IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON UTILITY PATTERN MINING: THEORETICAL ANALYTICS AND APPLICATIONS

Received February 17, 2020, accepted March 4, 2020, date of publication March 9, 2020, date of current version March 23, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2979289

Efficient Approach for Damped Window-Based High Utility Pattern Mining With List Structure

HYOJU NAM[1], UNIL YUN[1], BAY VO[2], (Member, IEEE), TIN TRUONG[3], ZHI-HONG DENG[4], AND EUNCHUL YOON[5], (Senior Member, IEEE)

[1]Department of Computer Engineering, Sejong University, Seoul, South Korea
[2]Faculty of Information Technology, Ho Chi Minh City University of Technology (HUTECH), Ho Chi Minh City 700000, Vietnam
[3]Division of Computational Mathematics and Engineering, Faculty of Mathematics and Statistics, Institute for Computational Science, Ton Duc Thang University, Ho Chi Minh City 758307, Vietnam
[4]Department of Machine Intelligence, Peking University, Beijing 100871, China
[5]Department of Electronics Engineering, Konkuk University, Seoul 05029, South Korea

Corresponding author: Unil Yun (yunei@sejong.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) through the Ministry of Education, Science, and Technology under Grant NRF 018R1D1A1A09083109.

ABSTRACT Traditional pattern mining is designed to handle binary database that assume all items in the database have same importance, there is a limitation to recognize accurate information from real-world databases using traditional method. To solve this problem, the high utility pattern mining approaches from non-binary database have been proposed and actively studied by many researchers. Lately, new data is progressively created with the passage of time in diverse area such as biometric data of a patient diagnosed in a medical device and log data of an internet user, and the volume of a database is gradually increasing. A database with these characteristics is called a dynamic database. Under these circumstances, high utility mining techniques suitable for analyzing dynamic databases have recently been extensively studied. In this paper, we propose a new list-based algorithm that mines high utility patterns considering the arrival time of each transaction in an incremental database environment. That is, our algorithm efficiently performs pattern pruning by using a damped window model that considers the importance of the previously inputted data lower than that of recently inserted data and identifies high utility patterns. Experimental results indicate that our proposed method has better performance than the state-of-the-art techniques in terms of runtime, memory, and scalability.

INDEX TERMS Data mining, damped window model, pattern pruning, high utility patterns, stream data mining.

I. INTRODUCTION

As the size of data increases, data mining techniques, one of fields of pattern recognition, that analyze data from massive information and recognize meaningful information are getting attention as an interesting research issue in data analysis. Pattern mining [1], [2], one of various data mining method, is to find useful pattern information in the database. The fields of application of pattern mining are uncertain pattern mining [3], erasable pattern mining [4], high utility pattern mining [5], top-k pattern mining [6], high average utility pattern mining [7], and sequential pattern mining [8]. Among these techniques, high utility pattern mining method has been actively studied since it analyzes non-binary databases by considering the importance of each item in the mining process and extracts more useful patterns from the database generated in the real world. Recently, since new data continuously accumulates over time in diverse areas such as log data generated on the Internet and sensor data sensed by an IoT-based device, the volume of the database increases gradually. Since traditional mining techniques cannot handle aforementioned database called dynamic database or stream database, many new techniques for dealing with these databases have been proposed. The proposed methods include sliding window model [9] and damped window model [10]. Among these techniques, the damped window model is also called a time decaying model. It assigns different weights to each data according to the arrival time of corresponding data and finds meaningful information from the data. In recent years, it is an important issue to identify meaningful information in

The associate editor coordinating the review of this manuscript and approving it for publication was Jerry Chun-Wei Lin.

an incremental environment where the latest data considered more important than previous data. In this environment, since the data comes in real time, it is necessary to scan the newly inserted data only once, reflect it in the existing structure, and quickly transmit the extracted results to the user. However, most of the existing utility pattern mining techniques [11] are suitable for processing static databases or dynamic databases where data arrival times are not considered. Recently, GENHUI [12], which finds the latest meaningful patterns while gradually decreasing the importance of old transactions, was proposed. It generates many candidate patterns during the mining process and requires operations that identifies actual patterns from them, so there is a limitation in processing real-time data. Motivated by the problems, we propose an efficient algorithm named Damped High Utility Pattern mining based List (DHUPL) for mining high utility patterns without generating the candidate patterns using the list structure in which data reflecting the difference of importance according to the arrival time of data is stored.

The main contributions of our algorithm are as follows: (1) A novel list-based structure is suggested to store and process data more efficiently. As far as we know, this data structure is more efficient than the data structure used in the previously proposed algorithms. (2) An efficient mining technique is proposed that uses the damped window model to recognize the latest significant high utility patterns considering time decaying factor and is useful for processing the time-sensitive database. (3) Various experiments have been performed using diverse real datasets and synthetic datasets to compare our algorithms with previously proposed algorithms and compare. Results show that our approach guarantees better performance that competitors in terms of memory usage, execution time, and scalability.

To the best of our knowledge, ours is the first work to propose list-based high utility pattern mining that processes stream database using time decaying model. The remainder of this paper is organized as follows. Related studies are described in Section II. In Section III, several preliminaries and our proposed algorithm are introduced in detail. In Section IV, experimental results are provided that compare our technique with state-of-the-art algorithms using various real-world datasets and synthetic datasets. Finally, conclusions of this paper are presented in Section V.

II. RELATED WORK

A. HIGH UTILITY PATTERN MINING FROM STATIC DATABASES

Utility pattern mining has been proposed to solve the problem of frequent pattern mining that does not consider the characteristics of each item in the database. To satisfy the anti-monotone property in the high utility pattern mining, Transactions Weighted Utilization (TWU) obtained by estimating the maximum value of a utility of a pattern is widely used [5]. FUM and DCG+ [13] have been proposed, but these algorithms have poor performance because they use generation and test method to scan the database multiple times during the mining process. UP-Growth and UP-Growth+ [14] applied a unique pruning technique to reduce the number of candidate patterns by decreasing the TWU value in the tree structure. In these methods, up to three scans are required for the entire operation. In order to solve these performance limitations, one phase-based algorithms using the list structure have also been proposed. HUI-Miner [15] manages information for each pattern in a vertical data structure called the utility list. It recursively performs mining operations using utility lists and does not generate candidate patterns in this process. d^2HUP [16] discovers useful patterns without generating candidates using a linear data structure. It enumerates less patterns than HUI-Miner, so it is more efficient than HUI-Miner. HUP-Miner [17] is an algorithm that improves the performance of HUI-Miner by joining efficiently utility lists by using two pruning techniques. IMHUP [18] uses an indexed-list data structure, so it also finds meaningful patterns more efficiently than HUI-Miner. FDHUP [19] stores information in the unique data structures called EI-table and FU-table and mines discriminative high utility patterns using them. HUI-PR [20] extracts fewer candidate patterns than the d^2HUP by using new upper bound values that are more stringent than those used in the field of utility pattern mining and strict sub-tree utility pruning strategy. This technique uses a multiple threads framework to process big data efficiently. Moreover, recently, various algorithms applying or extending utility pattern mining have also been studied. dHAUIM [21] takes into account the length of the pattern and finds high average utility patterns. In traditional utility mining, because the resulting HUIs set is too large and requires a lot of computational resources, HMiner-Closed [22], a high utility closed pattern mining technique for finding small sets representing HUIs, was proposed. HUOPM [23] considers frequency, utility, and occupancy and discovers high utility occupancy patterns. However, the various algorithms described above are suitable for handling only static databases.

B. HIGH UTILITY PATTERN MINING FROM DYNAMIC DATABASES

Because most of the real-world databases are dynamic databases, various techniques [11] for extracting high utility patterns from these databases have been actively studied. The initial algorithms IUM and FIUM [24] mine high utility patterns from an incremental database by using generation and test method. Later, the proposed IHUP [25] builds three global trees and performs recursive mining operations while constructing a conditional tree using the global trees. Its main drawback is that it still generates many candidate patterns and uses a large amount of memory to maintain the tree structure. Since then, diverse one phase-based methods that do not generate candidate patterns have been proposed. EIHI [26] uses a list-based data structure and a trie-based data structure called HUI-trie to manage information about transaction. It requires two database scans. LIHUP [27] uses a list-based utility list data structure and scans the database only once to build and update global data structures. Id^{2}HUP+ [28]

is also one-phase method and uses various pruning strategies. Recently, two tree-based techniques have been proposed to find meaningful patterns using pre-large concept, which classifies patterns into three types, to efficiently process dynamic databases. PIHUP [29] extracts high utility patterns from an incremental database, and HUIPRED [30] finds high utility patterns from a dynamic database where transactions are deleted over time. The aforementioned techniques are only suitable for processing dynamic databases that do not consider the arrival time of data.

