IDHUP: Incremental Discovery of High Utility Pattern

Lele Yu[1], Wensheng Gan[2], Zhixiong Chen[3], Yining Liu[1]*

[1] Guangxi Key Laboratory of Trusted Software, School of Computer Science and Information Security, Guilin University of Electronic Technology, China
[2] College of Cyber Security, Jinan University, China
[3] Fujian Key Laboratory of Financial Information Processing, Putian University, China
yll101298@gmail.com, wsgan001@gmail.com, ptczx@126.com, ynliu@guet.edu.cn

Abstract

As a sub-problem of pattern discovery, utility-oriented pattern mining has recently emerged as a focus of researchers' attention and offers broad application prospects. Considering the dynamic characteristics of the input databases, incremental utility mining methods have been proposed, aiming to discover implicit information/ patterns whose importance/utility is not less than a user-specified threshold from incremental databases. However, due to the explosive growth of the search space, most existing methods perform unsatisfactorily under the low utility threshold, so there is still room for improvement in terms of running efficiency and pruning capacity. Motivated by this, we provide an effective and efficient method called IDHUP by designing an indexed partitioned utility list structure and employing four pruning strategies. With the proposed data structure, IDHUP can not only dynamically update the utility values of patterns but also avoid visiting non-occurred patterns. Moreover, to further exclude ineligible patterns and avoid unnecessary exploration, we put forward the remaining utility reducing strategy and three other revised pruning strategies. Experiments on various datasets demonstrated that the designed IDHUP algorithm has the best performance in terms of running time compared to state-of-the-art algorithms.

Keywords: Pattern discovery, Incremental mining, Utility mining, Dynamic data

1 Introduction

The prevalence of big data not only promotes rapid economic and social development, but also brings a lot of convenience to our lives. Managers from all walks of life use data mining and data analysis to extract potentially useful information from a large amount of data to assist intelligent decision-making. For example, pattern mining algorithms are widely used in market basket analysis, urban traffic congestion analysis, agrometeorological forecast, disease risk assessment, etc. [1]. The most common tasks of pattern mining are frequent pattern mining (FPM) and association rule mining (ARM). Apriori [2] and FP-growth [3] are two of the well-known methods for addressing these tasks. As the first step of ARM, FPM measures the usefulness or insightfulness of patterns based on co-occurrence frequency. Under this framework, all the items/objects are regarded as equally important, while other connotational factors such as weight, utility, or risk of items are ignored, thus losing the applicability of dealing with complex tasks.

Therefore, researchers have paid more attention to new pattern mining methods that integrate subjective metrics (e.g., weight, unit profit, and user preferences) and objective metrics (e.g., quantity and frequency) to enhance the usability and interest of the extracted patterns. Unlike the traditional frequency-based framework, HUPM focuses on the utility of patterns, which incorporates quantitative information and weights of items rather than only considering existence or confidence. In this way, HUPM can extract valuable and high profitable patterns for retailers and business managers, and is thus successfully used at the intersection of business and data science. Generally, utility is an ubiquitous concept in real life that is not limited to measuring the profit of patterns, but can also measure other subjective views of users on patterns, such as risk, usability, importance, satisfaction, and so on. Nowadays, utility mining plays an important role in the field of data analysis, and many studies have focused on mining efficiency, such as Two-Phase [4], HUI-Miner [5], FHM [6], HUP-Miner [7], mHUIMiner [8], and EFIM [9].

However, the above-mentioned methods are only suitable for dealing with static transaction databases. In order to adapt to the real world, various extension topics based on HUPM are concerned [10-11]. Incremental high utility pattern mining (IHUPM), as one of these hot topics, aims to effectively process the data continuously generated by various applications, avoiding the processing from scratch like static methods whenever new data is added. For example, assuming 100 new transactions are inserted into an original database containing 100,000 transactions, traditional static approaches would process 100,100 transactions, while IHUPM methods simply process 100 new transactions to achieve the same result. In the past few decades, many approaches have been developed to handle IHUPM tasks. According to [12], these methods can be roughly divided into three categories, respectively,

*Corresponding Author: Yining Liu; E-mail: ynliu@guet.edu.cn
DOI: 10.53106/160792642023012401013

Table 1. Comparison of existing IHUPM methods

Category	Algorithm	Theoretical basis	Limitations
Apriori-based algorithms	FUP-HUI-INS [13]	It relies on the *FUP* concept [22] and the *twu* model [4].	Apriori-based algorithms require multiple scans of the database and generate a large number of candidates.
	Pre-HUI-INS [14]	It level-wisely mine HUPs as an extension of literature [13].	
Tree-based algorithms	IHUP [15]	It constructs a global tree and uses it to mine HUPs.	Tree-based algorithms require frequent updating of tree nodes and creation of subtree structures, which is time-consuming.
	iCHUM [16]	It improves IHUP [15] and relies on the twu model [4].	
	PIHUP [17]	It relies on the pre-large concept from the literature [14].	
List-based algorithms	HUI-list-Ins [18]	It calculates utility values with utility-list structure and speeds up the algorithm with EUCS structure.	The joining process of the utility-list structure is inefficient.
	EIHI [19]	It uses partitioned utility-list structure and stores results in HUI-trie.	
	LIHUP [20]	It updates the global utility list through a novel reconstruction technique.	
	IIHUM [21]	It avoids inefficient intersection operations through the indexed utility list structure	It fails to take advantage of the properties of incremental data to simplify calculations.

They are Apriori-based methods [13-14], tree-based methods [15-17], and list-based methods [18-21].

Although the existing IHUPM algorithms can be applied to mining valuable patterns in dynamic environments, they still face many challenges. First, the calculation of utility values comprehensively considers multiple factors, this is more complicated than that of frequency. Second, utility does not maintain downward closures, which can be used to prune invalid patterns in the search space in advance. This means that traditional frequency-based pruning methods are not suitable for utility mining, and substitute robust pruning strategies must be used to effectively and efficiently exclude unqualified patterns. Third, the insertion of additional transactions changes the utility of patterns and may introduce some new items, rendering the original information invalid. What growable data structures are utilized to update the utility values without processing from scratch like with static methods and to retain identified patterns are key issues. Moreover, it is significant to minimize the time and space complexity without discarding qualified patterns and without reprocessing the original database, which ensures mining tasks can be completed precisely under limited time and space resources.

To address the above challenges and for further efficiency improvement, we designed a novel one-phase algorithm to efficiently and fully discover valuable patterns from incremental databases, called IDHUP (Incremental Discovery of High Utility Pattern). Thanks to the proposed several powerful pruning strategies and indexed partitioned utility list structure, IDHUP can complete the IHUPM task without generating many candidates. The main contributions of this paper are threefold:

1. Relying on the proposed novel and compact indexed partitioned utility list structure, the algorithm does not generate non-occurred patterns as candidates, and maintains the utilities of patterns easily in an incremental environment.

2. Considering the computational complexity of existing incremental algorithms, we investigate the Remaining Utility Reducing pruning strategy to decrease the remaining utility upper bound by subtracting the utilities of unpromising succeeding items. Furthermore, we adopt three complementary strategies to discover unpromising succeeding items.

3. Experiments on real and synthetic datasets demonstrated that IDHUP with all pruning strategies can discover intact high utility patterns with acceptable memory consumption and the shortest running time.

This paper is structured as follows. Section 2 reviews some existing IHUPM methods. In Section 3, some basic definitions are given. Then, we describe the proposed IDHUP algorithm in Section 4. In Section 5, we evaluate the effectiveness and efficiency of our method. Finally, we summarize this paper.

2 Related Work

Many dynamic HUPM methods have been studied recently, especially incremental mining methods dealing with databases with transactions insertion. A comparative summary of existing IHUPM methods is presented in Table 1.

