Mining Top-rank-k Erasable Patterns

Ye-In Chang[1], Chen-Chang Wu[2*] and Kuan-Chieh Lin[3]
National Sun Yat-Sen University[1,3] and Kaohsiung Medical University[2]
Kaohsiung, Taiwan
changyi@cse.nsysu.edu.tw[1], wucc5501@kmu.edu.tw[2], jackie10245@gmail.com[3]

Abstract

Erasable patterns extract items with low values conversely, which can help users to find meaningless information in databases, so as to reduce the cost in factories or stores. The previous algorithms for mining erasable patterns make users face the problem that the threshold is hard to decide. Le *et al.* proposed the algorithm named *TEPUS* to mine top-rank-k erasable patterns with the subsume concept and the dynamic threshold. However, the *TEPUS* will generate too many unnecessary number of candidates during the mining process. In this paper, we propose an algorithm that applies the timely updated dynamic threshold method for pruning unnecessary candidates during the mining process. From our experimental results, we show that our algorithm performs better than the *TEPUS* algorithm.

Key words: dynamic threshold pruning, erasable patterns, subsume concept, top-rank-k mining

Introduction

Data mining is to extract useful or interesting patterns from the enormous database, and the research of data mining has become more popular in recent years. There are many research topics for mining patterns, like frequent patterns [1, 2, 3, 4, 5, 6, 7]. There are also other patterns such as frequent weighted patterns [8, 9], maximal frequent patterns [10], high utility patterns [11, 12, 13, 14], and erasable patterns [15, 16, 17, 18, 19, 20].

Most traditional methods of data mining are usually to extract items with high frequency or values in databases. However, erasable patterns extract items with low values conversely, which can help users to find meaningless information in databases, so as to reduce the cost of the required components in factories or stores. There are many algorithms to deal with erasable patterns, such as the *MEI* algorithm [15], the *pMEIC* algorithm [19], *etc*.

However, it is hard for users to decide the threshold. Therefore, top-*k* and top-rank-*k* mining patterns [1, 4, 13, 16] are purposed to solve this problem. The top-rank-*k* concept which orders patterns by the related counts in the ascending order is applied to mine erasable patterns. Note that the result of top-rank-*k* may exist several patterns with the same rank.

Le *et at.* proposed the *TEPUS* algorithm [16] for mining top-rank-k erasable patterns with the subsume concept. During the mining process, erasable patterns are classified according to the dynamic threshold, which is a feature of top-rank-*k* mining. Nevertheless, the *TEPUS* algorithm requires additional sorting for finding subsume relations for size 1 patterns. In addition, the *TEPUS* algorithm always update the dynamic threshold at the end of the current size of patterns mining. Therefore, the frequency of updating the dynamic threshold is too slow in the *TEPUS* algorithm, which will lead to generate too many unnecessary candidates in the mining process.

To get rid of these disadvantages, in this paper, we propose an algorithm called Inverted Subsume Table (*IST*) algorithm for mining top-rank-*k* erasable patterns. We construct Product-Value list (*PVlist*) and Inverted-Subsume table (*IsubT*) to record the information of produces and patterns, respectively. We use the timely updated dynamic threshold *MaxGain* to mine erasable patterns. Furthermore, during the mining process, we update the dynamic threshold *MaxGain*, once the gain value of the last entry of erasable patterns table (*EpsT*) has been modified, instead of updating the dynamic threshold at the end of current size of patterns mining in the *TEPUS* algorithm. It means that we can prune more number of candidates than the *TEPUS* algorithm, since the dynamic threshold could be reduced while generating candidates. From our experiment results, we show that the performance of our algorithm is better than that of the *TEPUS* algorithm.

The rest of the paper is organized as follows. In Section 2, we show a survey about the *TEPUS* algorithm. In Section 3, we present our proposed IST algorithm. In Section 4, we present the comparison of the performance between our *IST* algorithm and the *TEPUS* algorithm. Finally, in Section 5, we give the conclusion.

The TEPUS Algorithm

Le *et al.* proposed the *TEPUS* algorithms for mining top-rank-*k* erasable patterns [16]. Let $I = \{i_1, i_2, ..., i_m\}$ be a set of all items, and $DB = \{P_1, P_2, ..., P_n\}$, where P_i $(1 \le i \le n)$ is a product, which is presented in the form of (*Items*, *Val*), where Items are the components of P_i and *Val* is the profit of selling the product P_i. Figure 1 is an example of dataset used throughout this section [16].