C. TIME DECAYING MODEL

One of the methods for processing time-sensitive stream database is to use a time decaying model or damped window model that differentiates the importance of data according to the arrival time of data. In this model, a value called decaying factor given by the user is used. It is a value between 0 and 1 and denoted as f. If the utility value or the frequent value of the transaction is multiplied by the decaying factor, this value is decreased. Then, the difference in importance between old and newly inserted transactions becomes larger, and the importance of new data becomes relatively high with the passage of time. As a result, the user can obtain the recent valid patterns from time-sensitive stream database. At this time, the flow time of one transaction can be obtained by the difference between the TID value of the corresponding transaction and that of the last inserted transaction. Then, the utility value of the corresponding transaction is multiplied by the decaying factor by this difference. The GENHUI [12] for mining high utility patterns by applying a time decaying model has been proposed previously. It uses a tree structure called RHUI-tree, so it is inefficient for processing stream database. The reason is that it requires a large amount of memory to manage the data structure and generates many candidate patterns to find the actual patterns. To overcome these limitations, we propose list-based algorithm for analyzing stream databases that do not generate candidate patterns during the mining process.

III. LIST BASED INCREMENTAL HIGH UTILITY PATTERN MINING USING DAMPED WINDOW MODEL

In this section, we describe the preliminary knowledge used in our method. Moreover, we explain the overall procedures of our technique sequentially.

A. PRELIMINARIES

In the utility pattern mining, a non-binary database $D = \{T_1, T_2, \ldots, T_n\}$ is used. It consists of n non-binary transactions and includes m distinct items $I = \{i_1, i_2, \ldots, i_m\}$. Each transaction has a unique identifier called a TID and consists of a set of items of the form '$i_p : iu(i_p, T_d)$'. In this notation, $iu(i_p, T_d)$ means the frequency of the i_p in the T_d and is called the internal utility. $(1 \leq p \leq m)eu(i_p)$ means the relatively importance of the i_p and is called the external utility. Fig. 1 shows the example of transaction database and utility table of items. The example database consists of eight non-binary transactions and contains seven distinct items. As shown in the figure, a TID of 1-4 is the original database, a TID of 5-6 is the first incremental database DB_1+, and a TID of 7-8 is the second incremental database DB_2+.

	TID	Transaction	TU
Original DB	1	A : 2, E : 1, F : 1	10
	2	A : 2, C : 2, D : 1, E : 2	22
	3	B : 1, D : 1, F : 2	11
	4	C : 1, D : 3, F : 2	18
DB_1+	5	B : 2, C : 3, D : 1, E : 2	28
	6	D : 2, E : 1, F : 1	8
DB_2+	7	A : 1, E : 2, G : 1	9
	8	A : 2, B : 2, C : 1	18

Item	A	B	C	D	E	F	G
Utility	3	3	6	2	1	3	4

FIGURE 1. Example of transaction database and utility table.

The following are preliminaries widely used in utility pattern mining. When an item i_p, a pattern P, a transaction T_d, database D, and a threshold δ is given, the utility of P in T_d is computed by $u(p, T_d) = \sum u(i_p, T_d) = \sum eu(i_p) \times iu(i_p, T_d)$, where $i_p \in P$ and $P \subseteq T_d$. In addition, the utility of T_d is calculated as $tu(T_d) = \sum u(i_p T_d)$, where $i_p \in T_d$, and the utility of D is calculated as $u(D) = \sum tu(T_d)$, where $T_d \in D$. At this time, minimum utility threshold is computed by $minutil = u(D) \times \delta$. If the $u(P) \geq minutil$, P is high utility pattern.

Example 1: Let's calculate the following values for the original database D in Fig.1. For example, $u(A, T_1) = eu(A) \times iu(A, T_1) = 3 \times 2 = 6$. Since the pattern $\{AE\}$ is composed of item A and E, $u(AE, T_1) = u(A, T_1) + u(E, T_1) = 6 + 1 = 7$. Likewise, $\{AE\}$ is included in T_1 and T_2, so $u(AE) = u(AE, T_1) + u(AE, T_2) = 7 + 8 = 15$. Because T_1 is composed of items A, E, and F, $tu(T_1) = u(A, T_1) + u(E, T_1) + u(F, T_1) = 6 + 1 + 3 = 10$. Thus, $u(D) = tu(T_1) + tu(T_2) + tu(T_3) + tu(T_4) = 61$. If the δ is 15%, $minutil = 61 \times 0.15 = 9.15$. Since $u(AE) \geq minutil$, $\{AE\}$ is high utility pattern.

In this manuscript, we suggest a method to extract high utility patterns from time-sensitive stream data by applying damped window model. Therefore, the following definitions are additionally required for this purpose. In all definitions, let t be the arrival-time of T_d, T be the current time and f be a user-defined decaying factor.

Definition 1: Damped utility of a P in a D is denoted as $du(P)$ and computed by $du(P) = \sum u(P, T_d) \times f^{T-t}$, where $P \subseteq T_d$ and $T_d \in D$.

Definition 2: Damped transaction weighted utility of a P in a D is expressed as $dtwu(P)$ and calculated as $dtwu(P) = \sum tu(T_d) \times f^{T-t}$, where $P \subseteq T_d$ and $T_d \in D$.

Example 2: When all the transactions in original database D of Fig. 1 are inserted, let's compute the following values for $\{AE\}$. In this case, as the value of the decaying factor is decreased, the importance of the old transaction is greatly reduced. Therefore, in this example we introduce the results

of two decaying factors f_1 and f_2. Assume that f_1 is set to 0.9 and f_2 is set to 0.5. Since the last transaction in the original database is T_4, the current time T is 4. T_1 and T_2 contain pattern *{AE}*, each transaction arrives at time 1 and 2, respectively. Therefore, when the f_1 is used, $du(AE) = u(AE, T_1) \times f_1^{4-1} + u(AE, T_2) \times f_1^{4-2} = 7 \times 0.9^{4-1} + 8 \times 0.9^{4-2} = 5.103 + 6.48 = 11.583$. $dtwu(AE) = tu(T_1) \times 0.9^{4-1} + tu(T_2) \times 0.9^{4-2} = 7.29 + 16.038 = 23.328$. In the same way, when the f_2 is used, $du(AE) = u(AE, T_1) \times f_2^{4-1} + u(AE, T_2) \times f_2^{4-2} = 7 \times 0.5^{4-1} + 8 \times 0.5^{4-2} = 0.875 + 2 = 2.875$. $dtwu(AE) = tu(T_1) \times 0.5^{4-1} + tu(T_2) \times 0.5^{4-2} = 1.25 + 5.5 = 6.75$. Therefore, when the *minutil* is 9.15, if the decaying factor is 0.9, *{AE}* is a high utility pattern ($du(AE) = 11.583 > minutil$), but if the decaying factor is 0.5, it is not a high utility pattern ($du(AE) = 2.875 < minutil$).

Definition 3: When a set of all the items after P in T_d is expressed as $T_d \backslash P$, the remaining utility of a P in T_d is denoted as $ru(P, T_d)$ and calculated as $ru(P, T_d) = \sum du(i_p, T_d)$, where $i_p \in T_d \backslash P$.

Definition 4: Utility upper-bound of P in D is expressed as $ub(P)$ and computed by $ub(P) = du(P) + ru(P)$.

Example 3: Assume that transactions in D are sorted alphabetically. Then, the remaining item after *{AE}* in T_1 of Fig. 1 is F. Therefore, $ru(AE, T_1) = u(F, T_1) = 3$. Since there are no remaining items after *{AE}* in T_2, $ru(AE, T_2) = 0$. Thus, $ru(AE) = ru(AE, T_1) + ru(AE, T_2) = 3$. $ub(AE) = du(AE) + ru(AE) = 2.875 + 3 = 5.875$. In the process of expanding P, if the $ub(P)$ is greater than or equal to the *minutil*, P can be recognized as a prefix pattern. In the above case, $ub(AE) < minutil$, so no pattern extension for *{AE}* occurs.