As shown in Table 1, IHUPM algorithms are classified as Apriori-based [13-14], Tree-base [15-17], and List-based [18-21], depending on the data structure they use. FUP-HUI-INS [13], as an Apriori-based approach, combines the concepts of *FUP* [22] and *twu* [4] to divide all patterns in the database into four categories, and reduces some unnecessary processing by classification and discussion. Pre-HUI-INS [14], an extension of FUP-HUI-INS, presents the concept of pre-large to divide patterns into nine categories to further improve efficiency. However, Apriori-based algorithms require multiple scans of the database. To solve this problem, tree-

based algorithms such as IHUP [15], iCHUM [16], and PIH-UP [17] have been proposed. But all the above algorithms are two-stage algorithms that would generate lots of candidate itemsets instead of finding HUPs directly. Thus, inspired by HUI-Miner [4], many one-stage algorithms without candidate generation are proposed, such as HUI-list-Ins [18], EIHI [19], LIHUP [20], and IIHUM [21]. EIHI uses a global structure HUI-trie to store previously obtained results and accelerates exploration by classifying patterns by their presence or absence in incremental data. Through the developed utility lists reconstruction mechanism, LIHUP achieves effective mining with only a single scan of incremental data. However, the above algorithms based on traditional utility lists suffer from inefficient joining processes. IIHUM overcomes it with a novel indexed utility list structure, but still leaves room for improvement in filtering invalid patterns and fails to take advantage of the property of incremental data to simplify calculations. Recently, Liu *et al.* proposed the incremental algorithm Id^{2}HUP+ [23]. Note that the mining objective of this algorithm differs from previous algorithms in that its mining results do not contain the HUPs in the original database, while other incremental algorithms update the utility of these patterns and output them. In other words, Id^{2}HUP+ cannot obtain the complete set of HUPs and their corresponding actual utility values in the entire database.

In addition to the above incremental algorithms for mining HUPs, there are many incremental algorithms for different mining purposes, such as IncCHUI [24] for mining closed high utility patterns, and HUIPRED [25] for handling database where transactions are deleted. Moreover, there are many methods for other dynamic scenarios. For example, M- PM [26] and SOHUPDS [27] for dealing with data streams, iMEFIM [28] and CHUI-Power [29] for processing the dynamic unit profit databases. As the common and basic situation of dynamic change, incremental high utility mining is still our focus. For further efficiency improvement, we propose our IDHUP algorithm.

3 Preliminaries

Let $I = \{i_1, i_2, ..., i_m\}$ be a set of m distinct products/ items. Given an original database $OGD = \{T_1, T_2, ..., T_n\}$ containing *n* transactions and an additional database db^+ consisting of *k* transactions that need to be inserted into *OGD*, the entire updated database to be mined is represented as $D = OGD \cup db^+$. Each transaction in *D* has a unique identifier called *tid* and is a subset of *I*. In addition, each item i in a transaction T_r is associated with a non-binary value $q(i, T_r)$ that records occur quantity and a positive number $pr(i)$ that measures unit profit/weight. Tables 2 and Table 3 show examples of an entire incremental database and a unit profit table.

Table 2. Example of incremental transaction database

	tid	Transaction	tu
OGD	T_1	$(a:3), (b:1), (c:3), (d:2)$	\$22
	T_2	$(b:1), (c:3), (d:2), (e:3), (f:1)$	\$23
	T_3	$(b:5), (d:2), (e:3), (f:2)$	\$41
	T_4	$(a:2), (c:2), (d:3), (e:1)$	\$15
db^+	T_5	$(c:1), (d:3), (e:2)$	\$9
	T_6	$(d:3), (f:1)$	\$7
	T_7	$(a:3)$	\$9

Table 3. Example of static unit profit database

Item	*a*	*b*	*c*	*d*	*e*	*f*
Unit profit	\$3	\$5	\$2	\$1	\$2	\$4

Definition 1: The utility of a pattern X in a transaction T_r is $u(X, T_r)$ and can be calculated as in equation (1), where $u(i, T_r)$ is the utility of item $i \in X$ in T_r and obtained by multiplying $q(i, T_r)$ and $pr(i)$.

$$u(X, T_r) = \sum_{i \in X} u(i, T_r) = \sum_{i \in X} \{ q(i, T_r) \times pr(i) \}. \quad \textbf{(1)}$$

Definition 2: The total utility of pattern X in the entire database D, denoted as $u_D(X)$, is calculated as in equation (2).

$$u_D(X) = \sum_{X \subseteq T_r \wedge T_r \in D} u(X, T_r). \quad \textbf{(2)}$$

For example, the utility of item a in T_1 is $u(a, T_1) = q(a, T_1) \times pr(a) = 3 \times \$3 = \$9$. Further, utility of pattern $\{a,b,c\}$ in T_1 can be calculated as $u(\{a,b,c\}, T_1) = u(a, T_1) + u(b, T_1) + u(c, T_1)$ = \$9 + \$5 + \$6 = \$20. Similarly, $u(\{b,c\}, T_1) = 1 \times \$5 + 3 \times \$2 = \$11$, and the utility of pattern $\{b,c\}$ in the updated database D is $u_D(\{b,c\}) = u(\{b,c\}, T_1) + u(\{b,c\}, T_2) = \$11 + \$11 = \$22$.

Definition 3: Given a minimum utility threshold *minUtil* defined by an experienced decision maker, if the utility of pattern X in D is not less than *minUtil*, then X is called high utility pattern (HUP) in D, otherwise it is a low utility pattern in it. That is,

$$HUPs = \{X \mid u_D(X) \geq minUtil, X \subseteq I\}. \quad \textbf{(3)}$$

For example, if the *minUtil* set as 30, then the pattern $\{b,c\}$ is a low utility pattern since $u_D(\{b,c\}) = 22 < 30$ while pattern $\{b\}$ is a HUP in D because $u_D(\{b\}) =$

$u(\{b\},T_1)+u(\{b\},T_2)+u(\{b\},T_3)=\$5+\$5+\$25=35>30$.

Definition 4: The utility of a transaction T_r, denoted as $tu(T_r)$ and is calculated as,

$$tu(T_r)=\sum_{i\in T_r}u(i,T_r). \tag{4}$$

Definition 5: The transaction weighted utilization (*twu* [4]) of a pattern X in the updated database D is the sum of the utilities of transactions containing X, and is defined by,

$$twu_D(X)=\sum_{X\subseteq T_r\wedge T_r\in D}tu(T_r). \tag{5}$$

For example, the utility of T_1 is $tu(T_1)=u(a,T_1)+u(b,T_1)+u(c,T_1)+u(d,T_1)=3\times\$3+1\times\$5+3\times\$2+2\times\$1=\$9+\$5+\$6+\$2=\$22$. The utility of each transaction in the example database of Table 2 is given in it. Then, $twu_D(\{a\})=tu(T_1)+tu(T_4)+tu(T_7)=\$22+\$15+\$9=\$46$ and $twu_D(\{a,b\})=tu(T_1)=\$22$.

Property 1 (Overestimation and downward closure of twu): The *twu* of pattern X is not less than its utility, that is,

$$twu(X)\geq u(X). \tag{6}$$

In addition, it satisfies downward closure property, that is, the *twu* of X is not less than the *twu* of any of its super patterns. Combined, the following property holds.

$$twu(X)\geq twu(Y)\geq u(Y),\ if\ X\subseteq Y. \tag{7}$$

Definition 6: Generally, items in a database are assumed to be sorted in a certain order, which is denoted as total order $\prec$. In particular, the total order $\prec$ used in the designed IDHUP algorithm is obtained as follows: Items in the first *OGD* database are sorted by twu_{OGD} ascending order. For subsequent batches, this order is maintained, and items that have never appeared before always succeed these sorted items.

Definition 7: The remaining utility of a pattern X in a transaction T_r is defined by,

$$ru(X,T_r)=\sum_{(\forall i\in X)i\prec j\wedge j\in T_r}u(j,T_r). \tag{8}$$

Definition 8: The remaining utility of pattern X in the entire database D, is denoted as $ru_D(X)$.

$$ru_D(X)=\sum_{X\subseteq T_r\wedge T_r\in D}ru(X,T_r). \tag{9}$$

For example, twu_{OGD} of items *a*, *b*, *c*, *d*, *e*, *f* are 37, 86, 60, 101, 79, 64, respectively, thus the total order $\prec$ is set as: $a\prec c\prec f\prec e\prec b\prec d$. In addition, the remaining utility of pattern $\{ab\}$ in T_1 is $ru(\{ab\},T_1)=u(\{d\},T_1)=2\times\$1=\$2$. And the remaining utility of $\{ab\}$ in D is $ru_D(\{ab\})=ru(\{ab\},T_1)=\$2$.

