec B$". Specially, if *auub* of items are equal, each item sorts in the lexicographical order. All in all, the global order plays an important role in projection and merging process.

Definition 4.1. (**Extension** [45]) An 2-itemset can be obtained by an item extending another distinct item. We suppose $E(X)$ is the set of all extended items of itemset X according to the depth-first method, where $E(X) = \{x_j \mid x_i \prec x_j, \forall x_i \in X \land x_j \in I\}$. Thus, an extension itemset X' of X appears in a subtree of X in set-enumeration tree. The formula can be denoted as $X' = X \cup \{x_j\}$, where $x_j \in E(X)$.

Definition 4.2. (**Remaining items**) Given an itemset X in transaction T_j, we suppose remaining items are $re(X, T_j) = \{x_i \mid x_i \in T_j \land x_z \prec x_i, \forall x_z \in X\}$. In addition, the number of the remaining items of X in T_j is $|re(X, T_j)|$.

Definition 4.3. (**Maximum utility of local remaining items**) In a transaction T_j, local remaining items are the extension items of an itemset X. Thus the maximum utility of local remaining item of X is defined as $lru(X, T_j) = Max(\{u(x_i) | x_i \in re(X, T_j)\})$.

Definition 4.4. (**Maximum utility of remaining items**) Given an itemset X and an item x_z, where $x_z \in E(X)$, the maximum remaining utility of x_i w.r.t. X is $mru(X, T_j) = Max(\{u(x_i) \mid re(X \cup x_z, T_j) \cup \{x_z\}\})$.

Definition 4.5. (**Database projection technology** [45]) Consider the definition of extension, it is clearly that all items $x_i \notin E(X)$ can be directly ignored when traversals the subtrees of itemset X in each transaction T_j. Hence, a database without these irrelevant items is called a projected database. Formally, we use X-T_j represents the projection of X in T_j, where $X\text{-}T_j = \{x_i \mid x_i \in T_j \land x_i \in E(X)\}$. Similarly, the projection of X in $\mathcal{D}$ is defined as $X\text{-}\mathcal{D} = \{X\text{-}T_j \mid T_j \in \mathcal{D} \land X\text{-}T_j \neq \emptyset\}$.

Definition 4.6. (**Transaction merging technology** [45]) In a projected database, there often appears some identical transactions. A transaction T_i is identical to another transaction T_j only if they contain same items. To reduce the cost of database scans, these sets of identical transactions can be replaced by a single new transaction T_M, where $T_1 = T_2 = \ldots = T_j$. And the internal utility of each item x_i in T_M is denoted as $iu(x_i, T_k) = \sum_{1 \leq k \leq j} iu(x_i, T_k)$.

In order to facility the transaction merging process, we initially sort transactions of the original database by lexicographical order, and then compare elements of two transactions starting backwards. The study in [45] described four cases will happen in detail. Due to the limitation of this paper, we simply conclude that the cost of merging all transactions in a projected database is $O(ml)$, where m denotes the number of the transactions and l represents the average length of transaction.

4.2 Several pruning strategies

In this subsection, we discuss the efficient pruning strategies adopted in our novel algorithm. We design a novel revised version of upper-bound (i.e., local utility) employed by the EFIM algorithm [45] and utilize an efficient upper-bound called maximum average-utility [42]. Before calculating these upper-bounds (except *auub*), we need to prune those unpromising items. Thus, these upper-bounds will be tighter and perform better when mining HAUIs.

Strategy 1. ***(auub-based pruning strategy)*** Considering the previous *auub* definition, if the *auub* value of an itemset X is less than δ, X and its supersets can be pruned in the search space directly.

Definition 4.7. (**Local average-utility**) Given an itemset X and an extension item $x_i \in E(X)$. The local average-utility of x_i is denoted as $lau(X, x_i) = \sum_{\{X \cup x_i\} \subseteq T_j \land T_j \subseteq \mathcal{D}} \frac{u(X) + lru(X, T_j)}{|X|}$.

Property 1. ***(Overestimation using the local average-utility)*** Let be an itemset X, an item $x_i \in E(X)$, and X' is one of super-itemsets of X such as $x_i \in X'$. Then $lau(X, x_i)$ is always higher than $au(X')$.

