AN EFFICIENT ALGORITHM FOR MINING HIGH UTILITY ITEMSETS

Nguyen Thi Thanh Thuy*, Nguyen Van Le, Manh Thien Ly
Ho Chi Minh City University of Industry and Trade, Vietnam

ARTICLE INFORMATION

Journal: *Vinh University Journal of Science Natural Sciences, Engineering and Technology*
p-ISSN*: 3030-4563*
e-ISSN: *3030-4180*

Volume: 53
Issue: 2A
****Correspondence:***
thuyntt@huit.edu.vn

Received: *21 November 2023*
Accepted: *24 January 2024*
Published: *20 June 2024*

Citation:
Nguyen Thi Thanh Thuy, Nguyen Van Le, Manh Thien Ly (2024). An efficient algorithm for mining high utility itemsets. ***Vinh Uni. J. Sci.*** *Vol. 53 (2A), pp. 56-72*
doi: 10.56824/vujs.2023a147

OPEN ACCESS

Copyright © 2024. This is an Open Access article distributed under the terms of the Creative Commons Attribution License (CC BY NC), which permits non-commercially to share (copy and redistribute the material in any medium) or adapt (remix, transform, and build upon the material), provided the original work is properly cited.

ABSTRACT

High utility itemsets (HUIs) mining is the finding of itemsets that satisfy a user-defined minimum utility threshold. Many successful studies in this field have been carried out, however they are all reliant on Tidset techniques, which records the intersection of transactions in a data structure. This paper presents the DCHUIM algorithm which mines the high utility itemset based on the Diffset techniques. Essentially, this mechanism stores the subtraction set of transactions rather than the intersection set. In order to achieve this, a DUL data structure is proposed to store utilities information and subtraction transactions of an itemset. Furthermore, the algorithm also applies pruning strategies such as U-Prune, EUCS-Prune and the concept of closed utility to effectively compress data. Thus, in the mining process, the search space is greatly diminished. Experiment on large datasets including Accidents, Mushroom, Retail, Chainstore and compare the performance of DCHUIM algorithm with HMiner algorithm. The findings indicate that the DCHUIM method outperforms the HMiner algorithm in terms of memory utilization across all databases and outperforms it in terms of time on sparse databases.

Keywords: Frequent itemsets mining; high utility itemsets mining; transaction database.

1. Introduction

Nowadays, data mining is a topic of interest to many people. In the business sector, mined data will provide businesses with deeper insights into customer behavior and preferences. From there, managers can come up with business strategies to maximize profits. Studies on frequent itemsets mining (FIM) [1-4] and association rule mining (ARM) [5-7] have been conducted with the purpose of finding the itemset commonly purchased together by customers in the transaction database. Later, research on mining high utility itemsets (HUIM) arose to exploit and find sets of items that bring high profits to businesses by considering both the quantity and profit. Many studies on HUIM have been performed [8]. Initially, the exploitation of HUIs is carried out through 2 phases: Phase 1 identifies the candidate set and Phase 2 determines the exact HUIs by eliminating unsuitable candidates in Phase 1. Two-phase algorithms can be mentioned as UMining and

Umining-H [9], TWU-Mining [10], Two-Phase [11], UP-Growth [12], UP-Growth+ [13], etc. Reality shows that 2-phase algorithms often consume a lot of time and memory to identify and store the candidate set, especially when the candidate set is too large. Therefore, recently, HUIs mining algorithms have been implemented faster and more effectively with just 1 phase such as d2HUP [14], HUI-Miner [15], FHM [16], EFIM [17], HMiner [18], HUIM-SU [19], etc. Single-phase algorithms significantly improve execution time, however, memory consumption is still quite large. In addition, the common point of these algorithms is to store the intersection of transactions in a data structure.

In this paper, a new approach was proposed for HUIs mining, which is based on the Diffset mechanism. To accomplish that, the DUL structure has been built to store the utility of an itemset and its Diffset transactions in the database. Within the same equivalence class, DiffSet is calculated based on the difference between two Tidsets. For example, if the itemset {fde} has the prefix {fd} and the extension {e}, the transactions appearing in DUL(fde) will be the transactions that contain {fd} and do not contain {e}. This approach results in significantly improved execution time and memory space compared to previous algorithms due to the very small number of stored transactions, especially when applied on sparse databases.

The major and important contributions of this paper include:

- Build a DUL structure to store information about the utility of itemsets, as a basis for pruning during the mining process.

- Proposing the DCHUIM algorithm for efficient HUIs mining, combining the application of effective pruning strategies proposed in previous studies on HUIs mining such as U-Prune, EUCS-Prune to reduce search time and storage space.

- Experimental results on sparse and dense datasets demonstrate that compared to HMiner - the most efficient algorithm in recent times, the DCHUIM algorithm has better performance in terms of execution time and memory usage, especially on sparse databases.

2. Related works

2.1. Frequent itemsets

The concept of frequent itemsets mining was proposed by Agrawal in 1993 with the aim of finding relationships between items sold in supermarkets. The Apriori algorithm [4], proposed in 1994, is a popular algorithm in level-wise approaches with candidates generated at multiple levels. Next, the frequent itemset mining algorithm should be referred to as FP-Growth [2], in which the database is traversed twice, projection is then applied to create a local database of each single item, generate a local FP-Tree and ecursively mine the local tree. This algorithm uses the divide and conquer method to mine frequent itemsets and is a candidate-free method, which is often very effective on databases with high data duplication density. In addition, a number of studies on mining frequent itemsets and association rules have also been conducted such as Eclat, Clique [5], MGARs [6], NSFI [3], UniqAR [7]. However, these studies did not address other factors such as profit, weight, or interestingness of the items.

2.2. High utility itemsets

Recently, many studies have focused on mining high utility itemsets and certain achievements have been achieved. The initial HUIs mining algorithms were two-phase algorithms, typically the Two-Phase algorithm [11] proposed by Liu et al., in which phase 1 finds a candidate set that satisfies the TWU condition greater than the given minimum utility threshold, denoted minUtil; HUIs will be mined in phase 2 from the list of candidate sets found in phase 1. The TWU-Mining algorithm [10] implemented by Le et al., used the WIT tree structure, in which each node in the tree contains an itemset X, a set of transactions containing X, TWU and utility of X. In addition, other 2-phase algorithms can be mentioned such as UMining and UMining H [8], UP-Growth [11], UP-Growth+ [13], etc. However, these algorithms often consume a lot of time and memory to identify and store the candidate set, especially when the number of candidates found in phase 1 is quite large.

Therefore, the following algorithms focus on mining HUIs more quickly and effectively with just 1 phase. The original typical 1-phase algorithm is the HUI-Miner algorithm [14], proposed by Liu et al. in 2012 using the utility-list (UL) structure. In this structure, the authors used iutil to store information about the utility of an itemset and used ruutil to store the remaining utility of that itemset. Thanks to these two quantities, the algorithm can check whether an itemset is 1 HUI and can be expanded or not (iutil + rutil > minUtil). The 1-element UL set is built from the original database after pruning and sorting the items in ascending order according to TWU. The k element UL set is constructed by combining (k-1) elements ULs without rescanning the database. The FHM algorithm [16] then proposes a pruning strategy called EUCP, which will prune 2-element itemsets with TWU smaller than minUtil to reduce the search space.