B. CONSTRUCTING GLOBAL LISTS DATA STRUCTURE WITH ORIGINAL DATABASE

In this section, we explain the process of constructing a global lists structure when scanning a database once in the incremental database environment. The global lists constructed after scanning the database once consist of utility lists of items and the construction process as follows. First, each transaction in the original database is scanned once, and the following process is repeated for each item constituting the transaction. Assume that we process the i_p of T_d. If the *dtwu-table* does not contain the value of i_p, a new entry for i_p is created and $tu(T_d)$ is inserted as the $dtwu(i_p)$ of that entry. If the table already contains the value of i_p, the algorithm calculates the difference between the TID value of the last inserted i_p stored in the *dtwu-table* and the current TID value and multiplies the existing value by the calculated difference of the decaying factor. Then, add $tu(T_d)$ to the corresponding value. After the $dtwu(i_p)$ is updated, the $endTID(i_p)$ in the *dtwu-table* is changed to the current TID value. If the utility list of i_p is included in the global lists, the TID value of T_d, $u(i_p, T_d)$, and 0 as the *ru* value are inserted into the utility list. Otherwise, create a new utility list of i_p and insert $<d, u(i_p, T_d), 0>$ into the entry. At this time, the utility list of i_p is expressed as $UL(i_p)$. Once all the items in a transaction have been processed, the *dtwu* value of all the entries in the *dtwu-table* whose *endTID* value is not the current TID is multiplied by the decaying factor value. In addition, among the entries in the utility list existing in the global lists, all entries that are not inserted in the current transaction are multiplied by the decaying factor, thereby increasing the importance of recently inserted data. This process is repeated for all transactions.

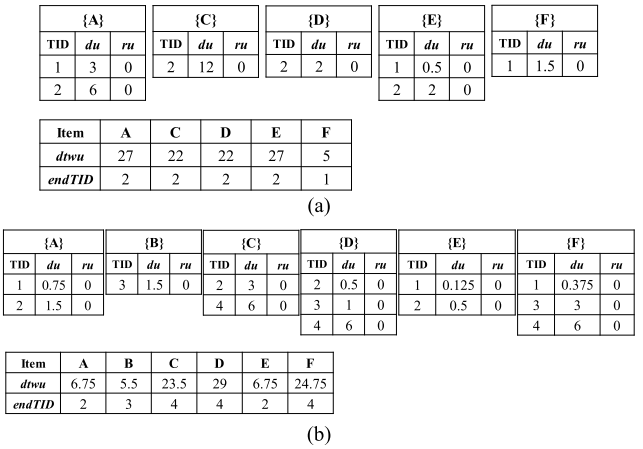


FIGURE 2. Process of original database in Fig. 1.: (a) After inserting T_1-T_2, (b) After inserting T_1-T_4.

Example 4: In this example, we show how to build global lists and *dtwu-table* using the original database shown in Fig. 1. The construction process is as follows. Since the transaction being scanned is the T_1, there is no entry in the *dtwu-table*, and there is no utility list in the global lists. T_1 consists of items A, E, and F. Create an entry for each item in the *dtwu-table* and store a value of $tu(T_1) = 10$ and a current TID value of 1. Then, a utility list is created for each item. In the $UL(A)$, the current TID value 1, $u(A, T_1) = 6$, and $ru = 0$ are inserted into the entry. <1, 1, 0> and <1, 3, 0> are inserted into the utility list of E and F in the same way as A. Then, T_2 is scanned. T_2 consists of items A, C, D, and E. First, process from A. There is an entry for A in the *dtwu-table*. Therefore, the previously stored value 10 is multiplied by decaying factor 0.5 and $tu(T_2) = 22$ is added to the value. Then, $dtwu(A) = 10 \times 0.5 + 22 = 27$ and the current TID value 2 is stored as the *endTID* value. Since there is a $UL(A)$ in the global lists, the current TID value 2, $u(A, T_2) = 6$, and $ru = 0$ are inserted into that list as the entry. We process the other items in the same way as above. After processing the last item of the transaction, the *dtwu-table* and global lists are checked once to ensure that the importance of the recent transaction is higher than the importance of the previously inserted transaction. Then, we can get the result of Fig. 2(a). Fig 2(b) describes the global list and *dtwu-table* after inserting original database in Fig. 1.

After all transactions in the database are inserted, if no more new transactions are inserted, the constructed global lists are reordered in *dtwu* ascending order. If the items have the same *dtwu* value, they are sorted in the inserted order. Suppose that the *dtwu-table* contains items of $\{i_1, i_2, \cdots, i_{m-1}, i_m\}$ and *dtwu* ascending order is 'i_1- i_2-$\cdots$- i_{m-1}- i_m'. Then, the global lists are rearranged in the order

'$UL(i_1)$-$UL(i_2)$-$\cdots$-$UL(i_{m-1})$-$UL(i_m)$'. After that, the *ru* values for each utility list are calculated. At this time, the calculation is carried out from the utility list of i_m, which is the item having the largest value of *dtwu*, and a temporary array is used for calculation efficiency. In this array, the TID value and the *SDU* value, which is the sum of the *du* values of each entry, are managed in pairs. In a transaction containing i_m, i_m is the last item, so there are no remaining items. Therefore, the *ru* value of all entries of $UL(i_m)$ is zero. Both the TID value and the *du* value of the entries existing in the $UL(i_m)$ are stored in the temporary array. Then, the $UL(i_{m-1})$ is processed. If there is an entry with a TID value existing in the $UL(i_{m-1})$ in the temporary array, the stored *du* value is stored as the *ru* value. Otherwise, the *ru* value is stored as zero. And then updates the value stored in the temporary array. If there is an entry having a TID value k existing in the $UL(i_{m-1})$ in the temporary array, the value $du(i_{m-1}, T_k)$ is added to the *SDU* value. If there is no entry with TID value k in the temporary array, create a new entry and store $du(i_{m-1}, T_k)$ in the *SDU* value. We can find the *ru* values by repeating the above calculation for all utility lists in the global lists. When the rebuilding of the global lists is completed, the temporary array is removed.

Example 5: In this example, we describe the process of the restructuring global lists of Fig. 2(b). Since the *dtwu* ascending order of items is 'B-A-E-C-F-D', the global lists are resorted in this order. Then, the *ru* values are computed from the *UL(D)*. Since the D is last item in all transactions, so, *ru(D, 2), ru(D, 3),* and *ru(D, 4)* all zero. Then, the <TID, *SDU*> pairs present in the *UL(D)* are stored in the temporary array. Thus, <2, 0.5>, <3, 1>, and <4, 6> are stored in the temporary array. After the processing of *UL(D),* the *UL(F)* is treated. The TID values included in the *UL(F)* are 1, 3, and 4. In the temporary array, the value when the TID value is 3 and 4 are stored, but the value when the TID value is 1 is not stored. Therefore , *ru(F, 1)* is 0, and *ru(F, 3)* and *ru(F, 4)* are the values 1 and 6 stored in the temporary array, respectively. After processing the *UL(F),* update the temporary array. Since there is no entry in the temporary array when the TID value is 1, add <1, *du(F, 1)*> in the array. Then $du(F, 3) = 3$ and $du(F, 4) = 6$ are added to the *SDU* values when the existing TID values are 3 and 4, respectively. Fig. 3(a) expresses the restructured *UL(F)* and temporary array. Fig. 3(b) indicates the status of the reconstructed global lists.

Lemma 1: In the damped high utility pattern mining, the *dtwu* value of the pattern is used to apply the anti-monotone property.