Proof. For $\forall x_i \in E(X)$, we assume $Y = X \cup x_i$, and X' is a super-itemset of X where $x_i \in X'$, and $Y \subseteq X'$. Then we have:

$$au(X') = \sum_{X' \subseteq T_n \land T_n \subseteq \mathcal{D}} \frac{u(X', T_n)}{|X'|}$$
$$= \sum_{X' \subseteq T_n \land T_n \subseteq \mathcal{D} \land x_j \in E(X)} \frac{u(X, T_n) + u(x_j, T_n)}{|X'|}, \text{ and}$$
$$lau(X, x_i) = \sum_{Y \subseteq T_m \land T_m \subseteq \mathcal{D}} \frac{u(X, T_m) + lru(X, T_m)}{|X|}.$$
$\because T_n \subseteq T_m$, $u(x_j, T_n) \leq lru(X, T_n)$, and $|X| < |X'|$,
$\therefore au(X') < lau(X, x_i)$.

□

Strategy 2. ***(Local average-utility pruning strategy)*** Given an itemset X and an item $x_i \in E(X)$. If $lau(X, x_i) < \delta$, then all itemsets containing $\{X \cup x_i\}$ are of low average-utility. This represents x_i can be ignored when exploring subtrees of X.

Definition 4.8. (**Maximum average-utility** [42]) Given an itemset X and its extension item $x_i \in E(X)$ in transaction T_j, the maximum average-utility of X in T_j is defined as

$$mau(X, x_i) = \begin{cases} \frac{u(X, T_j) + mru(X, T_j) \times |re(X, T_j))|}{|X| + |re(X, T_j)|}, & mru(X, T_j) > \frac{u(X, T_j)}{|X|} \\ \frac{u(X, T_j) + mru(X, T_j)}{|X| + 1}, & 0 < mru(X, T_j) \leq \frac{u(X, T_j)}{|X|} \\ 0, & mru(X, T_j) = 0. \end{cases}$$

Then, the maximum average-utility of X in database $\mathcal{D}$ is denoted as $mau(X) = \sum_{\{X \cup x_i\} \subseteq T_j \land T_j \subseteq \mathcal{D}} mau(X, x_i)$.

Property 2. ***(Overestimation using the maximum average-utility)*** Given an itemset X and its superset X', $mau(X)$ is always higher than or equal to $au(X')$.

Proof. In the first condition, $mru(X, T_j) > au(X, T_j)$ represents that the average-utility of X' can be maximally increased by the $mru(X, T_j) \times |re(X, T_j))|$ value in transaction T_j. In the second condition, $0 < mru(X, T_j) \leq au(X, T_j)$ means that the average-utility of X' can be minimally decreased by the $mru(X, T_j) \times 1$ value in transaction T_j. In the last condition, $mru(X, T_j) = 0$ can be viewed as that there is no extension of X. □

Strategy 3. ***(Maximum average-utility pruning strategy)*** If the maximum average-utility of itemset X is less than δ, all supersets of X are low average-utility itemsets. In other words, it is no need to search the subtrees of node X.

4.3 Calculate upper-bounds using utility array

In this subsection, we adopt an array-based structure called utility-bin. This efficient structure can compute three upper-bounds in linear time and space. Given an itemset X, we assume $LauSet(X) = \{x_i \mid x_i \in E(X) \wedge lau(X, x_i) \geq \delta\}$ and $MauSet(X) = \{x_i \mid x_i \in E(X) \wedge mau(X, x_i) \geq \delta\}$.

Definition 4.9. (**Utility-Bin** [45]) A utility-bin array U is denoted as $U[x_i]$, where $x_i \in I$ and length $|U| = |I|$. $U[x_i]$ is initialized as 0 and then is used to store utility value.

Calculating the *auub* of each item by *U*. For each $x_i \in I$, $U[x_i] = U[x_i] + tmu(T_j)$. After scanning all transactions of database, it is clear that U contains *auub* of each item $x_i \in I$.

Calculating the *lau* of each itemset by *U*. Given a utility-bin array U initialized with 0 for each T_j, $U[x_i] = U[x_i] + lau(X, x_i)$, where $x_i \in E(X) \wedge \{X \cup x_i\} \subseteq T_j$.

Calculating the *mau* of each itemset by *U*. Given a utility-bin array U initialized with 0 for each T_j, $U[x_i] = U[x_i] + mau(X, x_i)$, where $x_i \in E(X) \wedge \{X \cup x_i\} \subseteq T_j$.

4.4 Threshold raising strategies

Here, we propose a revised version of RIU strategy employed by the REPT algorithm [31], which is adopted in our proposed PPT algorithm after scanning the database.