Product	Items	Val
P_1	*a, b*	1000
P_2	*a, b, e*	200
P_3	*c, e*	150
P_4	*b, d, e, f*	50
P_5	*c, d, e*	100
P_6	*d, e, f, h*	200
P_7	*d, h*	150
P_8	*d, f, h*	100

Fig. 1 An example product dataset[16].

Let top-rank-*k* table be denoted by *TR*, which is applied to maintain the current top rank-*k* erasable itemsets, and itemsets in the *TR* are sorted by the ascending order of the gain value. When the TR has *k* entries, the dynamic threshold is defined as

ISBN 978-1-6654-9650-6

the gain value of the last entry in *TR*.

First of all, let k = 6. The *TEPUS* algorithm constructs *dPidset* and subsume index of each item from the product dataset Figure 2 is the resulting table after finding *dPidset* and subsume index.

Rank	Items	Gain	dPidset	Subsume
1	*c*	250	3,5	
2	*f*	350	4,6,8	
3	*h*	450	6,7,8	
4	*d*	600	4,5,6,7,8	*f, h*
5	*e*	700	2,3,4,5,6	*c*
6	*a*	1200	1,2	

Fig. 2 The gains, dPidsets and subsume for the example dataset [16].

Second, the *TEPUS* algorithm extracts erasable 1-patterns with these items' *dPidset* and then sorts them in the ascending order of gain values. Figure 3 shows the current step.

Rank	Items	Gain
1	*c*	250
2	*f*	350
3	*h*	450
4	*d*	600
5	*e*	700
6	*a*	1200

Fig. 3 Top six ranked erasable itemsets in TR [16].

Next, the *TEPUS* algorithm generates size (l+1) candidates using size l itemsets. Note that the dynamic threshold is set to 1200 so far, if the gain value of any two erasable itemsets is greater than the dynamic threshold and the *TR* has k entries, the algorithm will not create the next candidate. Finally, the algorithm repeats these steps until no candidates can be inserted into *TR*. Then, the mining process is finished here. Figure 4 shows the final result of TR [16].

Rank	Items	Gain
1	*c*	250
2	*f*	350
3	*h*	450
4	*fh*	500
5	*d, cf, fd, hd, fhd*	600
6	*e, ch, ce*	700

Fig. 4 Top six ranked erasable itemsets for the example datatset [16].

Proposed Method

In this section, we will use an example database to illustrate our algorithm. During the mining process of erasable patterns, we use the gain value of an itemset, which means the total profits of products that the itemset exists, as a main factor to decide whether the itemset is the erasable pattern or not. Thus, we use the Product-Value list and the Inverted-Subsume table to record the information. Note that we only have to scan the database one time to record the information of the database. During the mining process, we always use these data structures to extract erasable patterns. Figure 5 shows an example database (DB_e) as the input.

Product	Items	Value
P_1	a, b, c, f, g	200
P_2	a, e, g	200
P_3	a, b, e, f	150
P_4	c, f	150
P_5	b, c, f	100
P_6	a, d, f	100
P_7	e, g	1200
P_8	a, d	150

Fig. 5 An example of product database DB_e.

A. The Product-Value List and Inverted-Subsume Table

The Product-value list, called *PVlist*, is composed of the identifications (*Pi*) and values (*Pval*) of products. For example, in Figure 5, there are eight products in DB_e, we will record each product and its value into the *PVlist*. The main feature of *PV* list is to improve the algorithm when computing the gain values of candidates. That is, we can compute gain values without scanning the database again. Figure 6 shows the result of this example.

Product	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
Value	200	200	150	150	100	100	1200	150

Fig. 6 The *PVlist* of the original data in the DB_e.

The Inverted-Subsume table, which is called *IsubT*, is the item-based table, which means that it records the information of databases with each item in the set of items. The structure of *IsubT* is composed of four fields, which are *ItemName* denoted by *Ix*, *gain(Ix)*, *dPid(Ix)* and *subsumeList(Ix)*, containing the items, the gain values of items, the *dPidset* of items and the subsume lists of items, respectively. Figure 7 shows the size 1 items in *IsubT* after scanning DB_e. Then, we use *IsubT* as the main data structure during the remaining mining process.

