An Efficient Approach for Mining Closed High Utility Patterns in the Incremental Database

Ye-In Chang
Dept. of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan, Republic of China
changyi@mail.cse.nsysu.edu.tw

Po-Chun Chuang
Dept. of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan, Republic of China
zhungboqun@gmail.com

Chun-Yu Wang
Dept. of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan, Republic of China
cliff860429@gmail.com

Yu-Hao Liao
Dept. of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan, Republic of China
karta88821@gmail.com

***Abstract*—The utility of an itemset is the product of the profit and the frequency of the itemset in the database. If it is larger than the given threshold, it is defined as the high utility pattern. If there is no superset which has the same frequency as the frequency of its subset, the itemset can be considered as the closed itemset. If the itemset satisfies both the high utility and the closed property, the itemset is defined as the closed high utility itemset/pattern (CHUI). Among those algorithms based on the utility-list structure, the HMiner- Closed algorithm is the most efficient algorithm. However, if the value of the threshold decreases, the number of candidate CHUIs increases and decreases the efficiency of the algorithm. Therefore, in this paper, we propose an efficient algorithm based on a closed-set lattice structure. Our algorithm first finds the closed itemsets and then finds CHUIs in those itemsets. By checking the set relationship between the transactions, we first insert transactions into the closed-set lattice structure. Our algorithm also considers data insertion/deletion in the incremental database. From our simulation results, we show that our proposed approach is more efficient than the HMiner-Closed algorithm.**

***Keywords*—Closed High Utility Patterns, Closed-Set Lattice, Data mining, Incremental Database, Lattice**

I. Introduction

Frequent Itemset mining (FIM) discovers frequent itemsets based on the frequency of the itemset in the database. The importance of each item in the database is equal in association rule mining. However, this approach may have problems, when mining in the market database or other databases which concern the profit and weight of each item. Because the profits of the items are different in the real world. Moreover, the quantity of an item must be considered. Therefore, *High Utility Itemset Mining* (HUIM) is proposed. HUIM is used in many areas nowadays, such as retail marketing organization, machine learning, stock market analysis [1], *etc*. The approaches in HUIM includes static high utility itemset mining [2–4], dynamic high utility itemset mining [5–8] and closed itemset mining [9–11].

TABLE I: An example database D_1

Tid	$Items$	$Utility$
T_1	$(a,1),(b,4)$	$5,4$
T_2	$(a,1),(b,2),(c,1)$	$5,2,2$
T_3	$(a,1),(c,3)$	$5,6$
T_4	$(a,2),(b,1),(d,2)$	$10,1,6$
T_5	$(a,1),(b,2),(d,1)$	$5,2,3$

If the utility value of an itemset is larger than the threshold, the itemset is determined as a high utility pattern. Table I shows an example database D_1. The $(a,1)$ in transaction T_1 means the quantity of the item a is 1. HUIM is based on two kinds of consideration, the quantities and the profits. The profits of items a, b, c, d are 5, 1, 2, 3, respectively. The utility of an itemset X in transaction T_i is defined as $u(X,T_i)$, which is the product of the quantity of the itemset and the profit of the itemset. For example, for itemset $\{c\}$ in transaction T_3, we have the $u(\{c\},T_3) = 2*3$.

The total utility of an itemset X is defined as $totalu(X)$, which is the sum of every utility of the itemset in each transaction. For example, the itemset $\{c\}$ appears in transaction T_2 and transaction T_3. We have $totalu(\{c\}) = 2+6 = 8$. The transaction utility of transaction T_i is denoted as $TU(T_i)$, which is the sum of all utilities of items in the transaction. For example, we have $TU(T_4) = 10+1+6 = 17$.

The minimum utility threshold $miniutil$ is a value given by the user. If the value of $totalu(X)$ is not less than the $miniutil$, the itemset X is considered as a High Utility Itemset (HUI). Liu *et al.* propose an upper bound called *Transaction Weight Utility* (TWU) [12]. The TWU value of itemset X is the sum of all TU values for those transactions which contains itemset X. For example, the itemset $\{ac\}$ exists in transactions T_2 and T_3. We have $TWU(\{ac\}) = 9+11 = 20$.

The *downward-closure* property, which is known as the

Apriori property, is proposed in the Apriori algorithm [13,14]. If itemset X is infrequent, then all of its supersets, itemset Y, must be infrequent. However, The *downward-closure* property does not hold in HUIM.

Based on the transactions in database D_1, the final result of HUIs are $\{a\}$, $\{ab\}$, $\{ac\}$, $\{ad\}$ and $\{abd\}$. If we have $miniutil = 18$, then $totalu(b) = 9$ is not a HUI. However, for the superset of itemset $\{b\}$, itemset $\{ab\}$ is a HUI, because the value of $totalu(ab) = 34$. Therefore, it is possible that itemset X is not a HUI, but its superset Y, $X \subseteq Y$, is a HUI.

Many mining algorithms focus on static datasets, but datasets may be changed in the real life. Therefore, *Dynamic High Utility Itemset Mining* (DHUIM) is proposed. There are three situations, when the new transaction is stored in the database D_1. First, If itemset X is a HUI in database D_1, itemset X will remain as a HUI. Second, itemset X is not a HUI in database D_1, but itemset X becomes HUI in database $D_1 \cup D_2$. Third, the itemset X remains a non-HUI in database $D_1 \cup D_2$. For all of the above three cases, it seems that every itemset in the database must be considered again, when new transactions are stored. Therefore, the DHUIM approach [6] has been proposed to process new stored data without additional database scan and inserts the result into the previously processed result.

The limitation of HUIM is that it often produces too many candidates. To decrease the number of candidates and keep the representation lossless, *Closed High Utility Itemset Mining* (CUIM) is proposed [15]. The CUIM extends the concept of HUIM, if an itemset X does not have a superset, which has the same support count, the itemset X is a closed itemset. If the itemset X is a high utility itemset and a closed itemset. The itemset X is defined as a Closed High Utility Itemset (CHUI).

