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

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, HU) is higher than or equal to the user-specified minimal utility threshold. Obviously, HUs 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., HUs) 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 HUs 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-\(k\) HUs [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-\(k\) Profitable Patterns with average-uTility (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-\(k\) HAUIM. 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 waste 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 named EMUP is also adopted several threshold raising strategies like RIU, minUtil and HAUIs. Meanwhile, TKAU helps the algorithm perform well in sparse dataset. 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 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 itemsets or constructing their AUO-Lists requires a significant amount of memory, since massive lists need to be maintained in memory while searching a high average-utility upper-bound (abbreviated as auub) of an itemset is auub(X) = \( \sum_{t_j \subseteq X} \text{au}(X, T_j) \) [20]

\[
\text{au}(X, T_j) = \frac{u(x, T_j)}{|X|},
\]

If auub(X) is no less than the current \( \delta \), 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 \( \delta \) 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 \( D \) are defined as TopHAUIs.

Definition 3.4. The average-utility upper-bound (abbreviated as auub) of an itemset is auub(X) = \( \sum_{t_j \subseteq X} \text{au}(X, T_j) \) [20].

In this paper, 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 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 < 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 < E < D < C < 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) = {xj | xj < xj, ∀xj ∈ X ∧ xj ∈ 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 ∪ {xj}, where xj ∈ E(X).

Definition 4.2. (Remaining items) Given an itemset X in transaction Tj, we suppose remaining items are re(T, Tj) = {xj | xj ∈ Tj ∧ xj < xj, ∀xj ∈ X}. In addition, the number of the remaining items of X in Tj is |re(T, Tj)|.

Definition 4.3. (Maximum utility of local remaining items) In a transaction Tj, local remaining items are the extension items of an itemset X. Thus the maximum utility of local remaining item of X is defined as lr(T, Tj) = Max{|u(xj)| ∈ re(T, Tj)}.

Definition 4.4. (Maximum utility of remaining items) Given an itemset X and an item xz, where xz ∈ E(X), the maximum remaining utility of xz w.r.t. X is mru(T, Tj) = Max{|u(xz)| ∈ re(T, Tj)}.

Definition 4.5. (Database projection technology [45]) Consider the definition of extension, it is clearly that all items xj ∈ E(X) can be directly ignored when traversals the subtrees of itemset X in each transaction Tj. Hence, a database without these irrelevant items is called a projected database. Formally, we use X-Tj represents the projection of X in Tj, where X-Tj = {xj | xj ∈ Tj ∧ xj ∈ E(X)}. Similarly, the projection of X in D is defined as X-D = {X-Tj | Tj ∈ D ∧ X-Tj ≠ ∅}.

Definition 4.6. (Transaction merging technology [45]) In a projected database, there often appears some identical transactions. A transaction Tj is identical to another transaction Tj 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 Tm, where T1 = T2 = . . . = Tj. And the internal utility of each item xk in Tm is denoted as iu(xk, Tj) = Σ 1≤k≤l iu(xk, Tk).

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 xj ∈ E(X). The local average-utility of xj is denoted as lau(T, xj) = Σ i∈Tj∪xj∪{xj} |u(xj)| |X|.

Property 1. (Overestimation using the local average-utility) Let be an itemset X, an item xj ∈ E(X), and X’ is one of super-items of X such as xj ∈ X’. Then lau(T, xj) is always higher than au(X’).

Proof. For ∀xj ∈ E(X), we assume Y = X ∪ xj, and X’ is a super-itemset of X where xj ∈ X’, and Y ⊂ X’. Then we have:

\[ au(X’) = \sum_{X’ \subseteq Tn \land Tn \subseteq D} \frac{|u(X’, Tn)|}{|X’|} \]

\[ = \sum_{X’ \subseteq Tn \land Tn \subseteq D} \frac{|u(X) + u(xj, Tn)|}{|X|} \]

\[ \leq \sum_{X’ \subseteq Tn \land Tn \subseteq D} \frac{|u(X)| + |u(xj, Tn)|}{|X|} \]

\[ \leq \sum_{X’ \subseteq Tn \land Tn \subseteq D} \frac{|u(X) + u(xj, Tn)|}{|X|} \]

\[ ≺ \sum_{X’ \subseteq Tn \land Tn \subseteq D} \frac{|u(X)| + |u(xj, Tn)|}{|X|} \]

\[ \therefore au(X’) < lau(T, xj). \]

Strategy 2. (Local average-utility pruning strategy) Given an itemset X and an item xj ∈ E(X). If lau(T, xj) < δ, then all itemsets containing X ∪ xj are of low average-utility. This represents xj can be ignored when exploring subtrees of X.