Strategy 4. ***(RAU: Raising the threshold by average-utility of items)*** Based on the definition of average-utility of itemsets, each item $x_i \in I$ in $\mathcal{D}$ can be viewed as a 1-itemset. Then the real average utility $rau(x_i)$ is equal to $au(x_i)$ actually. In other words, these 1-itemsets also may be part of HAUIs. Therefore, let *rau-list* = $\{rau(x_1), rau(x_2), \ldots, rau(x_i)\}$ be a set of real average-utility of each item $x_i \in I$, and the k-th highest value in *rau-list* is denoted as $rau(x_k)$. If $rau(x_k)$ is no less than the current δ, we can safely raise δ to that value.

Then, after the RAU strategy increases the current threshold, we utilize the CAD strategy employed by study [39]. It is a revised version of CUD strategy employed by KHMC [8]. The CAD strategy uses a matrix (named CADM) to store the average-utility of 2-itemsets $X = x_i \cup x_j$ ($x_i \neq x_j$ and $x_i, x_j \in I$). Moreover, using hashmaps instead of a triangular matrix and considering items with $auub \geq \delta$ will save more memory.

Strategy 5. ***(CAD: Co-occurrence average-utility descending order raising)*** Based on previous content, the CADM structure actually stores the real average-utility of 2-itemsets. Therefore, let *cad-list* = $\{cad(X_1), cad(X_2), \ldots, cad(X_i)\}$ where $cad(X_i)$ is the average-utility of 2-itemset X_i, and the k-th highest value in *cad-list* is denoted as $cad(X_k)$. If $cad(X_k)$ is no less than the current δ, we can safely raise δ to that value.

At last, we adopt another common strategy employed by TKO algorithm called RUC [36]. In this paper, it uses a *ruc-list* structure to maintain top-k HAUIs. Each element of *ruc-list* are sorted by descending order of their own average-utility, and the maximum size of *ruc-list* is k.

Strategy 6. ***(RUC: Raising the threshold by average-utility of candidates)*** During the mining process, if the average-utility of an itemset X is higher than the current δ, X will be added into *ruc-list*. We suppose the k-th highest value in *ruc-list* is denoted as $ruc(x_k)$. If $ruc(x_k)$ is no less than the current δ, it can be safely that we raise δ to that value.

4.5 Procedure of the proposed algorithm

So far, the key technologies, pruning strategies with upper-bounds, and threshold raising strategies have been presented in detail. In this subsection, we continue introduce the complete procedure-code of the proposed PPT algorithm. **Algorithm** 1 posts the main procedure of the PPT algorithm.

Algorithm 1: Proposed PPT algorithm

Input: $\mathcal{D}$: a transaction database, k: the desired number of HAUIs.
Output: A priority queue of top-k HAUIs.

1 initialize itemset α as $\emptyset$;
2 initialize utility list *rau-list* as $\emptyset$;
3 set the initial minimum average-utility δ as 1;
4 initialize an empty priority queue *TopHAUIs* of size k;
5 scan $\mathcal{D}$, compute real utility of all items $x_i \in I$, and store *rau-list*(x_i) in *rau-list*;
`// the RAU strategy`
6 raise the current δ by *rau-list*;
7 using utility-bin array U calculates *auub* of all items x_i;
8 sort each item in U by the global order $\prec$ of *auub* increasing values;
9 compute $LauSet(\alpha) = \{x_i \mid auub(x_i) \geq \delta\}$;
`// the auub strategy`
10 scan $\mathcal{D}$, remove items $x_i \notin LauSet(\alpha)$ in each $T_j \in \mathcal{D}$;
`// the transaction merging technology`
11 merging and then deleting empty transactions;
12 sort all changed transactions and their consist of items, then obtain a new database $\mathcal{D}'$;
13 compute $MauSet(\alpha) = \{x_i \mid x_i \in LauSet(\alpha) \wedge mau(\alpha, x_i) \geq \delta\}$;
`// the CAD strategy`
14 scan $\mathcal{D}'$, and create *cad-list* to raise δ;
15 call **Search**(α, $\mathcal{D}'$, $MauSet(\alpha)$, $LauSet(\alpha)$, δ, *TopHAUIs*);
16 **return** *TopHAUIs*