Ix	$gain(Ix)$	$dPid(Ix)$	$subsumeList(Ix)$
d	250	[6, 8]	[a]
b	450	[1, 3, 5]	[f]
c	450	[1, 4, 5]	[f]
f	700	[1, 3, 4, 5, 6]	ϕ
a	800	[1, 2, 3, 6, 8]	ϕ
e	1550	[2, 3, 7]	ϕ
g	1600	[1, 2, 7]	ϕ

Fig. 7 Size 1 items in *IsubT*.

B. The Mining Process for the Original Data

According to the given example database DB_e above, we let *rank* = 6 to illustrate our method. Considering the original data, first of all, we scan database one time, so as to construct *PVlist* shown in Figure 6. After that, we store the *dPidset* of each size 1 item in DB_e, and calculate the gain values of them. Then, we sort these items in the ascending order of gain values. Since the

rank is six, we set the rank size of *EpsT* to six, where *EpsT* is the table to store top-rank-*k* erasable itemsets. Figure 8 shows the size 1 items in *EpsT* .

Rank	Gain	Items
1	250	*d*
2	450	*b, c*
3	700	*f*
4	800	*a*
5	1550	*e*
6	1600	*g*

Fig. 8 The size 1 items in *EpsT* with rank = 6.

After setting size 1 erasable patterns, we have a sorted size 1 item list [*d, b, c, f, a, e, g*], which is sorted in the ascending order of gain values. The items in the list are those patterns which are in *EpsT*, and they will be used to create size 2 candidates. We check subsume relations of each item in this item list. The checking process of subsume relation is shown in Figure 9.

Items X and Y	$pid(X)$	$pid(Y)$	subsume relation
$[d, b]$	[6, 8]	[1, 3, 5]	-
$[d, c]$	[6, 8]	[1, 4, 5]	-
$[d, f]$	[6, 8]	[1, 3, 4, 5, 6]	-
$[d, a]$	[6, 8]	[1, 2, 3, 6, 8]	$pid(d) \subseteq pid(a)$
$[d, e]$	[6, 8]	[2, 3, 7]	-
$[d, g]$	[6, 8]	[1, 2, 7]	-
$[b, c]$	[1, 3, 5]	[1, 4, 5]	-
$[b, f]$	[1, 3, 5]	[1, 3, 4, 5, 6]	$pid(b) \subseteq pid(f)$
$[b, a]$	[1, 3, 5]	[1, 2, 3, 6, 8]	-
$[b, e]$	[1, 3, 5]	[2, 3, 7]	-
$[b, g]$	[1, 3, 5]	[1, 2, 7]	-
$[c, f]$	[1, 4, 5]	[1, 3, 4, 5, 6]	$pid(c) \subseteq pid(f)$
$[c, a]$	[1, 4, 5]	[1, 2, 3, 6, 8]	-
$[c, e]$	[1, 4, 5]	[2, 3, 7]	-
$[c, g]$	[1, 4, 5]	[1, 2, 7]	-
$[f, a]$	[1, 3, 4, 5, 6]	[1, 2, 3, 6, 8]	-
$[f, e]$	[1, 3, 4, 5, 6]	[2, 3, 7]	-
$[f, g]$	[1, 3, 4, 5, 6]	[1, 2, 7]	-
$[a, e]$	[1, 2, 3, 6, 8]	[2, 3, 7]	-
$[a, g]$	[1, 2, 3, 6, 8]	[1, 2, 7]	-
$[e, g]$	[2, 3, 7]	[1, 2, 7]	-

Fig. 9 The processing table of finding subsume relation of size 1 items.

For the following mining process, we use *PVlist*, *IsubT* for mining without scanning the database again. For size 2 candidates, we combine each item in size 1 item list, which means that if size 1 item list has n number of items, we have at most (n*(n-1)/2) number of candidates. In the above case, we have 7 items in the size 1 item list, so we will have at most 21 (= 7*6/2) number of candidates. For size l ($l \geq 3$) candidates, we check whether items have the same prefix to generate candidates. For example, given two items *XY* and *XZ*, they have the same prefix *X*. If two items have the same prefix, we will generate the candidate of these two items, *i.e.*, the itemset *XYZ* (or *XZY*).

Moreover, we use the dynamic threshold method to prune unnecessary candidates and improve the performance of generating next candidates. The dynamic threshold is denoted by *MaxGain*, which is defined as follows, where $EpsT_{last}$ means the patterns in the rank *k* in *EpsT* :

$$MaxGain = gain(EpsT_{last}).$$

While generating next candidates, we will check whether the gain value of the candidate is lower than or equal to *MaxGain*. There are three possible cases while we updating *EpsT* with the candidate.