The EFIM algorithm [17] was launched in 2017 by proposing three effective techniques to mine HUIs. Firstly, high-utility database projection (HDP) is performed, so that when exploring expanded itemsets, the algorithm only needs to scan through the items behind that itemset in each transaction, thereby reducing database scanning time. Next, high-utility transaction merging (HTM) is used to group similar transactions into one transaction, thereby reducing database size; At the same time, the algorithm also proposes two thresholds on sub-tree utility and local utility to reduce the search space. Another efficient algorithm is HMiner [18] proposed by Krishnamoorthy along with compact utility list (CUL) data structure. The advantage of this structure is that it applies the closed utility set concept to itemsets with 2 or more elements to effectively compress data during the mining process. Accordingly, for transactions that are in all CULs at the same level (with the same prefix), the information of those transactions will be counted in the closing utility and vice versa. Therefore, the time for reviewing transactions is significantly reduced, increasing the efficiency of the algorithm. At the same time, this algorithm is also combined with many different pruning strategies for effective HUIs mining. However, the HMiner algorithm still consumes quite a lot of memory across all databases.

3. Theoretical foundation

Let $I = \{x_1, x_2, \ldots, x_m\}$ be a set of m distinct items; X is a finite itemset such that $X \subseteq I$. A transaction database $D = \{T_1, T_2, \ldots, T_n\}$ has n transaction. For $\forall\, T_j \in D$, $T_j =$

$\{x_k | k = 1, 2, \dots, N_j, x_k \in I\}$, where N_j is the number of items in transaction T_j. Each x_i in I has a profit value of $p(x_i)$. Each x_k in transaction T_j has a purchase quantity of $q(x_k, T_j)$. An example of transaction database D is given in Table 1. The profits of the items are given in Table 2.

Table 1: *Transaction database D*

TID	Transaction (T)	Purchase quantity (q)	Utility (U)	Transactional utility (TU)
T_1	b, d, e	5, 1, 1	5, 1, 2	8
T_2	b, c, e, f	3, 6, 3, 4	3, 6, 6, 8	23
T_3	a, b, c, d, e, f	1, 4, 1, 8, 2, 3	5, 4, 1, 8, 4, 6	28
T_4	$b, c, d,$ e	2, 4, 6, 2	2, 4, 6, 4	16
T_5	a, c, e, f	1, 2, 3, 2	5, 2, 6, 2	15
T_6	a, c, d	2, 3, 6	10, 3, 6	19

Table 2: *Profit of items*

Items	*a*	*b*	*c*	*d*	*e*	*f*
Profit	5	1	1	1	2	2

The utility of an item x_i in a transaction T_j, denoted by $U(x_i, T_j)$. For example, the utility of item $\{b\}$ in transaction T_1 is calculated as $u(b, T_1) = p(b) * q(b, T1) = 1 * 5 = 5$.

The utility of an itemset $X = \{x_1, x_2, \dots, x_k\}$ in transaction T_j, denoted by $U(X, T_j)$ and is defined as $U(X, T_j) = \sum_{x_i \in X} U(x_i, T_j)$. For example: $U(bd, T_1) = u(b, T_1) + u(d, T_1) = 6$.

The utility of an itemset X in transaction database D, denoted by $U(X)$ and is defined as $U(X) = \sum_{X \subseteq T_j \wedge T_j \in D} U(X, T_j)$. For example: $U(bd) = U(bd, T_1) + U(bd, T_3) + U(bd, T_4) = 6 + 12 + 8 = 26$.

The utility of a transaction T_j in database D, denoted by $TU(T_j)$ and is defined as $TU(T_j) = \sum_{x_i \in T_j} U(x_i, T_j)$. For example: $TU(T_1) = U(b, T_1) + U(d, T_1) + U(e, T_1) = 8$ and $TU(T_2) = 23$.

Transaction-weighted utility of an itemset X in database D is denoted by $TWU(X)$ and is defined as $TWU(X) = \sum_{X \subseteq T_j \in D} TU(T_j)$. For example: $TWU(ac) = TU(T_3) + TU(T_5) + TU(T_6) = 62$.

A complete ordering $\prec$ is constructed based on the TWU ascending sorting of the items in database D. This sorting is intended to eliminate items with TWU < minUtil in the algorithm that follows. In the database given in Table 1, the complete order of the items is $a \prec f \prec d \prec b \prec e \prec c$. Table 3 represents the TWU of the items after ascending and Table 4 shows the database D after ascending according to TWU.

Table 3: *TWU of the items after being sorted ascending*

Items	*a*	*f*	*d*	*b*	*e*	*c*
TWU	62	66	71	75	90	101

Table 4: *Ordered purchase history*

TID	Transaction (T)	Purchase quantity (q)	Utility (U)	Transactional utility (TU)
T_1	d, b, e	1, 5, 1	1, 5, 2	8
T_2	f, b, e, c	4, 3, 3, 6	8, 3, 6, 6	23
T_3	a, f, d, b, e, c	1, 3, 8, 4, 2, 1	5, 6, 8, 4, 4, 1	28
T_4	d, b, e, c	6, 2, 2, 4	6, 2, 4, 4	16
T_5	a, f, e, c	1, 2, 2, 3	5, 2, 6, 2	15
T_6	a, d, c	2, 6, 3	10, 6, 3	19

All the following itemset of X in T_j are denoted as $T_j|X$, are the items that come after X in transaction T_j. For example, in Talbe 4, $T_2|\{f,b\} = \{e,c\}$ and $T_1|\{d\} = \{b,e\}$. The subsequent utility of an itemset X in a transaction T_j, denoted as $RU(X,T_j)$, is the total utility of all the following items of X in T_j, and is defined as $RU(X,T_j) = \sum_{x_i \in (T_j|X)} U(x_i,T_j)$. For example, $RU(d,T_1) = U(b,T_1) + U(e,T_1) = 7$.

4. DCHUIM Algorithms

4.1. DUL structure

In this paper, DUL structure is proposed to store information about the utility of an itemset X (composed of the prefix P and the extension A). This structure divides the utility of X into two parts, the closed part and the non-closed part. The closed part is stored through the values $CU/CRU/CPU$ and the non-closed part is stored through the values $NPU/NEU/NRU$. At the same time, transactions T_j in the structure below are sets of tuples with the structure $< T_j, PU, RPU >$, where T_j are transactions containing P but not A, PU is the prefix utility of X in the transaction, RPU is the post - utility of prefix X in transaction T_j. Figure 1 represents the structure of a DUL.