Property 2 (Pruning by remaining utility): For pattern X, define its extensions as the patterns obtained by appending such an item j to X, where $i\prec j,\forall i\in X$. If the sum calculated by equation (10) is less than *minUtil*, then all extensions of *X* are not HUPs.

$$reu(X)=u(X)+ru(X). \tag{10}$$

Problem statement. Given an original transaction database *OGD*, a static unit profit table, a user-specified minimum utility threshold (*minUtil*), and a set of transactions db^+ are inserted into *OGD*. The incremental part db^+ and the original database *OGD* together form an updated database D to be mined. The goal of IHUPM is to output all HUPs in the currently updated database *D*.

4 Proposed IDHUP Algorithm

This section presents a novel algorithm for addressing the IHUPM task, called IDHUP. IDHUP first scans the original database or the additional database to set up a total order of items, then constructs global lists structure of single items according to the total order to maintain the utility information, and finally recursively mines high utility patterns and stores them in the HUI-Tire. The framework of the proposed IDHUP algorithm is presented in Figure 1. Details of the indexed partitioned utility list structure, pruning strategies, and the main procedure of IDHUP are respectively described below.

4.1 Proposed Indexed Partitioned Utility List Structure

Several utility list-based methods have been proposed to achieve IHUPM task without multiple scans. However, the traditional utility list structure is constructed through time-consuming intersection operations. The utility-list* structure in the static algorithm HUI-miner* [30] directly points to the element to be intersected to solve this problem, but its corresponding dynamic maintenance and update mechanism has not been proposed. Therefore, in the proposed IDHUP algorithm, the utility-list* is modified to adapt to incremental environments. Furthermore, also inspired by the partitioned structure, the indexed partitioned utility list structure is defined below. For convenience, we abbreviate the indexed partitioned utility list of pattern *X* as *X.ipul*.

Definition 9: The *X.ipul* consists of the pattern name (*name*), utility (*sumU*), remaining utility (*sumRU*), a set of *tuples* associated with transactions containing *X* in the original database *OGD* (*X.ipul.OGD*), and such *tuples* associated with transactions in the additional database db^+ (*X.ipul.* db^+). A tuple is defined as < *next*, *tid*, *item*, *iutil* > for each transaction T_r containing *X*.

next : a pointer to the next tuple associated with T_r.

tid : the transaction identifier of T_r.

item : the last item in pattern X.
iutil : the utility of X in T_r , i.e., $u(X,T_r)$.

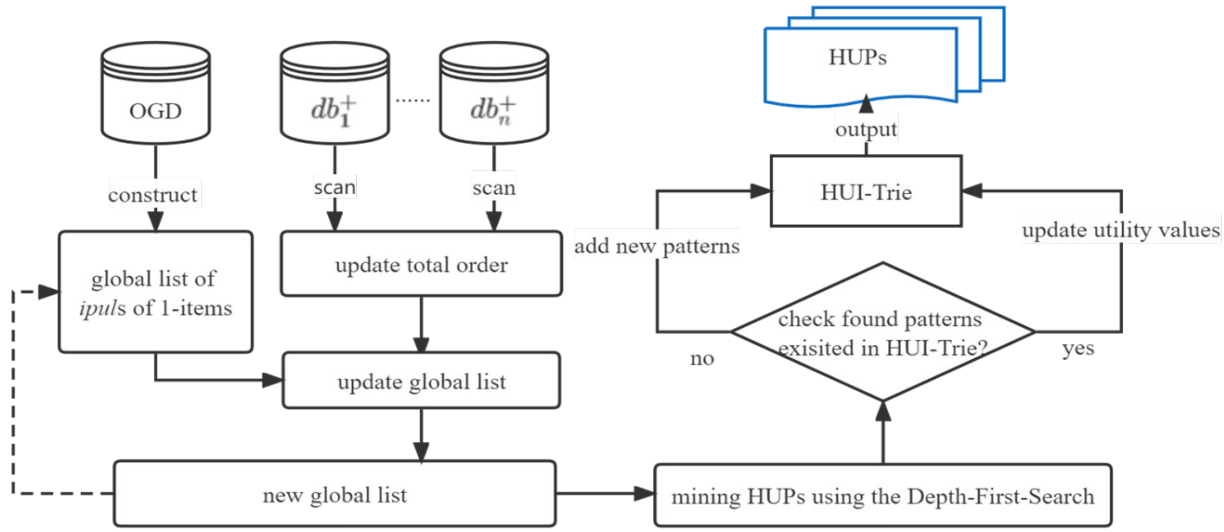

Figure 1. Flowchart of the IDHUP algorithm

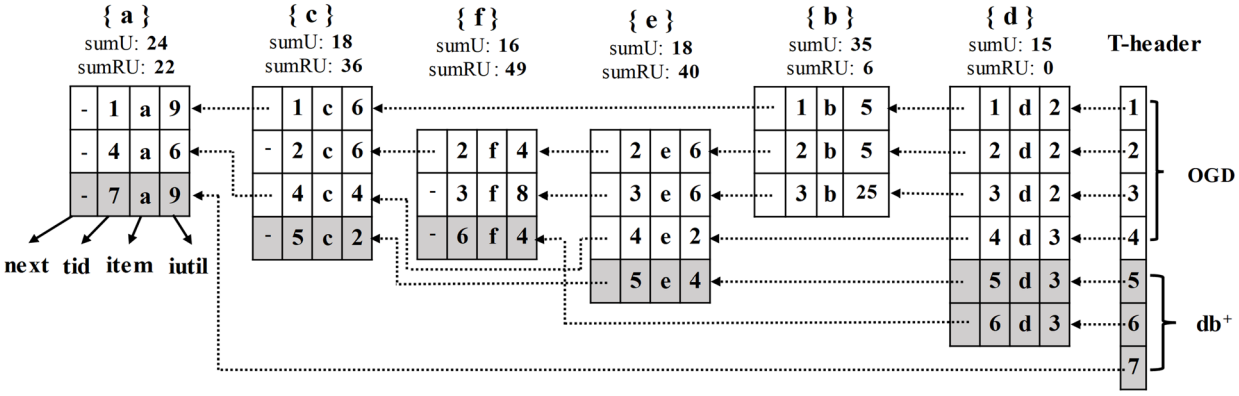

Figure 2. Global list of each 1-item

In addition, IDHUP links the *i*-th element in a table called *T-Header* to the tuple corresponding to the last item in the *i*-th transaction according to the total order $\prec$. Figure 2 shows the *ipuls* of all single items (1-itemsets) in the running example with *T-header*. IDHUP builds them horizontally and processes them in reverse order. For example, consider T_3 in Table 2. According to the Def. 6, the total order we adopt is $a \prec c \prec f \prec e \prec b \prec d$. Then, when T_3 is processed, IDHUP first stores a tuple t_d : $(tid:3,\ item:d,\ iutil:2)$ in $d.ipul.OGD$ and links the 3-rd component in *T-Header* to t_d , secondly stores a tuple t_b : $(tid:3, item:b, iutil:25)$ in $b.ipul.OGD$ and links the *next* field of t_d to t_b ; Then it updates the lists of remaining items in T_3 in turn. Note that the summary information, such as *sumU* and *sumRU*, are calculated during the building process and updated after all tuples are processed. For example, in the above process, when the tuple t_b is processed, $ru(b,T_3)=u(d,T_3)=2$ and $u(b,T_3)=25$ can be obtained, thus add 2 to the *sumRU* of *b.ipul* and add 25 to the *sumU* of *b.ipul*.

Property 3: Given a pattern X and $X.ipul$, then *sumU* and *sumRU* in $X.ipul$ calculated by the utility and remaining utility of the tuples are respectively equivalent to $u_D(X)$ and $ru_D(X)$.