Proof: Suppose that n transactions in the database are inserted, and data structures are constructed. There are pattern $P = \{i_1, i_2, \ldots, i_m\}$ with m length and pattern $P' = \{i_1, i_2, \ldots, i_m, i_{m+1}\}$, which is supper pattern of P, with $m+1$ length. In the worst case, both P and P' are included in $T_1, T_2, \cdots, T_k (k \leq n)$. In the damped high utility pattern mining, the f value is used to decrease the importance of previously arrived transaction. Moreover, the anti-monotone property, which means that if any P is an invalid pattern, the supper pattern P' of P is also invalid pattern, must be satisfied in this method. Since P and P' are included in the same transactions, *dtwu* values of both patterns are the same as in the following results. $dtwu(P) = tu(T_1) \times f^{n-1} + tu(T_2) \times f^{n-2} + \cdots + tu(T_k) \times f^{n-k}$. $dtwu(P') = tu(T_1) \times f^{n-1} + tu(T_2) \times f^{n-2} + \cdots + tu(T_k) \times f^{n-k}$. However, in the general case, the set of transactions containing P' is a subset of the set of transactions containing P. Thus, $dtwu(P') \leq dtwu(P)$ is always established. If the P is an invalid pattern, $dtwu(P') \leq dtwu(P) \leq minutil$ is valid. As a result, using the *dtwu* value of pattern satisfies the anti-monotone property.

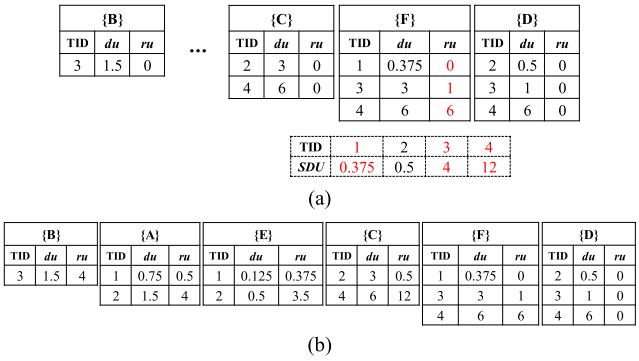


FIGURE 3. Restructuring process of global lists: (a) Restructured utility list of {F}, (b) Restructured global lists.

C. UPDATING GLOBAL LISTS WITH INCREMENTED DATABASE

If existing global lists are already established and the new data is additionally inputted, the algorithm must efficiently reflect it in the global lists. Thus, newly added data is also scanned once and inserted into the global lists and *dtwu-table*. If the newly added data does not exist in the global lists, the utility list for the corresponding data is created, respectively. Otherwise, the data is inserted in the existing data structures in the same way as described in the previous section, and the entire data structures are checked to multiply the previously inserted data by the decaying factor. This can make a difference in the importance of newly inserted data and previously inserted data. After newly added data are reflected in the data structures, the *dtwu* values of the items and the *dtwu* ascending order are changed. Thus, the global lists are rearranged with the updated *dtwu* ascending order. Then, *ru* values of all utility lists is initialized to zero, and new *ru* values are calculated using the same method as introduced in the previous section. After completing the above procedures for all the utility lists in the global lists, the reconstruction of the global lists is completed. Fig. 4 shows the restructured global lists after inserting transactions of DB_1+ and DB_2+ in Fig. 1.

D. MINING HIGH UTILITY PATTERNS WITH THE PROPOSED DATA STRUCTURE

When the construction of global lists and *dtwu-table* about the given incremental database is completed, the mining request

{F}		
TID	*du*	*ru*
1	0.0235	0.0547
3	0.1875	0.1563
4	0.375	0.75
6	0.75	1.25

{G}		
TID	*du*	*ru*
7	2	2.5

{D}		
TID	*du*	*ru*
2	0.0313	0.3126
3	0.0625	0.0938
4	0.375	0.375
5	0.25	3.25
6	1	0.25

{E}		
TID	*du*	*ru*
1	0.0078	0.0469
2	0.0313	0.2813
5	0.25	3
6	0.25	0
7	1	1.5

{B}		
TID	*du*	*ru*
3	0.0938	0
5	0.75	2.25
8	6	12

{A}		
TID	*du*	*ru*
1	0.0469	0
2	0.0938	0.1875
7	1.5	0
8	6	6

{C}		
TID	*du*	*ru*
2	0.1875	0
4	0.375	0
5	2.25	0
8	6	0

FIGURE 4. Updated and restructured global lists with DB_2+ in Fig. 1.

{FD}		
TID	*du*	*ru*
3	0.25	0.0938
4	0.75	0.375
6	1.75	0.25

{FE}		
TID	*du*	*ru*
1	0.0313	0.0469
6	1	0

{FB}		
TID	*du*	*ru*
3	0.2813	0

{FA}		
TID	*du*	*ru*
1	0.0704	0

{FC}		
TID	*du*	*ru*
4	0.75	0

(a)

{GE}		
TID	*du*	*ru*
7	3	1.5

{GA}		
TID	*du*	*ru*
7	3.5	0

(b)

FIGURE 5. Utility lists of candidate patterns with the length 2: (a) Created utility list for a prefix pattern {F}, (b) Created utility list for a prefix pattern {G}.

Algorithm 1 : ***DHUPL***

Initialize ***dT*** and ***GL*****;**
For each transaction, T_d, in db_{+1} or db_{+k}, **do**
  tid ← T_d.TID;
  For each item, i_k, in T_d, **do**
    If there is no entry for i_k in ***dT*****, then**
      Create a new entry, ***e***, into ***dT*** and **Initialize** ***e***.*dtwu*;
    e.*dtwu* ← ***e***.*dtwu*×$f^{tid - e.endTID}$ + $tu(T_d)$;
    e.*endTID* ← ***tid***;
    If there is no utility list for i_k in ***GL*****, then**
      Create a new utility list, $UL(i_k)$, in ***GL***;
    Else
      For each entry, ***k***, in $UL(i_k)$, **do** ***k***.*du* ← ***k***.*du* ×***f***;
    Create a new entry, ***e***, into $UL(i_k)$;
    Set ***e***.TID, ***e***.*du*, and ***e***.*ru* to the ***tid***, $u(i_k, T_d)$, and 0, respectively;
  End for
End for
For each entry, ***e***, in ***dT*****, do**
  If ***e.endTID*** is not ***tid*****, then** ***e***.*dtwu* ← ***e***.*dtwu*×$f^{tid - e.endTID}$;
End for
For each utility list, $UL(i_k)$, in ***GL*****, do**
  If the TID of the end entry of $UL(i_k)$ is not ***tid*****, then**
    endTID ← TID of the end entry of $UL(i_k)$;
    For each entry, ***k***, in $UL(i_k)$, **do** *k*.*du* ← *k*.*du* ×$f^{tid - endTID}$;
End for
Restructure the ***GL*** with the *dtwu* ascending order;
If there is a mining request from user, **then**
  Calculate a minimum utility ***minutil*** and **Initialize** ***HUP*** and ***Prefix***;
  Mine HUPs using ***GL, minutil, HUP,*** and ***Prefix***;
Return ***HUP***;

FIGURE 6. DHUPL Algorithm.

from the user is occurred. Then, the proposed method computes the *minutil* value by multiplying entire utility value of database by a user given threshold δ. And then, our algorithm recursively extracts valid high utility patterns by extending from the utility list of the item with the lowest *dtwu* value in the global lists. Assume that the utility lists are sorted in the order '$UL(i_1)$-$UL(i_2)$- $\cdots$-$UL(i_{m-1})$-$UL(i_m)$'. First, calculate the $du(i_1)$ and $ru(i_1)$ from $UL(i_1)$. The $du(i_1)$ and $ru(i_1)$ are sum of *du* values and *ru* values of entries in the $UL(i_1)$, respectively. If the $du(i_1)$ value is greater than or equal to *minutil* value, $\{i_1\}$ recognizes as a high utility pattern. The $ub(i_1)$ which is the value obtained by adding $du(i_1)$ and $ru(i_1)$, is greater than or equal to *minutil* value, so pattern expansion starts from $UL(i_1)$. Even though the *du* value of an item is less than the *minutil* value, if the *ub* value of the item is greater than or equal to *minutil*, the pattern expansion is performed with the corresponding item as a prefix. The pattern extension method performed with i_1 as a prefix is as follows. Combine '$UL(i_2)$-$\cdots$-$UL(i_{m-1})$-$UL(i_m)$' existing after $UL(i_1)$ with the $UL(i_1)$, respectively. When combining different utility lists, make sure that each utility has an entry with the same TID value. If an entry having the same TID value d exists in $UL(i_1)$ and $UL(i_2)$, $UL(i_1i_2)$ having an entry whose TID value is d is generated. The remaining values $du(i_1i_2, T_d)$ and $ru(i_1i_2, T_d)$ of this entry are obtained through the following equations: $du(i_1i_2, T_d) = du(i_1, T_d) + du(i_2, T_d)$. $ru(i_1i_2, T_d) = min(ru(i_1, T_d), ru(i_2, T_d))$. In this way, extended operations are performed on existing utility lists of the global lists. There is no utility list after $UL(i_m)$, which is the last utility list in the global lists. Thus, the extension of the utility list is conducted up to $UL(i_{m-1})$.

