An Efficient Algorithm for Mining Top-k High On-shelf Utility Itemsets with Positive/Negative Profits of the Local/Global Minimum Count

1st Ye-In Chang
dept. of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan
changyi@mail.cse.nsysu.edu.tw

2nd Po-Chun Chuang
dept. of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan
zhungboqun@gmail.com

3rd Yu-Hao Liao
dept. of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan
karta88821@gmail.com

4th Po-Yu Hu
dept. of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan
HuPY@db.cse.nsysu.edu.tw

5th Ting-Wei Chen
dept. of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan
ChenTW@db.cse.nsysu.edu.tw

***Abstract*—High Utility Itemset Mining (HUIM) utilizes the threshold value to extract the High Utility Itemset (HUI) from the transactional database. However, it is hard to define a suitable threshold value, since it depends on the domain knowledge of the application. Therefore, Top-k High Utility Itemset Mining (Top-k HUIM) solves the problem of setting a suitable threshold. The user can define a k value, which represents the number of HUIs. Moreover, there may exist some itemsets occurring at a specific time interval, which have a chance to become HUI. Since the traditional HUIM algorithm does not consider the transaction with the time interval, we can not apply the HUIM algorithm directly. Thus, High On-shelf Utility Itemset Mining (HOUIM) addresses the above problem. The proportion of the utility value of the item in all of the time intervals, when the itemset appears is used for determining wether the itemset is a High On-shelf Utility Itemset (HOUI) or not. In the Top-k High On-shelf Utility Itemset Mining (Top-k HOUIM), the KOSHU algorithm utilizes the utility list based data structure and ignores the item with the negative profit in the process of overestimating the utility of the itemset. Thus, the KOSHU algorithm needs the less processing time. However, the KOSHU algorithm has to scan the database two times and sort the database one time. Therefore, in this paper, we propose an efficient algorithm based on the TIPN Table to mines Top-k HOUIs. Our proposed data structures include TIPN Table, MINC Table, IO Bitmap and TIUL. In the TIPN Table, we record positive items, positive utilities, negative items and negative counts. MINC Table is used for storing the local/global counts of all of the items with negative profits. In our algorithm, we only need to scan the database once. From our performance study, we have shown that our proposed algorithm is more efficient than the KOSHU algorithm.**

***Index Terms*—Data Mining, High On-shelf Utility Itemset Mining, Negative Unit Profits, Static Transactional Database, Top-k High Utility Itemset Mining**

I. Introduction

The *Frequent Weighted Itemsets Mining* (FWIM) [10] considers the frequency and the weight of the item. The *High Utility Itemsets Mining* (HUIM) [7], [8] has become highly popular in the recent years. If the utility value of itemset X is not less than the minimum threshold value, then itemset X is a High Utility Itemsets (HUIs). However, it is hard to define an appropriate threshold value. To address the above issue, the *Top-k High Utility Pattern Mining* (Top-k HUIM) [8] has been proposed. The default threshold value is set to 0, and then applied several threshold raising strategies to increase the threshold value, such as RIU [8], RUC [8], *etc.*

The traditional HUIM algorithm considers all of the transactions in the database are on-shelf. The *High On-shelf Utility Itemset Mining* (HOUIM) [3] has been proposed by considering the on-shelf time interval of the transaction, and it can find itemsets having the high utility in the specific time interval. All of the transactions are added with another field called the *time interval*, which indicates the time interval that the transaction

This research was supported in part by the Office of Research and Development, National Sun Yat-sen University under Grant No. 14DS02

is on-shelf as shown in Table I. The related profit of item a, b, c, and d is 2, 4, 1 and 3, respectively.

TABLE I
THE EXAMPLE DATABASE D_1 WITH ON-SHELF TIME INTERVALS

TID	*Transaction*	*Time Interval*
T_1	(a, 1)(b, 2)(c, 1)	1
T_2	(c, 3)(d, 2)	1
T_3	(b, 2)(c, 4)(d, 3)	2
T_4	(b, 6)(d, 2)	2

The total utility of time interval h is denoted as $TIU(h)$, which is calculated as the sum of the utility of transactions that the time interval is equal to h. Note that the utility of itemsets in a certain time interval is equal to the summation of the utility of all of itemsets in that certain interval. The result of the total utility of time interval 1 is 20 and that of time interval 2 is 51. The *Relative Utility* of the given itemset X is used for determining whether the itemset is a HOUI.