Proof: each tuple in $X.ipul$ is associated with a transaction containing X, hence the sum of the *iutil* values of tuples in $X.ipul.OGD$ and in $X.ipul.db^+$ is equals to,

$$\begin{aligned} & sumU \\ &= \sum_{X \subseteq T_r \wedge T_r \in ODG} u(X,T_r) + \sum_{X \subseteq T_k \wedge T_k \in db^+} u(X,T_k) \\ &= u_{OGD}(X) + u_{db^+}(X) \\ &= u_D(X). \end{aligned} \quad (11)$$

Similarly, we have,

$$sumRU = ru_{OGD}(X) + ru_{db^+}(X) = ru_D(X). \quad (12)$$

IDHUP horizontally constructs the *ipuls* of 1-extensions of a single item x (2-itemsets) by traversing *x.ipul*. Suppose the tuple being traversed is the u -th tuple in $x.ipul$ called t_u and $t_u.tid = k$. It first locates the tuple t_1 based on the k -th component in the *T-Header* related to $x.ipul$. Then, starting from $t_1.next$, a sequence of tuples $t_1, t_2, ..., t_m$ until t_u can be derived by following $t_i.next$ $(1 \le i \le m)$. For each t_i , IDHUP will construct a new $ipul$ for pattern $\{x, t_i.item\}$ and store a new tuple t_{xi} : $(tid: u, item: i, iutil: t_i.iutil + t_u.iutil)$ in its OGD or db^+ . Simultaneously, IDHUP updates the *next* field of these new tuples in sequence and links the first constructed tuple to the u -th component in a new *T-Header*. In addition, the summary information of these new *ipuls* are calculated during the building process.

Take the second tuple t_2: $(4, a, 6)$ in *a.ipul.OGD* in Figure 2 as an example, IDHUP traces a sequence of tuples $(4, d, 3)$, $(4, e, 2)$ and $(4, \ , 4)$. Thus, new tuples $(2, d, 3+6)$, $(2, e, 2+6)$ and $(2, c, 4+6)$ are stored in the *ad.ipul.OGD*, *ae.ipul.OGD* and *ac.ipul.OGD,* respectively. Simultaneously, IDHUP updates their summary information by adding $(sumU+9, sunRU+0)$ in *ad.ipul*, adding $(sumU+8, sunRU+3)$ in *ae.ipul*, and adding $(sumU+10, sunRU+5)$ in *ac.ipul*, respectively. Where 3 is the *iutil* of tuple $(4, d, 3)$ and 5 is summed by 3 and the *iutil* of tuple $(4, e, 2)$, i.e., $3+2$. In addition, IDHUP connects these tuples in sequence and links the second component in a new *T-Header* to $(2, d, 9)$. Figure 3 shows the result of constructed *ipuls* of all 1-extensions of $\{a\}$.

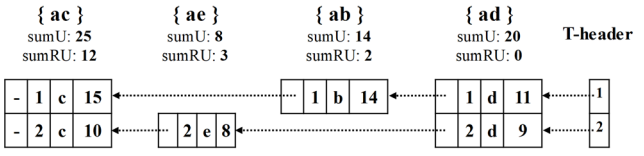

Figure 3. The *ipuls* of all 1-extensions of $\{a\}$

The construction of the *ipuls* of *k-itemsets* ($k \ge 3$) is similar to that of 2-itemsets. For k - 1 pattern X and its prefix pattern P, assume that the tuple being traversed is the u-th tuple in $X.ipul$ called t_u and $t_u.tid = z$. Compared with the above construction process, the only difference is that, for each t_i in sequence derived by t_u , IDHUP constructs a new $ipul$ for pattern $\{X, t_i.item\}$ and stores a new tuple t_{Xi} : $(tid: u, item: i, iutil: t_i.iutil + t_u.iutil - preUtil)$ in area OGD or db^+ of it. Where $preUtil$ is the utility of P in the transaction associated with t_u and can be obtained by $t_u.tid = z$ without traversing $P.ipul$, since the $iutil$ of the z-th tuple in $P.ipul$ is equal to $preUtil$. For example, the *ipuls* of 1-extensions of pattern $\{ac\}$ are shown in Figure 4.

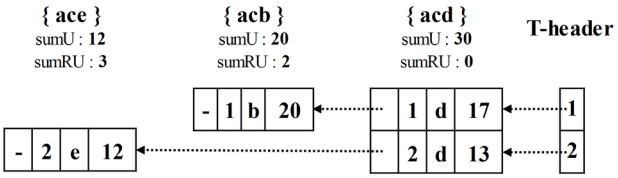

Figure 4. The *ipuls* of all 1-extensions of $\{ac\}$

4.2 Pruning Strategies

Similar to the previous studies [5-6], a set enumeration tree is used to represent the complete search space for mining HUPs. This tree structure represents all possible subsets of $I = \{i_1, i_2, ..., i_m\}$, where each node represents a pattern and its children nodes represent extensions of this pattern [31]. Note that this tree structure is not real, it is used to facilitate the description of the search space and strategies for filtering unqualified patterns, and is not actually constructed during the mining process. In the worst case, there may be 2^m candidate patterns (i.e., all the subsets of *I*), corresponding to 2^m nodes in the enumeration tree, which indicates a huge search space, thus showing the difficulty and complexity of mining. Therefore, it is necessary to use several effective pruning strategies to find the unqualified patterns in advance and terminate exploration early. Key pruning strategies used in this paper are described below.

Lemma 1 (Extension upper bound): The extension upper bound of a pattern X can be calculated as:

$$extUB(X) = \sum_{X \subseteq T_r \wedge ru(X,T_r) > 0} [u(X,T_r) + ru(X,T_r)]. \quad (13)$$

If $extUB(X) < minUtil$, then all extensions of X are not HUPs and do not need to extend pattern X further. A complete proof of this lemma can be referred to [32].

Strategy 1 (EUB-Prune) For each pattern, IDHUP creates a new summary information field called *sumU'* in *X.ipul*. During the horizontal construction of *X.ipul*, IDHUP determines whether the remaining utility of the tuple t_i being stored is zero. If it is not zero, then add $t_i.iutil$ to the *sumU* and *sumU'* fields in *X.ipul*, otherwise add it to *sumU* only. While performing a depth-first search, for any pattern/node X in the search space, if $(X.ipul.sumU' + X.ipul.sumRU) < minUtil$, then no further extension for X is needed. Because all its children nodes/patterns are not HUPs, and these unpromising patterns can be directly pruned. Conversely, X

should be further extended.

For example, as *a.ipul* shown in Figure 2, the $sumU'(a) = 9 + 6 = 15$ since the remaining utility of the tuple $(-,7,a,9)$ is zero, and $extUB(a) = sumU'(a) + sumRU(a) = 15 + 22 = 37 > minUtil = 30$ implies that further expansion of {a} is needed.

Lemma 2 (EUCS Structure [6]*):* The Estimated Utility Co-Occurrence Structure (EUCS) is a triangular matrix to stores the *twu* values of all pairs of items and is defined as,

$$EUCS = \{(a;b;twu(\{ab\})) \mid a \in I, b \in I\}. \quad \textbf{(14)}$$

If the *twu* of a 2-pattern is less than *minUtil*, then all of its supersets are not HUP according to the property 1.

Strategy 2 (EUCS-Prune) While constructing the *ipuls* for all 1-extensions of pattern *X* as described in Sec. 4.1, for each tuple t_i derived by tuple t_u in *X.ipul*, if $twu(t_u.item, t_i.item) < minUtil$ is obtained from EUCS, then the extension pattern $\{X, t_i.item\}$ as a superset of $\{t_u.item, t_i.item\}$ is not a HUP. Thus, instead of constructing a new *ipul* for $\{X, t_i.item\}$, we ignore $t_i.item$ and skip tuple t_i.

For example, during the construction of *ipuls* for all 1-extensions of $\{a\}$ from Figure 2, IDHUP locates the first component in the *T-Header* and traces a sequence of tuples $(1,d,2)$, $(1,b,5)$ and $(1,c,6)$. Since $twu(a,d) = 31 > 30$, $twu(a,b) = 22 < 30$ and $twu(a,c) = 31 > 30$, IDHUP creates new *ipuls* for patterns $\{ad\}$ and $\{ac\}$, and respectively stores tuples $(1,d,11(=2+9))$ and $(1,c,15(=6+9))$ in them, but ignores pattern {ab} and skips tuple $(1,b,5)$. The construction result with EUCS-prune is shown in Figure 5.

