TRƯỜNG ĐẠI HỌC FPT

GRADUATION THESIS REPORT

- Artificial Intelligence -

Effective High Utility Itemsets Mining Algorithm for Incremental Database

Supervisor

Associate Professor Phan Duy Hung

Group Members

1. Do Thanh Cong HE150385
2. Do Mai Phuong HE150375
3. Pham Duc Duong HE160384

ACKNOWLEDGEMENT

We wish to express our appreciation for the invaluable assistance and support extended to us, provided by various organizations, agencies, and individuals. As a result, we successfully completed the capstone project and obtained our Bachelors of Artificial Intelligence from FPT University.

First and foremost, our group is eager to convey our gratitude to FPT University for fostering an environment characterized by vibrancy and creativity. We are sincerely thankful for the opportunity it afforded us to engage in practical experiences, acquire priceless life lessons, and instill strong personal values.

Secondly, our heartfelt gratitude goes to our esteemed supervisors, Mr.Phan Duy Hung and Mr.Tran Van Ha, for their unwavering guidance, support, and encouragement during the course of our graduation project. Without their profound expertise and motivating influence, the successful completion of this project would not have been feasible.

Last but certainly not least, we wish to express our gratitude to our friends, families, and parents, who served as a continual source of inspiration and offered invaluable assistance in bringing this project to its successful conclusion.

This journey would not have been possible without the nurturing environment of FPT University and the support of all those who contributed to our accomplishment.

FPT University, Hanoi, Fall Semester 2023

The authors of this thesis

ABSTRACT

High-utility itemset mining (HUIM) majors have done a lot of research lately, the past few years. Almost all published algorithms focus on processing static databases, which do not utilize previously mined information to mine incremental databases. To solve this problem, some incremental HUIM algorithms were published and showed the possibility of development. In this study, a new algorithm named iHUIM based on the EIHI algorithm was improved. Unlike EIHI, which requires twice database scans, the iHUIM just scans the database only once. Additionally, using compact utility lists and some pruning strategies, iHUIM shows outperformance EIHI regarding the length of execution time and has a slight improvement in memory consumption.

Keywords: Compact Utility List, High-utility Itemset Mining, Incremental Databases, Data Mining

Table of contents

List of abbreviations

Number	Abbreviations	Description
1	HUIM	High Utility Itemset Mining
2	iHUIM	Incremental High Utility Itemset Mining
3	EIHI	Efficient Incremental High Utility Itemset
4	FIM	Frequent Itemset Mining
5	HUI	High Utility Itemset
6	CUList	Compact Utility List
7	MCUL	Modify Compact Utility List

List of Tables

List of Figures

1. INTRODUCTION

1.1. Motivation

In the era of data explosion, knowledge discovery is receiving more research attention. One prominent area in there is Data Mining. Data Mining has been applied a lot in life and has contributed greatly to the business industry. In recent years, HUIM task has been focused on research because of the benefits it brings. Unlike Frequent Itemsets Mining (FIM), which searches for high-frequency itemsets, HUIM focuses on itemsets that bring high profits. Many algorithms have been developed to solve this problem for instance HUI-Miner, MLHMiner, HMiner,…However, most algorithms are developed to search on static databases, but in real life, the database will be continuously updated incrementally, requiring flexible and effective mining methods to handle that change. This is a challenge and it motivated us to research this topic.

1.2. Related work

HUIM is a task that extends the FIM task. HUIM focuses on finding an itemset which appears frequently in each transaction and has high profit. Almost all HUIM algorithms using the user-defined minimum utility threshold to consider an itemset is a HUI. In the early period of HUIM, based on Apriori [1] – an FIM algorithm, researchers proposed a concept that was suitable for HUIM, named TwoPhase. Main process includes generating candidates in one phase and determining which set is high-utility sets based on those candidates in the other phase. Some algorithms using this concept may be mentioned as: Two-Phase [2] , UP-Growth [3], and UP-Growth+ [4]… This concept effectively eliminates the candidates and precisely identifies a comprehensive set of HUIs. Nevertheless, these algorithms also face time-consuming and intensive memory challenges when handling long transactions or threshold is set too low. To overcome these challenges, many studies in the subsequent phase aimed to overcome these drawbacks. The algorithms that have one-phase are proposed like HUI-Miner [5], FHM [6], EFIM [7], and HMiner [8] to reduce computation time and minimize memory usage.

The proposed algorithms mentioned above have shown high efficiency and practical applications. But in real life, a store's database continuously grows over time as transactions are recorded and updated either in real-time or at regular intervals. Therefore, the database becomes increasingly larger. In this case, even if the algorithm works effectively in a static database, these still have a problem with memory usage and execution time. When adding a small number of transactions, previous algorithms still require rescanning the whole database. This is time-consuming, wastes resources and does not take advantage of the results of the previous calculations. Therefore, a new problem has attracted researchers to explore and find a solution: high utility itemset mining on incremental databases.

It is obvious that many researches about incremental high-utility itemsets mining have been done and got good results. Basically, incremental HUIM algorithms can be classified into three main categories: list-based, tree-based and Apriori-based approaches.

Forward to 2012, Lin et al [9] suggested an algorithm called FUP-HUI-INS. This algorithm's core is the Fast Update concept (FUP). When a new transaction was added, it was treated as 1 in 4 cases: small-small, small-large, large-large or large-small. The disadvantage of this concept is the large search space and it increases exponentially. The authors continued their research and proposed an algorithm called PRE-HUI-INS [10], applied the pre-large concept by Hong et al [11] and the TWU model using tree-base structure; It employs a level-wise approach for mining updated HUIs [12]. A pre-large itemset is not genuinely considered large but is expected to become large at a later time [11]. After labeling, itemsets were divided into one of nine cases. Only a few cases needed to be considered because they affected the formation of HUIMs. PRE-HUI-INS demonstrated efficiency compared to the HUIM algorithms for static databases and reduced the count of database scans in comparison to FUP-HUI-INS [9]. Following that, two algorithms, PRE-HUI-MOD [13] and PRE-HUI-DEL [14], were proposed. PRE-HUI-MOD was found to have the shortest runtime but required more memory compared to the three algorithms used for comparison: Two-Phase, HUI-Miner and FHM. The PRE-

HUI-DEL algorithm ran faster in most cases but became less efficient when faced with a large number of deleted transactions. In [15], the authors developed the PIHUP (Pre-large Incremental High Utility Patterns) algorithm that scanned databases only once. When new transactions were added, the PIHUP-tree was updated based on the pre-large concept.