Definition 4.8. (Maximum average-utility [42]) Given an itemset X and its extension item xj ∈ E(X) in transaction Tj, the maximum average-utility of X in Tj is defined as

\[ ma(T, xj) = \begin{cases} \frac{|u(xj, Tj)|}{|X|}, & |X| > 0 \\ \frac{|u(xj, Tj)|}{|X|}, & |X| = 0, \end{cases} \]

\[ mrd(T, xj) > \frac{|u(xj)|}{|X|}. \]

Then, the maximum average-utility of X in database D is denoted as ma(T, xj) = \sum_{X, \subseteq X, \subseteq D} ma(T, xj).

Property 2. (Overestimation using the maximum average-utility) Given an itemset X and its superset X’, ma(T, xj) is always higher than or equal to au(X’).

Proof. In the first condition, mrd(T, xj) > au(X’) represents that the average-utility of X’ can be maximally increased by the mrd(T, xj) × |re(X’, Tj)| value in transaction Tj. In the second condition, 0 < mrd(T, xj) ≤ au(X’) means that the average-utility of X’ can be minimally decreased by the mrd(T, xj) × |1 value in transaction Tj. In the last condition, mrd(T, xj) = 0 can be viewed as that there is no extension of 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 \( \text{LauSet}(X) = \{ x_i \mid x_i \in E(X) \wedge \text{lau}(X, x_i) \geq \delta \} \) and \( \text{MauSet}(X) = \{ x_i \mid x_i \in E(X) \wedge \text{mau}(X, x_i) \geq \delta \} \).

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

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

Calculating the \( lau \) of each itemset by \( U \). Given a utility-bin array \( U \) initialized with 0 for each \( T_j \), \( U[x_i] = U[x_i] + \text{lau}(X, x_i) \), 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 \( T_j \), \( U[x_i] = U[x_i] + \text{mau}(X, x_i) \), 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 \( D \) can be viewed as a 1-itemset. Then the real average utility \( \text{rau}(x_i) \) is equal to \( au(x_i) \) actually. In other words, these 1-itemsets also may be part of HAUUs. Therefore, let \( \text{rau-list} = \{ \text{rau}(x_1), \text{rau}(x_2), \ldots, \text{rau}(x_k) \} \) be a set of real average-utility of each item \( x_i \in I \), and the k-th highest value in \( \text{rau-list} \) is denoted as \( \text{rau}(x_k) \). If \( \text{rau}(x_k) \) is no less than the current \( \delta \), we can safely raise \( \delta \) 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 \( X = x_i \cup x_j \) \((x_i \neq x_j \text{ and } x_i, x_j \in I)\). 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 \( \text{cad-list} = \{ \text{cad}(X_1), \text{cad}(X_2), \ldots, \text{cad}(X_k) \} \) where \( \text{cad}(X_i) \) is the average-utility of 2-itemset \( X_i \), and the k-th highest value in \( \text{cad-list} \) is denoted as \( \text{cad}(X_k) \). If \( \text{cad}(X_k) \) is no less than the current \( \delta \), we can safely raise \( \delta \) to that value.

At last, we adopt another common strategy employed by TKO algorithm called RUC [36]. In this paper, it uses a \( \text{ruc-list} \) structure to maintain top-\( k \) HAUUs. Each element of \( \text{ruc-list} \) are sorted by descending order of their own average-utility, and the maximum size of \( \text{ruc-list} \) 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 \( \delta \), \( X \) will be added into \( \text{ruc-list} \). We suppose the k-th highest value in \( \text{ruc-list} \) is denoted as \( \text{ruc}(x_k) \). If \( \text{ruc}(x_k) \) is no less than the current \( \delta \), it can be safely that we raise \( \delta \) 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:** \( D \): a transaction database; \( k \): the desired number of HAUUs.

**Output:** A priority queue of top-\( k \) HAUUs.

1. initialize itemset \( \alpha \) as \( \emptyset \);
2. initialize utility list \( \text{rau-list} \) as \( \emptyset \);
3. set the initial minimum average-utility \( \delta \) as 1;
4. initialize an empty priority queue \( \text{TopHAUIs} \) of size \( k \);
5. scan \( D \), compute real utility of all items \( x_i \in I \), and store \( \text{rau-list}(x_i) \) in \( \text{rau-list} \);
   // the RAU strategy
6. raise the current \( \delta \) by \( \text{rau-list} \);
7. using utility-bin array \( U \) calculates \( auub \) of all items \( x_i \);
8. sort each item in \( U \) by the global order \( < \) of \( auub \) increasing values;
9. compute \( \text{LauSet}(\alpha) = \{ x_i \mid auub(x_i) \geq \delta \} \);
   // the auub strategy
10. scan \( D \), remove items \( x_j \notin \text{LauSet}(\alpha) \) in each \( T_j \in D \);
   // 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 \( D' \);
13. compute \( \text{MauSet}(\alpha) = \{ x_i \mid x_i \in \text{LauSet}(\alpha) \wedge \text{mau}(\alpha, x_i) \geq \delta \} \);
   // the CAD strategy
14. scan \( D' \), and create \( \text{cad-list} \) to raise \( \delta \);
15. call \( \text{Search}(\alpha, D', \text{MauSet}(\alpha), \text{LauSet}(\alpha), \delta, \text{TopHAUIs}) \);
16. return \( \text{TopHAUIs} \).