$NPU/NEU/NRU$		$CU/CRU/CPU$
T_j	$PU(X,T_j)$	$RPU(X,T_j)$

Figure 1: Structure of a DUL

The closed utility of an itemset X in transaction T_j, is denoted as $CU(X,T_j)$ and is defined:

$$CU(X,T_j) = \begin{cases} U(X,T_j), \ if \ |X| > 1 \ and \ C(X - x_k) = S(T_j/\{X - x_k\}) \\ 0, \ vice \ versa \end{cases} \quad [18]$$

Where: $C(X - x_k)$ is the itemset following $X - x_k$ in database and $S(T_j/\{X - x_k\})$ is the itemset following $X - x_k$ in transaction T_j.

The closed utility of an itemset X in database D, is denoted as $CU(X)$ and is defined: $CU(X) = \sum_{X \subseteq T_j \in D} CU(X,T_j)$ [18]

For example, in Table 4, $CU(af,T_3) = 11$, $CU(f,T_2 = 0)$ and $CU(ad) = CU(ad,T_3) + CU(cd,T_6) = 9 + 0 = 9$.

The closed remaining utility of an itemset X in a transaction T_j, denoted as $CRU(X,T_j)$, and is defined:

$$CRU(X,T_j) = \begin{cases} RU(X,T_j),\ if\ |X| > 1\ and\ C(X - x_k) = S(T_j/\{X\ - x_k\}) \\ 0,\ vice\ versa \end{cases} \quad [18]$$

The closed remaining utility of an itemset X in database D is defined as:

$CRU(X) = \sum_{X \subseteq T_j \in D} CRU(X,T_j)$ [18]

For example: $CRU(af,T_5) = 17$ and $CRU(de) = 8 + 4 = 12$.

Closed prefix utility of an itemset X in a transaction T_j, denoted as $CPU(X,T_j)$, is defined as:

$$CPU(X,T_j) = \begin{cases} PU(X,T_j), if\ |X| > 1\ and\ C(X - x_k) = S(T_j/\{X\ - x_k\}) \\ 0, vice\ versa \end{cases} \quad [18]$$

Closed prefix utility of an itemset X in database D is defined as:

$CPU(X) = \sum_{X \subseteq T_j \in D} CPU(X,T_j)$ [18]

For example: $CPU(af,T_5) = 5$ and $CPU(de) = 4 + 6 = 10$.

In addition, in this article, to calculate non-closed utility, we propose some additional definitions as follows:

Definition 1: Given an itemset X composed of prefix P and extension A. The non-closing prefix utility of itemset X in the database, denoted as $NPU(X)$, is the total non-closing utility of prefix P in transactions containing X and is defined:

$$NPU(X) = \sum_{X \subseteq T_j \in D \wedge CU(X,T_j)=0} U(P,T_j)$$

For example, in Table 4, consider the itemset $\{ad\}$ composed of the prefix $\{a\}$ and the extension $\{d\}$, $NPU(ad) = U(a,T_6) = 10$. Transaction T_3 contains $\{ad\}$ but is not considered because $\{ad\}$ in transaction T_3 is closed.

Definition 2: Given an itemset X composed of prefix P and extension A. The non-closing extended utility of itemset X in the database, denoted as $NEU(X)$, is the total non-closing utility of A in transactions containing X and is defined:

$$NEU(X) = \sum_{X \subseteq T_j \in D \wedge CU(X,T_j)=0} U(A,T_j)$$

For example, in Table 4, consider the itemset $\{ad\}$ composed of the prefix $\{a\}$ and the extension $\{d\}$, $NEU(ad) = U(d,T_6) = 6$. Transaction T_3 contains $\{ad\}$ but is not considered because $\{ad\}$ in transaction T_3 is closed.

Definition 3: The utility of an itemset X in the database, denoted as $U(X)$, is the sum of non-closed utility and closed utility of X in the transactions that contain X and is defined:

$$U(X) = NPU(X) + NEU(X) + CU(X)$$

For example, in Table 4, $U(ad) = NPU(ad) + NEU(ad) + CU(ad) = 10 + 6 + 13 = 29$.

Definition 4: Given an itemset X composed of prefix P and extension A. The remaining non-closed utility of item

set X in the database, denoted as $NRU(X)$, is the total remaining non-closed utility of A n transactions containing X and is defined:

$$NRU(X) = RU(X) - CRU(X)$$

For example, in Table 4, $NRU(ad) = RU(ad) - CRU(ad) = 12 - 9 = 3$.

Definition 5: Given an itemset X composed of prefix P and extension A. The prefix utility of the itemset X in a transaction T_j, denoted as $PU(X, T_j)$, is the utility of P in transaction T_j and is defined:

$$PU(X, T_j) = U(P, T_j)$$

For example, in Table 4, consider the itemset $\{fdb\}$ composed of the prefix $\{fd\}$ and the extension $\{b\}$, $PU(fdb, T_3) = U(fd, T_3) = 6 + 4 = 10$.

Definition 6: Given an itemset X composed of prefix P and extension A. Post - prefix utility of itemset X in a transaction T_j, denoted as $RPU(X, T_j)$, is the posterior utility of P in transaction T_j and is defined:

$$RPU(X, T_j) = RU(P, T_j)$$

For example, in Table 4, consider the itemset $\{fdb\}$ composed of the prefix $\{fd\}$ và and the extension $\{b\}$, $RPU(fdb, T_3) = RU(fd, T_3) = 1 + 4 + 8 = 13$.

An itemset set X is called *High Utility Itemset - HUI* if $U(X) \geq minUtil$, where minUtil is the minimum utility threshold value.

$HUIs = \{X \subseteq I \mid U(X) \geq minUtil\}$ [18]

For example, in Table 4, $U(ec) = 33$. For $minUtil = 32$, the itemset $\{fe\}$ is one HUI. HUIs can be found in the database are $\{adc, fbec, fe, fec, dbe, dbec, bec, ec\}$.

4.2. Construction of DUL

This section will illustrate how to construct single-element DUL ($1-DUL$), two-element ($2-DUL$) and k-element ($k-DUL$) with $k \geq 3$ in database D (after sorting the items in ascending order according to TWU in Table 4).