The HUI-List-INS algorithm was suggested in paper [16] for Mining High Utility Itemset when adding new transactions. This algorithm uses a data structure that was first used in HUI-Miner named Utility list. The utility list contains both utility details regarding the itemsets and heuristic information indicating whether prune should be applied to the itemsets [17]. To construct a utility list of larger itemset, we only need to join the utility list of smaller items. Using utility lists also helps to prune by remaining utility of an item. Beside advantage, utility list requires additional memory to store heuristic information that is prepared for prunning strategies. An extension of HUI-List-INS, HUI-List-DEL [18] was proposed in 2016. In the same year, Philippe et al [19] published the EIHI (Efficient Incremental High-utility Itemset) algorithm. This algorithm effectively combined list-based and tree-based structures. Utility lists were built for each itemset and transaction, while HUIs were represented in a trie-like structure. The flexible use of these two structures, combined with LA prune optimization, allowed EIHI to demonstrate superior efficiency in terms of runtime while maintaining similar memory requirements to HUI-List-INS.

In 2017, Unil Yun and colleagues developed an efficient algorithm with one database scan called LIHUP [20]. The authors used a global list structure that contained utility lists of candidates. Updating the global list did not require rescanning the original database, resulting in less time consumption. This thesis proposed a technique to enhance the efficiency of the updated data structure through a restructuring procedure and reflect newly added transactions by scanning them, rather than the entire database, just once.

Incremental HUIM has done much research in recent years. Many solutions were proposed to resolve the problem of memory consumption and execution time.

Pre-large concept, utility list, CUList, and many pruning strategies work effectively when applied to build an algorithm. We expect to combine some method pruning and data structure to enhance the performance of the HUIM algorithm.

1.3. Objectives

The aim of this research is to enhance the efficiency of the algorithm used in High utility itemset mining on incremental databases. We present EIHI's extension algorithm that uses the Modify Compact Utility List (MCUL) data structure to store information while conducting the mining process. Based on the data compression advantages of the CUList structure, combined with pruning strategies and efficient use of using trie-like structure to store High-utility itemset after mining, thereby shortening calculation time and improving performance.

2. METHODOLOGY

2.1. Preliminaries

Within this section, we provide the definitions for important terminologies utilized in this study. The essential definitions and the statement of the problem are as follow:

Given $I = \{i_1, i_2, ...i_m\}$ is a finite set of m items. X is a finite collection of items, denoted as $X \subseteq I$. An itemset $X = \{i_1, i_2, ...i_k\}$ is *k-itemset*, where k is the length of X and i_k is the last item in X.

Each item $i \in I$ has a positive integer $pr(i)$ that associated with it is called *price* of an item.

$D = \{T_1, T_2, ...T_n\}$ is an original *transaction database* that contains a collection of transactions, each transaction is identified by T_{id}.

$q(i, T_{id})$ is the *quantity* of i in T_{id}.

N is *additional transactions incorporated into* the original database.

U represents the *database after adding new transactions* $U = D \cup N$

Table 1 illustrates an example original database, example of new transaction N is provided in Table 2 and Table 3 is sample price table.

Table 1: An original database D

Tid	Transactions	Quantity (q)	Transaction utility
T1	a, b, c, d, e, f	1, 5, 1, 3, 1, 1	30
T2	b, c, d, e	4, 3, 3, 1	20
T3	a, c, d	1, 1, 1	8
T4	a, c, e, g	2, 6, 2, 5	27

Table 2: A sample of new transaction N

Tid	Transactions	Quantity (q)	Transaction utility
T5	b, c, e, g	2, 2, 1, 2	11

Table 3: A sample price table

Item	a	b	c	d	e	f	g
Price of item	5	2	1	2	3	5	1

Definition 1 (Utility value):

- The utility of an item i_x in a transaction T_{id} is calculated by multiplying the utility of item and its quantity, denoted as:

$$u(i_x, T_{id}) = q(i_x, T_{id}) \times pr(i_x)$$

- The utility of an item i_x in database is calculated by following formula:

$$u(i_x) = \sum_{i_x \in T_{id}, T_{id} \in D} u(i_x, T_{id})$$

- The utility of an itemset X in a transaction T_{id}, denoted as $u(X, T_{id})$, is calculated by following formula:

$$u(X, T_{id}) = \sum_{i_x \in X} u(i_x, T_{id})$$

- The utility of an itemset X in database D, denoted and calculated as:

$$u(X) = \sum_{X \subseteq T_{id}, T_{id} \in D} u(X, T_{id})$$

- The utility of a transaction $u(T_{id})$ is total utility of all items belong in transaction T_{id}, formula is:

$$u(T_{id}) = \sum_{i_x \in X, X \subseteq T_{id}} u(i_x, T_{id})$$

From Table 1:

Utility of item {a} in T_1 is:

$$u(\{a\}, T_1) = q(\{a\}, T_1) \times pr(\{a\}) = 1 \times 5 = 5$$

Utility of item {a} in database is:

$$u(\{a\}) = u(\{a\}, T_1) + u(\{a\}, T_3) + u(\{a\}, T_4) = 5 + 5 + 10 = 20$$

Utility of $X = \{a,b\}$ in transaction T_1 :

$$u(X,T_1) = u(\{a\},T_1) + u(\{b\},T_1) = 5+10 = 15$$

Utility of $X = \{c,d\}$ in database is:

$$u(X) = u(X,T_1) + u(X,T_2) + u(X,T_3) = 7+9+3 = 19$$

Utility of transaction T_3 is calculated as:

$$u(T_3) = u(\{a\},T_3) + u(\{c\},T_3) + u(\{d\},T_3) = 5+1+2 = 8$$

Definition 2 (Transaction weighted utility): The transaction-weighted utility of an itemset ($TWU(X)$) is the summing up of the transaction utilities of all transactions that X belong to. The formula is: $TWU(X) = \sum_{X \subseteq T_{id} \in D} u(T_{id})$

As an illustration, with itemset $X = \{b\}$, $TWU(X) = u(T_1) + u(T_2) = 30+20 = 50$

The TWU of 1-itemsets are shown in Table 4 below.

Table 4: Transaction weighted utility of 1-itemset

Item	g	f	b	d	a	e	c
TWU	27	30	50	58	65	77	85

The TWU of larger itemsets are calculated similarly. Consider itemset $X = \{a,b\}$, TWU value of X is $TWU(X) = u(T_1) = 30$

Property 1 (TWU pruning strategy):

For any itemset X, if TWU of X is less than minUtil then TWU of all any suppersets of X is also less than minUtil. Mathematically, If $TWU(X) < \min Util$ *then* $\forall X' \supseteq X$, *we have* $TWU(X') \le TWU(X) < \min Util$.