---

It takes a transaction database \( D \) and a desired number of HAUUs \( k \) as input and output a priority queue of top-\( k \) HAUUs as result. In Lines 1-4, the algorithm initializes several key parameters (i.e., \( \alpha \), \( \text{rau-list}, \delta \) and \( \text{TopHAUIs} \)). Then the algorithm scans the database and computes each \( \text{rau-list}(x_i) \) of item \( x_i \in I \) in Line 5. After that, strategy 4 raises the current threshold \( \delta \) 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 $\alpha$ by comparing $a_{\alpha\beta}$ values and $\delta$ (Line 9). Note that the $\alpha$ 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 $D'$ (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 $\delta$ 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: $\alpha$: the current itemset, $\alpha\cdot D$: the projected database of $\alpha$, $\text{MauSet}(\alpha)$: the $\text{MauSet}$ items of $\alpha$, $\text{LauSet}(\alpha)$: the $\text{LauSet}$ items of $\alpha$, $\delta$: a minimum average-utility threshold, and $\text{TopHAUIs}$: a priority queue of top-$k$ HAUIs.

Output: a set of top-$k$ HAUIs which contains extensions of $\alpha$.

1. for each item $x_i \in \text{MauSet}(\alpha)$ do
  
  // the transaction merging technology
  
  $\beta = \alpha \cup \{x_i\}$;

  // the RUC strategy
  
  scan $\alpha\cdot D$, calculate $a_{\beta\cdot D}$, and then create $\beta\cdot D$;

  if $a_{\beta\cdot D} \geq \delta$ then
    if $|\text{TopHAUIs}| \geq k$ then
      delete the $k$-th itemset in $\text{TopHAUIs}$;
    end
    add $\beta$ into $\text{TopHAUIs}$, and update the current $\delta$;
  end

  scan $\beta\cdot D$, calculate $\text{mau}(\beta, x_i)$, and $\text{lau}(\beta, x_i)$ where items $x_i \in \text{LauSet}(\alpha)$;

  obtain MauSet($\beta$) items where the extension items $x_i \in \text{LauSet}(\alpha)$;

  obtain LauSet($\beta$) items where the extension items $x_i \in \text{LauSet}(\alpha)$;

  call Search($\beta$, $\beta\cdot D$, MauSet($\beta$), LauSet($\beta$), $\text{TopHAUIs}$);

end

Algorithm 2 presents the Search procedure. It takes six parameters as input: the current extended itemset $\alpha$, the projected database of $\alpha$, the $\text{MauSet}$ and $\text{LauSet}$ items of $\alpha$, the current minimum average-utility threshold $\delta$ and a priority queue of top-$k$ HAUIs. The procedure mainly focuses on finding all single item extensions of $\alpha$ (that is $\beta = \alpha \cup \{x_i\}$), where $x_i$ is $\text{MauSet}$ item. For each $\beta$, the procedure first checks whether it is an HAUI or not. If it is true, according to the Strategy 6, the procedure adds $\beta$ into $\text{TopHAUIs}$ and then raises $\delta$ value if the length of HAUIs is already no less than $k$ (Lines 4-10). At the same time, the projected database $\beta\cdot D$ is created in Line 3. The procedure scans $\beta\cdot D$ to obtain $\text{MauSet}$ and $\text{LauSet}$ items of $\beta$ (Lines 11-13). At last, the procedure inputs the new parameters about $\beta$ 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 (T104D100K). 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.

Table 3: Basic information about datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Trans</th>
<th>#Items</th>
<th>#AvgLen</th>
<th>#Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>chess</td>
<td>3,196</td>
<td>75</td>
<td>37</td>
<td>Dense</td>
</tr>
<tr>
<td>mushroom</td>
<td>8,124</td>
<td>119</td>
<td>23</td>
<td>Dense</td>
</tr>
<tr>
<td>retail</td>
<td>88,162</td>
<td>16,470</td>
<td>10.3</td>
<td>Sparse</td>
</tr>
<tr>
<td>T104D100K</td>
<td>100,000</td>
<td>870</td>
<td>10.1</td>
<td>Sparse</td>
</tr>
</tbody>
</table>

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 T104D100K datasets, PPT always takes less memory than PPT$^*$ with various $k$. This means maximum average-utility upper-bound plays an important role during mining process. Therefore, we can take a conclusion that PPT performs better than TKAU in terms of memory consumption.

We can learn that PPT usually generates less candidates in sparse which is bigger than that of TKAU greatly, PPT AUO-List will be constructed and will be kept in memory all the time. The reason is TKAU is a list-based algorithm, and in addition, comparing the number of candidates generated between two lists to determine whether the new itemset can be generated. In time. To obtain high level itemset from AUO-Lists, TKAU must scan a list contains many tuples. If TKAU discovers an itemset, a new 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 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...
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


