INTELLIGENT AUTOMATION & SOFT COMPUTING, 2017
https://doi.org/10.1080/10798587.2017.1340135

An Algorithm for Fast Mining Top-rank-k Frequent Patterns based on Node-list Data Structure

Qian Wang[a,b,c], Jiadong Ren[a,b], Darryl N Davis[c] and Yongqiang Cheng[c]

[a]College of Information Science and Engineering, Yanshan University, Qinhuangdao, Hebei, China; [b]Computer Virtual Technology and System Integration Laboratory of Hebei Province, China; [c]Department of Computer Science, University of Hull, Hull, UK

ABSTRACT

Frequent pattern mining usually requires much run time and memory usage. In some applications, only the patterns with top frequency rank are needed. Because of the limited pattern numbers, quality of the results is even more important than time and memory consumption. A Frequent Pattern algorithm for mining Top-rank-K patterns, FP_TopK, is proposed. It is based on a Node-list data structure extracted from FTPP-tree. Each node is with one or more triple sets, which contain supports, preorder and post-order transversal orders for candidate pattern generation and top-rank-k frequent pattern mining. FP_TopK uses the minimal support threshold for pruning strategy to guarantee that each pattern in the top-rank-k table is really frequent and this further improves the efficiency. Experiments are conducted to compare FP_TopK with iNTK and BTK on four datasets. The results show that FP_TopK achieves better performance.

KEYWORDS

Data mining; frequent pattern; top-rank-k frequent pattern; FTPP-tree; Node-list

1. Introduction

The task of frequent pattern mining is to discover the relationships between items in a data-set. It is important to build a knowledge base, which is the basic component of an expert system (Sadik, 2008) or decision support system (Chae et al., 2003). These systems can make a contribution to an intelligent life for people by providing concise and accurate results. Previous frequent pattern mining algorithms are usually based on Apriori (Agrawal & Srikant, 1994) and FP-growth (Han, Pei, & Yin, 2000). Apriori algorithm employs a candidate generation and test strategy to discover frequent patterns. It is expensive for repeatedly scanning the database and checking a large set of candidates. There are many improvements of Apriori (Shenoy et al., 2000; Zaki & Gouda, 2003) that achieve good performance by reducing the candidate number and the database scanning times. FP-growth algorithm mines frequent patterns by divide-and-conquer approach without candidate generation. It achieves better efficiency by adopting a condensed FP-tree data structure. More improved algorithms (Liu et al., 2004; Liu et al. , 2007; Tanbeer et al., 2008) of FP-growth are followed. However, when the datasets are sparse, building FP-tree and conditional pattern bases recurrently make the methods inefficient. There are also other various data structures, like lattice-based algorithms (Vo, Hong, & Le, 2013; Vo, Le, Hong, & Le, 2014) and node based algorithm (Deng, 2016; Le & Vo, 2014). Frequent pattern mining is still an active topic in data mining, ranging from various extended mining tasks (Deng, 2016; Lee & Yun, 2016; Vo et al., 2017) and a variety of new applications (Awad, Ekanayake, & Jenkins, 2010; Wei et al., 2015). In general, frequent pattern mining needs a minimal support threshold to generate real frequent patterns. Whether the threshold is large or small, there will be too many frequent patterns for those applications such as the expert systems and so on. Because only a small scale of the frequent patterns are used in the final results. To this point, TFP algorithm (Wang et al. , 2005) is proposed for mining top-k frequent closed patterns, and k is the desired number of frequent closed patterns to be mined. It does not use a minimal support threshold, and a threshold min_l is set as the minimal length of each pattern. However, it is not easy to decide the value of min_l and new algorithms for top-rank-k pattern mining are proposed to solve the problem. FAE (Deng & Fang, 2007) and VTK (Fang & Deng, 2008) select patterns according to their frequency rank instead of using min_l. FAE reduces the searching space by using heuristic rules. It filters the undesired patterns and the useful patterns are selected for pattern extension. VTK performs better than FAE, because it achieves the desired results by intersecting the Tid-lists of candidate frequent patterns without scanning the entire data-set. NTK algorithm (Deng, 2014) is built for mining top-rank-k frequent patterns using a Node-list structure extracted from a PPC-tree, which is helpful for reducing the run time and memory consumption, but NTK must always generate and test all the candidates. iNTK (Huynh-Thi-Le et al., 2015) uses an improved N-list structure and employs the subsume index without candidate generation to achieve higher efficiency, however, it costs lots of time for finding subsume index especially when the data-set is sparse. These algorithms mine top-rank-k frequent patterns without minimal support threshold. BTK (Dam et al., 2016) employs a TB-tree structure and a B-list structure for mining top-rank-k frequent patterns, and pruning strategy with minimal support threshold is also used, but its efficiency is also dropped if the database is sparse.