For Case *I*, if the gain value of the candidate is greater than *MaxGain*, we can assure that the candidate is not the erasable pattern, it can be pruned. The reason is that the rank of the candidate will be greater than *k* due to the feature of gain values. For Case *II*, the gain value of the candidate is lower than or equal to *MaxGain*, and *EpsT* also has patterns with this gain value. Therefore, we insert the candidate into *EpsT* with the same rank of these patterns. For Case *III*, the gain value of the candidate is lower than *MaxGain*, and *EpsT* has no patterns with this gain value. It means that we have to add the candidate into *EpsT* with a new rank, and let the rank of patterns whose gain values are greater than the gain value of the candidate be added 1. Moreover, we will update *MaxGain* according to the definition above. Figure 10 shows erasable patterns with their ranks and gain values.

Rank	Gain	Items
1	250	*d*
2	450	*b, c*
3	600	*cb*
4	700	*f, bd, cd, fb, fc, fcb*
5	800	*a, ad*
6	850	*fd, cdb, fdb, fdc, fdcb*

Fig. 10 The *EpsT* after mining erasable patterns of the original data in DB_e.

Performance

In this section, we show the performance of mining top-rank-*k* erasable patterns with our *IST* algorithm and the *TEPUS* algorithm. We use the Chess dataset from the dataset repository (http://fimi.uantwerpen.be/data/) [16]. Both of these two algorithms are implemented in Java on the personal computer with the Intel Core i7-8700 3.2 GHz CPU and 16 GB of RAM.

We consider not only the processing time of mining erasable patterns, but also the number of candidates generated in the mining process. The profits of products of the Chess dataset are generated between 1 to 5000. The value of rank k which we considered are range of 40 to 200.

From Figure 14 and Figure 15 for the Chess dataset, we have set the value of k to 40, 80, 120, 160, 200, respectively. These figures below show the processing time of our *IST* algorithm is faster than the processing time of the *TEPUS* algorithm. Besides, our algorithm also generated less number of candidates than the *TEPUS* algorithm. It is because the *TEPUS* does not update the *MaxGain* timely. Thus, the *TEPUS* algorithm generates too many unnecessary candidates during the mining process. Therefore, the performance of our *IST* algorithm is much better than that of *TEPUS* algorithm.

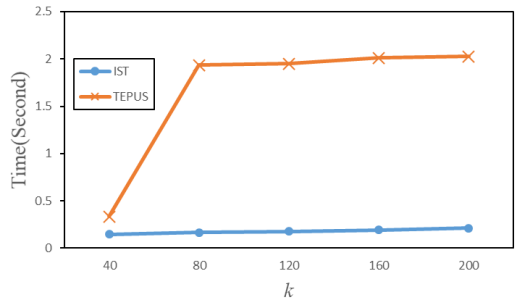

Fig. 14 A comparison of the processing time between our *IST* algorithm and the *TEPUS* algorithm for mining the Chess dataset under the change of *k*.

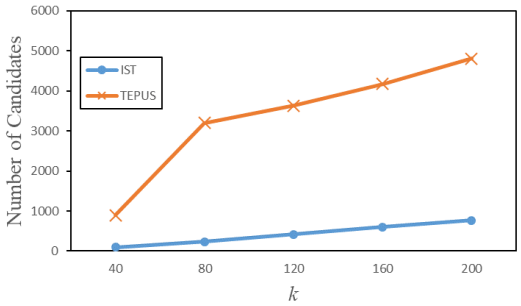

Fig. 15 A comparison of the number of candidates generated between our *IST* algorithm and the *TEPUS* algorithm for mining the Chess dataset under the change of *k*.

Conclusion

In this paper, we have proposed the *IST* algorithm for incremental mining top-rank-*k* erasable patterns efficiently. We have used data structures, the *PVlist* and the *IsubT* to store the information of items for the mining process. Furthermore, we have proposed *MaxGain* to prune more candidates than the *TEPUS* does. The experiment results have shown that the performance of our *IST* algorithm is better than that of the *TEPUS* algorithm.

Acknowledgment

This research was supported in part by the Ministry of Science and Technology of Republic of China under Grant No. MOST-108-2221-E-110-060.