From Table 4, let *minUtil=30* , itemset $X = \{g\}$ has $TWU(X) = 27 < \min Util$. According to Property 1, any supersets of {g} cannot be greater than 27. Therefore, this itemset is pruned from search space and superset of X do not need to be formed.

Definition 3 (Order of items):

For the purpose of this work, we sorted all items in the database according to TWU of each item in ascending order.

The ordering of items in this example is: g, f, b, d, a, e, c

Table 5 below shows the database after sorting by TWU.

Table 5: The database after sorted by the order of TWU

Tid	Transactions	Profit	Transaction utility
T1	f, b, d, a, e, c	5, 10, 6, 5, 3, 1	30
T2	b, d, e, c	8, 6, 3, 3	20
T3	d, a, c	2, 5, 1	8
T4	g, a, e, c	5, 10, 6, 6	27
T5	g, b, e, c	4, 2, 3, 2	11

Definition 4 (Set of items after an itemset):

In sorted transaction T_{id}, the collection of all items after X is denoted as T_{id} / X. The overall count of items in this set is $s(T_{id} / X)$

For instance, in Table 5, consider $X = \{g, a\}$:

Set of all items after X in transaction T_4 : $T_4 / X = \{e, c\}$

Total number of items after X in T_4 : $s(T_4 / X) = |\{e, c\}| = 2$

Definition 5 (Remaining utility):

- The calculation of the remaining utility of X in a transaction involves summing up the utility of all items after X in transaction. The formula is:

$$ru(X, T_{id}) = \sum_{i_x \in (T_{id} / X)} u(i_x, T_{id})$$

- The remaining utility of an itemset in database is denoted and calculated as:

$$ru(X) = \sum_{X \subseteq T_{id} \in D} ru(X, T_{id})$$

In Table 5, consider itemset $X = \{b, d\}$ in T_1 :

Set of item after X in T_1 is $\{a, u, c\}$. So, the remaining utility of X in T_1 is:

$$ru(X, T_1) = u(a, T_1) + u(e, T_1) + u(c, T_1) = 5 + 3 + 1 = 9$$

In sample database, itemset X appears in two transactions: T_1 , T_2 .

Therefore, remaining utility of X in database is:

$$ru(X) = ru(X, T_1) + ru(X, T_2) = 9 + 6 = 15$$

Definition 6 (Extension of itemset):

In the set of all items after ordering by TWU, extension of itemset X is all items after X. The number of all X's extensions if denoted as $c(X)$.

From the order that presented in previous, the extension of itemset $X = \{f, d\}$ is $\{a, c, e\}$ and $c(\{a, c, e\}) = 3$

Definition 7 (Closed utility of itemset):

- Closed utility of an itemset X in transaction T_{id}, is calculated as:

$$cu(X, T_{id}) = \begin{cases} u(X, T_{id}) \text{ if } |X| > 1 \text{ and } c(X - i_k) = s(T_{id} / X - i_k) \\ 0, \text{ otherwise} \end{cases}$$

  In formula, $c(X - i_k)$ is the number of items in extension of X except last item

- Closed utility of an itemset X with size ≤ 2 in database, denoted and calculated as: $cu(X) = \sum_{X \subseteq T_{id} \in D} cu(X, T_{id})$

For instance, in Table 5:

Closed utility of itemset $X = \{g\}$ in T_4 is $cu(X, T_4) = 0$ because $|X| = 1$

Consider itemset $X = \{d, c\}$ in T_1 :

Following are steps to calculate closed utility of itemset X in transaction T_1 :

Firstly, the number of items in itemset X is $|X| = 2 > 1$

Secondly, the number of set of X's extension except last item is

$$c(X - \{c\}) = c(d) = |\{a, e, c\}| = 3$$

Finally, total number of item after X exclude last item is

$$s(T_1 / (X - \{c\})) = s(T_1 / d) = |\{a, e, c\}| = 3$$

Therefore, $cu(X, T_1) = u(X, T_1) = 7$

Closed utility of X in database is $cu(X) = cu(X, T_1) + cu(X, T_2) + cu(X, T_3)$

$cu(X, T_2)$ and $cu(X, T_3)$ values are calculated similarly with $cu(X, T_1)$ that presented before.

Because $s(T_2 / X - \{c\}) = |\{e,c\}| = 2 \neq 3$ and $s(T_3 / X - \{c\}) = |\{a,c\}| = 2 \neq 3$, so $cu(X, T_2) = cu(X, T_3) = 0$. Therefore, $cu(X) = cu(X, T_1) = 7$

Definition 8 (Closed remaining utility):

- Closed remaining utility of an itemset X in transaction T_{id}, is denoted and calculated as:

$$cru(X, T_{id}) = \begin{cases} ru(X, T_{id}), \text{ if } |X| > 1 \text{ and } c(X - i_k) = s(T_{id} / X - i_k) \\ 0, \text{ otherwise} \end{cases}$$

- Closed remaining utility of an itemset X with size ≤ 2 in database, denoted and calculated as: $cru(X) = \sum_{X \subseteq T_{id} \in D} cru(X, T_{id})$

For example, in Table 5:

Closed remaining utility of itemset $X = \{e\}$ in T_2 is $cru(X, T_2) = 0$ because $|X| = 1$

Consider itemset $X = \{a,e\}$ in T_4 :

Following steps are used to calculated the closed remaining utility of X in T_4 :

Firstly, calculating the number of item in X: $|X| = 2 > 1$

Secondly, calculating the number of item after X exclude last item in ordered set of all item: $c(X - \{e\}) = |\{e,c\}| = 2$

Finally, calculating the number of item after X exclude last item in T_4 :

$s(T_4 / X - \{e\}) = s(T_1 / a) = |\{e,c\}| = 2$

Therefore, $cru(X, T_4) = ru(X, T_4) = 6 + 6 = 12$

Consider itemset $X = \{d,a\}$ in database:

Closed remaining utility of X in database is $cru(X) = cru(X, T_1) + cru(X, T_3)$

Process of computing $cru(X, T_1)$ and $cru(X, T_3)$ are similar to previous.

Because $s(T_1 / X - \{a\}) = |\{a,e,c\}| = 3$ and $s(T_3 / X - \{a\}) = |\{a,c\}| = 2 \neq 3$, so $cru(X, T_1) = ru(X, T_1) = 4$ and $cru(X, T_3) = 0$

Therefore, $cru(X) = ru(X, T_1) = 4$

Definition 9 (Non-closed utility and non-closed remaining utility of itemset):

- The definition of non-closed utility of an itemset is:

$$nu(X) = u(X) - cu(X)$$

- The definition of non-closed remaining utility of an itemset is:

$$nru(X) = ru(X) - cru(X)$$

For example, itemset $X = \{d, c\}$ has $nu(X) = u(X) - cu(X) = 19 - 7 = 12$ and $nru(X) = ru(X) - cru(X) = 19 - 9 = 10$.