To achieve the above goal, Singh *et al.* have proposed the TKEH algorithm [8] for mining the Top-k HUIs. Srikumar Krishnamoorthy has proposed the THUI algorithm [6] to mine the Top-k HUIs. Ashraf *et al.* have proposed the TKN algorithm [2]. Later, *High On-shelf Utility Itemsets Mining* (HOUIM) [4] considers the time interval. The *Redefined transaction Weighted Utility* (RTWU) [9] is an utility overestimated value which ignores items with negative profits. The KOSHU algorithm [4] is based on the utility list structure to mine *Top-k High On-shelf Utility Itemsets* (Top-k HOUIs). The KOSHU algorithm try to reduce the number of candidates during the mining process. However, the KOSHU algorithm needs to scan the database twice.

To reduce the number of database scans in the static database, in this paper, we propose an efficient algorithm called *TIPN-Table-based* to extract Top-k HOUIs. We apply the local and the global concepts to deal with items with the negative profit. We also propose the *IO_Bitmap* to record the occurrence status of all of the items according to different time intervals. We propose two pruning strategies. From our performance study, we show that our TIPN-Table-based algorithm is more efficient than the KOSHU algorithm.

II. A SURVEY OF THE KOSH ALGORITHM

The KOSHU algorithm [4] has been proposed to mine the Top-k High On-shelf Utility Itemsets (Top-k HOUIs). The KOSHU algorithm calculates RTWU values for all of the items and sorts them in the defending order. Then, they apply two threshold increased strategies to increase the threshold value. Moreover, they propose a pruning strategy to effectively diminishes the processing time. However, the KOSHU algorithm requires to iterate the database twice to mine the Top-k HOUIs. Moreover, the utility list for each item has been constructed by the KOSHU algorithm.

III. THE TIPN-TABLE-BASED ALGORITHM

In this section, we use an example database to illustrate our algorithm. Next, we describe some variables, four data structures, three pruning strategies, two strategies for increasing the threshold value quickly, and our proposed algorithm in details.

A. An Example Database

We use an example database D_2 as shown in Figure 1 to illustrate our algorithm. Moreover, each item in the set I has its own profit = [a:5, b:-2, c:3, d:-1, e:3, f:4]. Each transaction T_j in database D_2 contains an unique identifier TID. Moreover, in each transaction T_j, we have a subset of items I with the related count and a related time interval.

TID	Transaction	Time Interval
T_1	(b, 3)(d, 2)(f, 4)	1
T_2	(b, 2)(c, 7)(d, 4)(e, 5)	1
T_3	(a, 6)(c, 3)(d, 4)	2
T_4	(a, 4)(d, 2)(e, 2)	2
T_5	(b, 3)(c, 8)(d, 5)(e, 4)	3
T_6	(b, 1)(c, 6)(d, 3)	3

Fig. 1. An example database D_2 with on-shelf time intervals

B. Variables

$P\_Val(i)$ and $Q\_Val(i)$ represents the Profit and the Quantity value of item i, respectively. Moreover, $UT(i, T_j)$ is defined as the Utility of item i in Transaction T_j , which can be calculated as the product of $P\_Val(i)$ and $Q\_Val(i)$. $UT(X, T_j)$ represents the utility of itemset X in transaction T_j, which is calculated as the cumulative utility of all itemset X within transaction T_j . $TotalU(T)$ represents the Total Utility of transaction T, which is the sum of the utility of each item in transaction T. The total utility of each transaction in database D_2 is 8, 23, 35, 22, 21, and 13, for transactions T_1, T_2, T_3, T_4, T_5, and T_6, respectively. Moreover, each product is on-shelf in different time intervals. We define TI_{List} as the List of whole Time Intervals in database D_2. $TotalTI(h)$ represents the utility of time interval h. The total utilities of all of the time intervals in database D_2 are [a, 59], [b, 83], [c, 106], [d, 142], [e, 77], and [f, 14]. The utility of itemset X in time interval h is denoted as $UTI(X, h)$. The relative utility of the itemset is used for determining whether the itemset is an HOUI or not. $RelativeU(X)$ represents the Relative Utility of the itemset, which is computed by the utility of itemset X divided by the total utility of each time interval that contains itemset X in the database.