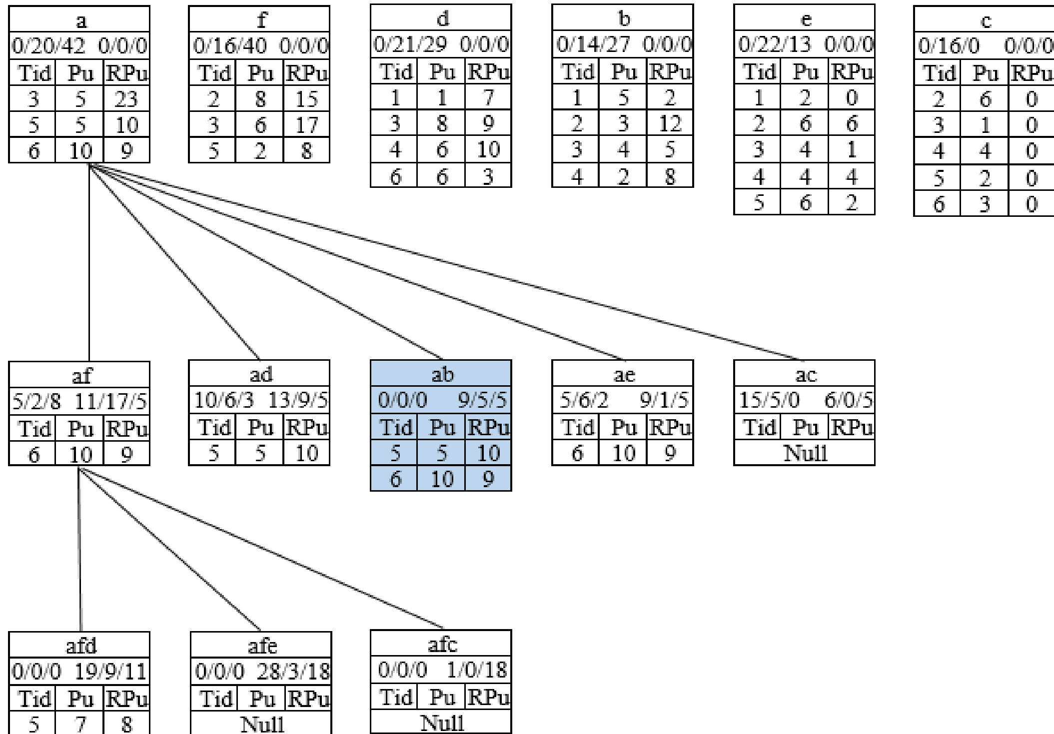

Figure 2: *Illustrative example*

4.2.1. Construction 1-itemset $\boldsymbol{DUL}$ (*1*$-\boldsymbol{DUL}$)

1-itemset $\boldsymbol{DUL}$ is constructed by scanning the database in Table 4, to find transactions containing that element and calculate the corresponding PU and RPU values. This transaction is then included in the list of transactions below $\boldsymbol{DUL}$, recalculate NEU and NRU. Note that 1-itemset DUL have $CU/CRU/CPU/NPU$ value of 0 because $|X| = 1$ and the prefix is empty.

For example: $DUL(a)$ includes 3 sets corresponding to transactions T_3, T_5, T_6. T_3 has $PU(a, T_3) = 5$, $RPU(a, T_3) = 23$; T_5 has $PU(a, T_5) = 5$, $RPU(a, T_5) = 10$; T_6 has $PU(a, T_6) = 10$, $RPU(a, T_6) = 9$. $NEU(a) = 5 + 5 + 10 = 20$; $NRU(a) = 23 + 10 + 9 = 42$. Similar for other DULs.

4.2.2. Construction of two-itemsets DUL

A two-itemsets DUL is constructed by the combination of two 1-itemset DULs. Suppose we need to construct $DUL(af)$, we need $DUL(a)$ and $DUL(f)$. The construction of $DUL(af)$ is as follows:

First, initialize $DUL(af)$ from $DUL(a)$ and $DUL(f)$ with the following general information: $NPU(af) = 0$; $NEU(af) = 0$; $NRU(af) = 0$; $CU(af) = 0$; $CRU(af) = 0$; $CPU(af) = 0$; and the list of Tid set below of $DUL(af)$ is empty.

The idea of the algorithm is based on Diffset, so the transactions appearing in $DUL(af)$ must be transactions contained in $DUL(a)$ and not contained in $DUL(f)$. Therefore, browse the Tid in $DUL(a)$ one by one. If this Tid is not in $DUL(f)$, then a set $< Tid, PU(af, Tid), RPU(af, Tid) >$ will be included in $DUL(af)$. On the contrary, if the Tid under consideration is in both $DUL(a)$ and $DUL(f)$, then consider the itemset $\{af\}$ as closed or not closed in that transaction. If closed, update the $CU/CRU/CPU$ values. If not closed, update the $NPU/NEU/NRU$ values. In this example, T_3 and T_5 are in the Tid list of both $DUL(a)$ and $DUL(f)$, but $\{af\}$ in T_3 is closed, $\{af\}$ in T_5 is not closed. Therefore $CU(af) = PU(a, T_3) + PU(f, T_3) = 11$, $CRU(af) = RPU(f, T_3) = 17$, $CPU(af) = PU(a, T_3) = 5$; $NPU(af) = PU(a, T_5) = 5$, $NEU(af) = PU(f, T_5) = 2$, $NRU(af) = RPU(f, T_5) = 8$. Particularly T_6 is in the Tid list of *DUL(a)* and not in the Tid list of *DUL(f)*, so the Tid list of $DUL(af)$ contains T_6 with $PU(af, T_6) = 10$ and $RPU(af, T_6) = 9$. Similar for other DULs.

4.2.3. Construction of k-itemsets DUL (k≥3)

A k-itemsets DUL is constructed by combining 2 DUL of (k-1) itemsets that share the same prefix. Suppose we need to construct $DUL(afd)$, we need $DUL(af)$ and $DUL(ad)$. The construction of $DUL(afd)$ is as follows:

First, initialize $DUL(afd)$ from $DUL(af)$ and $DUL(ad)$ with the following general information: $NPU(afd) = 7$; $NEU(afd) = 6$; $NRU(afd) = 3$; $CU(afd) = 19$; $CRU(afd) = 9$; $CPU(afd) = 11$; and the list of Tid set below of $DUL(afd)$ is empty.

Transactions appearing in $DUL(afd)$ must contain $\{af\}$ but not $\{d\}$. Therefore, to find these Tid, perform sequential browsing of Tid in $DUL(ad)$, if this Tid is not in the Tid list of $DUL(af)$, then a set $< Tid, PU(afd, Tid), RPU(afd, Tid) >$ will be included in $DUL(afd)$ and recalculate the values $NPU(afd) = NPU(afd) - PU(afd, Tid)$. Conversely, browse the Tid in $DUL(af)$ one by one. If this Tid is not in $DUL(ad)$, then recalculate the values $NEU(afd) = NEU(afd) - PU(ad, Tid)$, $NRU(afd) =$

$NRU(afd) - RPU(ad, Tid)$. In this example, T_5 is in $DUL(ad)$ and not in $DUL(af)$. Therefore, a set $< 5, 7, 8 >$ is put into $DUL(afd)$ and recalculated $NPU(afd) = NPU(afd) - PU(afd, T_5) = 0$. At the same time, transaction T_6 is only in $DUL(af)$ but not in $DUL(ad)$, so $NEU(afd) = NEU(afd) - PU(d, T_6) = 0$, $NRU(afd) = NRU(afd) - RPU(d, T_6) = 0$.

4.3. Pruning strategies

In mining high utility itemsets, pruning strategies play a very important role in improving algorithm performance thanks to the ability to narrow the search space; thereby optimizing execution time and storage space. In this paper, two pruning strategies U-Prune and EUCS-Prune [15] are used to increase the execution performance of the algorithm.