Definition 10 (High-utility itemset):

An itemset is called a *high-utility itemset (HUI)* if its utility is equal to or greater than a *user-defined minimum utility threshold (minUtil)*. Alternatively, it is referred to as *a low-utility itemset*.

For example original database, let *minUtil=30*, itemsets $\{b,d\}$, $\{a,c,e\}$ with utility values 30 and 36 respectively, are HUIs. Itemset $\{a,g\}$ is an example of a low utility itemset.

Property 2 (Downward closure):

For any itemset X, if X is not a high-utility itemset, any superset of X is not a high-utility itemset.

This property brings an efficient way to reduce the number of candidates in the candidate generation phase. That means if an itemset X is not a HUI, it does not need to form a superset of X anymore.

Property 3 (U-Prune):

If the cumulative suffix utility of X is below the minUtil threshold, then none of the supersets of X qualifies as a high-utility itemset. That is, if $u(X) + ru(X) < \min Util$ *then* $X' \notin HUIs, \forall X' \supseteq X$.

Property 4 (LA-Prune):

Given two itemset X and Y, if

$$cu(X) + cru(X) + nu(X) + nru(X) - \sum_{\forall T_j \in D, X \subseteq T_j \text{ and } Y \not\subset T_j} nu(X, T_j) + nru(X, T_j) < \min Util$$

then $\forall Y' \supseteq Y$ and $X' \supseteq X$, $X'Y' \notin HUIs$

Two properties 3 and 4 are proposed and proof in [8].

2.2. Problem definition

Given a *transaction database* D, new transactions N, a minimum utility threshold *minUtil*-specific by user, the objective is to mine all HUIs with minimum cost of utilization of memory and the time taken for execution.

The set of all HUIs found in original database D is called H. The set of all HUIs found when added new transactions N is called H'.

For example, with $\min Util = 30$, the set H is all HUIs in D is presented in Table 6. After adding new transaction N, $H' = H \cup \{be, bce\}$ is the set of HUIs found and updated utility value of existing HUIs are presented in Table 7.

Table 6: The set H of all HUIs found in D

HUIs	Utility value
b, d	30
b, c, d	31
b, d, e	34
a, c, e	36
b, c, d, e	40
a, b, c, d, e	30

Table 7: The set of all HUIs found in U

HUIs	Utility value
b, d	30
b, e	**31**
b, c, d	31
b, d, e	34
b, c, e	**37**
a, c, e	36
b, c, d, e	40
a, b, c, d, e	30

2.3. Proposed algorithm

2.3.1. Modified Compact Utility List Structure

The first time Compact Utility List was introduced is in paper [8]. Authors presented form of CUList include: $cu(X), cru(X), cpu(X)$ and

$< T_{id}, nu(X,T_j), nru(X,T_j), npu(X,T_j), PPOS(X,T_j) >$. This structure, according to experiments conducted by authors, shows efficiency in compactly storing

information and during mining process. Results show reduction in utility list size for both sparse and dense datasets [8].

Based on that result, we modify CUList to fit our process and strategy. A Modify Compact Utility List (MCUL) of an itemset is a data structure that contains: cu(X), cru(X) and list of quintuples $< T_{id}, nu(X, T_{id}), nru(X, T_{id}) >$. The overall structure of MCUL is presented below.

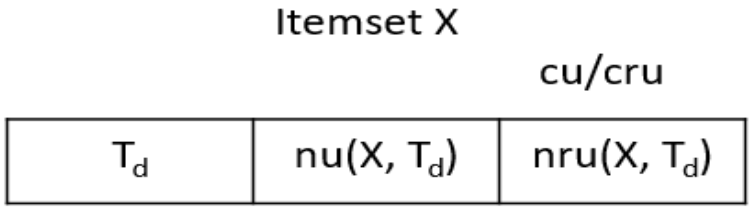


Figure 1: Modified Compact Utility List

2.3.2. HUIs storage structure

After finding an itemset X is a HUI, we need a structure to store HUI and a mechanism to quickly retrieve and update utility of itemsets. A trie-like [8] is proposed for this reason. In trie, an item is a node and an itemset is a path from the root. Last node in the path, in addition to representing the last item, also contains the utility value of that itemset. This design allows easy modification of the utility value of an itemset when detecting the updating of HUIs.

For previous result, an example of HUIs found in D that stored in H is present in Figure 2.

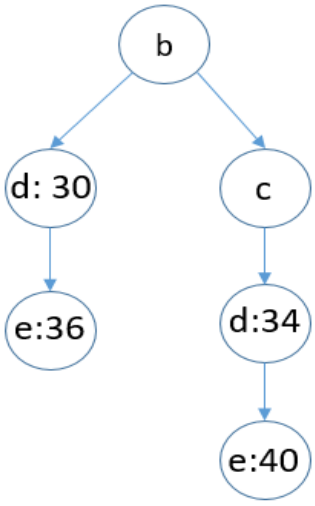


Figure 2: Example of HUI-trie structure

According to Table 7, when database is updated, {*be*} and {*bce*} are two new HUIs. Utility values of some existing HUIs are updated. Part of those updates are shown in Figure 3 below.

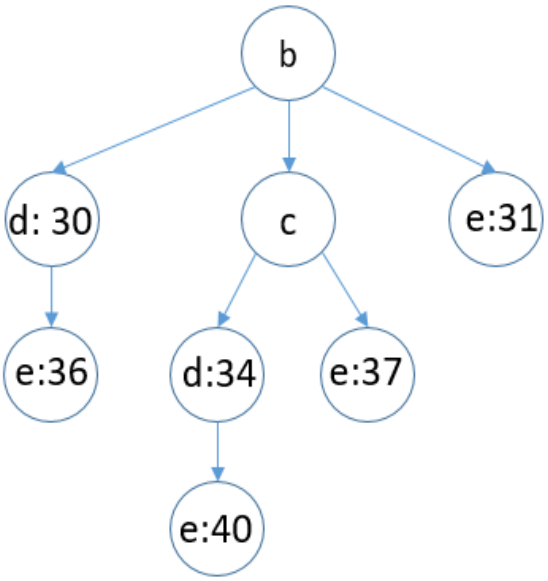


Figure 3: Example of updated HUI-trie

2.3.3. iHUIM Algorithm

This section presents our algorithm, named iHUIM algorithm. First, the inputs of Main procedure (algorithm 1) are transaction database D (for the first time) or N (when we want to add new transactions) and *minUtil* threshold specific by user. Main

process starts by scanning database to build MCUL of each distinct item and calculate TWU of them. If the item was processed in previous, the MCUL and TWU values are still needed to be updated. Then, sort MCUL of all items according to ascending TWU and store MCUL of all items that have TWU larger than *minUtil*. Second, Restruct procedure (algorithm 2) will be called to restruct each MCUL in *MCULs*. After that, the Search procedure (algorithm 3) starts first with *prefix = null*, the list of all MCUL and *minUtil*, then this algorithm is called recursive to find all HUIs.