It takes a transaction database $\mathcal{D}$ and a desired number of HAUIs k as input and output a priority queue of top-k HAUIs as result. In Lines 1-4, the algorithm initializes several key parameters (i.e., α, *rau-list*, δ and *TopHAUIs*). Then the algorithm scans the database and computes each *rau-list*(x_i) of item $x_i \in I$ in Line 5. After that, strategy 4 raises the current threshold δ for the first time (Line 6). According to the utility-bin array structure, the *auub* upper-bound of each x_i is obtained and we sort them by the *auub* increasing order (Lines 7 and 8). The algorithm then finds all extension items of α by comparing *auub* values and δ (Line 9). Note that the α is still empty in the case. The database is scanned by the second time and then we remove all unpromising items by Strategy 1 (Line 10). At the same time, due to the modified transactions, we can obtain a novel database $\mathcal{D}'$ (Lines 11 and 12). Then, the algorithm computes the maximum average-utility of each promising item by utility-bin array structure (Line 13). In Line 14, the algorithm scans the new database and then adopts Strategy 5 to raise δ again. Thereafter, the algorithm calls the recursive *Search* procedure (**Algorithm** 2) to mine promising itemsets with depth-first mechanism (Line 15). Finally, the PPT algorithm will return a priority queue of top-k HAUIs as result.

Algorithm 2: The Search procedure

Input: α: the current itemset, α-$\mathcal{D}$: the projected database of α, *MauSet*(α): the *MauSet* items of α, *LauSet*(α): the *LauSet* items of α, δ: a minimum average-utility threshold, and *TopHAUIs*: a priority queue of top-k HAUIs.
Output: a set of top-k HAUIs which contains extensions of α.

1 **for** *each item* $x_i \in$ *MauSet*(α) **do**
2   $\beta = \alpha \cup \{x_i\}$;
  `// the transaction merging technology`
3   scan α-$\mathcal{D}$, calculate *au*(β), and then create β-$\mathcal{D}$;
  `// the RUC strategy`
4   **if** *au*(β) $\geq \delta$ **then**
5   **if** $|$*TopHAUIs*$| \geq k$ **then**
6   delete the k-th itemset in *TopHAUIs*;
7   **end**
8   add β into *TopHAUIs*, and update the current δ;
9   **end**
10   scan β-$\mathcal{D}$, calculate *mau*(β, x_i), and *lau*(β, x_i) where items $x_i \in$*LauSet*(α);
11   obtain *MauSet*(β) items where the extension items $x_i \in$ *LauSet*(α);
12   obtain *LauSet*(β) items where the extension items $x_i \in$ *LauSet*(α);
13   call **Search**(β, β-$\mathcal{D}$, *MauSet*(β), *LauSet*(β), *TopHAUIs*);
14 **end**

Algorithm 2 presents the *Search* procedure. It takes six parameters as input: the current extended itemset α, the projected database of α, the *MauSet* and *LauSet* items of α, the current minimum average-utility threshold δ and a priority queue of top-k HAUIs. The procedure mainly focuses on finding all single item extensions of α (that is $\beta = \alpha \cup x_i$), where x_i is *MauSet* item. For each β, the procedure first checks whether it is an HAUI or not. If it is true, according to the Strategy 6, the procedure adds β into *TopHAUIs* and then raises δ value if the length of *HAUIs* is already no less than k (Lines 4-10). At the same time, the projected database β-$\mathcal{D}$ is created in Line 3. The procedure scans β-$\mathcal{D}$ to obtain *MauSet* and *LauSet* items of β (Lines 11-13). At last, the procedure inputs the new parameters about β into the *Search* procedure until no extension item occurs (Line 14).

5 PERFORMANCE EVALUATION

To the best of our knowledge, TKAU is the most efficient algorithm for mining top-k HAUIs. We have tried our best to make the experimental TKAU algorithm performed as well as the results in study [39]. To evaluate the performance of upper-bounds of PPT, we also designed another version of PPT which did not adopt the maximum average-utility pruning strategy (named PPT*).

5.1 Experimental setup and data description

To demonstrate the effectiveness and efficiency of PPT, we have conducted some experiments. The experimental equipment is a PC with 64 bit Intel(R) Core(TM) i5-8300H @2.30GHz Intel Core Processor with 16GB RAM running on Windows 10. All tested algorithms are implemented by Java language.

We used four datasets to evaluate the performance of PPT, including three real dataset (chess, mushroom, and retail) and a synthetic dataset (T10I4D100K). As shown in Table 3, the retail is a sparse dataset and has the most distinct items. The chess is the densest dataset which average length of transaction is 37. All of these datasets can be downloaded from the open source website[1].