This paper presents top-rank-k frequent pattern mining algorithm using Node-list data structure, called FP_TopK.

CONTACT Jiadong Ren jdren@ysu.edu.cn

Table 1. A Database DB.

TID	Items
1	a,b
2	a,b,c,d
3	a,c,e
4	a,b,c,e
5	c,d,e,f
6	c,d

Table 2. Rank of the Items in DB.

Rank	Support	Items
1	5	c
2	4	a
3	3	b,e,d
4	1	f

Table 3. The Sorted Database DB.

TID	Sorted Items
1	a,b
2	c,a,b,d
3	c,a,e
4	c,a,b,e
5	c,d,e
6	c,d

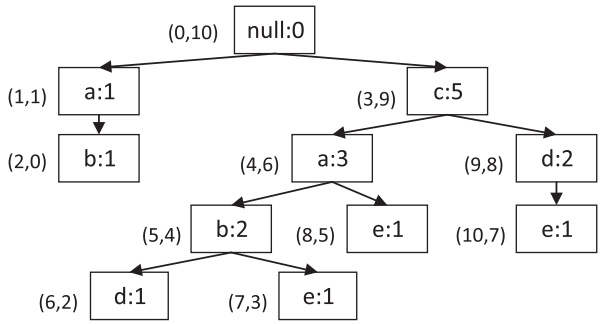


Figure 1. FTPP-tree for DB.

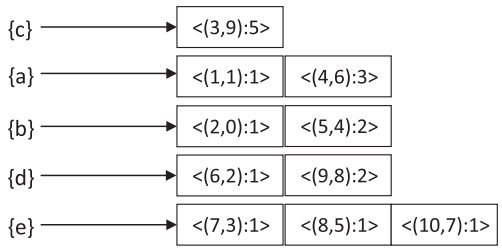


Figure 2. Node-lists of all the 1-patterns in Tab_4.

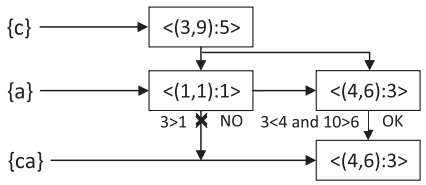


Figure 3. Node-list Generation of 2-pattern ca.

With a minimal support threshold, a frequent pattern must satisfy the minimal support requirement. FTPP-tree is built by frequent 1-patterns instead of using all the 1-patterns like PCC-tree. The number of Node-lists is reduced, because there are fewer nodes in the FTPP-tree. The minimal support threshold works for pruning infrequent patterns and guaranteeing the quality of the top-rank-k patterns. It can interact with value of rank k. When there is specific requirement for frequency, minimal support plays a leading role. And its value can be adjusted not to affect the top-k result when rank k is dominant.

The remainder of the paper is organized as follows: Section 2 introduces the problem definition. Section 3 develops FP_TopK algorithm and gives some examples. Section 4 presents our performance study. Section 5 contains concluding remarks.

2. Problem Definition

Let $I = \{I_1, I_2, I_3 \ldots I_m\}$ be a set of items, and $DB = \{T_1, T_2, T_3 \ldots T_n\}$ be a database, where each transaction $T_i (1 < i < n)$ is a set of items such that $T_i \subseteq I$. Given a pattern P, it is said that T contains P if $P \subseteq T$. If P contains t items, P is a t-pattern. Given a database DB and a pattern P. The support of the P in DB, denoted as Sup(P), is the number of the transactions containing P. A pattern P is a frequent pattern if Sup(P) is no less than a minimal support min_sup. The min_sup is calculated by $\xi^*|DB|$ where ξ is a given threshold and $|DB|$ is the number of transaction in DB.

2.1. A Problem of Top-rank-k Frequent Patterns

Deng et al.[28] described the problem of mining top-rank-k frequent patterns, and here some improvements are made.

Definition 1 (The rank of a pattern). Given a transaction database DB and a pattern $A(A \subseteq I)$, R_A, the rank of A, is defined as $R_A = |\{Sup(X)|X \subseteq I \text{ and } Sup(X) \geq Sup(A)\}|$, where $|Y|$ is the number of elements in Y.