Algorithm 1: Main

Input: *D (or N), minUtil*

Output: *HUIs*

```
1:  foreach transaction T in D do
2:          foreach item X in T do
3:                  create or update MCUL(X), TWU(C)
4:          end for
5:  end for
6:  Sort All_MCULs according to TWU
7:  if TWU(X) ≥ minUtil then
8:      MCULs ←X
9:      FinalMCULs = Restruct (MCULs)
10:     SearchHUI (null, FinalMCULs, minUtil)
```

Algorithm 2 takes input as a list of MCUL that have $TWU \geq \min Util$. This procedure starts from *lastMCUL* in the list. For each element *ex* in *lastMCUL*, algorithm calculates the sum of *ex.nu* and *ex.nru* to store in *TempTable* in the *ex.tid* index (Line 3). Then calculating the remaining utility of each element in MCUL before *lastMCUL* by taking the value according to the *ey.tid* index in *TempTable* (Line 7). Line 8 updates the value in *TempTable* by the sum of *ey.Nu* and *ey.Nru*. After every MCUL was processed, return *FinalMCULs* after restructuring (Line 11).

Algorithm 2: Restruct

Input: *MCULs*

Output: *FinalMCULs*

1: lastMCUL = Last MCUL in MCULs

2: **foreach** ex ∈ lastMCUL **do**

3: TempTable(ex.tid) ← ex.nu + ex.nru

4: **end for**

5: **foreach** Y before lastMCUL in MCULs **do**

6: **foreach** ey ∈ Y **do**

7: ey.nru ← TempTable(ey.tid)

8: TempTable(ey.tid) ← ey.nu + ey.nru

9: **end for**

10: **end for**

11: **Return** *FinalMCULs*

Algorithm 3 (SearchHUI) takes input as *prefix-item*, *FinalMCULs* and *minUtil* value. For each item X in *FinalMCULs*, first check the condition to be a HUI (Line 2-3). If yes, X is HUI. If not, check pruning condition was discussed in Property 3 (Line 5). If the property is satisfied, call ConstructExtOfX (Line 6) then call SearchHUI algorithm for item X and extension of X (Line 7).

Algorithm 3: SearchHUI

Input: *prefix-item, FinalMCULs, minUtil*

Output: *HUIs*

1: **foreach** X in FinalMCULs **do**

2: **if** u(X) = nu(X) + cu(X) ≥ minUtil **then**

3: HUIs ← X

4: **end if**

5: **if** u(X) + ru(X) ≥ minUtil **then**

6: exMCULs = ConstructExtOfX(X, MCULs)

7: SearchHUI (X, exMCULs, minUtil)

8: **end if**

9: **end for**

Algorithm 4 presents the process of constructing extension MCUL (*exMCULs*) of prefix X. Line 1-3 process each MCUL Y after X in *FinalMCULs*, initialize extension MCUL of Y called *Pxy* and *uLA* value to prepare for LA-prune. Line 4 traverses each element *ex* in X to find element *ey* in Y that satisfy *ey.tid =ex.tid* (Line 5). Line 6 add element *ey* to *Pxy* if found *ey*. Line 7 handles case where *ey* doesn't exist. *uLA* value will decrease by (*ex.nu* + *ex.nru*) and then check LA-prune. If not satisfy, stop construct *Pxy*. Line 13 add *Pxy* into *exMCULs*. Line 15 returns output of Algorithm 4 is *exMCULs* of prefix X.

Algorithm 4: ConstructExtOfX

Input: *X: MCUList Prefix, FinalMCULs, minUtil*

Output: *exMCULs*

```
1:  foreach Y after X in FinalMCULs do
2:      Pxy = {}
3:      U_LA = nu(X) + nru(X) + cu(X) + cru(X)
4:      foreach element ex in X do
5:          if ∃ ey ⊂ Y && ey.tid = ex.tid then
6:              Pxy ← Pxy U {ey}
7:          else:
8:              U_LA = U_LA - (ex.Nu + ex.Nru)
9:              if U_LA < minUtil then
10:                 stop construct Pxy
11:             end if
12:     end for
13:     exMCULs ← exMCULs U {Pxy}
14: end for
15: Return exMCULs
```

When adding new transactions (N), a naïve approach to find the set H' in N, given D and H is applied to Algorithm 1 on database U. It is inefficient because it would mine from scratch that does not take advantage of results from previous mining. One more reason is the abundance of itemsets appearing in D may not appear in N. From these observations, three properties below describe applying pruning techniques to decrease the search space and maximize the advantage of previous minings.

Property 5 (Pruning condition 1): *Let X is HUI appearing in D but not in N and denote U=D U N. The utility of X in U is the same as in D. So, if X is a HUI in D, it is also a HUI in U. Similarly, if X is not HUI in D, it is still not HUI in U.*

Property 6 (Pruning condition 2): *If MCUL of itemset X in N is empty, then X and all extensions of X do not need to be explored.*

Property 7: *For any itemset X, if the combined value of utility and remaining utility in dataset D, along with the utility and remaining utility in dataset N, is below the minUtil threshold, then both X and its extensions are considered low-utility itemsets.*

3. EXPERIMENTS

3.1. Data preparation

To benchmark iHUIM algorithm, three datasets Chess, Mushroom, Fruithut are used. These Datasets are acquired or retrieved from the SPMF open-source platform data mining library [21]. Properties of these datasets such as: |D| presents transaction count of database D, |I| presents the count of unique items in the database, AvgLen calculates the average length of transaction, and is provided in Table 8 below.