C. Data Structures

Here, we will introduce our proposed 6 data structures. First, we store the information of each transaction in the *TIPN_Table*. Moreover, negative items are ignored in the overestimated value of the itemset. Furthermore, we design the *NIMC_Table* to keep the local minimum count of each negative item according to different time intervals. Obviously,

in the process of mining top-k high on-shelf utility itemset, the threshold is initially set to 0 and it is increased during the mining process. We propose the *Bitmap* to store the status of the occurrence of each item according to different time intervals. We propose the *Time Interval Utility List* to achieve the goal.

TIPN_Table has six columns as shown in Figure 2. Each row in *TIPN_Table* contains a set of transactions appeared at that time time interval, and each transaction is classified as positive items and negative items. Furthermore, the utility of each positive item in the transaction is stored. Here, we store the count of each negative item, which is helpful for constructing *NIMC_Table* described later.

Time Interval	TID	Positive Items	Positive Utility	Negative Items	Negative Count
1	T_1	f	f: 16	b, d	b: 3, d: 2
	T_2	c, e	c: 21, e: 10	b, d	b: 2, d: 4
2	T_3	a, c	a: 30, c: 9	d	d: 4
	T_4	a, e	a: 20, e: 4	d	d: 2
3	T_5	c, e	c: 24, e: 8	b, d	b: 3, d: 5
	T_6	c	c: 18	b, d	b: 1, d: 3

Fig. 2. $TIPN\_Table$ of database D_2

We propose the data structure called *NIMC_Table* to decrease the overestimated utility. Moveover, we propose two counts called *GNC*, and *LNC* to record the minimum count of each negative item (b and d). We record the *Global Minimum Count* (GMC) of each negative item. For each negative item, we calculate the minimum count of each time interval and store it to the GMC column. If we have count = 0, we just skip it. For example, for negative item b, we record 2, 0, 1 ,for time intervals 1, 2, 3, respectively, and record its GMC = 1.

IO_Bitmap is used for storing the occurrence of each item in the database and it is constructed during the initial pass of the database. For example, for item c in time interval 1, the related bits are 01.

We propose the *Time Interval Utility List* for discovering HOUIs efficiently. The *Time Interval Utility List* is inspired by the *Utility List*. We define such a list of itemset X as $TIUL(X) = \{(TI, \{(TID, P\_Util, N\_Util, NGC\_Util, R\_Util)\})\}$, where TI is the time interval, ***P_Util*** is the positive utility of itemset X in transaction T_{TID}. Moreover, ***N_Util*** is the negative utility of itemset X in transaction T_{TID} and ***NGC_Util*** is the negative utility of itemset X that considers the **Global Minimum Count** (denoted as ***GMC***) in transaction T_{TID}. Note that we have $UTGC(i, T_j) = TU(i, T_j)$, if $P\_Val(i) > 0$. Moreover, $UTGC(i, T_j) = GC(i) \times$ ***P_Val(i)***, if $P\_Val(i) < 0$. Furthermore, ***R_Util*** is the Remaining Utility of itemset X in transaction T_{TID} and it is defined as follows:

$$R\_Util(X, T_j) = \sum_{i \in T_j \wedge i \succ x \forall x \in X} UTLC(i, T_j).$$

Here, the ***UTGC(i, T_j)*** (Utility of the item i in transaction T_j with Global minimum Count) adopts the global minimum count of each **negative** item according to the *NIMC_Table*. Then, the utility of item i is calculated as $UTLC(b, T_1) = GC(b) \times$ ***P_Val(i)*** $= 2 \times$ **(-4)** $= -8$.

The *Time Interval Utility List* of itemset X stores the positive utility and negative utility of itemset X. Moreover, such a List stores the Negative Utility by considering the ***Global minimum Count*** of each time period. Figure 3 shows the *Time Interval Utility List* of item e. In the mining process, the algorithm constructs such a list for each single item in the database. Then, these Lists can be used for extracting HOUIs in the *Mining* procedure. To calculate the utility of the itemset and the overestimated utility of the item, we define several variables as follows. (1) ***sumP_Util(X)*** means the sum of Positive Utilities ***P_Util(X)*** of *TIUL* of X. (2) ***sumN_Util(X)*** means the sum of Negative Utilities ***N_Util(X)*** of *TIUL* of X. (3) ***sumNLC_Util(X)*** means the sum of Negative Utilities with Local minimum Count *NNGC_Util* of *TIUL* of X. (4) ***sumR_Util(X)*** means the sum of Remaining Utilities *R_Util* of *TIUL* of X. (5) ***sumN_Util(X)*** means the accumulation of Positive Utilities and Negative Utilities of *TIUL* of X.