Example 6: In this example, we show the process of expanding the pattern using the restructured global lists in Fig. 6. Since the global lists is the result after inserting all transactions in Fig. 1, $u(D) = 124$. At this time, if the threshold δ inputted by user is 2%, $minutil = 124 \times 0.02 = 2.48$. The global lists of Fig. 7(b) are arranged in order of '*UL(F)-UL(G)-UL(D)-UL(E)-UL(B)-UL(A)-UL(C)*'. Therefore, the mining operation is performed from the *UL(F)* having the smallest *dtwu* value. $du(F) = 1.33$, which is smaller than *minutil*, so $\{F\}$ is not extracted as a high utility pattern. However, *ub(F)* is greater than *minutil* because *ru(F)* = 2.211 and *ub(F)* = *du(F)* + *ru(F)* = 3.547, so pattern expansion using $\{F\}$ as a prefix is performed. First, it can be confirmed that there is no common TID value in the *UL(G)* following the *UL(F)*. Therefore, the algorithm searches *UL(D)*, which is a next to *UL(G)*. There are common TID values 3, 4, and 6 in *UL(F)* and *UL(D)*. Thus, *UL(FD)* is generated, and *du(FD)* and *ru(FD)* values for each TID value are calculated and inserted into *UL(FD)*. As a result, <3, 0.25, 0.0938>, <4, 0.75, 0.375>, and <6, 1.75, 0.25> are inserted into the entries of *UL(FD)*. When the operation combining *UL(F)* and *UL(D)* is finished, operations that combine *UL(F)* and other lists after *UL(D)* are performed sequentially in the same manner as above. Figs. 5(a)-(b) show the utility lists of candidate patterns of length 2 with *{F}* and *{G}*as a prefix, respectively.

Lemma 2: In the pattern expansion step of the damped high utility pattern mining method implemented in list structure, the *ub* value of the pattern is used instead of the *dtwu* value of pattern for efficient pre-pruning of patterns.

Proof: In the traditional damped high utility pattern mining, if the *dtwu* value of pattern is greater than or equal to *minutil* value, the corresponding pattern is recognized as candidate pattern and the pattern expansion is performed using extracted candidate patterns. However, since the *dtwu* value of pattern is so overestimated value that the efficiency of the mining operation is not good because the algorithm extracts many invalid patterns during the mining process. To solve this problem, the $ub(P) = du(P) + ru(P)$ is used. In the equation, $ru(P)$ means the sum of the utility values of the items remaining after the pattern X in each sorted transaction. The *dtwu* value is the sum of the utility values of all items in a transaction. Therefore, $ub(P) \leq dtwu(P)$ is always established. When expanding a pattern in transaction sorted with *dtwu* value ascending order, the method does not consider items existing before pattern P. Only if there are items remaining after the pattern P, supper patterns P' of P are created by combining P with these items. Therefore, using the *ub* value of the pattern is more effective because it reduces unnecessary pattern expansion. Let P' be the supper pattern of P, $du(P') \leq ub(P) = du(P) + ru(P)$ is valid. Therefore, if $minutil \leq ub(P)$ is established, $minutil \leq du(P') \leq ub(P)$ may also be satisfied. Thus, before performing the extension operation with the pattern P as the prefix, the method compares whether the upper bound of P, $ub(P)$, is greater than or equal to the *minutil* value. If $minutil \leq ub(P)$, the pattern extension operation for P is performed.

E. ALGORITHM FOR MINING HIGH UTILITY PATTERNS FROM INCREMENTAL DATABASES

In this section, we explain the overall procedure of the proposed algorithm DHUPL using the pseudo-code of Fig. 6. The algorithm performs high utility pattern mining operation with the original and incremental databases, the utility information of the items, the minimum utility threshold, and decaying factor as input parameters. First, the algorithm initializes the *dtwu-table*, *dT* and the global lists, *GL*. Then, it scans each transaction of the original database db_{+1} or the newly inserted database db_{+k} in the set of databases *DB* only once. To check the TID value of the last transaction in processed database, store the TID value of the current transaction in *tid*. (lines 01-03) Then, for each item i_k in the transaction T_d, do the following: If there is no entry for i_k in the *dT*, a new entry $\boldsymbol{e}$ is created and inserted into the *dT* and the *dtwu* value of $\boldsymbol{e}$ is initialized. Then, it multiplies the *dtwu* value of the $\boldsymbol{e}$ by the decaying factor to the power difference between the current TID value and the last TID value of the entry and adds the utility value of the T_d to the value. The TID value of the current transaction is stored in the *endTID* value of the corresponding entry. (lines 04-08) If there is no utility list of i_k in the global-list, $UL(i_k)$ is created. Otherwise, the *du* values of entries present in $UL(i_k)$ are multiplied by the decaying factor. And then, in both cases, a new entry is created in $UL(i_k)$, and the TID value of the current transaction, the utility value of the item in the transaction, and 0 as *ru* value are stored in the entry. (lines 09-16) Once all transactions in the database are processed, the method checks the generated *dT* and *GL*. In the *dT*, it finds entries whose *endTID* value is not a *tid* and multiplies the *dtwu* value of the corresponding entry by the decaying factor to the power difference between the *tid* and the *endTID* value. (lines 17-19) In the *GL*, if the TID value of the last entry in the utility list is not the *tid*, the method multiplies the *du* value of all the entries in corresponding utility list the decaying factor to the power difference between the *tid* and the TID value of the last entry in the utility list. (lines 20-24) At the end of this process, the global lists are restructured by rearranging lists and calculating *ru* values according to the *dtwu* ascending order. (line 25) When a mining request is received from the user, *minutil* is calculated using the total utility value of the inserted database and the threshold value δ and the *HUP* to store the high utility pattern and the *Prefix* to store the prefix pattern are initialized. Then, it mines the high utility patterns recursively using *GL*, *minutil*, *HUP* and *Prefix*. (lines 26-28) When all operations are completed, the *HUP* is returned. (line 29)

TABLE 1. Features of the real datasets.

Dataset	Number of transactions	Number of items	Average length of transactions
Accidents	340,183	468	33.808
Mushroom	8,124	120	23
Chain-store	1,112,949	46,086	7.2
Retail	88,162	16,470	10.306

TABLE 2. Features of the real datasets.

Dataset	Number of transactions	Number of items	Average length of transactions
T10N10000L1000	100,000	10,000	10
T20N20000L2000	100,000	20,000	20
T30N30000L3000	100,000	30,000	30
T40N40000L4000	100,000	40,000	40

IV. PERFORMANCE EVALUATION

A. EXPERIMENTAL ENVIRONMENT AND DATASETS

In this section, we conduct various experiments to compare the proposed algorithm DHUPL with the state-of-the-art methods, GENHUI [12] and LIHUP [27] using diverse datasets shown in Tables 1 and 2. GENHUI is a tree-based technique for discovering high utility patterns by applying damped window techniques. It is used to compare the performance differences in data structure and pruning techniques with our algorithm. LIHUP is a recent list-based high utility pattern mining algorithm that does not consider the arrival time of transaction, so it is used to compare the effect of the damped window applied in our approach. All the algorithms

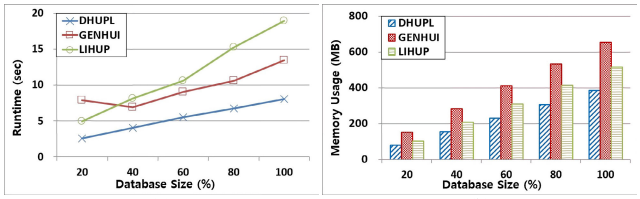


FIGURE 7. The results on accidents ($\delta = 30\%$): (a) Runtime test results, (b) Memory test results.