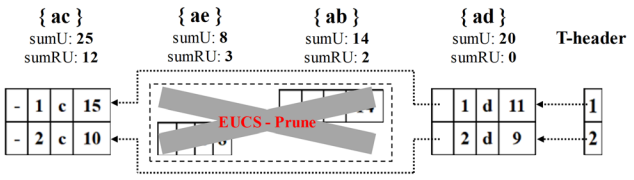

Figure 5. An example of EUCS-prune

Lemma 3 (Absent pattern): Given the entire incremental database $D = OGD \cup db^+$, a pattern *X* is called an absent pattern if it appears only in *OGD* but not in db^+. The utility of an absent pattern in *D* remains the same as that in *OGD*, that is, $u_D(X) = u_{OGD}(X)$. And two situations are considered.

1. If an absent pattern *X* is a HUP in *OGD*, then *X* is also a HUP in *D* that should be output.
2. If an absent pattern *X* is not a HUP in *OGD*, then it is also not a HUP in *D* and can be pruned directly.

Strategy 3 (ABS-Prune) IDHUP stores HUPs into a global structure HUI-trie [19], and these stored patterns are regarded as the HUPs in the *OGD* when the next additional database db^+ arrives. On the one hand, the absent patterns in situation (1) have been stored in the global HUI-trie structure and can be output directly from it. On the other hand, the absent patterns in situation (2) are hopeless patterns that can be excluded. Therefore, those patterns whose $ipul.db^+$ are empty (i.e., absent patterns) can be ignored during depth-first search or be skipped in the process of constructing *ipuls* for extensions.

Definition 10: An item is said to be unpromising if none of its super-patterns are HUPs. For a subtree of pattern *X* in the set enumeration tree, items that not contained in the HUPs on the subtree are called locally unpromising items.

Lemma 4 (Remaining utility reducing) We propose a compact remaining utility by reducing the utilities of locally unpromising items for *X*. That is,

$$rur(X) = ru(X) - \sum_{X \subseteq T_r \wedge X \prec i \wedge i \in LUI(X)} u(i, T_r), \quad \textbf{(15)}$$

where *LUI(X)* is the set of unpromising items on the subtree rooted at node *X*.

Proof: Since *LUI(X)* is the set of locally unpromising items of *X*, the HUPs generated from *X* must not contain the items in *LUI(X)* according to Def. 10, thus the items in *LUI(X)* do not contribute any utility to the HUPs in the extensions of *X*. So, the locally unpromising items and their utilities can be discarded when calculating the remaining utility upper bound.

Property 4: For item $i \in X$ and item $j \notin X$, if $twu(i, j) < minutil$, then $j \in LUI(X)$.

Rationale: If $twu(i, j) < minutil$, by property 1, the superset $\{X \cup j\}$ of $\{i \cup j\}$ and any pattern generated from $\{X \cup j\}$ have utility values less than *minutil* and are not HUP, thus $j \in LUI(X)$ according to Def. 10.

Strategy 4 (RUR-Prune) Owing to the above property, during the constructing process of *ipuls* for extensions, IDHUP algorithm decreases remaining utilities of extension patterns by utilities of locally unpromising items found by EUCS-Prune. Put another way, instead of adding the utilities of these items to the remaining utilities of the following tuples, IDHUP simply skips them without updating the remaining utilities in the sequence.

For example, as shown in Figure 6, when updating the remaining utility of {ac}, the utilities of items {b} and {e} in the sequence that are pruned by EUCS-Prune are not taken into account, and $sumRU(ac) = 2 + 3 = 5$. Similarly, since the supersets of absent patterns are also excludable absent patterns, for other extended patterns, absent items do not contribute to their utilities and thus can be directly skipped without updating the remaining utilities.

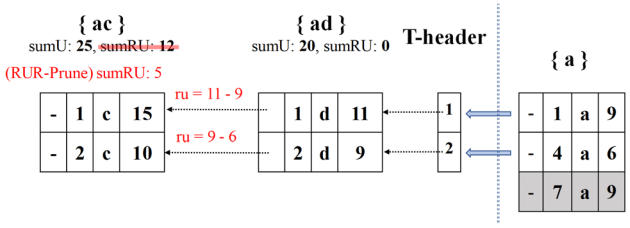

Figure 6. An example of RUR-prune

4.3 IDHUP

The pseudocode of the main procedure of IDHUP is given in Algorithm 1. It takes as input the original database *OGD*, the additional database db^+, and the user-specified threshold *minUtil* and outputs the complete set of HUPs in the current entire incremental database ($OGD \cup db^+$). In order not to duplicate processed data, IDHUP initializes HUI-trie, GIPULs, and EUCS as global variables if it is the first time (Line 1). It first scans each transaction in *OGD* or db^+ to: (1) calculate the *twu* value of all individual items in the database, which is used to specify the total order $\prec$ to improve mining efficiency;

Algorithm 1. The IDHUP algorithm

1: *Global variables*: a tree structure for storing HUPs (HUI-trie), the Estimated Utility Co-Occurrence Structure (EUCS), a set of *ipuls* for single items (GIPULs).
2: Initialize Global variables for the first time;
3: Scan *OGD* or db^+ to calculate *twu*(*i*) for each *I* and specify the total order $\prec$;
4: Scan *OGD* or db^+ once again in total order to build *i.ipul* for each *i*, add to GIPULs;
5: Update the *sumU* , *sumRU* and *sumU* ′ field of each *ipul* synchronously; //EUB-Prune
6: Call ***Search***($\varnothing$, *GIPULs, minUtil)* and insert results intoHUI-trie;
7: Extract HUPs from HUI-trie;
8: **for** each *ipu* $\in$ GIPULs **do**
9: Merge entries in *ipul.db*$^+$ into *ipul.OGD*;
10: *ipul.db*$^+ = \varnothing$;
11: **return** HUPs;

(2) Update the global EUCS for subsequent pruning (Lines 2-3). After rearranging the *OGD* or db^+ in order $\prec$, the revised database is re-scanned once to build *ipuls* for all individual items (Line 4). During this process, the summary information of each *ipul* is updated simultaneously (Line 5). After that, IDHUP recursively searches for HUPs by calling the ***Search*** procedure that traverses the set enumeration tree with a Depth-First-Search strategy (Lines 6-7). Finally, IDHUP merges the db^+ area of each *ipul* into *ipul.OGD* in preparation for the next execution (Lines 8-10).

Algorithm 2. The Search procedure

1: **for** each *ipul* $\in$ IPULs **do**
2: Obtain *ipul.name* pattern X;
3: **if** $X.ipul.db^+ == null$ or $twu(X.lastItem) < minUtil$ **then**
4: continue; //ABS-prune
5: **if** $X.ipul.sumU \geq minUtil$ **then**
6: insert *X* into HUI-trie;
7: **if** $X.ipul.sumU' + X.ipul.sumRU \geq minUtil$ **then** //EUB-prune
8: Call ***Construct***(*X.ipul*, *IPULs, preList*) $\rightarrow$ *exULs* with $T\text{-}header_{new}$
9: **If** $exULs.size \geq 2$ **then**
10: Call ***Search***(*X, X.ipul, exULs, minUtil)*;
11: **Else If** $exULs.size == 1$ **then**
12: Get the unique *ipul* and its name Y in exULs;
13: **If** $Y.sumU \geq minUtil$ **then**
14: insert *Y* into HUI-trie;
15: **return** *HUI-trie*;

The details of our ***Search*** procedure are presented in Algorithm 2. It takes four inputs: a prefix pattern *P*, the *ipul* of *P*, a set of *ipuls* of 1-extensions of *P* with *T-Header* named *IPULs* and the *minUtil* threshold. For each *ipul* in *IPULs*, the ***Search*** procedure first gets its corresponding pattern *X* by *ipul.name* (Lines 1-2). Next, it aims to explore *X* and all extensions of *X*, finding HUPs in them (Lines 3-14). For efficiency, we apply ABS-Prune (Strategy 3) and the overestimation property of *twu* (Property 1) to exclude absent patterns and unpromising patterns (Lines 3-4). Next, we check the utility of *X* by *X.ipul.sumU* , and if $u(X) > minUtil$, then *X* is inserted into HUI-trie as HUP (Lines 5-6). After that, we apply EUB-Prune (Strategy 1) to avoid unnecessary exploration by calculating the sum of *ipul.sumU'* and *ipul.sumRU*. If the sum is not less than *minUtil*, then further exploration of *X* is achieved by calling the ***Construct*** procedure and recursively performing the ***Search*** procedure (Lines 7-10). Otherwise, those extensions of *X* and their descendant nodes are regarded as low utility, and they are moved in this step. To enhance the flexibility of the algorithm, we count the number of extensions returned by the ***Construct*** procedure. If this number is 1, instead of further recursion, we directly output or delete the returned extension by judging its utility (Lines 11-14). When no candidates are generated, the method terminates and returns the HUI-trie back to the main procedure.