HUIM is used in many areas nowadays, such as retail marketing organization, machine learning, stock market analysis [1], *etc*. High Utility Itemset Mining (HUIM) is one of the approaches in association rule mining. For the concern of mining in the static database, Liu *et al.* propose the Two-Phased algorithm [3]. To improve the Two-Phased algorithm, Liu and Qu propose the HUI-Miner algorithm [2]. For those transactions which have similar items but different quantities, EFIM [4] is proposed. The local utility is a tighter upper-bound than Transaction Weight Utility (TWU) [3]. For the concern of the dynamic database in high utility itemset mining, there are two types of ways to mine the candidates in the dynamic database. The first type is the incremental database. The whole database is considered, when new transactions are stored. MEFIM and iMEFIM algorithm [8] have been proposed to achieve this goal. iMEFIM is an improved version of MEFIM. Lee *et al.* propose PIHUP (*Pre-large Incremental High Utility pattern*) algorithm [5]. Yun *et al.* propose IIHUM (*List based Incremental High Utility pattern Mining*) to deal with incremental mining [1]. The second type of dynamic database is under the assumption of the window model. The algorithm considers the arrival time of the data. ILDHUP algorithm [7] has been proposed to handle this problem. The closed high utility mining combines the requirements of closed itemset in FIM and the traditional HUIM. The EFIM-Closed algorithm [10] has been proposed for mining CHUIs. As a modified version of the EFIM algorithm, the EFIM-Closed algorithm has several improvements. The HMiner-Closed algorithm [11] aims for dealing with both sparse and dense datasets. The HMiner-Closed algorithm uses *Backward checking* and *Forward checking* to reduce the search space and runtime. The IncCHUI [9] algorithm is proposed to not only CHUI mining, but also the incremental database. The main problem in incremental closed high utility mining is the reconstruction process. Since the TWU order might be changed, when a new transaction is stored. Therefore, the mining order must be sorted again to maintain the *downward closure property*. However, the *remaining utility* (rutil) has to be recalculated every time, when a new transaction is stored. The cost of the reconstruction can be improved.

The main purpose of mining High Utility Patterns (HUP) is to consider both the frequency and the profit. However, the number of high utility patterns might be large, if the threshold is set too low. Closed high utility mining can decrease the number of resulting itemsets without information loss. If pattern X satisfies both closed property and high utility property, the pattern X is considered as a closed high utility pattern.

Among those algorithms based on the utility-list structure for mining closed high utility patterns, [11], which needs only one scan of the database, the HMiner-Closed algorithm, is the most efficient algorithm. However, the CHUI mining algorithm first focuses on mining HUIs and then determines the closed property among the itemsets. The number of high utility itemsets is affected by the minimum threshold. If the threshold is set too low, there are lots of utility lists which have to be constructed. The efficiency of the algorithm is affected by the number of high utility patterns. Because the utility lists have to be reconstructed again, when new transactions are inserted.

Therefore, to reduce the cost for evaluating a large number of high utility patterns, in this paper, we propose a new algorithm to efficiently discover CHUIs with a closed-set lattice structure. Instead of finding HUI first then finding the closed frequent itemsets, we first find all closed frequent itemsets and then find CHUI among the closed frequent itemsets. By mining closed frequent itemsets first, we can prune candidate high utility itemsets, which do not satisfy the closed property. In the algorithm, we use a closed-set lattice to store transactions from the dataset. Our algorithm first finds all closed frequent itemsets and then find CHUI among the closed frequent itemsets. By mining closed frequent itemsets first, our algorithm can prune candidate high utility itemsets, which do not satisfy the closed property. In the algorithm, we use a closed-set lattice to store transactions from the dataset. Our algorithm first checks the relationship between sets between the incoming transaction and the old transactions. The key to find closed patterns is by comparing support counts between patterns. The relationship between sets can help to calculate the support counts of patterns. There are several advantages to consider in the relationship between itemsets. First, every node in the closed-set lattice is certainly a closed itemset. Those high utility itemsets, which our algorithm has discovered in the lattice structure are CHUIs. Different from traditional CHUI mining algorithms, our algorithm first finds all closed itemsets then finds high utility itemsets among those closed frequent itemsets. The number of candidate itemsets is fixed and do not affect by the transaction value. Second, the lattice structure is the only structure, which we need to find CHUIs. Therefore, no extra space is needed. Third, when new transactions are stored, our algorithm does not need to rescan the old transactions. Our algorithm can mine CHUIs by updating the closed-set lattice. Our algorithm uses a binary list called $bit\text{-}represent$ to speed up the comparing process between itemsets. Fourth, no CHUI is lost in the algorithm. From the simulation result, we show that our proposed algorithm has better performance than HMiner-Closed algorithm in dense databases and sparse databases.

II. A Survey of the HMiner-Closed Algorithm

Nguyen *et al.* have introduced the HMiner-Closed algorithm [11] in 2019. They have proposed a modified version of *Compact Utility List* (CUL) called *Modified Compact Utility List* (MCUL) as shown in Figure 1. The information inside each node can help the extended process of HMiner-Closed algorithm constructs utility lists without rescanning the database. A MCUL of an itemset with more than one item is called a k-MCUL, and it is built from (k-1)-MCULs without rescanning the database. The mining process of HMiner-Closed algorithm is as follows. First, the algorithm scans the database once and constructs 1-MCULs. Items with TWU values smaller than the threshold are pruned, since the extension of those items can not be closed high utility itemsets. The algorithm then sorts the 1-MCULs by the accending TWU order. Next, the algorithm constructs k-MCUL and finds closed high utility itemsets. If $U(X)$ of itemsets X is greater than the threshold, the itemset will be added into CHUI-list. When the algorithm completes, the itemsets remaining in the CHUI-list are the CHUIs. There are some downsides in the HMiner-Closed algorithm. If the threshold value is small, there are many itemsets will be considered as high utility itemset. The HMiner-Closed algorithm has to construct many utility lists and consider many itemsets to find closed high utility itemsets. Also, the algorithm does not consider mining in dynamic databases. If there are new transactions inserted into the database, the TWU order of items may be changed. Because TWU values of items are changed. Therefore, the whole database needs to be rescanned, the TWU order of itemset needs to be sorted again, and the MCULs needs to be reconstructed.

III. The Closed-Set Lattice Approach

In this section, we present a closed-set lattice algorithm to identify the closed high utility patterns efficiently by using the modified subset-lattice [16,17] as the foundation of our data structure.

$NU/NPU/NRU$		$CU/CPU/CRU$		$CSUP/NSUP$	
T_j	$NU(X,T_j)$	$NPU(X,T_j)$	$NRU(X,T_j)$	$W(X,T_j)$	$PPOS(X,T_j)$

Fig. 1: The data structure MCUL applied in the HMiner-Closed Algorithm [11]

abcd	
QuantitySeq	*utility*
[1,2,6,1,0]	22
bit-present	*TransactionId*
[11110]	$\{T_5\}$

Fig. 2: The example node

The subset-lattice proposed by Chen [16] only stores the frequency of each item in the database. Each node of the subset lattice can store at most 3 batches and a batch represents two transactions. The utility of an itemset is calculated by multiplying the frequency and the profit of the itemset. The subset-lattice structure is proposed by Peng [17] for mining frequent itemsets. Each node records the itemsets by a bit-represent list, and the node also records the transactions which contain the itemsets

The difference between our closed-set lattice and their subset-lattices [16,17] is that each node in our structure records the utility value, bit-represent list of itemset, the frequency of every item of the itemset, and the list of transactions. Our algorithm does not separate transactions into batches. On the contrary, the information in each node is calculated from all transactions in the dataset. The example node is shown in Figure 2.