Table 8: Characteristic of datasets

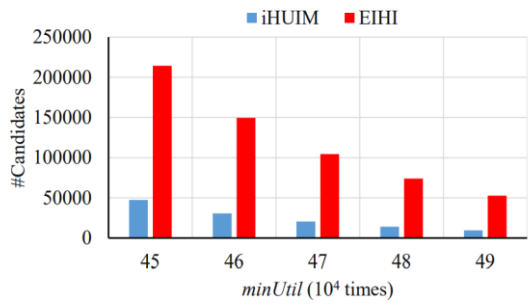
Dataset	|D|	|I|	AvgLen	Type	Has real utility value
Chess	3196	75	37	Dense	No
Fruithut	181970	1265	3.58	Sparse	Yes
Mushroom	8124	119	23	Dense	No

3.2. Experiments setup

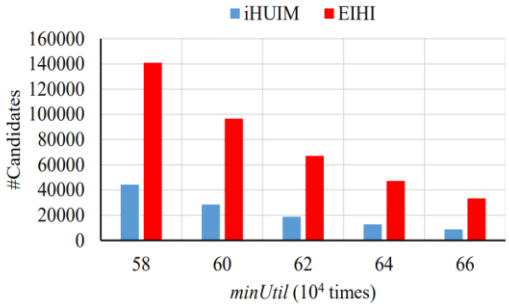
All experiments are carried out in an environment with the following configuration: Intel® Core I7, 2.20GHz, 16GB RAM, Microsoft Window 11. We implemented our algorithm- iHUIM and compared it with the EIHI algorithm to evaluate 3 criterias: number of generated candidates, execution time and memory usage. We run experiments with various parameters. We set up parameters by fixing *minUtil* varying *addRatio* and fixing *addRatio* varying *minUtil*. Each pair of parameters, we run 5 times then calculate the average of them.

3.3. Results and Analysis

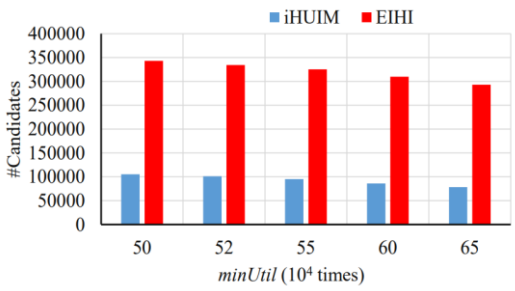
3.3.1. Number of generated candidates



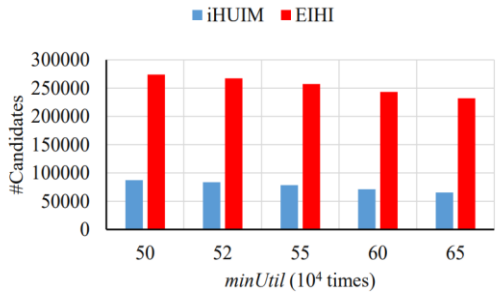
a-Chess-addRatio=20%



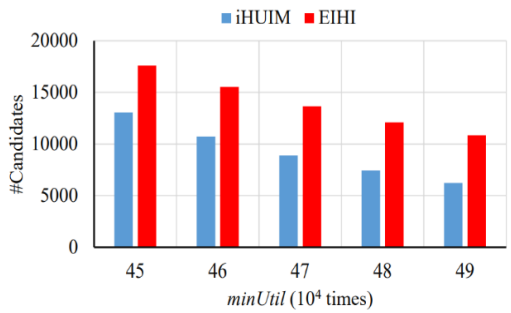
b-Chess-addRatio=25%



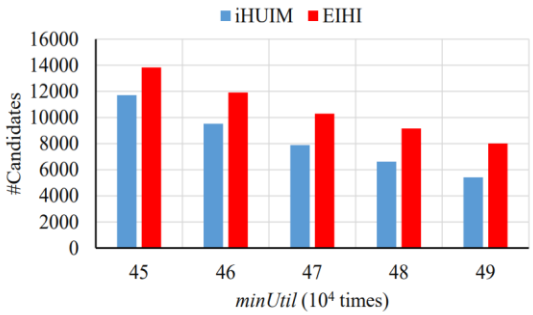
c-Fruithut-addRatio=20%



d-Fruithut-addRatio=25%



e-Mushroom-addRatio=20%

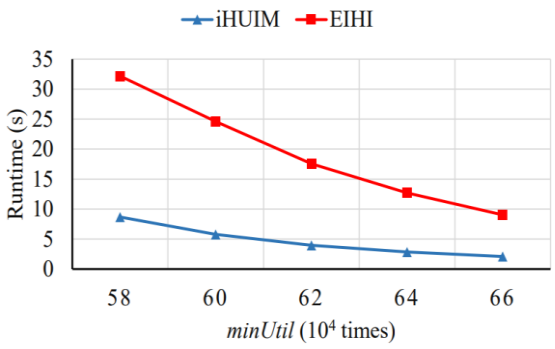


f-Mushroom-addRatio=25%

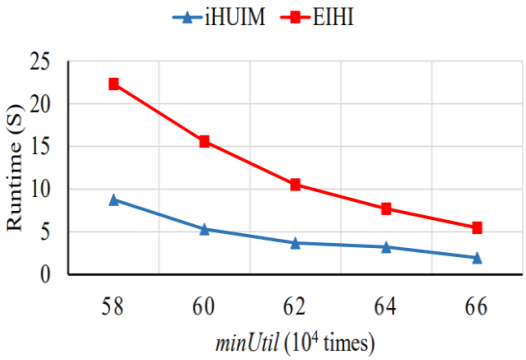
Figure 4: Comparison of number of generated candidates

In the first experiment, we evaluated the number of generated candidates that can be reduced by our algorithm. Figure 4 illustrates the count of candidates for two algorithms that execute with dissimilar databases with different *minUtil* values. As shown in Figure 4, the iHUIM reduces the large number of candidates compared with EIHI. For example, when running the algorithm with a Chess dataset, iHUIM can reduce over 68% of candidates compared to EIHI in case *minUtil=580000* and *addRatio=25%*, or with Fruithut dataset, the candidate generated by iHUIM less than 70% in case *minUtil=500000* and *addRatio=20%.* Throughout all test cases, iHUIM showed outperformance when the candidate generated by iHUIM was quite fewer than EIHI.