Itemset	Time Interval	TID	P_Util	N_Util	NLC_Util	R_Util
e	1	2	10	0	0	15
	2	4	4	0	0	-2
	3	5	8	0	0	19

Fig. 3. The *Time Interval Utility List* of item e

D. Pruning Strategies

Here, we will introduce our three pruning strategies. In the first pruning strategy called TWUGC, which is motivated by the RTWU [5], we define the TWUGC as follows:

$$\textbf{\textit{TWUGC(X)}} = \sum_{X \in T_j \wedge T_j \in D} TotalU\_GC(T_j).$$

$$\textbf{\textit{TotalU\_GC(}}\mathbf{T_j}\textbf{\textit{)}} = \sum_{i \in T_j \wedge T_j \in D} \textbf{\textit{UTGC(i, }}\mathbf{T_j}\textbf{\textit{)}}.$$

$$\textbf{\textit{UTGC(i, }}\mathbf{T_j}\textbf{\textit{)}} = \begin{cases} TU(i, T_j), & \text{if } P\_Val(i) > 0. \\ GC(i) \times P\_Val(i), & \text{if } P\_Val(i) < 0. \end{cases}$$

The **TWUGC** of itemset X is the **sum** of $TotalU\_GC(T_j)$ of transactions, where itemset X appears. ***TotalU_GC(*$\mathbf{T_j}$*)*** is the total utility of transaction T_j by considering the **global** minimum count of each negative item. Moreover, $TotalU\_GC(T_j)$ is calculated as the **sum of utilities** of items in transaction T_j. If the item is negative, the utility of the item is calculated as the product of global minimum count of the item and the profit of the item. For the second pruning strategy, the *RLC pruning strategy* prunes some of hopeless candidates. We utilize the utility list to calculate the overestimated utility value of the relative utility of the itemset. For the third pruning

strategy, the TIO pruning strategy prunes subtrees of the set-enumeration tree which do not appear in the database during the mining process. We have introduced the IO Bitmap which records occurrence statuses of all of the items.

E. Threshold Increased Strategies

Here, we will introduce two threshold increased strategies. The $RPRU\_Size1$ Strategy calculates the relative utility for all of the positive items in database D and it inserts those positive items into the $RPRU\_Size1\_List$. Then, the $RPRU\_Size1$ Strategy sorts the $RPRU\_Size1\_List$ according to the descending order of relative utilities and it increases the threshold value to the k-highest relative utility in the $RPRU\_Size1\_List$. The RRU_Size2 Strategy calculates the relative utilities for all of the items in database D and it inserts them into RRU_Size2_List. The RRU_Size2 Strategy sorts the RRU_All_List according to the descending order of relative utilities and it increases the threshold value to the k-highest relative utility in the RRU_All_List.

F. The Mining Process

In the section, we will introduce the mining process of our algorithm.

1) The Preprocessing Step: In this step, we construct *TI_List*, *TotalTI_Table*, *TIPN_Table*, *IO_Table* and *NIMC_Table*. In addition, the minimum count of negative items at a time interval is called *Local Minimum Count*. In contrast, the minimum count of negative items is called *Global Minimum Count*, which is the maximum count of all of the local minimum counts of the negative item.

After scanning the database once, the algorithm performs the *RPRU-Size1 Strategy* to increase the value of *ThreVal*. To obtain the real relative utility of positive item a, we calculate the utility of positive item a by using *TIPN_Table*. We have $UD(a) = 115$. Moreover, the list of time intervals which contain item a is $TI\_Occu\_List(a) = \{1, 2, 3\}$. The total utility of the list of time intervals which contain item a is calculated as $TI\_Occu\_Total(a) = 133$. The real relative utility of item a is calculated as $UD(a)/TI\_Occu\_Total(a) = 0.86$. The result of *RPRU_Size1* List which stores real relative utilities of positive items is [Each positive item, RPRU] = [[a, 0.86], [c, 0.54], [e, 0.17], [f, 0.09]]. If the size of *RPRU_Size1_List* is not less than k, we increase the threshold value $ThreVal$ to the k-highest value in *RPRU_Size1_List*. On the other hand, TK_List stores the k-highest HOUIs and it is sorted by the descending order of the relative utility. Therefore, TK_List is updated by *RPRU_Size1_List*.