Strategy 1 (U-Prune): If the total of utility and remaining utility of the itemset X and the total utility of the following items of X is less than $minUtil$, any extension from X is not a HUI. That is, if $U(X) + RU(X) < minUtil$ then $\forall\, Y \supseteq X$, $Y \notin HUI$. Then stop expanding with the itemset X [15].

Strategy 2 (EUCS-Prune): If $TWU(X) < minUtil$ (with X is a 2-itemset), then $X' \notin HUI, \forall X' \supseteq X$. [15].

4.4. $DCHUIM$ algorithm

In this paper, the $DCHUIM$ algorithm is proposed for high utility itemset mining. The main algorithm $DCHUIM$ has as input the transaction database D and the $minUtil$ utility threshold. The output of the algorithm is a list of high utility itemsets. The algorithm first scans the database D to calculate the $TWU(i)$ value for each item i in the database. Then, remove items i with $TWU(i) < minUtil$ from I and only put items i that satisfy the condition into I^*. Then sort I^* ascending according to TWU and rearrange the items in D according to the order of I^* (line 3). In line 4, calculate the values for the $EUCS$ structure. Scan the database D lần 2 a second time and create a list of DUL for each $i \notin I^*$ (line 5). Finally, call $MiningDCHUI$ to mine high utility itemsets with $minUtil$ threshold.

Algorithm 1: $Main\ algorithm -\ DCHUIM$

Input: D: Transactional database, $minUtil$: Minimum utility threshold.
Output: $HUIs$: High utility itemsets.
1. Scan database D to calculate $TWU(i)$ for each i contained in I.
2. Calculate $I^* = \{i \in I \mid TWU(i) \geq minUtil\}$, remove from D for $i \notin I^*$.
3. Sort I^* ascending by TWU, arranges the items in D in order of I^*.
4. Initialize the $EUCS$ structure.
5. Scan database D to create a list of DUL for each $i \notin I^*$ as $oneDULs$
6. $MiningDCHUI(\varnothing, oneDULs, minUtil)$

The input data of algorithm 2 ($MiningDCHUI$) includes a P as a prefix, a list of DUL as $oneDULs$ and $minUtil$. From lines 1 to 5, the algorithm goes through all $DUL\ X$ in the list of $oneDULs$ và check the condition that if $U(X) = X.NPU + X.NEU + X.CU \geq minUtil$ then put X into $HUIs$. Line 6 checks the scaling condition by applying the U-Prune pruning strategy. Line 7 calls the function $twoDUL$Construct to create two-itemsets DUL ($twoDULs$). Line 9 calls the $Hui_diffset_Miner$ function to continue mining $HUIs$ on $twoDULs$.

Algorithm 2: $MiningDCHUI$

Input: P: prefix; $oneDULs$: List of $DULs$ prefixed with P; $minUtil$: Minimum utility threshold.

Output: $HUIs$: High utility itemsets.

1. **for** each position i in $oneDULs$ **do**
2. $X = oneDULs[i]$;
3. **if** $X.NPU + X.NEU + X.CU \geq minUtil$ **then**
4. $HUIs \leftarrow X$
5 **end if**
6. **if** $(X.NPU + X.NEU + X.CU + X.NRU + X.CRU \geq minUtil)$ **then**//U-Prune
7. $twoDULs \leftarrow twoDUL$Construct$(X, oneULs, i, minUtil)$; // two-itemset DUL
8. $P = \{P \cup X.item\}$;
9. $Hui_diffset_Miner(P, twoDULs, minUtil)$;
10. **end if**
11. **end for**

Algorithm 3 ($Hui_diffset_Miner$) has input data including a P as prefix, a list of DUL as $twoDULs$ and $minUtil$. From lines 1 to 5, the algorithm goes through all DUL X in the list of two$DULs$ and checks the condition that if $U(X) = X.NPU + X.NEU + X.CU \geq minUtil$ then put X in $HUIs$. Line 6 checks the scaling condition by applying the U-Prune pruning strategy. Line 7 calls the $kDUL$Construct function to create DULs with k itemsets ($kDULs$), k≥3. Line 9 recursively calls the $Hui_diffset_Miner$ function to continue mining $HUIs$ on $kDULs$.

Algorithm 3: $Hui_diffset_Miner$

Input: P: prefix; $twoDULs$: List of DULs prefixed with P, $minUtil$: Minimum utility threshold.

Output: $HUIs$: High utility itemsets.

1. **for** each position i in $twoDULs$ **do**
2. $X = twoDULs[i]$;
3. **if** $X.NPU + X.NEU + X.CU \geq minUtil$ **then**
4. $HUIs \leftarrow X$;
5 **end if**
6. **if** $(X.NPU + X.NEU + X.CU + X.NRU + X.CRU \geq minUtil)$ **then**//U-Prune
7. $kDULs \leftarrow kDUL$Construct$(X, twoDULs, i, minUtil)$; // k-itemsets DUL
8. $P = \{P \cup X.item\}$;
9. $Hui_diffset_Miner(P, kDULs, minUtil)$;
10. **end if**
11. **end for**

Algorithm 4 ($twoDULConstruct$) has input data including 1 DUL with prefix X, a list of DULs as $oneDULs$, st: starting position and $minUtil$. From lines 1 to 8, the algorithm traverses the $oneDULs$ Y after X, if $EUCS(X, Y)$ is greater than $minUtil$ then

initialize a $DUL(XY)$ that extends from X and included in $twoDULs$. In line 9, $extSz$ is used to store the DULs of $twoDULs$. Next, from line 11 to line 24, the algorithm browses each Tid in X in turn. If this Tid appears in any DUL of $twoDULs$, add the location of this DUL to newT. Otherwise, add the corresponding dataset of this Tid to the DUL under consideration. From line 25 to line 41, the algorithm checks if the itemset XY s closed then updates the values of $CU/CRU/CPU$, otherwise updates the $NPU/NEU/NRU$ values. Line 43 returns the result in $twoULs$.

Algorithm 4: $twoDUL$Construct

Input: X: DUL prefix; itemset $oneDULs$; st: starting position; $minUtil$: Minimum utility threshold.

Output: $twoDULs$: 2-itemsets DULs.