A. Database

In this subsection, we use an example database D_3 shown in Table II to illustrate our algorithm. In the process of mining CHUIs in the static and dynamic database, our algorithm assigns the profit and the quantity to each item. Our algorithm uses a modified subset-lattice structure, which is called closed-set lattice, to record the information of each transaction in the database. Note that our algorithm only scans the original database once, as compared to two times in Apriori-based algorithms, such as Two-Phase algorithm [3].

The non-binary database with n transactions is represented as $D = \{T_1, T_2, T_3, ..., T_k\}$, which includes a set of m distinct items, $I = \{i_1, i_2, i_3, ..., i_m\}$. Each transaction T_k $(1 \leq k \leq n)$ has a unique identifier, Tid, and contains multiple items belonging to set I $(T_k \subseteq I)$.

In real-world applications, different products have their own characteristics, such as the profit and the quantity. The closed high utility itemset mining allows users to give each item an independent value to satisfy the needs. For example, the profits of items a, b, c, d and e are assigned with 5, 1, 2, 3 and 7,

TABLE II: The example database D_3

Tid	$Items$
T_1	$(a, 1), (b, 4)$
T_2	$(c, 1), (d, 5)$
T_3	$(a, 2), (b, 3)$
T_4	$(c, 3)$
T_5	$(a, 1), (b, 2), (c, 6), (d, 1)$
T_6	$(b, 2), (c, 1)$

respectively. We define the profit of an item as $P(X)$. Take item {c} as example, and we have $P(c) = 2$. The quantities of itemset X in transaction T_i is represented as $Q(X, T_i)$.

The utility of itemset X in transaction T_i is denoted as $u(X, T_i)$. For example, for itemset $\{d\}$ in transaction T_2, its utility is calculated as $u(d, T_2) = Q(d, T_2)*P(d) = 5*3 = 15$. Similarly, for itemset $\{cd\}$ in transaction T_5, the $u(cd, T_5)$ is calculated as $u(cd, T_5) = Q(c, T_5)*P(c)+Q(d, T_5)*P(d) = 2*6+3*1 = 15$.

The Transaction Utility (TU) is the summation of utilities in a transaction. Take transaction T_1 as an example, we have $TU(T_1) = u(a, T_1) + u(b, T_1) = 5*1+1*4 = 9$. The $TWU(X)$ means the amount of the total utility related to the transactions which contain pattern X. The TWU value is calculated by summarizing all of the $TU(T_i)$, which contains pattern X. For example, the TWU of the itemset $\{d\}$ in database D_3 is computed as follows. Since itemset $\{d\}$ exists in transactions T_2, T_5, we have $TWU(d) = TU(T_2) + TU(T_5) = 38$. The TWU values of items a, b, c, d and e are 44, 48, 49, 39 and 0, respectively.

B. The Observations of the Relationships Between Itemsets

In this subsection, we present an observation of the set-relationships between itemsets. The set-relationships between itemsets can be utilized to identify closed itemsets. Our algorithm first finds all closed itemsets and we then find high utility itemsets between them. The reason for such an approach is that the number of the closed itemsets in the dataset is fixed. The number of closed itemsets is not affected by the threshold value. For the traditional CHUI mining algorithm, the number of high utility itemsets is affected by the threshold value. It might generate lots of high utility itemsets which do not satisfy the closure property. By finding all closed itemsets, our algorithm sets an upper bound for the number of CHUIs. The number of the CHUIs generated by our algorithm is at most equal to the number of closed itemsets. If our algorithm can not find any superset Y which satisfies the condition $sup(Y) = sup(X)$, the itemset X is considered as a closed high utility itemset. Let's consider a dataset with only one transaction $[a, b, c, d]$ with utility $[1, 2, 3, 4]$. The minimum threshold is set as 5. The utility of the itemset $\{abc\}$ is 6, which is greater than the threshold. However, itemset $\{abc\}$ has a superset $\{abcd\}$, which has the same support 1 as the itemset $\{abc\}$. Thus, itemset $\{abc\}$ is not a CHUI. Our algorithm classifies the set-relationships into five properties, which contains (a) equivalent, (b) disjoin, (c) belong, (d) contain and (e) partial overlap.

C. The Closed-set Lattice

The lattice structure is a data structure, which records the relationships between the sets. However, the traditional lattice lists all possible relationships between supersets and subsets. The edge between two itemsets represents that there is an intersection between two itemsets. But, first, the closed-set lattice structure does not contain all possible combinations of itemsets. Because the itemsets which our algorithm stores in the closed-set lattice are all closed itemset. Itemsets $\{abc\}$, $\{acd\}$ are the subsets of itemset $\{abcd\}$, which is in transaction T_5. They are not stored in the closed-set lattice because they are not closed itemsets. Because itemset $\{abcd\}$ has the same frequency as two itemsets, $\{abc\}$ and $\{acd\}$, and itemset $\{abcd\}$ is their superset. The second difference is that the level between superset and subset in the closed-set lattice can be more than one. In the traditional lattice, the itemset with size k has the relationship with supersets which have size (k+1) and subsets which have size (k-1). The level between two itemsets can be more than 1 in the closed-set lattice structure.

The closed-set lattice structure is consisted of many lattice nodes and edges between the nodes. An edge represents the relationships between the superset and the subset. A lattice node records the related data about the itemset and its T_{id}. Each node in the closed-set lattice contains five variables, which are $itemset$, $QuantitySeq$, $bit\text{-}present$, $utility$, and the $TransactionId$ list. The $itemset$ represents the set of items. The $QuantitySeq$ is a list which records the quantity of each item in each transaction. The $bit\text{-}present$ is a bit-vector, which stores the information of each transaction. The size of the $bit\text{-}present$ and $QuantitySeq$ is equal to the number of the items in the database. Our algorithm can easily find relationships between two transactions with $bit\text{-}present$. When our algorithm has the information of the quantity of each item, our algorithm can calculate the real contribution value to determine whether it is a closed high utility pattern. For example, transaction T_5= {(a,1), (b,2), (c,6), (d,1)} is denoted as $QuantitySeq[1, 2, 6, 1, 0]$. Because there are five distinct items, {a, b, c, d, e} in the example database and item e does not appear in the transaction T_5, the Quantity of item e in the transaction T_5 is recorded as 0. The $Utility$ of the itemset is also recorded in the node. Our algorithm records the transaction id of an itemset in the $TransactionId$ list. If the $TransactionId$ list of an itemset is empty, it means that the itemset does not appear in any transaction in the database. The situation may happen when the old transaction is deleted from the database. Therefore, the itemset has to be removed from the lattice structure.