Then, we calculate the TWUGC value of all of the items in the database for overestimating the relative utility. Here, we define a total order called *TWUGC_Order* for finding the Top-k HOUIs efficiently. *TWUGC_Order* has three ordering rules. First, positive items are sorted by the descending order of TWUGC. Second, negative items are sorted by the descending order of TWUGC. Third, negative items succeed positive items. *TWUGC_Order* of database D_2 is [f, a, e, c, b, d]. Figure 4 shows the result of sorted *TIPN_Table*.

After *TIPN_Table* is sorted by the *TWUGC_Order*, our algorithm creates *Time Interval Utility List* for all of the single items in the database. *Time Interval Utility List* of each single items can be used for discovering itemsets with the large size by using the itemset expansion method.

Time Interval	TID	Positive Items	Positive Utility	Negative Items	Negative Count
1	T_1	f	f: 16	b, d	b: 3, d: 2
	T_2	e, c	c: 21, e: 10	b, d	b: 2, d: 4
2	T_3	a, c	a: 30, c: 9	d	d: 4
	T_4	a, e	a: 20, e: 4	d	d: 2
3	T_5	e, c	c: 24, e: 8	b, d	b: 3, d: 5
	T_6	c	c: 18	b, d	b: 1, d: 3

Fig. 4. Sorting $TIPN\_Table$ in *TWUGC_Order*

To further increase the minimum threshold value for finding Top-k HOUIs, our algorithm applies another threshold increased strategy called the *RRU-Size2 Strategy*. The *RRU-Size2 Strategy* calculates the real relative utility for each size 2 itemsets and stores them in *RRU_Size2_List*. Moreover, the algorithm uses the *Time Interval Utility List* and *IO_Table* to calculate the real relative utility of the size 2 itemset efficiently. For obtaining the real relative utility of itemset $x \cup y$, we need to calculate the utility of itemset $x \cup y$ and the total utility of each time Interval which contains the occurrence of itemset $x \cup y$. The real relative utility of $x \cup y$ can be calculated as $Relative(x \cup y) = UD(x \cup y)/TI\_Occu\_Total(x \cup y)$. TK_List is updated by *RRU_ALL_List*.

2) The Mining Step: In the mining step, our algorithm applies the pattern growth method to discover Top-k HOUIs. At the beginning of the mining process, our algorithm traverses through all of the TIULs of items from the root node of the set-enumeration tree, *i.e.* empty itemset $\{\}$. In the each iteration, the original itemset P is appended with the current item x to obtain new itemset $NewP = P \cup \{x\}$. Then, our algorithm checks whether new itemset $NewP$ is a HOUI or not. If the relative utility of the itemset is not less than the threshold value *ThreVal*, the itemset is a HOUI. If the relative utility of the new itemset $NewP$ is greater than the k-highest relative utility in TK_List, our algorithm removes the k-highest itemset, and inserts the new itemset $NewP$ into TK_List. Moreover, the threshold value $ThreVal$ is updated as the k-highest relative utility in TK_List. In order to calculate the relative utility of new itemset $NewP$, we have to calculate the utility of new itemset $NewP$ in the database and the total utility of each time interval that contains the occurrence of the new itemset in the database TI_Occu_Total. For calculating the utility of new itemset $NewP$, our algorithm uses *TIUL* of itemset $NewP$. The sum of positive utilities and negative utilities of itemset $NewP$ contains $sumPN\_Util(NewP)$, which is the accumulation of positive utilities $sumP\_Util(NewP)$, and the accumulation of negative utilities $sumN\_Util(NewP)$.

Moreover, if $sumPN\_Util(NewP)$ is less than 0, our algorithm skips it directly. For calculating TI_Occu_Total, our algorithm performs the AND operation of all of the time intervals of itemset $NewP$ in *IO_Bitmap*.