1. **for** $i = st + 1$ to $oneDULs.size$-1 **do**
2. $Y = oneDULs[i]$; //Y sau X
3. **if** $EUCS(X.item, Y.item) \geq minUtil$ **then** // EUCS pruning strategy
4. Initial $newUL(Y.item)$; // Initialize a new DUL extended from X
5. $twoDULs \leftarrow newDUL$;
6. $ey_tid[i] = 0$; // mark tid position in newDUL
7. **end if**
8. **end for**
9. $extSz = twoDULs.size$;
10. $newT = null$;
11. **for** each ex in $X.tidList$ **do**
12. **for** $j = 0$ to $twoDULs.size - 1$ **do**
13. **if** $twoDULs[j] = null$ **then**
14. continue;
15. **end if**
16. $eylist = oneDULs[twoDULs[j].item].tidList$;
17. **while** $ey_tid[j] < eylist.size$ and $eylist[ey_tid[j]].tid < ex.tid$ **do**
18. $ey_tid[j] += 1$;
19. **if** $ey_tid[j] < eylist.size$ and $eylist[ey_tid[j]].tid = ex.tid$ **then**
20. $newT \leftarrow j$;
21. **else**
22. $twoDULs[j].tidList \leftarrow ex$;
23. **end if**
24. **end for**
25. **if** $newT.size = extSz$ **then**
26. **for** $j = 0$ to $newT.size - 1$ **do**
27. $ey = oneDULs[twoDULs[j].item]$;
28. $eyy = ey.tidList[ey_tid[newT[j]]]$;

29. $twoDULs[newT[j]].CU += ex.PU + eyy.PU$;
30. $twoDULs[newT[j]].CRU += eyy.RPU$;
31. $twoDULs[newT[j]].CPU += ex.PU$;

32. **end for**

33. **else**
34. **for** $j = 0$ to $newT.size - 1$ **do**
35. $ey = oneDULs[twoDULs[j].item]$;
36. $eyy = ey.tidList[ey_tid[newT[j]]]$;
37. $twoDULs[newT[j]].NPU += ex.PU$;
38. $twoDULs[newT[j]].NEU += eyy.PU$;
39. $twoDULs[newT[j]].NRU += eyy.RPU$;
40. **end for**
41. **end if**
42. **end for**
43. **return** $twoDULs$;

Algorithm 5 ($kULConstruct$) has input data including 1 DUL with prefix X, a list of DUL $- twoULs$, st: starting position and $minUtil$. First, the algorithm initializes the $kULs$ which has the same number of elements as $twoULs$. Next, perform an X expansion with the Y_s after the X and update the corresponding $kULs$. From lines 4 to 9, the algorithm assigns the initial $NPU/NEU/NRU/CU/CRU/CPU$ values; use $preX$ sand $preY$ as 2 DUL at 1-itemset level corresponding to X and Y. From line 12 to line 19, browse each element in turn $ey \in Y$ and find $ex \in X$ such that $ex.tid = ey.tid$. If not, find $x \in preX$ such that $x.tid = ey.tid$; create element exy combined from x and ey (line 14, 15); add exy to $DUL(XY)$; update the NPU value at the same time. From line 20 to line 27, browse each element in turn $ex \in X$ and find $ey \in Y$ such that $ex.tid = ey.tid$. If not, find $y \in preY$ such that $y.tid = ex.tid$; update the NEU, NRU values. Line 28 removes empty DULs at the k-element level. Line 29 returns $kULs$.

Algorithm 5: kULConstruct

Input: X: DUL prefix; itemset $twoULs$; st: starting position; $minUtil$: Minimum utility threshold.

Output: $kULs$: k-element DULs.

1. Initialize $kULs$ with an initial number of elements equal to the number of elements of $twoULs$
2. **for** $i = st + 1$ to $twoULs.size - 1$ **do**
3. $Y = twoULs[i]$; //Y sau X
4. $kULs[i].NPU = X.NPU + X.NEU$;
5. $kULs[i].NEU = Y.NEU$;
6. $kULs[i].NRU = Y.NRU$;
7. $kULs[i].CU = X.CU + Y.CU - X.CPU$;
8. $kULs[i].CRU = Y.CRU$;
9. $kULs[i].CPU = X.CU$;

10. $preX = oneULs[X.item]$;
11. $preY = oneULs[Y.item]$;
12. **for** each element ey in Y **do**
13. **if** ! $\exists ex \in X \wedge ex.tid = ey.tid$ **then**
14. Find $x \in preX$ such that $x.tid = ey.tid$;
15. $exy = < ey.tid, ey.PU + x.PU, x.RPU >$;
16. $kULs[i] \leftarrow exy$;
17. $kULs[i].NPU -= exy.PU$;
18. **end if**
19. **end for**
20. **for** each element ex in X **do**
21. **if** ! $\exists ey \in Y \wedge ey.tid = ex.tid$ **then**
22. Find $y \in preY$ such that $y.tid = ex.tid$
23. $kULs[i].NEU - = y.PU$;
24. $kULs[i].NRU -= y.RPU$;
25. **end if**
26. **end for**
27. **end for**
28. Remove empty $kULs$
29. **return** $kULs$;

5. Experimentation

Experiments are carried out to evaluate the proposals in the $DCHUIM$ algorithm. The algorithm is installed in Java language, experimentally run on an HP 14-bs1 computer with Intel Core i5-8250 CPU @1.6GHz configuration, 8GB RAM and Windows 10 operating system. The standard dataset used in HUI mining documents is used, downloaded from the $SPMF$ library [20], including: Accidents, Mushroom, Retail and Chainstore. These data sets have completely characteristics of representing different types of data such as sparse, dense and diverse transaction lengths. Descriptive information for the datasets is presented in Table 5.

The experimental results of the $DCHUIM$ algorithm are compared with the most effective recent algorithm $HMiner$ [18] (the original implementation of the $HMiner$ algorithm is downloaded from the SPMF library SPMF [20]) based on an assessment of execution time and memory usage.

Table 5: *Characteristics of experimental databases*

Database	Transaction quantity	Number of items (I)	Average length (A)	Dense (A/I) %
Accidents	340,183	468	33.8	7.2222
Mushroom	8,124	119	23	19.3277
Retail	88,162	16,470	10.3	0.0625
Chainstore	1,112,949	46,086	7.3	0.0158

Figure 3 compares the performance of the proposed algorithm $DCHUIM$ and the $HMiner$ algorithm in terms of execution time on four databases $Accidents$, $Mushroom$, $Retail$ and $Chainstore$. With a high density database (Accidents), the execution time of the $HMiner$ algorithm and the $DCHUIM$ algorithm are almost the same. Specifically with the $Accidents$ database, at the thresholds $minUtil = 27{,}000{,}000$ to $minUtil = 30{,}000{,}000$, the execution time of the proposed algorithm is lower than that of the $HMiner$ algorithm. However, when the $minUtil$ threshold decreases to 26,000,000, the execution time of the $DCHUIM$ algorithm increases slightly compared to $HMiner$.

With an average database of Mushroom, these two algorithms have roughly equal execution time. With the Retail sparse database, the execution time of the $DCHUIM$ algorithm is almost twice as fast as the $HMiner$ algorithm at all $minUtil$ thresholds from 2,000 to 6,000. With the Chainstore sparse database, the $DCHUIM$ algorithm has an average execution time of 55.35 seconds, while the $HMiner$ algorithm has a higher execution time of 132.27 seconds. Also, in the Chainstore database, the lower the $minUtil$ threshold, the more effective the $DCHUIM$ algorithm is. This observation shows that the proposed DUL structure is indeed highly effective and the pruning strategies applied in the $DCHUIM$ algorithm have helped significantly narrow the search space, thereby increasing the algorithm's performance, especially on sparse databases. For dense databases, the execution time of $DCHUIM$ is approximately that of $HMiner$.