D. The Pruning Strategy

Here, we introduce a pruning strategy [3]. The first property is called "overestimation". The key point is that for any patten X, the value of $TWU(X)$ is greater than the value of $totalU(X)$. The second property is called "antomototonicity".

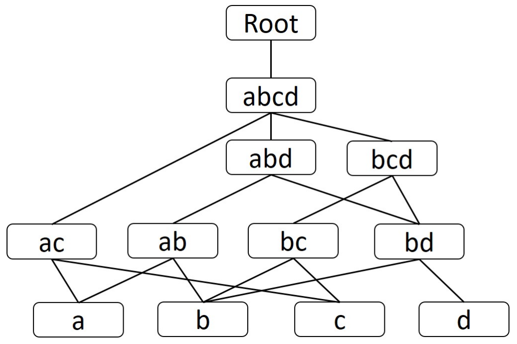


Fig. 3: The closed-set lattice structure

Given two itemsets X and Y, and X is the subset of Y, the value of $TWU(X)$ is greater than or equal to the value of $TWU(Y)$. The third property is called "pruning". Given an itemset X, if the value of $TWU(X)$ is smaller than the given en threshold $minutil$, then all of its superstars and itemset X itself are low utility itemsets. If an itemset X has $TWU(X) < minutil$, then itemset X and its supersets are low-utility itemsets. The TWU pruning strategy can prune items, which has smaller TWU value than the threshold in the database scanning stage. The strategy can decrease the search space of our algorithm.

E. The Mining Process

Here, we describe the mining process. There are two parts in the mining process, data insertion and data deletion. The first part is the data insertion. When a new transaction is inserted, the process considers its relationship to the nodes in the closed-set lattice. After the transactions are inserted, the process traverses the structure to find high utility patterns. Since the itemsets in the closed-set lattice structure are closed itemsets, the high utility patterns which we have found in the structure are closed high utility patterns. The second part of the process is data deletion. Our algorithm provides the flexibility to remove some transactions without rescanning the database. Our algorithm updates the nodes in the closed-set lattice, which are affected by the removal of the transactions. After the transactions have been removed, we update the CHUI list and output the updated result.

1) Data Insertion: In the process of data insertion, we introduce the steps to insert transactions into closed set lattice structure. The Closed-Set lattice algorithm is described as follows. First, our algorithm has to decide whether the TWU pruning technique is used or not. If the database is static, no transaction will be inserted or removed after this mining process. TWU pruning technique can be used to decrease the search space. If the database is dynamic, a new transaction will be inserted or removed after this mining process. The TWU pruning technique can not be used in this situation. Because when new transactions are inserted, the itemsets which have low TWU value may become larger than the threshold. Also, when transactions are removed, the itemsets which have high TWU value may become smaller than the threshold. When the above situation occurs, the closed-set lattice has to consider new items or ignore existing items. The database has to be rescanned to reconstruct the closed-set lattice. The reconstruction process is time-consuming. Therefore, our algorithm does not apply the TWU pruning technique, when mining in the dynamic database.

Next, our algorithm inserts each transaction into the closed-set lattice structure one by one. When a new transaction is inserted into the lattice structure, the algorithm starts finding set-relationships between transactions. In the checking step, our algorithm uses logical operators, AND, to make the subsumption checking more efficient. There are 5 different cases. Case 1 is the $equivalent$ relationship. The itemset $NewT$ has the same itemset with itemset $OldT$. Case 2 is the $disjoin$ relationship. There are no identical items between itemset $NewT$ and itemset $OldT$. Case 3 is the $belong$ relationship. The itemset $NewT$ belongs to itemset $OldT$. Case 4 is the $contain$ relationship. The itemset $NewT$ contains the itemset $OldT$. Case 5 is the $partial\ overlap$ relationship. Two itemsets have some items in common.

The insertion process is performed as follows. First, the process checks whether the transaction is the first transaction or not. If it is the first transaction. The process inserts the itemset as the new transaction, which is defined as itemset $NewT$ into the closed-set lattice structure. If the transaction is not the first transaction, the process will judge the relationship between the itemset $NewT$ with the already existing itemsets, which is defined as itemset $OldT$. The relationship between itemset $NewT$ and itemset $OldT$ can be classified into 5 cases. The first case is the $equivalent$ relationship. The process does not generate the new node. The process updates the information of the $OldT$ and its descendants. Since the itemset $NewT$ is the same as itemset $OldT$, the itemset $NewT$ does not be inserted into the closed-set lattice. The second case is the $disjoin$ relationship. The itemset $NewT$ is inserted into the closed-set lattice structure as a new node. The third case is the $belong$ relationship. The itemset $NewT$ is the subset of the itemset $OldT$. Therefore, the itemset $NewT$ is inserted into the closed-set lattice structure and becomes $OldT's$ subset. The fourth case is the $contain$ relationship. The itemset $NewT$ is the superset of the itemset $OldT$. Therefore, the itemset $NewT$ is inserted into the closed-set lattice structure and becomes $OldT's$ superset. The last case is the $partial\ overlap$ relationship. The itemset $NewT$ and itemset $OldT$ have some items in common. That is, they have an intersection itemset. The overlapping itemset becomes a new node, $childT$. If the new node $childT$ has already existed in the structure, we update the itemset and link it to itemset $NewT$. Itemset $NewT$ is then inserted into the structure. On the contrary, if the new node $childT$ does not exist in the data structure, itemset $childT$ will be inserted into the closed-set lattice structure. After checking all paths in the closed-set lattice and considered all relationships with the itemsets, the insertion process will terminate.