The ***Construct*** procedure is designed to build all 1-extensions and their *ipuls* for the pattern *Px* whose prefix is *P*. It takes *P.ipul*, *Px.ipul* and the set of *ipuls* of sibling patterns of *Px* with a *T-Header* as input. The ***Construct*** procedure operates as follows: First, it initializes the expected output: exULs and a new *T-Header* associated with exULs (Line 1). Next, for the input *Px.ipul* , we adopt the ABS-prune (Strategy 3) to expand its db^+ part first (Lines 2-21), and then perform similar operations on *Px.ipul.OGD* (Lines 22-37).

Algorithm 3. The Construct procedure

Initialize $exULs = \varnothing$, $T\text{-}header_{new} = \varnothing$;
for each tuple $t_u \in px.ipul.db^+$ **do**
  Initialize t_{Temp} = null, *remaining utility* = 0;
  Locate the tuple t_{mid} associated with the component pointed to by $t_u.tid$ in the *IPULs.T-header*;
  while t_{mid} != *null* **do**
    Obtain $twu(t_{mid}.item, t_u.item)$ from EUCS;
    if $twu(t_{mid}.item, t_u.item) < minUtil$ **then**
      $t_{mid} = t_{mid}.next$;
      Keep *remaining utility* unchanged;
      Continue;
    if *exULs* does not contain *ipul* of pattern $Px \cup t_{mid}.item$ **then**
      Create a new *ipul* for pattern $Px \cup t_{mid}.item$ and append it to *exULs* ;
    Set *PreUtil* ← P.length == 0? 0: $preList.db^+$. get($t_u.tid$ - *preList.OGD.size*-1).*iutil*
    Set $u_{new} \leftarrow u+1 + Px.ipul.OGD.size()$;
    Tuple $t_{new} \leftarrow (u_{new}, t_{mid}.item, t_u.iutil + t_{mid}.iutil - preUtil)$
    Associate the first created t_{new} to the u_{new}-th component of $T\text{-}header_{new}$
    Append t_{new} to the *ipul* of pattern $Px \cup t_{mid}.item$;
    Add the *remaining utility* to *sumRU* , add the $t_{new}.iutil$ to *sumU* and update *sumU'*;
    remaining utility += $(t_{mid}.iutil - PreUtil)$;
    $t_{Temp}.next = t_{new}$; $t_{Temp} = t_{new}$;
    $t_{mid} = t_{mid}.next$;
for each tuple $t_v \in px.ipul.OGD$ **do**
  Initialize a temporary tuple t_{Temp} = null, *remaining utility* = 0
  Locate the tuple t_{mid} associated with the component pointed to by $t_v.tid$ in the *IPULs.T-header*;
  while t_m != *null* **do**
    if *exULs* does not contain *ipul* of pattern $Px \cup t_m.item$ **then**
      $t_m = t_m.next$;
      Continue; // ABS-prune
    Set *PreUtil* ← P.length == 0? 0: $preList.db^+$. get($t_v.tid$ - 1).*iutil*
    Tuple $t_{new} \leftarrow (v+1, t_{mid}.item, t_v.iutil + t_m.iutil - preUtil)$
    Associate the first created t_{new} to the $v+1$-th component of $T\text{-}header_{new}$
    Append t_{new} to the *ipul* of pattern $Px \cup t_m.item$;
    Add the *remaining utility* to *sumRU* , add the $t_{new}.iutil$ to *sumU* and update *sumU'*;
    $t_{Temp}.next = t_{new}$; $t_{Temp} = t_{new}$;
    $t_m = t_m.next$;
return *exULs* with $T\text{-}header_{new}$

5 Experiments

Notice that many studies already exist on the topic of incremental mining HUPs. In addition, in recent studies, the EIHI algorithm [19] significantly outperforms HUI-list-Ins [18], and IIHUM [21] outperforms LIHUP [20] under certain conditions, but there is no literature comparing the IIHUM with the EIHI algorithm. Therefore, the state-of-the-art algorithms EIHI, LIHUP, and IIHUM are executed to provide benchmarks to validate the performance of our IDHUP algorithm.

Experimental configuration: The code of EIHI was from the open-source library SPMF[1], and since the codes of LIHUP and IIHUM are unavailable, these two algorithms were implemented by us. All the compared algorithms were implemented in Java language and developed in the Eclipse IDE (2021). Meanwhile, all the experiments were performed on a personal computer with an Intel(R) Core (TM) i7-7700 CPU @ 3.60GHz processor, 8 GB of RAM, and with 64-bit Windows 10 OS.

Parameter settings: In the experiments, algorithms were tested on different datasets with two varying parameters, i.e., *minUtil* and *IR*, to evaluate their performance. Where *IR* represents the insertion ratio, which means the ratio of the number of transactions in the additional database db^+. For example, for a database of size 200, *IR* = 10% implies that there are 20 transactions per insertion and ten updates in total.

5.1 Data Preprocessing

Our experiments were conducted on three publicly available real-world datasets (retail, foodmart, and mushroom[2]) and one synthetic dataset (T10I4D100K). These datasets cover common data features (e.g., sparse or dense, long or short) in real-life scenarios, each with its own characteristics. Details on the characteristics of the datasets are described below.

1. *retail* : it consists of 88,162 customer transactions from an anonymous Belgian store, including 16,470 distinct items, with an average transaction length of 10.3. It is a moderately long and sparse dataset.

2. *foodmart*: this dataset is obtained from the Microsoft SQL-Server 2000 and contains 4141 transactions with 1559 distinct items. It is a small and sparse dataset.

3. *mushroom* : it consists of 8124 transactions with 119 distinct items. Its average transaction length is 23 and density is 19.33%, implying that it is a dense dataset with long transactions.

4. *T10I4D100K* : it is synthesized by the IBM Quest Dataset Generator [3], which has 100K transactions with 870 distinct items, and the average transaction length is 10.1.

In these datasets, only the transactions of foodmart carry real utility information, that is, the quantitative and profit information are real. In consideration of this, we used a simulation model, which has been widely employed in several studies [5, 9], to synthesize external and internal utility values respectively in the [0.01, 1000] and [1, 5] intervals for

1 https://www.philippe-fournier-viger.com/spmf/
2 http://fimi.ua.ac.be/data/
3 http://www.Almaden.ibm.com/cs/quest/syndata/

items in retail and mushroom.

5.2 Effect of the *minUtil* Threshold

In this section, we first evaluated the performance of the proposed IDHUP algorithm on each dataset under a decreasing minimum utility threshold with a fixed insertion ratio of 5%. We ran all algorithms and recorded the accumulated execution time and peak memory usage after 20 updates on each dataset.