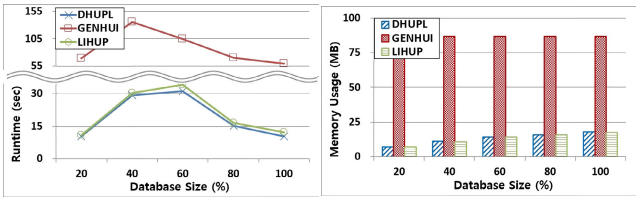


FIGURE 8. The results on mushroom ($\delta = 10\%$): (a) Runtime test results, (b) Memory test results.

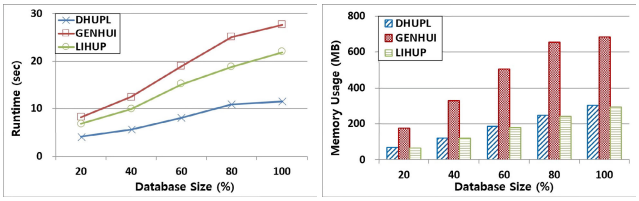


FIGURE 9. The results on Chain-store ($\delta = 0.25\%$): (a) Runtime test results, (b) Memory test results.

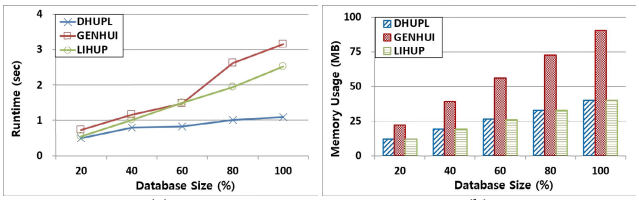


FIGURE 10. The results on retail ($\delta = 0.32\%$): (a) Runtime test results, (b) Memory test results.

are written in C++ language and executed on PC with an Intel Core i7-6700K@4.00GHz processor, 32.00GB of memory, and Windows 10 platform. Real datasets in Table 1 can be obtained from the FIMI repository and NU-MineBench 2.0, and synthetic datasets in Table 2 are available at the IBM data generator [12], [27]. The tables present the number of transactions, the number of distinct items, and the average length of transactions in each dataset. The four real datasets in Table 1 are used to evaluate runtime and memory usage of methods. These datasets are widely used for performance evaluation in the field of pattern mining. In Table 2, the synthetic datasets from T10N10000L1000 to T40N40000L4000 are called the $Tx_1Nx_2Lx_3$ group and used to test scalability of methods. In this group, the number of transactions is fixed at 100,000, while the number of items increases from 10,000 to 40,000. In this section, we perform experiments to mine high utility patterns, so each dataset needs external utility and internal utility information. Thus, we randomly generate internal utility values from 1 to 10 and external utility values from 0.01 to 1.00 for all datasets except for Chain-store. To perform tests in an environment where the new data is gradually accumulated, we divide all datasets evenly into five parts. In each database, the first part is considered the original database and the other four parts are regarded as additional databases. Moreover, we use a decaying factor of 0.99999 for all experiments.

B. EXPERIMENTAL RESULTS WITH INCREASING DATA SIZE

In this part, we show the results of the runtime and memory usage of three algorithms when processing 20%, 40%, 60%, 80%, and 100% of transactions in each database. Figs. 7(a) and (b) present experimental results of Accidents as the data size increases when the threshold is 30%. In these figures, we can confirm that all algorithms generally require more runtime and memory usage as the data size increases. In Fig. 7(a), when the size of database increases, DHUPL considering the arrival time of transactions guarantees the fastest runtime in every case. GENHUI performs worse than our algorithm because it uses tree structure and generates candidate patterns. So, it takes more time than DHUPL despite applying the damped window concept. LIHUP is closer, because of its list structure, but is still worse than ours because it extracts many patterns without considering the arrival time of the transactions. In Fig. 7(b), we can see that DHUPL has the best memory efficiency and GENHUI has the worst memory efficiency. The reason is that GENHUI generates a large amount of candidate patterns, so it requires a large amount of memory. In addition, LIHUP uses more memory than our approach because of the extraction of more invalid patterns than our algorithm. Figs. 8(a) and (b) present experimental results of Mushroom when the threshold is set 10%. Fig. 8(a) shows that all algorithms tend to decrease as the data size increases in some sections. This seems to be a feature of the Mushroom dataset. However, we can confirm that the runtime of DHUPL is the fastest in all cases. Fig. 8(b) presents that the memory usage of the DHUPL and LIHUP is about the same and they require less memory than GENHUI. Figs. 9-10 describe experimental results of Chain-store and Retail as the data size increases when the thresholds are 0.25% and 0.32%, respectively. In Figs. 9(a) and 10(a), DHUPL performs better than GENHUI and LIHUP because of its list-based data structure and pruning strategy. As the size of the database becomes larger, the runtime of DHUPL increases slowly, but the runtimes of GENHUI and LIHUP rise steeply. At this time, the runtime gaps of three algorithms become larger. In Figs. 9(b) and 10(b), DHUPL and LIHUP use almost the same amount of memory, and GENHUI among the three algorithms use the largest memory in every section. That is, memory efficiency of GENHUI is worst for the same reasons as aforementioned. From the experimental results, when the data size is increase, we can confirm that DHUPL guarantees the best runtime and memory efficiency.

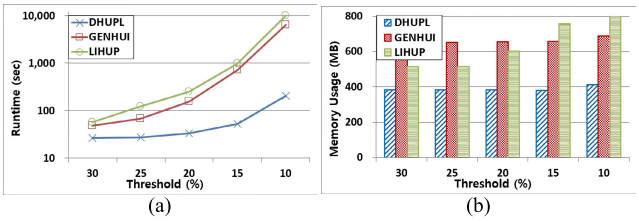


FIGURE 11. The results on accidents (δ is from 30% to 10%): (a) Runtime test results, (b) Memory test results.

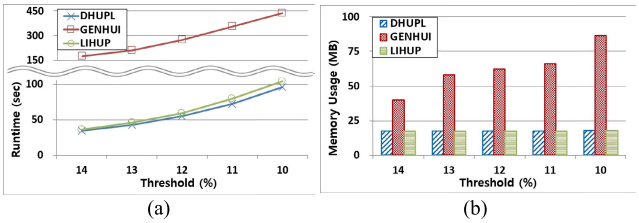


FIGURE 12. The results on mushroom (δ is from 14% to 10%): (a) Runtime test results, (b) Memory test results.

C. EXPERIMENTAL RESULTS WITH CHANGING THRESHOLD

In this section, we show the total runtime and memory usage results of three algorithms performing five high utility pattern mining operations for the gradually added real datasets while the threshold decreases. Figs. 11(a) and (b) present the result by reducing the threshold from 30% to 10% by 5% for the Accidents dataset. In Fig. 11(a), our DHUPL performs better than GENHUI and LIHUP because it is a list-based algorithm and considers arrival time of transactions, so it can be confirmed that the time required for data construction and operation is shorter than GENHUI and LIHUP. As shown in the figure, the execution time of our DHUPL rises gently, but the runtimes of GENHUI and LIHUP increase steeply. In Fig. 11(b), the memory usage of the three algorithms increases gradually as the threshold decreases. When the threshold was 30%, DHUPL, GENHUI, and LIHUP use about 384MB, 652MB, and 515MB, respectively. When the threshold was 10%, the memory usage of DHUPL, GENHUI, and LIHUP are about 414MB, 689MB, and 868MB, respectively. In this case, we can show that the memory efficiency of DHUPL is best and memory efficiency of LIHUP is getting worse. The reason is that as the threshold value decreases, all three algorithms extract more patterns, but LIHUP creates and manages more lists of invalid patterns than DHUPL. Figs. 12(a) and (b) present the results by reducing the threshold from 14% to 10% for the Mushroom dataset. In Fig. 12(a), GENHUI performs worse than our DHUPL because it manages tree data structures and create many candidate patterns. Mushroom is small and dense dataset. Due to the feature of dataset, the performance of LIHUP is not as good as DHUPL, but it does not show a very large gap. However, we can see that the runtime gap of the two algorithms increase as the threshold decreases. This is because the difference in the number

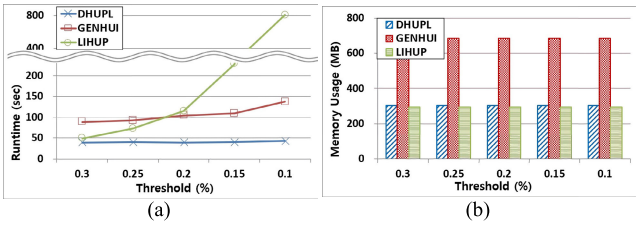


FIGURE 13. The results on chain-store (δ is from 0.3% to 0.1%): (a) Runtime test results, (b) Memory test results.