If the related utility of the new itemset $NewP$ is not less than the threshold value $ThreVal$, the new itemset $NewP$ is a HOUI. Moreover, Our objective is to discover the Top-k HOUIs within the database. Moreover, our algorithm creates a list called TK_List to store the Top-k HOUIs during the mining process. TK_List is a list which dynamically sorts all of the HOUIs in TK_List according to the descending order of relative utilities. If the size of TK_List is less than the user-defined parameter k, our algorithm inserts the new itemset $NewP$ into TK_List directly. If the size of TK_List is equal to the user-defined parameter k, our algorithm checks whether the relative utility of new itemset $NewP$ is greater than the k-highest relative utility in TK_List or not. If the result is true, the new itemset $NewP$ is inserted into TK_List. After the above checking, whether the new itemset $NewP$ is a HOUI or not, our algorithm applies the *TWUGC Pruning Strategy* to prune unpromising itemsets.

IV. Performance

In this section, we will show the performance study of our TIPN-Table-Based algorithm and the KOSHU algorithm [4]. In this performance study, we utilize two types of databases, which are real database and the synthetic database, to perform the experiment.

A. The Performance Model

The real database which we have used for experiments is *retail* (the **sparse** database). The above real database is downloaded from the SPMF library [5]. The ***retail*** database has the density lower than 1%, it is treated as the **sparse** database. The value of the time interval is 5, which is equal to the consideration of the KOSHU algorithm [4]. On the other hand, we set k to the range between 50 and 150. The real database has 88162 transactions (containing 16470 items) with desity=0.06.

For the synthetic database, we also utilize four parameters T, I, MI and NP to conduct the performance experiments of our TIPN-Table-Based algorithm and KOSHU algorithm, where T represents the total amount of transactions in the database, I represents the total amount of distinct items of the database. MI represents the maximum amount of distinct items of a single transaction and NP represents the percentage of counts of items with negative profits. For example, $T10000\_I4000\_MI10\_NP80$ is a synthetic database with 10000 transactions, 4000 distinct items, up to 10 distinct items in a single transaction and 80 percent of items with negative profits. These synthetic databases has been produced by the generator from the IBM Almaden Quest research group [1].

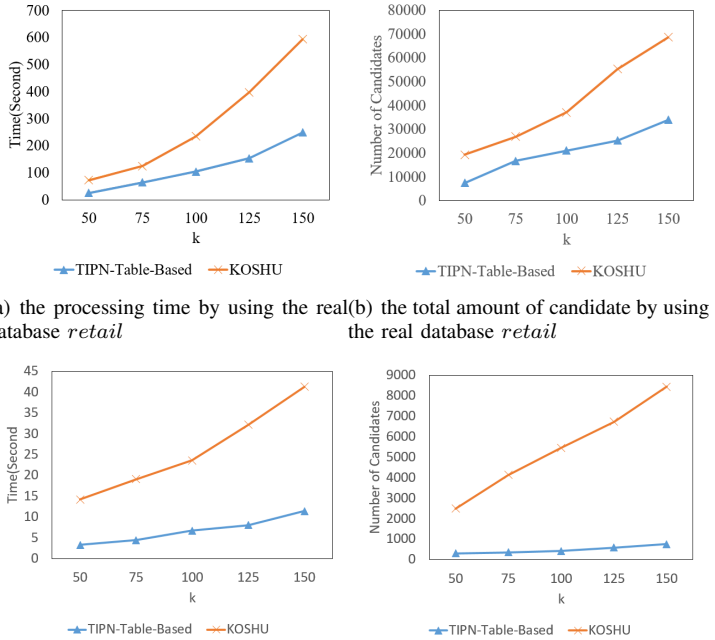
B. Experiment Results

Here, we will show the comparison of the performance between our TIPN-Table-Based algorithm (denoted as TIPN-Table-Based) and the KOSHU algorithm [4] (denoted as KOSHU).