References

[1] Z.-H. Deng, "Fast Mining Top-rank-k Frequent Patterns by Using Node-lists," *Expert Systems with Applications*, Vol. 41, pp. 1763-1768, March 2014.

[2] Z.-H. Deng, "Diffnodesets: an efficient structure for fast mining frequent itemsets," *Applied Soft Computing*, Vol. 41, pp. 214-223, April 2016.

[3] Z.-H. *Deng*, W. ZhongHui, and J. JiaJian, "A new algorithm for fast mining frequent itemsets using N-lists," *Information Sciences*, Vol. 55, pp. 2008-2030, Sept. 2012.

[4] Q. Huynh-*Thi*-Le, T. Le, B. Vo, and B. Le, "An efficient and effective algorithm for mining top-rank-k frequent patterns," *Expert Systems with Applications*, Vol. 42, pp. 156-164, Jan. 2015.

[5] A. Rakesh and S. Ramakrishnan, "Fast algorithms for mining association rules in large databases," *Proc. of the 20th Int. Conf. on Very Large Data Bases*, pp. 487-499, 1994.

[6] B. Vo, T. Le, F. Coenen, and T.-P. Hong, "Mining frequent itemsets using the N-list and subsume concepts," *International Journal of Machine Learning and Cybernetics*, Vol. 7, pp. 253-265, April 2016.

[7] W. Song, B. Yang, and Z. Xu, "Index-BitTableFI: an improved algorithm for mining frequent itemsets," *Knowledge-Based Systems*, Vol. 21, pp. 507-513, Aug. 2008.

[8] H. Bui, et al., "A weighted N-list-based method for mining frequent weighted itemsets," Expert Systems with Applications, Vol. 96, pp. 388-405, April 2018.

[9] B. Vo, F. Coenen, and B. Le, "A new method for mining frequent weighted itemsets based on WIT-trees," *Expert Systems with Applications*, Vol. 40, pp. 1256-1264, March 2013.

[10] B. Vo, S. Pham, T. Le, and Z.-H. Deng, "A novel approach for mining maximal frequent patterns," *Expert Systems with Applications*, Vol. 73, pp. 178–186, May 2017.

[11] S. Krishnamoorthy, "Hminer: efficiently mining high utility itemsets," *Expert Systems with Applications*, Vol. 90, pp. 168-183, Dec. 2017.

[12] S. Krishnamoorthy, "Efficient mining of high utility itemsets with multiple minimum utility thresholds," *Engineering Applications of Artificial Intelligence*, Vol. 69, pp. 112-126, March 2018.

[13] S. Lee and J. S. Park, "Top-k high utility itemset mining based on utility-list structures," *Proc. of the Int. Conf. on Big Data and Smart Computing (Big-Comp)*, pp. 101-108, 2016.

[14] U. Yun, H. Ryang, G. Lee, and H. Fujita, "An efficient algorithm for mining high utility patterns from incremental databases with one database scan," *Knowledge-Based Systems*, Vol. 124, pp. 188-206, May 2017.

[15] T. Le and B. Vo, "MEI: an efficient algorithm for mining erasable itemsets," *Engineering Applications of Artificial Intelligence*, Vol. 27, pp. 155-166, Jan. 2014.

[16] T. Le, B. Vo, and S. W. Baik, "Efficient algorithms for mining top-rank-k erasable patterns using pruning strategies and the subsume concept," *Engineering Applications of Artificial Intelligence*, Vol. 68, pp. 1-9, Feb. 2018.

[17] G. Lee, U. Yun, H. Ryang, and D. Kim, "Erasable itemset mining over incremental databases with weight conditions," *Engineering Applications of Artificial Intelligence*, Vol. 52, pp. 213–234, June 2016.

[18] G. Nguyen, T. Le, B. Vo, and B. Le, "A new approach for mining top-rank-k erasable itemsets," *Asian Conf. on Intelligent Information and Database Systems*, pp. 73-82, 2014.

[19] B. Vo, T. Le, W. Pedrycz, G. Nguyen, and S. W. Baik, "Mining erasable itemsets with subset and superset itemset constraints," *Expert Systems with Applications*, Vol. 69, pp. 50-61, March 2017.

[20] U. Yun and G. Lee, "Sliding window based weighted erasable stream pattern mining for stream data applications," *Future Generation Computer Systems*, Vol. 59, pp. 1-20, Jan. 2016.