Table 3: Basic information about datasets

Dataset	#Trans	#Items	#AvgLen	#Type
chess	3,196	75	37	Dense
mushroom	8,124	119	23	Dense
retail	88,162	16,470	10.3	Sparse
T10I4D100K	100,000	870	10.1	Sparse

5.2 Efficiency evaluation

In Figure 1, we first evaluated the runtime usage of three algorithms. In Figure 1, in chess and mushroom datasets, both PPT and PPT* perform better than TKAU. The most important reason is that PPT utilizes the transaction merging technique on the projected database. By replacing identical transactions with one transaction, the proposed algorithm significantly reduces the runtime of dataset scanning. Moreover, in sparse datasets, the gap between TKAU and PPT is not as much as that of dense datasets. This represents that our novel algorithm will have a better performance in mining cloud manufacturing data than that of TKAU. On the other hand, when k is 1,000, TKAU takes approximately three times execution time than that of PPT in Figure 1(d). In conclusion, the experimental results show that the performances of PPT and PPT* are better than TKAU in running time consumption.

Then, the memory usage is shown in Figure 2. It is clearly that PPT outperforms TKAU on all datasets. Due to chess is the densest dataset and merging technology is adopted, the memory consumption of TKAU is nearly up to one order of magnitude more than that of PPT and PPT*. In Figure 2(a), TKAU still costs 444 MB though k is 100. Consider the mushroom and T10I4D100K datasets, PPT always takes less memory than PPT* with various k. This means maximum average-utility upper-bound plays a important role during mining process. Therefore, we can take a conclusion that PPT performs better than TKAU in terms of memory consumption.

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

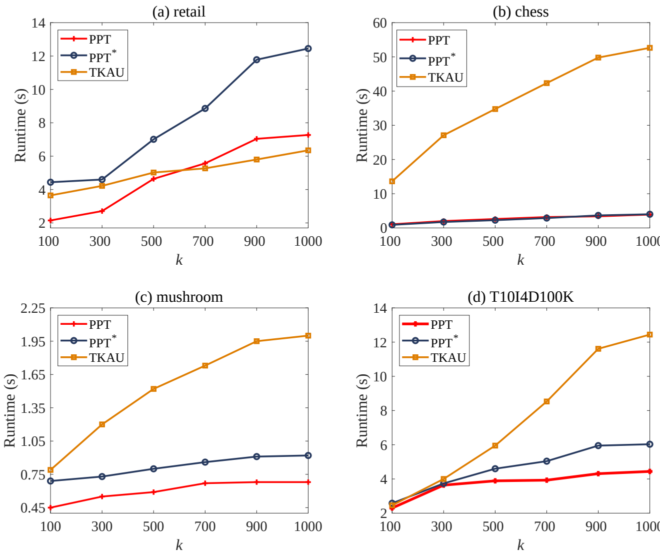

Figure 1: Runtime usage with various *k*.

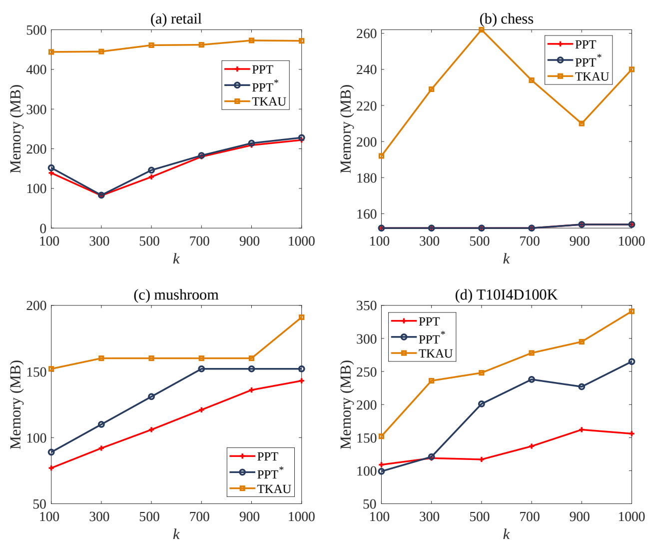

Figure 2: Memory consumption with various *k*.