Definition 2 (Top-rank-k frequent patterns). Given a transaction database DB, a rank threshold k, and minimal support min_sup, a pattern $A(A \subseteq I)$ is called to be a top-rank-k frequent pattern if and only if $R_A \leq k$ and $Sup(A) \geq min\_sup$.

Property 1 (Anti-monotone). If A is not a top-rank-k frequent pattern, any pattern B containing A, which is also called superset of A, cannot be a top-rank-k frequent pattern.

Definition 3 (Top-rank-k frequent table). A top-rank-k frequent table Tab_k records the top-rand-k frequent patterns with their rank and support.

Patterns with the same support are stored in the same entry of Tab_k. The number of entries in the top-rank-k table is no more than the threshold k.

Example 1. Table 1 shows a database DB. Sup(c) = 5, because there are five transactions, which contain c. Table 2 shows the supports and the ranks of all the 1-patterns. According to Table 2, Sup(c) is the biggest, so $R_c = 1$. Let rank = 4, min_sup = 2. Tab_4 for the frequent 1-pattern can be obtained after deleting the fourth row of Table 2. Although $R_f = 4$, it is not a frequent 1-pattern.

2.2. Node-list Structure

Deng et al.[28] presented the PPC-tree structure, and Node-list is extracted from PPC-tree. FP_TopK algorithm also uses Node-list structure, and it is created from FTPP-tree.

Definition 4 (FTPP-tree). FTPP-tree is a tree structure, which includes one root and a set of top-rank-k frequent 1-pattern nodes. Each node N is composed of five values; N.name, N.child, N.count, N.pre and N.post. N.name is the 1-pattern name, N.child is all the children of node N, N.count is Sup(N.

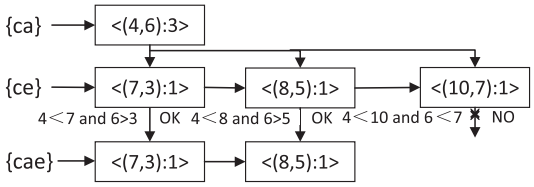


Figure 4. Node-list Formation of 3-pattern cae.

name), N.pre and N.post are the preorder and post-order of node N, respectively. The root of FTPP-tree, which is named R has R.name = null and R.count = 0.

Example 2. Table 3 shows the sorted database, and item f is removed. Figure 1 shows the FTPP-tree. In each rectangle, there are names and support of the 1-pattern. Next to the rectangle, pre-order and post-order of the 1-pattern are stored in pairs in a bracket.

Definition 5 (Node-list of a top-rank-k frequent 1-pattern). Given a FTPP-tree, Node-list of a top-rank-k frequent 1-pattern A is a sequence of all the PP-codes of nodes in the FTPP-tree whose name is A. In a Node-list, PP-codes are arranged in pre-order ascending order. Each PP-code in Node-list is denoted by PP = < (pre, post): count >. Node-list of a top-rank-k frequent pattern is denoted by $\{PP_1, PP_2, \cdots, PP_n\}$, where $PP_1.pre < PP_2.pre < \cdots < PP_n.pre$.

Property 2 (Ancestor-descendant relationship of 1-pattern PP-codes). Given PP_i and PP_j are two PP-codes, PP_i is an ancestor of PP_j if and only if $PP_i.pre < PP_j.pre$ and $PP_i.post > PP_j.post$.

Example 3. Node-list of 1-pattern b contains two PP-codes, $b.PP_1$ = < (2,0):1 > and $b.PP_2$ = < (5,4):2 >. They are arranged in pre-order ascending order. Figure 2 shows the Node-lists of all the top-rank-k frequent 1-patterns. PP-code for 1-pattern c is $c.PP_1$ = < (3,9):5 >, PP-codes for a is $a.PP_1$ = < (1,1):1 > and $a.PP_2$ = < (4,6):3 >. $c.PP_1$ is an ancestor of $a.PP_2$ = < (4,6):3>,because $c.PP_1.pre = 3 < a.PP_2.pre = 4$, $c.PP_1.post = 9 > a.PP_2.post = 6$. But $c.PP_1$ is not an ancestor of $a.PP_1$.

Definition 6 (Node-list of a top-rank-k frequent t-pattern). Let two t-patterns be PX_1 and PX_2, where PX_1.Node-list = $\{PP_{11}, PP_{12}, \ldots, PP_{1\,m}\}$ and PX_2.Node-list = $\{PP_{21}, PP_{22}, \ldots, PP_{2n}\}$, respectively. $\forall PP_i \in PX_1.NL(1 \le i \le m)$ and $PP_j \in PX_2.NL(1 \le j \le n)$, if $PP_i.pre < PP_j.pre$ and $PP_i.post > PP_j.post$, add the PP-code PX_2.Node-list[j] = < (PP_j.pre, PP_j.post):PP_j.count > to Node-list of PX_1X_2.