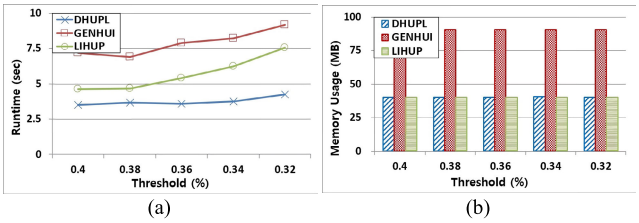
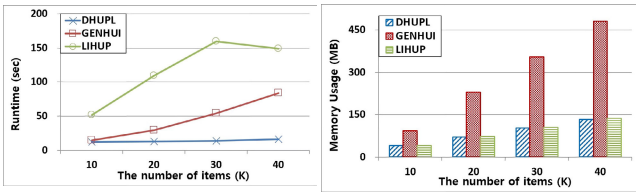


FIGURE 14. The results on retail (δ is from 0.4% to 0.32%): (a) Runtime test results, (b) Memory test results.

of pattern expansion operations between the two techniques becomes larger. In Fig. 12(b), as the threshold decreases, the memory usage of DHUPL and LIHUP is almost constant, but the memory usage of GENHUI increases sharply. Unlike the two algorithms, GENHUI has to construct a large number of local tree structures during the recursive mining operation, generate many candidate patterns, and verify actual patterns. So, it requires large memory space compared to DHUPL and LIHUP, which use simple list structures and do not generate candidate patterns. The Accidents and the Mushroom are dense datasets, but the Accidents are relatively large, so the overall memory usage results in Fig. 11(b) are higher than those of in Fig. 12(b). Figs. 13(a) and (b) describe the results by reducing the threshold from 0.3% to 0.1% by 0.05% for the Chain-store. Figs. 14(a) and (b) show the results of experiment for Retail, with 0.02% reduction of the threshold from 0.4% to 0.32%. In Figs. 13(a) and 14(a), DHUPL executes better than GENHUI and LIHUP in all sections because of aforementioned reasons. As the threshold decreases, the runtime of DHUPL changes little, but the runtime results of GENHUI and LIHUP increase relatively steeply compared to that of DHUPL. The Chain-store and the Retail are sparse datasets, but the Retail is relatively small dataset. Therefore, the overall memory usage results in Fig. 14(b) are lower than those of Fig. 13(b). In these figures, even if the threshold value decreases, the memory usage of the three algorithms is almost constant. This is because the number of patterns extracted from the measured thresholds is not large because of the characteristics of the databases. In addition, the memory usage of DHUPL and LIHUP is similar in all cases, and that of GENHUI is more than twice that of DHUPL. GENHUI uses a larger amount of memory than the list-based algorithms, DHUPL and LIHUP, because it is implemented as a tree structure that is costly to construct and maintain complicated data structures. In conclusion, we can determine that our approach can mine the high utility patterns most effectively



(a) (b)

FIGURE 15. The runtime test results on $Tx_1Nx_2Lx_3$ ($\delta = 0.4\%$): (a) Runtime test results, (b) Memory test results.

for all cases in an environment where the threshold value changes.

D. SCALABILITY TEST

We evaluate the scalability of the algorithms using the synthetic datasets shown in Table 2. Figs. 15(a) and (b) are the experimental results of the scalability for the synthetic dataset group $Tx_1Nx_2Lx_3$, and the threshold is set 0.4%. In Fig. 15(a), the runtime results of the three algorithms increase as the number of items increases in general. At this time, DHUPL performs better than other two methods due to its efficient data structures and pruning technique. To be more specific, DHUPL takes about 12 seconds when the number of items was 10K, but it takes about 16 seconds when the number of items was 40K. When 10K and 40K, the runtimes of GENHUI are about 15 seconds and about 84 seconds, respectively. LIHUP takes about 52 seconds and 149 seconds when 10K and 40K, respectively. The runtime of LIHUP is slightly less than 30K at 40K, but it shows the slowest runtime in all cases. Fig. 15(b) shows that the memory usage results of the three algorithms increase as the number of items in the dataset increases. The reason is that all algorithms have to store all information about distinct items. In the figure, the memory usage of DHUPL and LIHUP is almost the same because of their data structures and the number of extracted patterns. But, among the three algorithms, GENHUI consumes the most amount of memory. This is because GENHUI uses a tree structure and generates candidate patterns, while the other two algorithms use list structure and do not generate candidate patterns. As a result, we can confirm that DHUPL can discover the high utility patterns more effectively than state-of-the-art techniques in scalability test.

V. CONCLUSION

In this paper, we proposed a new efficient algorithm called DHUPL for discovering high utility patterns from an incremental non-binary database. Our technique can efficiently perform mining operations without generating candidate patterns by scanning incremental database once and managing them in list-based data structures. DHUPL also uses a decaying factor to handle the importance of data according to the arrival time of each data. As a result, significant results were obtained by increasing the importance of the most recently entered data and decreasing the importance of old data. From the experimental results, we confirmed that our proposed algorithm has better performance in terms of execution time, memory usage efficiency, and scalability than the existing techniques. The data structure proposed in this paper can be applied to various dynamic database pattern mining fields such as sliding window-based technique and other advanced pattern mining fields such as classifying and clustering. In the future work, the research that can improve the performance of the algorithm proposed in this paper or extend it from the results of the paper will be conducted.

REFERENCES

[1] A. Aggarwal and D. Toshniwal, "Frequent pattern mining on time and location aware air quality data," *IEEE Access*, vol. 7, pp. 98921–98933, 2019.

[2] J. C.-W. Lin, Y. Zhang, P. Fournier-Viger, and T.-P. Hong, "Efficiently updating the discovered multiple fuzzy frequent itemsets with transaction insertion," *Int. J. Fuzzy Syst.*, vol. 20, no. 8, pp. 2440–2457, Dec. 2018.

[3] W. Li, Y. Fan, W. Liu, M. Xin, H. Wang, and Q. Jin, "A self-adaptive process mining algorithm based on information entropy to deal with uncertain data," *IEEE Access*, vol. 7, pp. 131681–131691, 2019.

[4] H. Nam, U. Yun, E. Yoon, and J. C.-W. Lin, "Efficient approach for incremental weighted erasable pattern mining with list structure," *Expert Syst. Appl.*, vol. 143, no. 1, Apr. 2020, Art. no. 113087.

[5] W. Gan, C.-W. Lin, P. Fournier-Viger, H.-C. Chao, V. Tseng, and P. Yu, "A survey of utility-oriented pattern mining," *IEEE Trans. Knowl. Data Eng.*, to be published, doi: 10.1109/TKDE.2019.2942594.

[6] V. Tseng, C. Wu, P. Fournier-Viger, and P. S. Yu, "Efficient algorithms for mining top-K high utility itemsets," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 1, pp. 54–67, Jan. 2016.

[7] J. M.-T. Wu, J. C.-W. Lin, M. Pirouz, and P. Fournier-Viger, "TUB-HAUPM: Tighter upper bound for mining high average-utility patterns," *IEEE Access*, vol. 6, pp. 18655–18669, 2018.

[8] W. Gan, J. C.-W. Lin, P. Fournier-Viger, H.-C. Chao, and S. P. Yu, "A survey of parallel sequential pattern mining," *ACM Trans. Knowl. Discov. Data*, vol. 13, no. 3, pp. 25:1–25:34, Jun. 2019.

[9] S. Cai, S. Hao, R. Sun, and G. Wu, "Mining recent maximal frequent itemsets over data streams with sliding window," *Int. Arab J. Inf. Technol.*, vol. 16, no. 6, pp. 961–969, Nov. 2019.