We also have compared the ability of the PPT and TKAU algorithms in terms of pruning. Table 4 summarizes the results of the algorithms when k is 100, 300, 500, 700, 900 and 1,000, respectively. We can learn that PPT usually generates less candidates in sparse datasets. Although PPT* has always generate massive candidates which is bigger than that of TKAU greatly, PPT* still performs better than TKAU in terms of runtime and memory consumption (Figure 1 and Figure 2). The reason is TKAU is a list-based algorithm, and a list contains many tuples. If TKAU discovers an itemset, a new AUO-List will be constructed and will be kept in memory all the time. To obtain high level itemset from AUO-Lists, TKAU must scan two lists to determine whether the new itemset can be generated. In addition, comparing the number of candidates generated between

Table 4: The number of candidates generated

Dataset	k	PPT	PPT*	TKAU
chess	100	100,080	121,419	187,166
	300	364,067	439,036	396,938
	500	815,713	960,204	547,087
	700	1,299,749	1,527,461	680,409
	900	1,699,787	2,001,297	802,727
	1000	2,052,788	2,406,194	861,859
retail	100	213	793	202
	300	769	2,348	821
	500	1,331	3,885	1,694
	700	1,772	5,278	2,785
	900	2,268	6,629	4,004
	1000	2,491	7,297	4,641
mushroom	100	659	4,083	1,595
	300	6,365	21,504	4,821
	500	15,611	45,305	7,371
	700	23,403	74,003	10,055
	900	35,662	104,130	12,509
	1000	44,337	125,000	13,604
T10I4D100K	100	399	505	100
	300	1,066	2,308	672
	500	1,893	4,553	1,954
	700	2,922	6,987	4,226
	900	3,953	8,962	6,450
	1000	4,543	9,876	7,281

PPT and PPT*, the results prove that the maximum average-utility upper-bound is vital in our novel algorithm.

5.3 Scalability evaluation

Finally, we took two experiments in terms of the scalability of three algorithms in T10I4D100K. Especially, the symbol "20K" means the tested dataset has 20,000 transactions and others so on. We respectively set k as 1,000 and 5,000 in two experiments. Figure 3 shows the runtime and memory usage both linearly raise with increased dataset size. In addition, PPT always performs better than TKAU in all cases. All in all, we can draw a conclusion that PPT has a good scalability in terms of execution time and memory consumption.

6 CONCLUSION AND FUTURE WORK

Manufacturing datasets mostly contain numerous data which can be used to improve data intelligence. In this paper, by addressing the problem of data analytic for cloud manufacturing, we proposed an efficient data mining algorithm for discovering top-k profitable patterns with average-utility. Several novel upper-bounds and their corresponding pruning strategies are utilized to improve the performance of the novel algorithm. Especially, the maximum

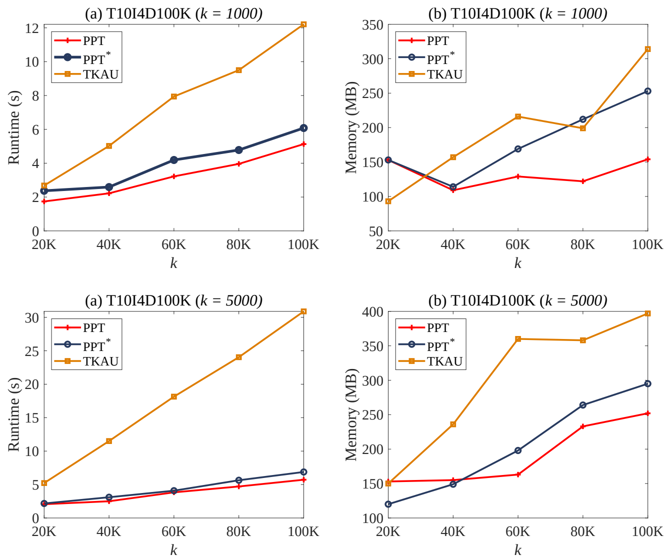

Figure 3: Scalability experiments on T10I4D100K.

average-utility pruning strategy can reduce massive useless items before mining high level itemsets. On the other hand, the combine maximum average-utility upper-bound and local average-utility upper-bound has greatly improve the pruning performance of the proposed algorithm. The results on both real and synthetic datasets show that the novel algorithm is more efficient in terms of execution time and memory usage than the existing algorithm.

Although the novel algorithm has a good performance in terms of runtime and memory consumption, it still generates too many uninteresting candidates during the intermediate mining processes. Thus, designing more tighter upper-bounds will be the next direction in our future work. Furthermore, using the novel algorithm to discover closed patterns or applying PPT technology in distributed environment is interesting for future work.

7 ACKNOWLEDGMENT

This research was supported in part by the National Natural Science Foundation of China (Grant Nos. 61902079 and 62002136), Guangzhou Basic and Applied Basic Research Foundation (Grant Nos. 202102020928 and 202102020277), and the project of Guangzhou Science and Technology (Grant No. 202007010004).