Figure 7 depicts the accumulated running time of the four algorithms on each dataset. Clearly, the IDHUP algorithm is significantly faster than other algorithms in any case. For example, for dataset retail with *minUtil* = 1500, IDHUP takes 10.89 seconds, EIHI takes 39.67 seconds, LIHUP takes 697.15 seconds, and IIHUM takes 35.58 seconds. It shows that IDHUP is three to four times faster than the EIHI and IIHUM algorithms, and even an order of magnitude faster than LIHUP. Moreover, the IHDUP algorithm is more adaptable than others. Generally, when *minUtil* is small, the runtime of utility mining algorithms increases sharply due to the growth of the actual search space. For dense dataset mushroom as shown in Figure 7(c), IDHUP works well even for low *minUtil* values, whereas the performance of the other

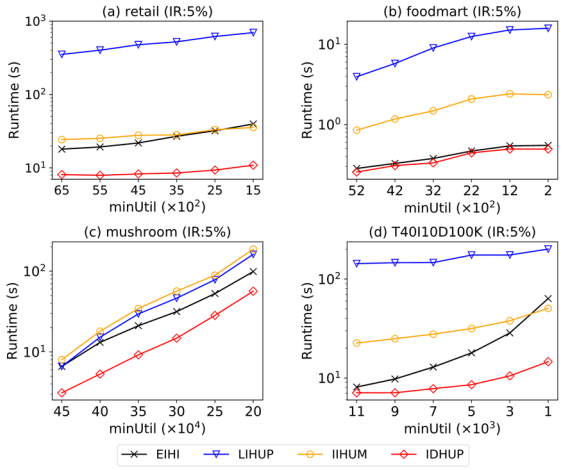

Figure 7. Runtime under varied *minUtil*

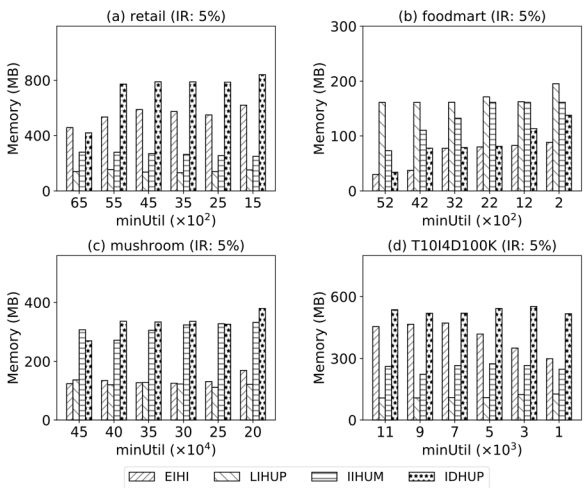

Figure 8. Memory under varied *minUtil*

algorithms degrades considerably. In contrast, for sparse datasets retail and foodmart as shown in Figure 7(a) and Figure 7(b), with the decrease of *minUtil*, the running time of IDHUP increases more slowly than that of the others. IDHUP also outperforms other algorithms on synthetic dataset T10I4D100K as shown in Figure 7(d). This demonstrates that the designed IDHUP algorithm is capable of significantly improving the performance in terms of running time.

Figure 8 presents the peak memory usage for all algorithms on the test datasets. As shown, the IDHUP algorithm performs poorly on other datasets except the foodmart dataset. The reason is that the construction of global structures, EUCS and HUI-trie, inevitably makes IDHUP consume additional memory. In addition, the size of the memory-consuming T-header list in the *ipul* structure is closely related to the number of transactions in datasets. Therefore, IDHUP performs poorly on large datasets retail and T10I4D100K. For dense dataset, the peak memory usage of IDHUP is comparable to that of IIHUM and more than the other two algorithms. In contrast, LIHUP and IIHUM have higher peak memory usage than IDHUP in small dataset foodmart, as this drawback of IDHUP is balanced by the quantitative disparity of visited candidates. It can be concluded that the designed IDHUP algorithm with EUCS-prune and ABS-prune provides a trade-off between running time and memory usage. Nevertheless, the designed IDHUP algorithm is still able to complete all mining tasks with less than 1 GB of memory consumption, making it more suitable for deployment in systems with strong real-time requirements and sufficient memory.

5.3 Effect of the Insertion Ratio

We further conducted experiments under increasing *IR* and a fixed *minUtil*. The *minUtil* of retail, foodmart, mushroom, and T10I4D100K are 4500, 1200, 300K, and 5K, respectively. The total execution time and peak memory usage are presented in Figure 9 and Figure 10.

From Figure 9, it can be observed that the IDHUP algorithm has the best performance on four datasets under various *IR* with a fixed *minUtil*. In some cases, IDHUP is two orders

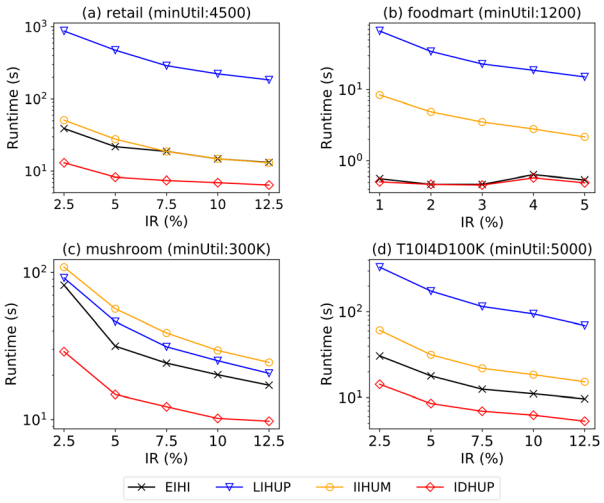

Figure 9. Runtime under varied *IR*

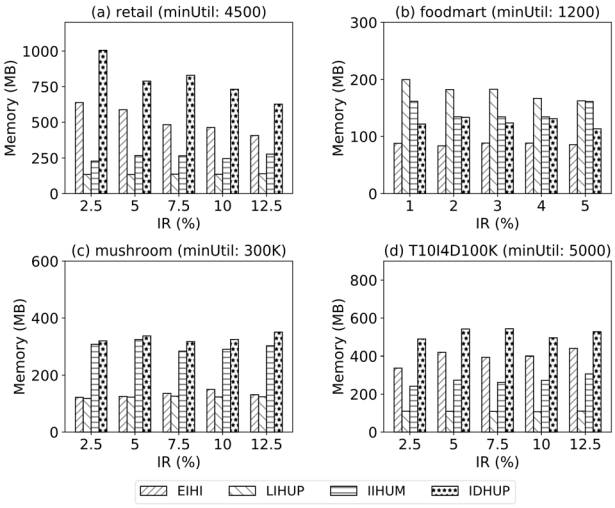

Figure 10. Memory under varied *IR*

of magnitude faster than LIHUP. In addition, the time spent by the IDHUP algorithm is relatively stable when the *IR* gradually decreases, while the other algorithms increase faster, indicating that IDHUP is insensitive to the changes in *IR* and therefore has better adaptability. Similar to the findings of the previous experiments, the designed IDHUP algorithm indeed shows a noticeable improvement in running time.

Figure 10 shows the peak memory consumption on the four datasets. IDHUP consumes more memory than the other algorithms on datasets, e.g., retail, mushroom, and T10I4D100K. Such results agree with our intuition. Similar to the previous analysis, our approach makes a compromise between runtime and memory. Fortunately, all mining tasks can be completed by IDHUP in less than 1 GB of memory, which is within acceptable limits.

6 Conclusions

In this paper, we present a novel and efficient method called IDHUP for discovering high utility patterns from incremental databases. To reduce unnecessary scanning and exploration of the database, we propose the *ipul* structure that stores utility information vertically and constructs extension lists horizontally. In addition, we employ four pruning strategies, namely EUB-prune, EUCS-prune, ABS-prune, and RUR-prune, to enhance the algorithm's ability to filter useless patterns by tightening the utility upper bound and excluding unpromising patterns. IDHUP was compared with EIHI, LIHUP and IIHUM, which are the state-of-the-art algorithms for incremental mining HUPs. Experiments showed that our IDHUP algorithm is more robust than existing algorithms and brings significant improvements in running time. In some cases, IDHUP is even three to four times faster than EIHI and IIHUM, and an order of magnitude faster than LIHUP.

Acknowledgements

This work was supported in part by the Natural Science Foundation of China under grant No. 62072133, Natural Science Foundation of Guangdong Province of China under grant No. 2022A1515011861, and Fujian Key Laboratory of Financial Information Processing (Putian University) (No. JXC202201).

References

[1] W. Gan, J. C.-W. Lin, P. Fournier-Viger, H.-C. Chao, V. S. Tseng, P. S. Yu, A Survey of Utility-Oriented Pattern Mining, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 33, No. 4, pp. 1306-1327, April, 2021.

[2] R. Agrawal, R. Srikant, Fast algorithms for mining association rules in Large Databases, *20th International Conference on Very Large Data Bases (VLDB)*, Santiago de Chile, Chile, 1994, pp. 487-499.

[3] J. Han, J. Pei, Y. Yin, R. Mao, Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach, *Data mining and knowledge discovery*, Vol. 8, No. 1, pp. 53-87, January, 2004.

[4] Y. Liu, W.-K. Liao, A. Choudhary, A two-phase algorithm for fast discovery of high utility itemsets, *9th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, Hanoi, Vietnam, 2005, pp. 689-695.