[10] U. Yun, D. Kim, E. Yoon, and H. Fujita, "Damped window based high average utility pattern mining over data streams," *Knowl.-Based Syst.*, vol. 144, pp. 188–205, Mar. 2018.

[11] W. Gan, J. C.-W. Lin, P. Fournier-Viger, H.-C. Chao, T.-P. Hong, and H. Fujita, "A survey of incremental high-utility itemset mining," *Wiley Interdiscipl. Rev., Data Mining Knowl. Discovery*, vol. 8, no. 2, 2018, Art. no. e1242.

[12] D. Kim and U. Yun, "Mining high utility itemsets based on the time decaying model," *Intell. Data Anal.*, vol. 20, no. 5, pp. 1157–1180, Sep. 2016.

[13] Y.-C. Li, J.-S. Yeh, and C.-C. Chang, "Isolated items discarding strategy for discovering high utility itemsets," *Data Knowl. Eng.*, vol. 64, no. 1, pp. 198–217, Jan. 2008.

[14] V. S. Tseng, B.-E. Shie, C.-W. Wu, and P. S. Yu, "Efficient algorithms for mining high utility itemsets from transactional databases," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 8, pp. 1772–1786, Aug. 2013.

[15] M. Liu and J. Qu, "Mining high utility itemsets without candidate generation," in *Proc. 21st ACM Int. Conf. Inf. Knowl. Manage. (CIKM)*, 2012, pp. 55–64.

[16] J. Liu, K. Wang, and B. C. M. Fung, "Mining high utility patterns in one phase without generating candidates," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 5, pp. 1245–1257, May 2016.

[17] S. Krishnamoorthy, "Pruning strategies for mining high utility itemsets," *Expert Syst. Appl.*, vol. 42, no. 5, pp. 2371–2381, Apr. 2015.

[18] H. Ryang and U. Yun, "Indexed list-based high utility pattern mining with utility upper-bound reduction and pattern combination techniques," *Knowl. Inf. Syst.*, vol. 51, no. 2, pp. 627–659, May 2017.

[19] J. C.-W. Lin, W. Gan, P. Fournier-Viger, T.-P. Hong, and H.-C. Chao, "FDHUP: Fast algorithm for mining discriminative high utility patterns," *Knowl. Inf. Syst.*, vol. 51, no. 3, pp. 873–909, Jun. 2017.

[20] J. M.-T. Wu, J. C.-W. Lin, and A. Tamrakar, "High-utility itemset mining with effective pruning strategies," *ACM Trans. Knowl. Discovery from Data*, vol. 13, no. 6, pp. 1–22, Nov. 2019.
[21] T. Truong, H. Duong, B. Le, and P. Fournier-Viger, "Efficient vertical mining of high average-utility itemsets based on novel upper-bounds," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 2, pp. 301–314, Feb. 2019.
[22] L. T. T. Nguyen, V. V. Vu, M. T. H. Lam, T. T. M. Duong, L. T. Manh, T. T. T. Nguyen, B. Vo, and H. Fujita, "An efficient method for mining high utility closed itemsets," *Inf. Sci.*, vol. 495, pp. 78–99, 2019.
[23] W. Gan, J. C.-W. Lin, P. Fournier-Viger, H.-C. Chao, and P. S. Yu, "HUOPM: High-utility occupancy pattern mining," *IEEE Trans. Cybern.*, vol. 50, no. 3, pp. 1195–1208, Mar. 2020, doi: 10.1109/TCYB.2019.2896267.
[24] J.-S. Yeh, C.-Y. Chang, and Y.-T. Wang, "Efficient algorithms for incremental utility mining," in *Proc. 2nd Int. Conf. Ubiquitous Inf. Manage. Commun. (ICUIMC)*, 2008, pp. 212–217.
[25] C. F. Ahmed, S. K. Tanbeer, B.-S. Jeong, and Y.-K. Lee, "Efficient tree structures for high utility pattern mining in incremental databases," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 12, pp. 1708–1721, Dec. 2009.
[26] P. Fournier-Viger, J. C.-W. Lin, T. Gueniche, and P. Barhate, "Efficient incremental high utility itemset mining," in *Proc. 5th Int. Conf. ASE Big Data Social Inform.*, Oct. 2015, Art. no. 53.
[27] U. Yun, H. Ryang, G. Lee, and H. Fujita, "An efficient algorithm for mining high utility patterns from incremental databases with one database scan," *Knowl.-Based Syst.*, vol. 124, pp. 188–206, May 2017.
[28] J. Liu, X. Ju, X. Zhang, B. C. M. Fung, X. Yang, and C. Yu, "Incremental mining of high utility patterns in one phase by absence and legacy-based pruning," *IEEE Access*, vol. 7, pp. 74168–74180, 2019.
[29] J. Lee, U. Yun, G. Lee, and E. Yoon, "Efficient incremental high utility pattern mining based on pre-large concept," *Eng. Appl. Artif. Intell.*, vol. 72, pp. 111–123, Jun. 2018.
[30] U. Yun, H. Nam, J. Kim, H. Kim, Y. Baek, J. Lee, E. Yoon, T. Truong, B. Vo, and W. Pedrycz, "Efficient transaction deleting approach of pre-large based high utility pattern mining in dynamic databases," *Future Gener. Comput. Syst.*, vol. 103, pp. 58–78, Feb. 2020.

HYOJU NAM received the B.S. degree in computer engineering from Sejong University, Seoul, South Korea, in 2018, where she is currently pursuing the M.S. degree. Her research interests include data mining, information retrieval, database systems, and artificial intelligence.

UNIL YUN received the M.S. degree in computer science and engineering from Korea University, Seoul, South Korea, in 1997, and the Ph.D. degree in computer science from Texas A&M University, College Station, TX, USA, in 2005.

He was with the Multimedia Laboratory, Korea Telecom, from 1997 to 2002. After receiving the Ph.D. degree, he was a Postdoctoral Associate for almost one year with the Computer Science Department, Texas A&M University. After that, he was a Senior Researcher with the Electronics and Telecommunications Research Institute. In March 2007, he joined the School of Electrical and Computer Engineering, Chungbuk National University, South Korea. Since August 2013, he has been with the Department of Computer Engineering, Sejong University, Seoul. His research interests include data mining, information retrieval, database systems, artificial intelligence, and digital libraries.

BAY VO (Member, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees in computer science from the University of Science, Vietnam National University, Ho Chi Minh City, Vietnam, in 2002, 2005, and 2011, respectively. He is currently an Associate Professor and the Dean of the Faculty of Information Technology, Ho Chi Minh City University of Technology, Vietnam. His research interests include association rules, classification, mining in incremental database, distributed databases, and privacy preserving in data mining.

TIN TRUONG received the B.S. degree in mathematics from the University of Dalat, in 1983, and the Ph.D. degree in stochastic optimal control from Vietnam National University, VNU-Hanoi, Vietnam, in 1990. He is currently a Researcher with the Division of Computational Mathematics and Engineering, Institute for Computational Science, and the Faculty of Mathematics and Statistics, Ton Duc Thang University, Ho Chi Minh City, Vietnam. His current research interests include artificial intelligence and data mining.

ZHI-HONG DENG received the M.Sc. and Ph.D. degrees in computer science from Peking University, Beijing, China, in 2000 and 2003, respectively. He is currently a Professor with the Department of Machine Intelligence, Peking University. He has published more than 80 articles in refereed journals and international conferences. His research interests include machine learning, natural language processing, and pattern mining.

EUNCHUL YOON (Senior Member, IEEE) received the B.S. and M.S. degrees in electronic engineering from Yonsei University, Seoul, South Korea, in 1993 and 1995, respectively, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 2005. From February 1995 to July 2000, he was with the Samsung CDMA System Development Team, Seoul, where he was involved in the design of system software. From December 2005 to February 2008, he was with the Samsung WiMAX System Development Group, Suwon, South Korea, where he developed modem algorithms for beamforming and MIMO. He has been with the Department of Electronic Engineering, Konkuk University, Seoul, since March 2008. His research interests include wireless networks, coding and modulation, interference cancellation, and time reversal.