REFERENCES

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

[2] 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.

[3] Jiahui Chen, Xu Guo, Wensheng Gan, Chien-Ming Chen, Weiping Ding, and Guoting Chen. 2020. OSUMI: On-shelf utility mining from itemset-based data. In *IEEE International Conference on Big Data*. IEEE, 5340–5349.

[4] Jiahui Chen, Xu Guo, Wensheng Gan, Chien-Ming Chen, Weiping Ding, and Guoting Chen. 2022. On-shelf utility mining from transaction database. *Engineering Applications of Artificial Intelligence* 107 (2022), 104516.

[5] Jiahui Chen, Shicheng Wan, Wensheng Gan, Guoting Chen, and Hamido Fujita. 2021. TOPIC: Top-k high-utility itemset discovering. *arXiv preprint, arXiv:2106.14811* (2021).

[6] Chunjung Chu, Vincent S Tseng, and Tyne Liang. 2008. An efficient algorithm for mining temporal high utility itemsets from data streams. *Journal of Systems and Software* 81, 7 (2008), 1105–1117.

[7] Alican Dogan and Derya Birant. 2021. Machine learning and data mining in manufacturing. *Expert Systems with Applications* 166 (2021), 114060.

[8] Quang-Huy Duong, Bo Liao, Philippe Fournier-Viger, and Thu-Lan Dam. 2016. An efficient algorithm for mining the top-k high utility itemsets, using novel threshold raising and pruning strategies. *Knowledge-Based Systems* 104 (2016), 106–122.

[9] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Bay Vo, Tin-Truong Chi, Ji Zhang, and Hoai Bac Le. 2017. A survey of itemset mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 7, 4 (2017), e1207.

[10] Wensheng Gan, Lili Chen, Shicheng Wan, Jiahui Chen, and Chien-Ming Chen. 2022. Anomaly Rule Detection in Sequence Data. *IEEE Transactions on Knowledge and Data Engineering. DOI: 10.1109/TKDE.2021.3139086* (2022), 1–14.

[11] Wensheng Gan, Chun-Wei Lin, Philippe Fournier-Viger, Han-Chieh Chao, Vincent 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.

[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, 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.

[14] Wensheng Gan, Jerry Chun-Wei Lin, Philippe Fournier-Viger, Han-Chieh Chao, and Philip S Yu. 2021. HUOPM: High-utility occupancy pattern mining. *IEEE Transactions on Cybernetics* 50, 3 (2021), 1195–1208.

[15] Wensheng Gan, Jerry Chun-Wei Lin, Philippe Fournier-Viger, Han-Chieh Chao, Justin Zhan, and Ji Zhang. 2018. Exploiting highly qualified pattern with frequency and weight occupancy. *Knowledge and Information Systems* 56, 1 (2018), 165–196.

[16] Wensheng Gan, Shicheng Wan, Jiahui Chen, Chien-Ming Chen, and Lina Qiu. 2020. TopHUI: Top-k high-utility itemset mining with negative utility. In *IEEE International Conference on Big Data*. IEEE, 5350–5359.

[17] Tzung-Pei Hong, Cho-Han Lee, and Shyue-Liang Wang. 2009. Mining high average-utility itemsets. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*. IEEE, 2526–2530.

[18] Heonho Kim, Unil Yun, Yoonji Baek, Jongseong Kim, Bay Vo, Eunchul Yoon, and Hamido Fujita. 2021. Efficient list based mining of high average utility patterns with maximum average pruning strategies. *Information Sciences* 543 (2021), 85–105.

[19] Jongseong Kim, Unil Yun, Eunchul Yoon, Jerry Chun-Wei Lin, and Philippe Fournier-Viger. 2020. One scan based high average-utility pattern mining in static and dynamic databases. *Future Generation Computer Systems* 111 (2020), 143–158.

[20] Guo-Cheng Lan, Tzung-Pei Hong, and Vincent S Tseng. 2012. Efficiently mining high average-utility itemsets with an improved upper-bound strategy. *International Journal of Information Technology & Decision Making* 11, 05 (2012), 1009–1030.

[21] Tuong Le, Bay Vo, Van-Nam Huynh, Ngoc Thanh Nguyen, and Sung Wook Baik. 2020. Mining top-k frequent patterns from uncertain databases. *Applied Intelligence* 50, 5 (2020), 1487–1497.

[22] Jingran Li, Fei Tao, Ying Cheng, and Liangjin Zhao. 2015. Big data in product lifecycle management. *The International Journal of Advanced Manufacturing Technology* 81, 1 (2015), 667–684.