The KOSHU algorithm has produced the EMPRS data structure during the preprocessing step, which is time-consuming. Moreover, the number of candidates of our TIPN-Table-Based algorithm is less than that of the KOSHU algorithm. There are two reasons why our algorithm could generate less number of candidates than the KOSHU algorithm. First, our *TWUGC Pruning Strategy* considers the global minimum count of all of the items with negative profits. Second, our *RLC Pruning Strategy* utilizes the **local** minimum count of all of the items with negative profits at each time interval. Therefore, the **remaining utility** of our *RLC Pruning Strategy* is tighter than that of the KOSHU algorithm. Moreover, the number of candidates could be pruned by our algorithm more than that of the KOSHU algorithm during the mining process.

Figure 5(a) and Figure 5(b) show the comparisons of the performance between these two algorithms by using the real database ***retail***. As the value of k is increased, the performance measures including the processing time and the number of candidates of our proposed algorithm are better than those of KOSHU algorithm. Moreover, the reasons for such results are the same as those reasons described before.

For the synthetic databases $T10000\_I100\_MI10\_NP80$ (the **dense** database). Figure 5(c) and Figure 5(d) show comparisons of the two concerned algorithms. Those result are similar to the comparison between the two algorithms.



(a) the processing time by using the real database $retail$

(b) the total amount of candidate by using the real database $retail$

(c) the processing time by using the synthetic database *T10000_I100_MI10_NP80* (the dense database)

(d) the total amount of candidate by using the synthetic database *T10000_I100_MI10_NP80* (the dense database)

Fig. 5. A comparison under the change of k

V. Conclusion

In this paper, we have proposed the TIPN-Table-Based algorithm, which is able to mine top-k high on-shelf utility itemsets efficiently. The TIPN-Table-Based algorithm only scans the database once and sorts the database once. Moreover,

we have proposed the global and local concepts to make the tight upper bound. Furthermore, we have utilized a bit map strategy to decrease the processing time. Our experiment results have shown that our TIPN-Table-Based algorithm has a better performance than the KOSHU algorithm.

REFERENCES

[1] "IBM Quest Synthetic Data Generation Code" http://www.almaden.ibm.com/cs/quest/syndata.html, 1996

[2] M. Ashraf, T. Abdelkader, S. Rady, and T. F. Gharib, "TKN: An Efficient Approach for Discovering Top-k High Utility Itemsets with Positive or Negative Profits," Information Sciences, Vol. 587, pp. 654–678, March 2022.

[3] J. Chen, X. Guo, W. Gan, C.-M. Chen, W. Ding, and G. Chen, "On-shelf Utility Mining from Transaction Database," Engineering Applications of Artificial Intelligence, Vol. 107, pp. 1–12, January 2022.

[4] T.-L. Dam, K. Li, P. Fournier-Viger, and Q.-H. Duong, "An Efficient Algorithm for Mining Top-k On-shelf High Utility Itemsets," Knowledge and Information Systems, Vol. 52, pp. 621–655, January 2017.

[5] P. Fournier-Viger, C. W. Lin, A. Gomariz, T. Gueniche, Z. D. A. Soltani, and H. T. Lam, "The Spmf Open-Source Data Mining Library Version 2," Proc. 19th European Conf. on Principles of Data Mining and Knowledge Discovery (PKDD 2016) Part III, pp. 36–40, 2016.

[6] S. Krishnamoorthy, "Mining Top-k High Utility Itemsets with Effective Threshold Raising Strategies," Expert Systems with Applications, Vol. 117, pp. 148–165, March 2019.

[7] J. Lee, U. Yun, G. Lee, and E. Yoon, "Efficient Incremental High Utility Pattern Mining Based on Pre-large Concept," Engineering Applications of Artificial Intelligence, Vol. 72, pp. 111–123, June 2018.

[8] K. Singh, S. S. Singh, A. Kumar, and B. Biswas, "TKEH: An Efficient Algorithm for Mining Top-k High Utility Itemsets," Applied Intelligence, Vol. 49, pp. 1078–1097, October 2018.

[9] K. Singh, A. Kumar, S. S. Singh, H. K. Shakya, and B. Biswas, "EHNL: An Efficient Algorithm for Mining High Utility Itemsets with Negative Utility Value and Length Constraints," Information Sciences, Vol. 484, pp. 44–70, May 2019.

[10] B. Vo, H. Bui, T. Vo, and T. Le, "Mining Top-Rank-k Frequent Weighted Itemsets Using WN-list Structures and an Early Pruning Strategy," Knowledge-Based Systems, Vol. 201-202, pp. 1–12, August 2020