[5] M. Liu, J. Qu, Mining high utility itemsets without candidate generation, *21st ACM international conference on Information and knowledge management (CIKM)*, Maui, Hawaii, USA, 2012, pp. 55-64.

[6] P. Fournier-Viger, C.-W. Wu, S. Zida, V. S. Tseng, FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning, *21st International symposium on methodologies for intelligent systems (ISMIS)*, Roskilde, Denmark, 2014, pp. 83-92.

[7] S. Krishnamoorthy, Pruning strategies for mining high utility itemsets, *Expert Systems with Applications*, Vol. 42, No. 5, pp. 2371-2381, April, 2015.

[8] A. Y. Peng, Y. S. Koh, P. Riddle, mHUIMiner: A fast high high utility itemset mining algorithm for sparse datasets, *21st Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, Jeju, South Korea, 2017, pp. 196-207.

[9] S. Zida, P. Fournier-Viger, J. C.-W. Lin, C.-W. Wu, V. S. Tseng, EFIM: a fast and memory efficient algorithm for high-utility itemset mining, *Knowledge and Information Systems*, Vol. 51, No. 2, pp. 595-625, May, 2017.

[10] Y. Baek, U. Yun, H. Kim, J. Kim, B. Vo, T. Truong, Z. H. Deng, Approximate high utility itemset mining in noisy environments, *Knowledge-Based Systems*, Vol. 212, Article No. 106596, January, 2021.

[11] L. T. T. Nguyen, V. V. Vu, M. T. H. Lam, T. T. M. Duong, L. T. Manh, T. T. T. Nguyen, B. Vo, H. Fujita, An efficient method for mining high utility closed itemsets, *Information Sciences*, Vol. 495, pp. 78-99, August, 2019.

[12] H. Cheng, M. Han, N. Zhang, X. Li, L. Wang, A Survey of incremental high-utility pattern mining based on storage structure, *Journal of intelligent & fuzzy systems*,

Vol. 41, No. 1, pp. 841-866, August, 2021.
[13] C.-W. Lin, G.-C. Lan, T.-P. Hong, An incremental mining algorithm for high utility itemsets, *Expert Systems with Applications*, Vol. 39, No. 8, pp. 7173-7180, June, 2012.
[14] C.-W. Lin, T.-P. Hong, G.-C. Lan, J.-W. Wong, W.-Y. Lin, Incrementally mining high utility patterns based on pre-large concept, *Applied intelligence*, Vol. 40, No. 2, pp. 343-357, March, 2014.
[15] C. F. Ahmed, S. K. Tanbeer, B.-S. Jeong, Y.-K. Lee, Efficient Tree Structures for High Utility Pattern Mining in Incremental Databases, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 21, No. 12, pp. 1708-1721, December, 2009.
[16] H.-T. Zheng, Z. Li, iCHUM: An Efficient Algorithm for High Utility Mining in Incremental Databases, *8th International Conference on Knowledge Science, Engineering and Management (KSEM)*, Chongqing, China, 2015, pp. 212-223.
[17] J. Lee, U. Yun, G. Lee, E. Yoon, Efficient incremental high utility pattern mining based on pre-large concept, *Engineering Applications of Artificial Intelligence*, Vol. 72, pp. 111-123, June, 2018.
[18] J. C.-W. Lin, W. Gan, T.-P. Hong, B. Zhang, An Incremental High-Utility Mining Algorithm with Transaction Insertion, *The Scientific World Journal*, Vol. 2015, Article No. 161564, February, 2015.
[19] P. Fournier-Viger, J. C.-W. Lin, T. Gueniche, P. Barhate, Efficient Incremental High Utility Itemset Mining, *Proceedings of the ASE BigData & Social Informatics 2015*, Kaohsiung, Taiwan, 2015, pp. 1-6.
[20] U. Yun, H. Ryang, G. Lee, H. Fujita, An efficient algorithm for mining high utility patterns from incremental databases with one database scan, *Knowledge-Based Systems*, Vol. 124, pp. 188-206, May, 2017.
[21] U. Yun, H. Nam, G. Lee, E. Yoon, Efficient approach for incremental high utility pattern mining with indexed list structure, *Future Generation Computer Systems*, Vol. 95, pp. 221-239, June, 2019.
[22] D. W. Cheung, J. Han, V. T. Ng, C. Wong, Maintenance of discovered association rules in large databases: An incremental updating technique, *12th International Conference on Data Engineering (ICDE)*, New Orleans, LA, USA, 1996, pp. 106-114.
[23] J. Liu, X. Ju, X. Zhang, B. C. M. Fung, X. Yang, C. Yu, Incremental mining of high utility patterns in one phase by absence and legacy-based pruning, *IEEE Access*, Vol. 7, pp. 74168-74180, July, 2019.
[24] T.-L. Dam, H. Ramampiaro, K. Norvag, Q.-H. Duong, Towards efficiently mining closed high utility itemsets from incremental databases, *Knowledge-Based Systems*, Vol. 165, pp. 13-29, February, 2019.
[25] U. Yun, H. Nam, J. Kim, H. Kim, Y. Baek, J. Lee, E. Yoon, T. C. Truong, B. Vo, W. Pedrycz, Efficient transaction deleting approach of pre-large based high utility pattern mining in dynamic databases, *Future Generation Computer Systems*, Vol. 103, pp. 58-78, February, 2020.
[26] U. Yun, D. Kim, E. Yoon, H. Fujita, Damped window based high average utility pattern mining over data streams, *Knowledge-Based Systems*, Vol. 144, pp. 188-205, March, 2018.
[27] B. P. Jaysawal, J-W. Huang, SOHUPDS: a single-pass one-phase algorithm for mining high utility patterns over a data stream, *Proceedings of the 35th Annual ACM Symposium on Applied Computing (SAC'20)*, Brno, Czech Republic, 2020, pp. 490-497.
[28] L. T. Nguyen, P. Nguyen, T. D. Nguyen, B. Vo, P. Fournier-Viger, V. S. Tseng, Mining high-utility itemsets in dynamic profit databases, *Knowledge-Based Systems*, Vol. 175, pp. 130-144, July, 2019.
[29] L. T. T. Nguyen, D.-B. Vu, T. D. D. Nguyen, B. Vo, Mining Maximal High Utility Itemsets on Dynamic Profit Databases, *Cybernetics and Systems*, Vol. 51, No. 2, pp. 140-160, February, 2020.
[30] J.-F. Qu, M. Liu, P. Fournier-Viger, Efficient Algorithms for High Utility Itemset Mining Without Candidate Generation, in: P. Fournier-Viger, J. W. Lin, R. Nkambou, B. Vo, V. Tseng (Eds), *High-Utility Pattern Mining: Theory, Algorithms and Applications*. Studies in Big Data, Vol. 51, Springer, Cham, 2019, pp. 131-160.
[31] R. Rymon, Search through systematic set enumeration, *3rd International Conference on Principles of Knowledge Representation and Reasoning (KR 92)*, Cambridge, MA, 1992, pp. 539-550.
[32] B. P. Jaysawal, J.-W. Huang, DMHUPS: Discovering Multiple High Utility Patterns Simultaneously, *Knowledge & Information Systems*, Vol. 59, No. 2, pp. 337-359, May, 2019.

Biographies

Lele Yu received the B.S. degree in Computer Science from Guangdong University of Technology, Guangzhou, China in 2020. She is currently a master student with Guilin University of Electronic Technology, Guilin, China. Her research interests include data mining, utility mining and big data.

Wensheng Gan received the Ph.D. in Computer Science and Technology, Harbin Institute of Technology (Shenzhen), China in 2019. He is currently an Associate Professor with the College of Cyber Security, Jinan University, Guangzhou, China. His research interests include data mining, big data, cybersecurity, artificial intelligence, and blockchain.

Zhixiong Chen received the Ph.D. degree in cryptography from Xidian University in 2006. He is currently a Professor with the School of Mathematics and Finance, Putian University. His research interests include finite fields and their applications, stream ciphers, and Boolean functions.

Yining Liu received the Ph.D. degree in mathematics from Hubei University, Wuhan, China in 2007. He is currently a professor with school of Computer and Information Security, Guilin University of Electronic Technology, Guilin, China. His research interests include data privacy, security and privacy in VANETs, data mining, and machine learning.