An example of data insertion is given as follows. Figure 4

Tid	*itemset*	*bit-present*	*QuantitySeq*	*Case*	*Relationship*
T_1	ab	[11000]	$[1, 4, 0, 0, 0]$		
T_2	cd	[00110]	$[0, 0, 1, 5, 0]$	2	$disjoint$
T_3	ab	[11000]	$[1, 3, 0, 0, 0]$	1	$equivalent$
T_4	c	[00100]	$[0, 0, 3, 0, 0]$	3	$belong$
T_5	$abcd$	[11110]	$[1, 2, 6, 1, 0]$	4	$contain$
T_6	bc	[01100]	$[0, 2, 1, 0, 0]$	5	$partial\ overlap$

Fig. 4: The example data based on database D_3.

shows the example dataset based on Database D_3 in Table II. The itemset $NewT$ represents the itemset in the new transaction. The itemset $OldT$ represents the set of old itemset in the closed-set lattice structure. Every relationship between two itemsets can be found in the example dataset. For itemset $\{ab\}$ in Transaction T_1, it is the first transaction in the database. The itemset $\{ab\}$ is inserted into closed-set lattice as itemset $NewT$. The closed-set lattice is shown in Figure 5-(a). When the Transaction T_2 is inserted into the lattice, our algorithm will check the relationship between the previous transactions and the transaction T_2. The *bit-present* of T_1 is [11000] and the *bit-present* of T_2 is [00110]. Our algorithm can easily find the relationship between T_1 and T_2 is $disjoint$ by having a AND operation between [11000] and [00110]. Because $T_1 \cap T_2 = \emptyset$, transaction T_2 will process Case 2, *i.e.* $disjoint$. There is no additional node which is generated. Since all paths are considered, the itemset $\{cd\}$ is inserted into closed-set lattice as itemset $NewT$ and the next transaction is inserted. The closed-set lattice after inserting transaction T_2 is shown in Figure 5-(b). Transaction T_3 has the same itemset as one of the itemset $OldT$, which is $\{ab\}$. The relationship is found by having a AND operation between [11000] and [11000]. The algorithm will process Case 1, *i.e.* $equivalent$. No new node is generated and the information of itemset $\{ab\}$ and its subsets are updated. The QuantitySeq of the itemset $\{ab\}$ is changed from $[1, 4, 0, 0, 0]$ to $[3, 7, 0, 0, 0]$. The total utility of itemset $\{ab\}$ is also changed from 9 to 22. Since not all paths are considered, the algorithm then check the relationship between transaction T_3 and the other itemset in itemset $OldT$, which is itemset $\{cd\}$. The relationship between two itemsets is Case 2, *i.e.* $disjoint$. Since the itemset $\{ab\}$ has been inserted before, the algorithm does not insert the new node. Since all paths are considered, the algorithm waits for new transactions. The closed-set lattice after inserting transaction T_3 is shown in Figure 5-(c).

Transaction T_4 has itemset $\{c\}$. Itemset $\{c\}$ is the subset of itemset $\{cd\}$ in the lattice. The relationship can be found by having an AND operation between [00100] and [00110]. The algorithm will process Case 3, *i.e. belong*. The process first makes itemset $\{c\}$ becomes the child of itemset $\{cd\}$. Then the process updates information of itemset $\{cd\}$ and its subsets. Itemset $\{c\}$ is inserted in to the closed-set lattice structure. Transaction T_5 has itemset $\{abcd\}$. It is the superset of itemsets $\{cd\}$ and $\{ab\}$. The algorithm will process Case 4, *i.e. contain*. The itemset $OldT$ will become the child of itemset $NewT$. Transaction T_6 has itemset $\{bc\}$. The itemset $\{bc\}$ has partially intersected with itemsets $\{cd\}$ and $\{ab\}$. The relationship can be found by having an AND operation to each of [01100], [11000] and [00110]. The algorithm will process Case 5, *partial overlap.* Itemset $X = NewT \cup OldT$. There are two itemsets, which are itemset $\{c\}$ [00100] and $\{b\}$ [01000]. Since itemset $\{c\}$ has already existed, the algorithm link $NewT$ to it. Since itemset $\{b\}$ is not in the descendants of Itemset $\{ab\}$, the algorithm creates a new node $\{b\}$ and links it to the itemset $NewT$ and itemset $OldT$. After all of the nodes are updated and all paths are considered, the intersection process terminates. The result of the data insertion is shown in Figure 6.

2) Mining CHUIs: After all transactions are inserted into the lattice structure, our algorithm can assure that all lattice nodes are "closed" itemset [17]. Next, our algorithm executes a lattice structure traversal to determine whether each lattice node is a high utility item or not. Note that the utility of a pattern has already been recorded in the node of the closed-set lattice structure. Therefore, to decide whether a pattern is a closed high utility is a simple checking task. However, the pay is to traversal all of the nodes in the closed-set hierarchy. But such a pay is little as compared to computing the utility for each candidate pattern before considering the closed-set property as used in all other algorithms based on the utility list structure. If the utility of the itemset is larger than or equal to the threshold, which is given by user, the itemset is considered as a CHUI and be recorded in to a list, $ClosedItemset\ Table$. Every node in the closed-set lattice is compared to the threshold once. After all itemsets is considered, the process terminates and outputs the result.

3) Data Deletion: In the deletion step, we show the procedure to delete the transactions in any order from the lattice structure. When a transaction is deleted from the dataset, the closed-set lattice structure will be updated. There are two situations, which will cause an itemset needs to be removed from the lattice structure. The first situation is that the itemset has an empty $TransactionId$ list and the second situation is that the itemset has the same support count as that of its supersets. If the $TransactionId$ list of itemset X is empty, it means that the itemset X does not appear in any transaction in the database. Therefore, itemset X does not be considered as a closed itemset and needs to be removed from the lattice.

Our algorithm traverses the lattice structure, which is corresponding to the deleted transaction, and updates the nodes. If a node is updated, the subset of the updated node will be affected, too. The node could be created by more than two itemsets. For instance, itemset $\{b\}$ is created by the intersection itemsets $\{ab\}$ and $\{bcd\}$. Therefore, the support count of $\{b\}$ contains that of itemsets $\{ab\}$ and that of $\{bcd\}$. If we delete a transaction T_i, which contains itemset $\{ab\}$. The itemset $\{b\}$ also has to deleted T_i from its $TransactionId$ list. After the deletion, if the itemset $\{b\}$ has the same support count to that of itemset $\{bcd\}$, itemset $\{b\}$ does not be considered as a closed itemset anymore and needs to be removed from the lattice structure.