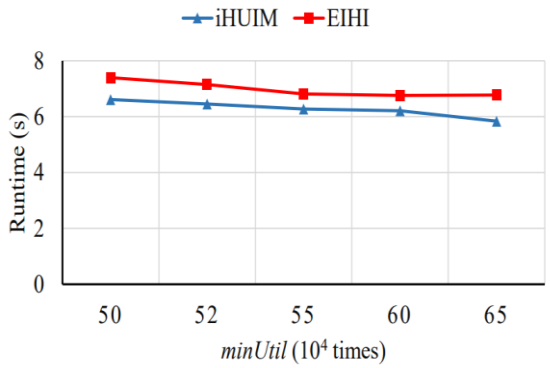
3.3.2. Execution time



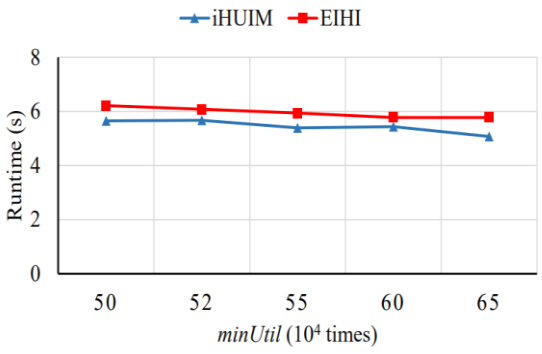
a-Chess-addRatio=20%



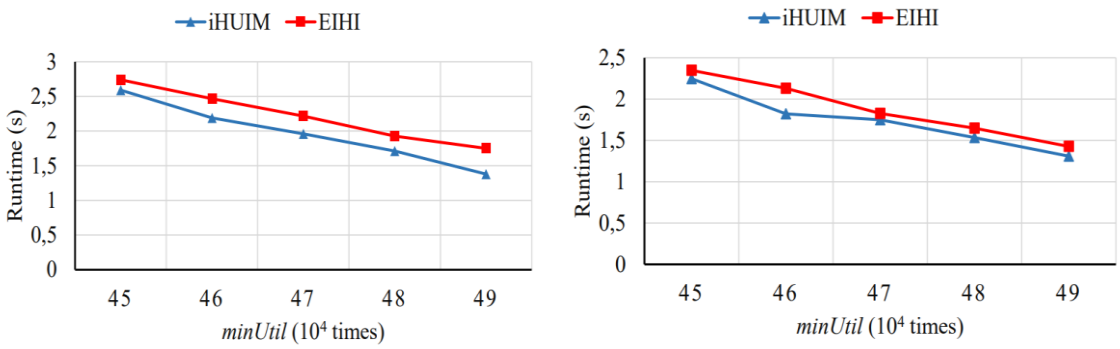
b-Chess-addRatio=25%



c-Fruithut-addRatio=20%



d-Fruithut-addRatio=25%

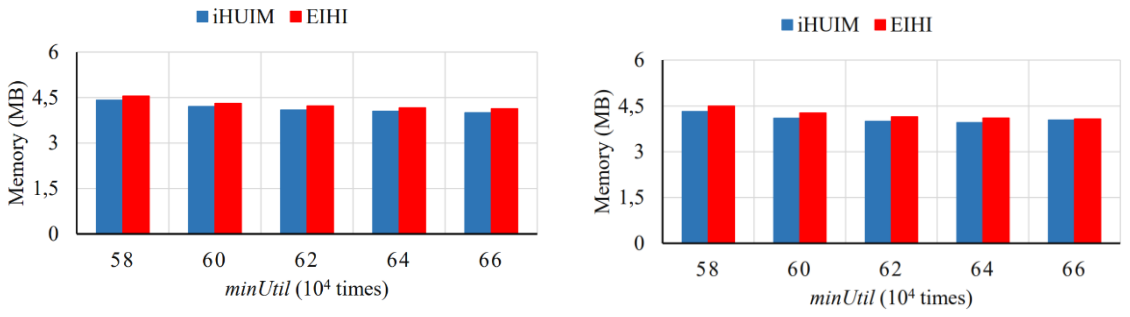


e-Mushroom-addRatio=20% f-Mushroom-addRatio=25%

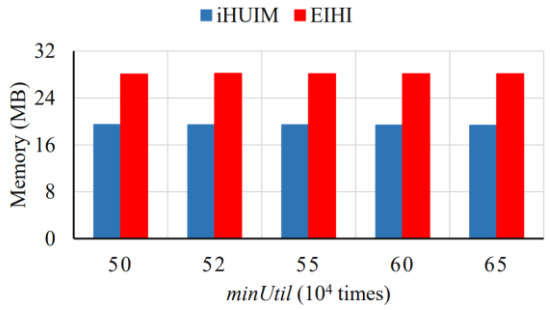
Figure 5: Comparison of execution time

In this section, we assess the runtime of the algorithms under varied minUtil or addRatio. The runtime of our algorithm to process each of the three datasets is demonstrated in Figure 5. Figure 5a and 5b shows clearly that iHUIM is quite faster than EIHI in each test case. When we run both algorithms with the Mushroom dataset and the Fruithut dataset, our algorithm is just slightly better than EIHI. With Mushroom, iHUIM reduces runtime in the range of 5% to 14% according to different parameters (Figure 5e, 5f), and with Fruithut, the average runtime our algorithm uses less than the EIHI is 10%. But with the Chess dataset, our algorithm is clearly faster than EIHI, the runtime when *minUtil=58000* is just 5,46 seconds when EIHI peaks at 22,31 seconds. As the experiment shows, the iHUIM algorithm can reduce runtime for incremental HUI mining.

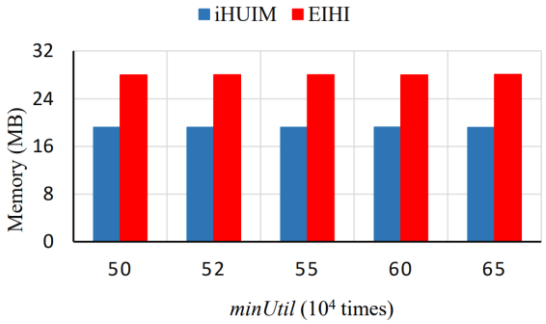
3.3.3. Memory usage



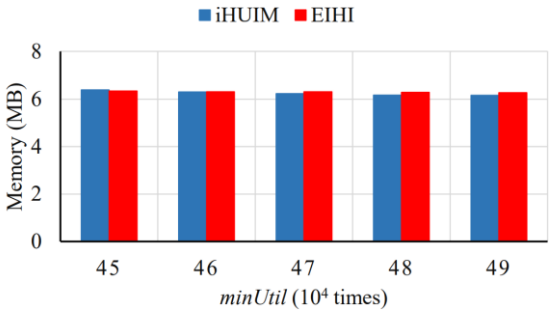
a-Chess-addRatio=20% b-Chess-addRatio=25%