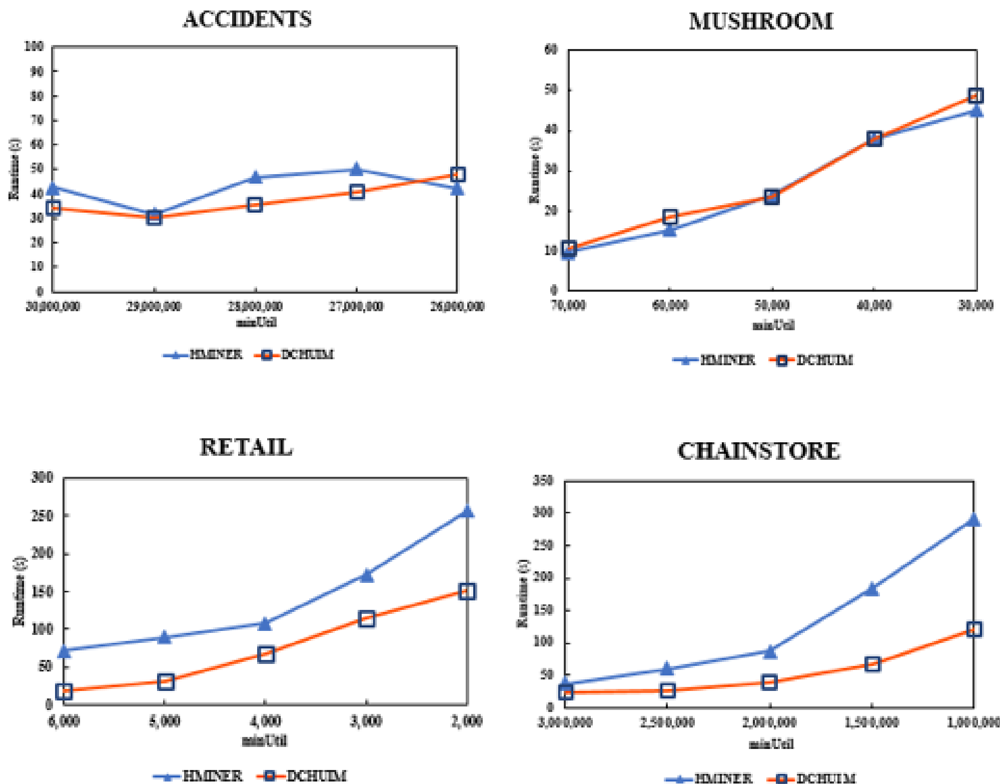

Figure 3: *Compare execution time*

Figure 4 compares memory usage between two algorithms $DHUIM$ and $HMiner$ on four databases: $Accidents$, $Mushroom$, $Retail$ and $Chainstore$. The results show

that for the *Accidents* database, *DCHUIM*'s memory capacity is only half that of *HMiner*. With the *Mushroom* database, *HMiner*'s memory is nearly 10 times that of *DCHUIM*. With *Retail* and *Chainstore*, the memory of *DCHUIM* s only half that of *HMiner*. Thus, experimental data shows that in both dense databases (*Accidents, Mushroom*) and sparse databases (*Retail* and *Chainstore*), the memory usage of *DHUIM* is much better than *HMiner* at all comparison thresholds. The reason is that *DCHUIM* only stores Diffset transactions and not Tidset transactions, resulting in significantly reduced memory capacity.

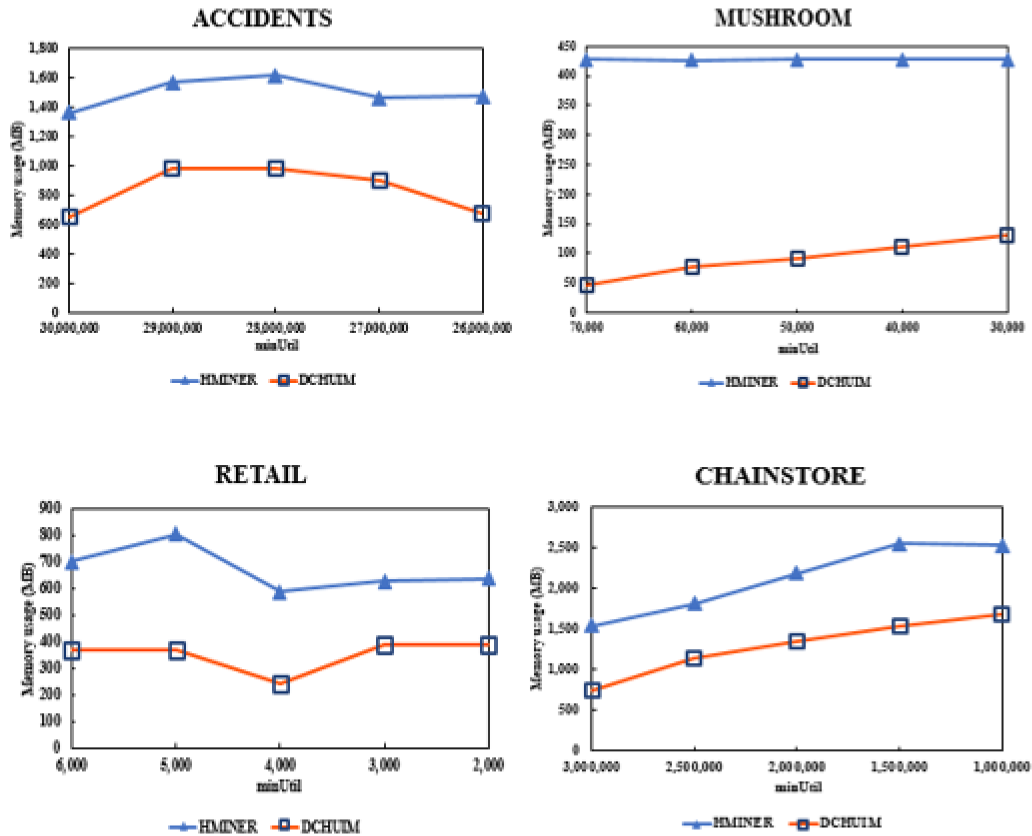

Figure 4: *Comparison of memory usage*

6. Conclusions

In this paper, a new structure called *DUL* has been proposed to mine high utility itemsets on transactional databases through the *DCHUIM* algorithm. This structure stores DiffSet transactions instead of storing TidSet transactions; At the same time, the algorithm also applies effective pruning strategies such as *U-Prune, TWU-Prune, EUCS-Prune* to optimize storage space during mining. Experimental results show that the *DCHUIM* algorithm has faster execution time than the *HMiner* algorithm in sparse databases. The memory usage of the *DCHUIM* algorithm is also better than the *HMiner* algorithm in all databases and at all minimum usefulness thresholds.