Here we have an example to illustrate the insertion and dele-

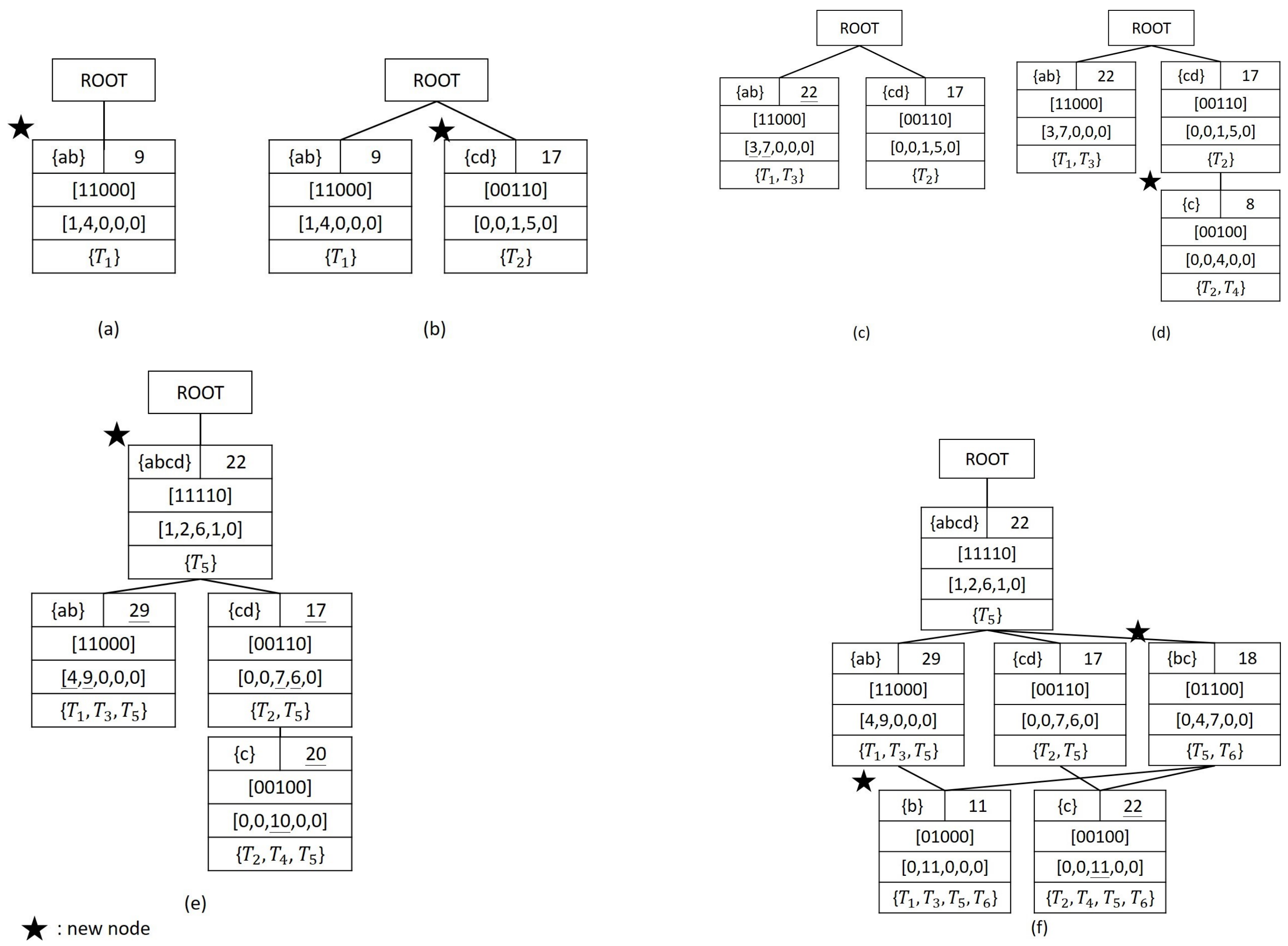
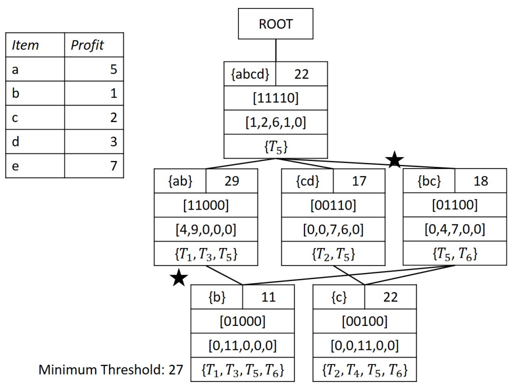


Fig. 5: The closed-set lattice structure after inserting transactions T_1, T_2, T_3, T_4, T_5 and T_6: (a) insertion of transaction T_1 $[a, b]$; (b) insertion of transaction T_2 $[c, d]$; (c) insertion of transaction T_3 $[a, b]$; (d) insertion of transaction T_4 $[c]$; (e) insertion of transaction T_5 $[a, b, c, d]$; (f) insertion of transaction T_6 $[b, c]$.



Item	*Profit*
a	5
b	1
c	2
d	3
e	7

Fig. 6: The Result of Insertion

tion process. Our algorithm first inserts a new transaction T_7 (which contains $(b, 2), (c, 4), (e, 1)$ as shown in Table III) into the closed-set lattice and then remove the oldest transaction T_1.

After the insertion of transaction T_7, our algorithm removes the oldest transaction T_1. Since we remove transaction T_1, the $QuantitySeq$ and the $totalU$ of itemset $\{ab\}$, $\{b\}$ need to be updated. The $QuantitySeq$ of itemset $\{ab\}$ becomes $[3, 5, 0, 0, 0]$ and its $totalU$ becomes $29 - 9 = 20$. The $QuantitySeq$ of itemset $\{b\}$ becomes $[0, 9, 0, 0, 0]$ and its $totalU$ becomes $13 - 4 = 9$. The $totalU$ of itemset $\{ab\}$ is no longer a CHUI, so it is removed from the $ClosedItemset\ Table$. After the Insertion process and the deletion process, the closed high utility itemsets in the $ClosedItemset\ Table$ are output as result. The result after insertion is shown in Figure 7. The CHUIs after Insertion of transaction T_7 and deletion of transaction T_1 are itemset $\{bc\}$ and $\{c\}$. Note that those two itemset are not CHUI in the previous mining result.

IV. Performance

In this section, we present the performance study of our algorithm and the HMiner-Closed algorithm [11] for mining closed high utility patterns. In this study, we use synthetic

TABLE III: The example data based on database D_4

Tid	$itemset$	$bit\text{-}present$	$QuantitySeq$
T_7	bce	[01101]	$[0, 2, 4, 0, 7]$

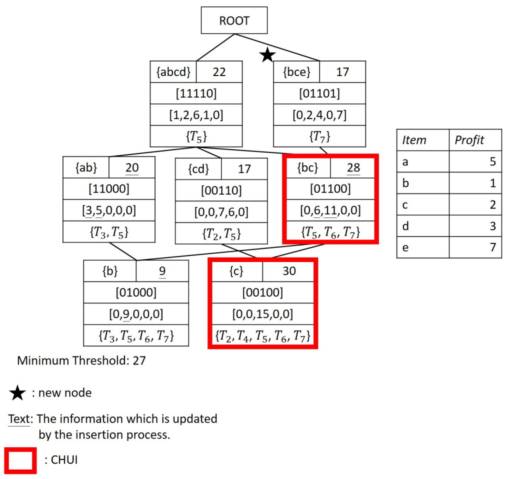


Fig. 7: The result after inserting transaction T_7 and removing transaction T_1

databases to simulate various conditions. We also use the real transaction dataset used in the previous studies of HUIM, including the dataset $foodmart$, which has been downloaded from the SPMF library [18].