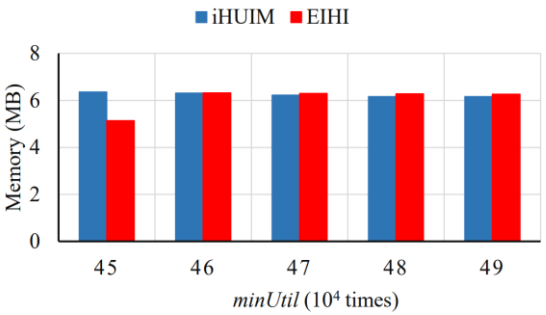
c-Fruithut-addRatio=20%



d-Fruithut-addRatio=25%



e-Mushroom-addRatio=20%



f-Mushroom-addRatio=25%

Figure 6: Comparison of memory usage

Memory consumption is the last evaluation criterion. Figure 6 reports the memory usage of iHUIM and EIHI when processing each of the three datasets. As the chart shows, our algorithm uses less memory than EIHI in most cases. The difference is not too much in Chess and Mushroom datasets but iHUIM significantly reduces memory consumption in the Fruithut dataset. The reason may come from MCUL, which can work effectively in a database that has a sparse density. In cases when MCUL achieves compression, memory usage can be lower. For example, with the Fruithut dataset, in case addRatio=25%, iHUIM reduces to 30% memory consumption, compared with EIHI. However, the use of MCUL for dense databases needs to be researched and modified in the future.

4. CONCLUSION AND FUTURE WORK

This thesis introduced the iHUIM algorithm that extends the EIHI algorithm. Unlike EIHI, our algorithm modifies Compact Utility List structure to store information of each item and scans the database once to build MCUL. Besides, our algorithm also has strategies to prune items that can't be HUIs before and during the mining process. Experimental outcomes demonstrate the efficacy of the pruning strategies. when the number of candidates generated by iHUIM is much smaller than that of EIHI. iHUIM also shows efficiency in terms of time but has limitations for sparse databases.

In the future, we will study and optimize the performance of iHUIM algorithm. Data structure is an aspect to optimize. We will continue to modify CUList or consider other efficient data structures to gain the efficiency of execution time and memory usage.

References

[1] Agrawal Rakesh and Srikant Ramakrishnan, "Fast Algorithms for Mining Association Rules in Large Databases," in *Proceedings of the 20th International Conference on Very Large Data Bases*, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., 1994, pp. 487-499.

[2] Liu Ying, Liao Wei-keng, Choudhary Alok, , "A Two-Phase Algorithm for Fast Discovery of High Utility Itemsets," in *Advances in Knowledge Discovery and Data Mining*, Berlin, Heidelberg, Springer Berlin Heidelberg, 2005, pp. 689-695.

[3] Vincent S. Tseng, Cheng-Wei Wu, Bai-En Shie, and Philip S. Yu., "UP-Growth: an efficient algorithm for high utility itemset mining," in *the 16th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '10)*, New York, NY, USA, 2010.

[4] Vincent S. Tseng; Bai-En Shie; Cheng-Wei Wu; Philip S. Yu, "Efficient Algorithms for Mining High Utility Itemsets from Transactional Databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 8, pp. 1772-1786, 2013.

[5] Mengchi Liu, Junfeng Qu, "Mining high utility itemsets without candidate generation," in *Proceedings of the 21st ACM international conference on Information and knowledge management*, Maui, Hawaii, USA, 2012.

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

[7] Souleymane Zida, Philippe Fournier-Viger, Jerry Chun-Wei Lin, Cheng-Wei Wu, and Vincent 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, 2017.

[8] S. Krishnamoorthy, "HMiner: Efficiently mining high utility itemsets," *Expert Systems with Applications*, vol. 90, pp. 168-183, 2017.

[9] Chun-Wei Lin, Guo-Cheng Lan, Tzung-Pei Hong, "An incremental mining algorithm for high utility itemsets," *Expert Systems with Applications*, vol. 39, no. 8, pp. 7173-7180, 2012.

[10] Chun-Wei Lin, Tzung-Pei Hong, Guo-Cheng Lan, Jia-Wei Wong, Wen-Yang Lin, "Incrementally mining high utility patterns based on pre-large concept," *Applied Intelligence*, vol. 40, no. 2, pp. 343-357, 2014.

[11] Tzung-Pei Hong, Ching-Yao Wang, Yu-Hui Tao, "A new incremental data mining algorithm using pre-large itemset," *Intelligent Data Analysis*, vol. 5, no. 2, pp. 111-129, 2001.

[12] Wensheng Gan, Jerry Chun-Wei Lin, Philippe Fournier-Viger, Han-Chieh Chao, Tzung-Pei Hong, Hamido Fujita, "A survey of incremental high-utility itemset mining," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 2, p. e1242, 2018.

[13] Jerry Chun-Wei Lin, Wensheng Gan, Tzung-Pei Hong, "A fast updated algorithm to maintain the discovered high utility itemsets for transaction modification," *Advanced Engineering Informatics*, vol. 29, no. 3, pp. 562-574, 2015.

[14] Chun-Wei Lin, Tzung-Pei Hong, Guo-Cheng Lan, Jia-Wei Wong, Wen-Yang Lin, "Efficient updating of discovered high-utility itemsets for transaction deletion in dynamic databases," *Advanced Engineering Informatics*, vol. 29, no. 1, pp. 16-27, 2015.

[15] Judae Lee, Unil Yun, Gangin Lee, Eunchul Yoon, "Efficient incremental high utility pattern mining based on pre-large concept," *Engineering Applications of Artificial Intelligence*, vol. 72, pp. 111-123, 2018.

[16] Jerry Chun-Wei Lin, Wensheng Gan, Tzung-Pei Hong, Binbin Zhang, "Incrementally Updating High-Utility Itemsets with Transaction Insertion," in *Advanced Data Mining and Applications*, Springer International Publishing, 2014, pp. 44-56.

[17] "A Survey of Key Technologies for High Utility Patterns Mining".

[18] Jerry Chun-Wei Lin, Wensheng Gan, Tzung-Pei Hong, "A fast maintenance algorithm of the discovered high-utility itemsets with transaction deletion," *Intelligent Data Analysis*, vol. 20, pp. 891-913, 2016.

[19] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Ted Gueniche, Prashant Barhate, "Efficient Incremental High Utility Itemset Mining," in *Proceedings of the ASE*

BigData & SocialInformatics 2015, Association for Computing Machinery, 2015, p. 6.

[20] Unil Yun, Heungmo Ryang, Gangin Lee,Hamido Fujita, "An efficient algorithm for mining high utility patterns from incremental databases with one database scan," *Knowledge-Based Systems*, vol. 124, pp. 188-206, 2017.

[21] P. Fournier-Viger, "SPMF:An Open-Source Data Mining Library," 25 12 2022. [Online]. Available: https://www.philippe-fournier-viger.com/spmf/index.php.