REFERENCES

[1] G. Grahne and J. Zhu, "Fast algorithms for frequent itemset mining using FP-Trees," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 10, pp. 1347-1362, 2005. DOI: 10.1109/TKDE.2005.166

[2] J. Han, J. Pei and Y. Yin, "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach," *Data Mining and Knowledge Discovery*, pp. 53-87, 2004. DOI: 10.1023/B:DAMI.0000005258.31418.83

[3] B. Vo, T. Le, F. F and T. P. Hong, "Mining frequent itemsets using the N-list and subsume concepts," *International Journal of Machine Learning and Cybernetics*, vol. 7, p. 253-265, 2016. DOI: 10.1007/s13042-014-0252-2

[4] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," *In Proc. 20th Int. Conf. Very Large Data Bases (VLDB*), pp. 487-499, 1994.

[5] M. Zaki, S. Parthasarathy, M. Ogihara and W. Li, "New algorithms for fast discovery of association rules," *Knowledge Discovery and Data Mining*, pp. 283-286, 1997. DOI: 10.1007/978-1-4615-5669-51

[6] B. Vo, T. P. Hong and B. Le, "A lattice-based approach for mining most generalization association rules," *Knowledge-Based Systems*, vol. 45, pp. 20-30, 2013. DOI: 10.1016/j.knosys.2013.02.003

[7] M. Nasr, M. Hamdy, D. Hegazy and K. Bahnasy, "An efficient algorithm for unique class association rule mining," *Expert Systems with Applications*, vol. 164, p. 113978, 2021. DOI: 10.1016/j.eswa.2020.113978

[8] Kumar R. And Singh Kuldeep, "High utility itemsets mining from transactional databases: a survey," *Applied Intelligence*, vol. 53, p. 27655-27703, 2023. DOI: 10.1007/s10489-023-04853-5

[9] H. Yao, H. J. Hamilton and a. C. J. Butz, "A foundational approach to mining Itemset Utilities from Databases," *Proceedings SIAM International Conference on Data Mining*, p. 482 - 486, 2004. DOI: 10.1137/1.9781611972740.51

[10] B. Le, H. Nguyen and a. B. Vo, "An efficient strategy for mining high utility itemsets," *International Journal of Intelligent Information and Database Systems*, vol. 5, pp. 164-176, 2011. DOI: 10.1504/IJIIDS.2011.038970

[11] 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*, pp. 689-695, 2005. DOI: 10.1007/1143091979

[12] V. S. Tseng, C. W. Wu, B. E. Shie and P. S. Yu, "UP-Growth: an efficient algorithm for high utility itemset mining," *In Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 253-262, 2010. DOI: 10.1145/1835804.1835839

[13] V. S. Tseng, B. E. Shie, C. W. Wu and S. Y. Philip, "Efficient algorithms for mining high utility itemsets from transactional databases," *IEEE transactions on knowledge and data engineering*, vol. 25, pp. 1772-1786, 2012. DOI: 10.1109/TKDE.2012.59

[14] J. Liu, K. Wang and B. C. Fung, "Direct discovery of high utility itemsets without candidate generation," *IEEE 12th international conference on data mining*, pp. 984-989, 2012. DOI: 10.1109/ICDM.2012.20

[15] M. Liu and J. Qu., "Mining high utility itemsets without candidate generation," *Proceedings of the 21st ACM international conference on Information and knowledge management*, pp. 55-64, 2012. DOI: 10.1145/2396761.2396773

[16] P. Fournier-Viger, C. W. Wu, S. Zida and V. S. Tseng, "FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning," *International Symposium on Methodologies for Intelligent Systems*, vol. 8502, pp. 83-92, 2014. DOI: 10.1007/978-3-319-08326-1_9

[17] S. Zida, P. Fournier-Viger, J. Lin, C. Wu and a. V. Tseng, "EFIM: A Highly Efficient Algorithm for High-Utility Itemset Mining," *Knowledge and Information Systems*, vol. 51, pp. 595-625, 2017. DOI: 10.1007/s10115-016-0986-0

[18] S. Krishnamoorthy, "HMiner: Efficiently mining high utility itemsets," *Expert Systems with Applications*, pp. 168-183, 2017. DOI: 10.1016/j.eswa.2017.08.028

[19] Z. Cheng, W. Fang, W. Shen, J. C. W. Lin and B. Yuan, "An efficient utility-list based high-utility itemset mining algorithm," *Applied Intelligence*, vol. 53, pp. 6992-7006, 2023. DOI: 10.1007/s10489-022-03850-4

[20] P. Fournier-Viger, A. Gomariz, A. Soltani and H. Lam, "An Open-Source Data Mining Library," 2014. [Online]. Available: http://www.philippe-fournier-viger.com.

TÓM TẮT

MỘT THUẬT TOÁN HỮU ÍCH ĐỂ KHAI THÁC TẬP HỮU ÍCH CAO

Nguyễn Thị Thanh Thủy*, Nguyễn Văn Lễ, Mạnh Thiên Lý
Trường Đại học Công Thương Thành phố Hồ Chí Minh, Việt Nam
Ngày nhận bài 21/11/2023, ngày nhận đăng 24/01/2024

Khai thác tập hữu ích cao (High Utility Itemsets - HUIs) là việc tìm ra các tập mục thỏa mãn một ngưỡng độ hữu ích tối thiểu do người dùng xác định. Đã có nhiều thuật toán khai thác tập hữu ích cao hiệu quả, tuy nhiên các thuật toán này đều dựa trên cơ chế Tidset (lưu trữ tập giao của các giao dịch trong cấu trúc dữ liệu). Ở bài báo này, thuật toán DCHUIM được đề xuất nhằm khai thác tập hữu ích cao bằng cơ chế Diffset, tức là thay vì lưu trữ tập giao thì cơ chế này sẽ lưu trữ tập trừ của các giao dịch. Để thực hiện việc này, một cấu trúc dữ liệu DUL được xây dựng để lưu trữ thông tin về độ hữu ích của một tập mục và các giao dịch thuộc tập Diffset. Ngoài ra, thuật toán còn áp dụng các chiến lược cắt tỉa như U-Prune, EUCS-Prune và sử dụng tập hữu ích đóng để nén dữ liệu một cách hiệu quả, từ đó giảm không gian tìm kiếm trong quá trình khai thác. Thực nghiệm được thực hiện trên các bộ dữ liệu lớn gồm Accidents, Mushroom, Retail, Chainstore và tiến hành so sánh hiệu suất thực thi giữa thuật toán DCHUIM với thuật toán HMINER. Kết quả cho thấy thuật toán DCHUIM có hiệu suất tốt hơn thuật toán HMINER về thời gian thực thi trên cơ sở dữ liệu thưa và vượt trội về bộ nhớ sử dụng trên tất cả các cơ sở dữ liệu.

Từ khóa: Khai thác tập phổ biến, khai thác tập hữu ích cao, cơ sở dữ liệu giao dịch.