A. The Performance Model

We compare the processing time of the synthetic database and the real database. The external utilities (profit) of the items are generated by the Gaussian distribution function, and the internal utilities (count) are generated randomly between 1 and 10.

For the preparing step, at first, all of the transactions are scanned to get the TWU value of each item. Only the HMiner-Closed algorithm needs to sort transactions according to TWU-ascending order. The HMiner-Closed algorithm inserts the sorted transactions to generate a single item utility list and our algorithm generates the lattice structure. The HMiner-Closed algorithm then generates many utility lists to discover CHUIs. Conversely, Our algorithm does not need to generate other data structures. After the above processes are finished, we will compare the processing time with each other.

The parameters, which are shown in Figure 8 are the variables of our simulation. The $MinUT$ represents the threshold values given by us. We give different threshold ranges for different databases. The NDI represents the number of distinct items in a database. The $MaxNDI$ represents the max number of distinct items in a single transaction. The synthetic database for simulation is showed in Table IV, where dataset 10000_15000_10 means the value of $TNum = 10000$, $NDI = 15000$ and $MaxNDI = 10$, respectively. Take dataset 10000_15000_10 for example. The dataset has 10000

$Parameters$	$Meaning$
$MaxNDI$	The Max Number of Distinct Items in a single transaction
NDI	The Number of Distinct Items in a database
$MinUT$	The Minimum Utility Threshold
$TNum$	The Total Number of transactions
$AvgLength$	The Average Length of transactions

Fig. 8: The parameters used in the experiments

TABLE IV: Dataset characteristics

	$Dataset$	$TNum$	NDI	$MaxNDI$
(a)	10000_15000_10	10000	15000	10
(b)	10000_12000_10	10000	12000	10
(c)	10000_10000_10	10000	10000	10
(d)	10000_6000_10	10000	6000	10

transactions, 15000 distinct Items and the max number of distinct items in the single transaction is 10. For dataset (a), it is a sparse dataset. Moreover, for datasets (a), (b), (c) and (d), they are only different in the value of NDI, $i.e.$, the number of distinct items in a database.

B. Experiments Results

In this section, we compare the performance of the HMiner-Closed algorithm [11] and our closed set lattice algorithm with synthetic and real databases. The transactions of each synthetic itemsets are set as 10000. For itemset (a) 10000_15000_10, the range of the $MinUT$ is set from 60 to 10. Since both HMiner-Closed and our algorithm have the same result of closed high utility itemsets, the number of CHUIs can judge the correctness of the algorithms.

1) Synthetic Dataset: A comparison is shown in Figure 9-(a) by using dataset (a) 10000_15000_10, $i.e.$, the sparse dataset, under different threshold values. Since the number of items is relatively large, the dataset is determined as a sparse dataset. For the HMiner-Closed algorithm, it has to consider many subsets to determine the CHUI. In the sparse datasets, itemsets have few partial overlap relationships with other itemsets, which causes the algorithm to construct more utility lists to consider all of them. Moreover, if the value of the $MinUT$ is small, then we will have large number of high utility itemsets. Therefore, HMiner-Closed has to consider more number of itemsets to find CHUIs. If the threshold is small, the pruning strategy of HMiner-Closed can not stop the mining process. Because the sum of the itemset utility and remaining utility can be easily greater than the threshold. In Figure 9-(a), the runtime gap between the HMiner-Closed algorithm and our algorithm becomes dramatically large. This is due to the fact that our algorithm first finds all closed itemsets. The amount of candidate itemsets is fixed. The change of the $MinUT$ has a smaller effect on our algorithm than the effect on the HMiner-Closed algorithm.

A comparison is shown in Figure 9-(b) by using dataset (b) 10000_12000_10 under different threshold values. This figure shows the similar result as Figure 9-(a). A comparison is shown Figure 9-(c) by using dataset (c) 10000_10000_10

TABLE V: Real Transaction Dataset characteristics

$Dataset$	$TNum$	NDI	$AvgLength$
$foodmart$	4141	1559	4.42

under different threshold values. This figure shows the similar result as Figure 9-(a). A comparison is shown in Figure 9 by using dataset (d) 10000_6000_10 under different threshold values. This figure shows the similar result as Figure 9-(a).

2) Real Transaction Dataset: For the real transaction dataset, we experiment the dataset, $foodmart$ "(http://www.philippe-fournier-viger.com/spmf/datasets/foodmart.txt)". The details of the dataset is shown in Table V. The dataset represents what the real life needs. Retailers want to find useful information from the datasets using CHUI mining.

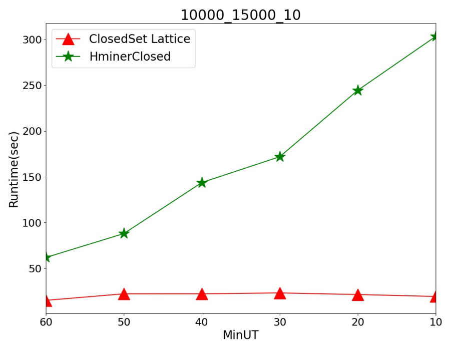
A comparison is shown in Figure 10 by using dataset $foodmart$ under different threshold values. For the HMiner-Closed algorithm, it has to consider many subsets to determine the CHUI. Moreover, if the value of the $MinUT$ is small, then we will have large number of high utility itemsets. Therefore, HMiner-Closed has to consider more number of itemsets to find CHUIs. If the threshold is small, the pruning strategy of HMiner-Closed can not stop the mining process. Because the sum of the itemset utility and remaining utility can be easily greater than the threshold. In Figure 10, the runtime gap between the HMiner-Closed algorithm and our algorithm becomes dramatically large. This is due to the fact that our algorithm first finds all closed itemsets. The amount of candidate itemsets is fixed. The change of the $MinUT$ has a smaller effect on our algorithm than the effect on the HMiner-Closed algorithm.