[23] Shancang Li, Li Da Xu, and Shanshan Zhao. 2015. The internet of things: a survey. *Information Systems Frontiers* 17, 2 (2015), 243–259.

[24] 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.

[25] Jerry Chun-Wei Lin, Wensheng Gan, Philippe Fournier-Viger, Tzung-Pei Hong, and Vincent S Tseng. 2016. Fast algorithms for mining high-utility itemsets with various discount strategies. *Advanced Engineering Informatics* 30, 2 (2016), 109–126.

[26] Jerry Chun-Wei Lin, Wensheng Gan, Philippe Fournier-Viger, Tzung-Pei Hong, and Vincent S Tseng. 2016. Weighted frequent itemset mining over uncertain databases. *Applied Intelligence* 44, 1 (2016), 232–250.

[27] Jerry Chun-Wei Lin, Shifeng Ren, Philippe Fournier-Viger, and Tzung-Pei Hong. 2017. EHAUPM: Efficient high average-utility pattern mining with tighter upper bounds. *IEEE Access* 5 (2017), 12927–12940.

[28] Ying Liu, Weikeng 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.

[29] Yongkui Liu, Lihui Wang, and Xi Vincent Wang. 2018. Cloud manufacturing: latest advancements and future trends. *Procedia Manufacturing* 25 (2018), 62–73.

[30] Jinbao Miao, Shicheng Wan, Wensheng Gan, Jiayi Sun, and Jiahui Chen. 2021. Targeted High-Utility Itemset Querying. In *Proceedings of the IEEE International*

Conference on Big Data. IEEE, 5534–5543.

[31] Heungmo Ryang and Unil Yun. 2015. Top-k high utility pattern mining with effective threshold raising strategies. *Knowledge Based Systems* 76 (2015), 109–126.

[32] Ron Rymon. 1992. Search through systematic set enumeration. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*. 539–550.

[33] Alberto Segura-Delgado, Augusto Anguita-Ruiz, Rafael Alcalá, and Jesús Alcalá-Fdez. 2022. Mining high average-utility sequential rules to identify high-utility gene expression sequences in longitudinal human studies. *Expert Systems with Applications* (2022), 116411.

[34] Krishan Kumar Sethi and Dharavath Ramesh. 2020. High average-utility itemset mining with multiple minimum utility threshold: A generalized approach. *Engineering Applications of Artificial Intelligence* 96 (2020), 103933.

[35] Baien Shie, Philip S Yu, and Vincent S Tseng. 2013. Mining interesting user behavior patterns in mobile commerce environments. *Applied Intelligence* 38, 3 (2013), 418–435.

[36] Vincent S Tseng, Chengwei Wu, Philippe Fournier Viger, and Philip S Yu. 2015. Efficient algorithms for mining top-k high utility itemsets. *IEEE Transactions on Knowledge and Data Engineering* 28, 1 (2015), 54–67.

[37] Chengwei Wu, Baien Shie, Vincent S Tseng, and Philip S Yu. 2012. Mining top-k high utility itemsets. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 78–86.

[38] Jimmy Ming-Tai Wu, Jerry Chun-Wei Lin, Matin Pirouz, and Philippe Fournier-Viger. 2018. TUB-HAUPM: Tighter upper bound for mining high average-utility patterns. *IEEE Access* 6 (2018), 18655–18669.

[39] Ronghui Wu and Zhan He. 2018. Top-k high average-utility itemsets mining with effective pruning strategies. *Applied Intelligence* 48, 10 (2018), 3429–3445.

[40] Chunyang Yu, Wei Zhang, Xun Xu, Yangjian Ji, and Shiqiang Yu. 2018. Data mining based multi-level aggregate service planning for cloud manufacturing. *Journal of Intelligent Manufacturing* 29, 6 (2018), 1351–1361.

[41] Minghai Yuan, Kun Deng, WA Chaovalitwongse, and Hongyan Yu. 2018. Research on technologies and application of data mining for cloud manufacturing resource services. *The International Journal of Advanced Manufacturing Technology* 99, 5 (2018), 1061–1075.

[42] Unil Yun and Donggyu Kim. 2017. Mining of high average-utility itemsets using novel list structure and pruning strategy. *Future Generation Computer Systems* 68 (2017), 346–360.

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

[44] 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.

[45] Souleymane Zida, Philippe Fournier-Viger, Jerry Chun-Wei Lin, Chengwei 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.