V. Conclusion

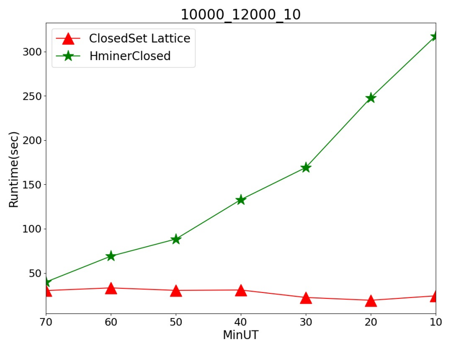
In this paper, we have proposed the closed-set lattice algorithm, which can mine closed high utility itemsets efficiently. The closed-set lattice algorithm first finds all of the closed frequent itemsets and then finds closed high utility itemsets from those closed frequent itemsets. Since the number of closed frequent itemsets will not be affected by the threshold value, our algorithm can mine closed high utility itemsets more efficient than the traditional algorithms for mining closed high utility patterns. We have also implemented the TWU pruning strategy to decrease the search space, when mining patterns in a static database. Moreover, We have considered mining CHUIs in the incremental database. From the simulation result, we have shown that our proposed algorithm has better performance than HMiner-Closed algorithm in dense databases and sparse databases.

VI. ACKNOWLEDGMENT

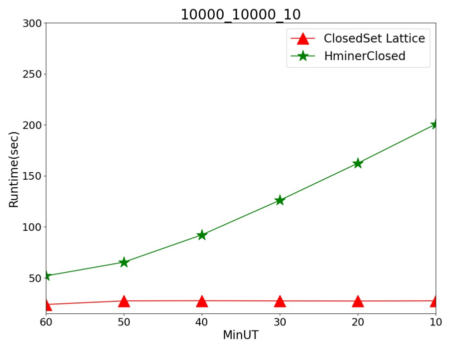
This research was supported in part by the National Science Council of Republic of China under Grant No. MOST 111-2221-E-110-054.



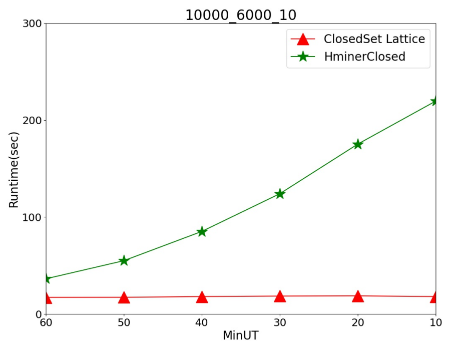
(a)



(b)



(c)



(d)

Fig. 9: A comparison by using four datasets under the change of $MinUT$: (a) 10000_15000_10; (b) 10000_12000_10; (c) 10000_10000_10; (d) 10000_6000_10.

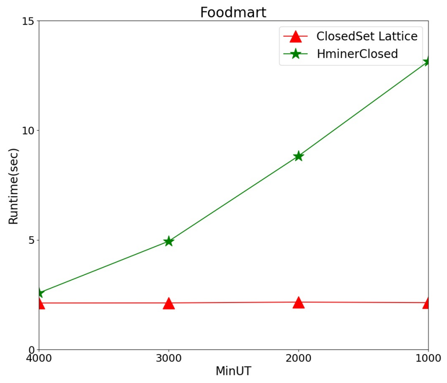


Fig. 10: A comparison by using dataset $foodmart$ under the change of $MinUT$

References

[1] U. Yun, H. Nam, G. Lee, and 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.

[2] M. Liu and J. Qu, "Mining High Utility Itemsets Without Candidate Generation," in *Proceedings of the 21st ACM international conference on Information and knowledge management*, 2012, pp. 55–64.

[3] Y. Liu, W.-k. Liao, and A. Choudhary, "A Two-phase Algorithm for Fast Discovery of High Utility Itemsets," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2005, pp. 689–695.

[4] S. Zida, P. Fournier-Viger, J. C.-W. Lin, C.-W. Wu, and 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, Feb. 2017.

[5] J. Lee, U. Yun, G. Lee, and 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.

[6] C.-W. Lin, G.-C. Lan, and T.-P. Hong, "An Incremental Mining Algorithm for High Utility Itemsets," *Expert Systems with Applications*, vol. 39, no. 8, pp. 7173–7180, June 2012.

[7] H. Nam, U. Yun, E. Yoon, and J. C.-W. Lin, "Efficient Approach of Recent High Utility Stream Pattern Mining with Indexed List Structure and Pruning Strategy Considering Arrival Times of Transactions," *Information Sciences*, vol. 529, Aug. 2020.

[8] L. T. Nguyen, P. Nguyen, T. D. Nguyen, B. Vo, P. Fournier-Viger, and V. S. Tseng, "Mining High-utility Itemsets in Dynamic Profit Databases," *Knowledge-Based Systems*, vol. 175, pp. 130–144, July 2019.

[9] T.-L. Dam, H. Ramampiaro, K. Nørvåg, and Q.-H. Duong, "Towards Efficiently Mining Closed High Utility Itemsets from Incremental Databases," *Knowledge-Based Systems*, vol. 165, pp. 13–29, Nov. 2019.

[10] P. Fournier-Viger, S. Zida, J. C.-W. Lin, C.-W. Wu, and V. S. Tseng, "EFIM-closed: Fast and Memory Efficient Discovery of Closed High-utility Itemsets," in *Proc. of International Conference on Machine Learning and Data Mining in Pattern Recognition*, 2016, pp. 199–213.

[11] L. T. Nguyen, V. V. Vu, M. T. Lam, T. T. Duong, L. T. Manh, T. T. Nguyen, B. Vo, and H. Fujita, "An Efficient Method for Mining High Utility Closed Itemsets," *Information Sciences*, vol. 495, pp. 78–99, Feb. 2019.

[12] B. Le, H. Nguyen, and B. Vo, "An Efficient Strategy for Mining High Utility Itemsets," *International Journal of Intelligent Information and Database Systems*, vol. 5, no. 2, pp. 164–176, March 2011.

[13] B. Goethals, *Apriori Property and Breadth-First Search Algorithms*, 2009, pp. 124–127.

[14] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation," *ACM Sigmod Records*, vol. 29, no. 2, pp. 1–12, Feb. 2000.

[15] V. S. Tseng, C.-W. Wu, P. Fournier-Viger, and S. Y. Philip, "Efficient Algorithms for Mining the Concise and Lossless Representation of High Utility Itemsets," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 3, pp. 726–739, Aug. 2014.

[16] R.-F. Chen, "A Subset-Lattice Algorithm for Mining High Utility Patterns over the Data Stream Sliding Window," Master's thesis, National Sun Yat-sen University, July 2017.

[17] W. hau Peng, "An Efficient Subset-Lattice Algorithm for Mining Closed Frequent Itemsets in Data Streams," Master's thesis, National Sun Yat-sen University, July 2009.

[18] P. Fournier-Viger, J. C.-W. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, and H. T. Lam, "The spmf open-source data mining library version 2," in *Machine Learning and Knowledge Discovery in Databases*, B. Berendt, B. Bringmann, É. Fromont, G. Garriga, P. Miettinen, N. Tatti, and V. Tresp, Eds. Cham: Springer International Publishing, 2016, pp. 36–40.