Property 3 (Support acquisition). P is a t-pattern and its Node-list P.Node-list = $\{PP_1, PP_2, \ldots, PP_n\}$. The support of P is determined by $Sup(P) = PP_1.count + \cdots + PP_n.count$.

Example 4. As shown in Figure 3. Take 1-pattern c and a as an example. c.Node-list[1] = < (3, 9):5 > and a.Node-list[1] = < (1, 1):1 >, a.Node-list[2] = < (4, 6):3 >. The ca.Node-list generation of candidate 2-pattern ca is as follows:

(1) c.Node-list[1].pre = 3 > a.Node-list[1].pre = 1. Connection is failed and it is then moved to a.Node-list[2].
(2) c.Node-list[1].pre = 3 < a.Node-list[2].pre = 4, c.Node-list[1].post = 9 > a.Node-list[2].post = 6. Then ca.Node-list[1] = a.Node-list[2] = < (4,6):3 >, Sup(ca) = 3.

Example 5. As shown in Figure 4. Take 2-pattern ca and ce as an example. ca.Node-list[1] = < (4, 6):3 > and ce.Node-list[1] = < (7, 3):1 >, ce.Node-list[2] = < (8, 5):1 >, ce.Node-list[3] = < (10, 7):1 >. The cae.Node-list generation of candidate 3-pattern cae is as follows:

(1) ca.Node-list[1].pre = 4<ce.Nnode-list[1].pre = 7, and ca.Node-list[1].post = 6 > ce.Node-list[1].post = 3. Connection is OK.
(2) ca.Node-list[1].pre = 4<ce.Node-list[2].pre = 8 and ca.Node-list[1].post = 6 > ce.Node-list[2].post = 5. OK.
(3) ca.Node-list[1].pre = 4<ce.Node-list[3].pre = 10 and ca.Node-list[1].post = 6<ce.Node-list[3].post = 7. Failed.
(4) cae.Node-list[1] = ce.Node-list[1] = < (7,3):1 >, cae.Node-list[2] = ce.Node-list[2] = < (8,5):1 >, Sup(cae) = 1+1 = 2.

3. FP_TopK Algorithm

3.1. Node-list Intersection Function

In FP_TopK algorithm, Node-list is extracted from FTPP-tree, so the function for Node-list intersection is presented after the FTPP-tree construction algorithm.

```
Function FTPP-tree-construction(DB, min_sup, k)
1. Scan DB and insert all frequent items whose supports are larger than
   min_sup and their supports to L1.
2. Sort L1 in support descending order and insert the top k items into Tabk.
   If the supports of some items are equal, they have the same rank and the
   orders among them in the transactions can be assigned by alphabetical or-
   der.
3. Create the root of a FTPP-tree, R, and name it as null.
4. For each transaction Tr in DB do.
5.    Sort all items in Tabk according to the support descending order.
6.    Call Insert_Tree(Tr, R).
7. Visit the FTPP-tree to generate the pre-order and the post-order values of
   each node by pre-order traverse and post-order traverse, respectively.
Function Insert_Tree(Tr, R)
1. t is the first element in Tr, Tr = Tr\t.
2. If R has a child node N such that N.name = t then N.count++.
3. Else create a new node N with N.count = 1 and N.name = t, R.child = N.
4. If Tr is not null then call Insert_Tree(Tr, N).

Function NL-intersection(NL1, NL2)
1. NL3 = ∅.
2. Let i = 0, j = 0.
3. While i < |NL1| and j < |NL2| do.
4.    If NL1[i].pre < NL2[j].pre.
5.       If NL1[i].post > NL2[j].post.
6.          Add the tuple (NL2[j].pre, NL2[j].post): NL2[j].count) to NL3.
7.          j++.
8.       Else.
9.          i+.+
10.   Else.
11.      j++.
12. Return NL3.
```

3.2. The Processing Procedure of FP_TopK Algorithm

FP_TopK algorithm explores (t+1)-patterns from t-patterns. By using FTPP-tree structure to extract Node-lists, FP_TopK algorithm can get the support of (t+1)-patterns without repeatedly scanning the database. When the PP-codes of two different Node-lists don't match each other, they immediately know the problem and decide who should move ahead. This can reduce the comparison times and further decrease the run time. The FP_TopK algorithm and the processing procedure are as follows:

```
Procedure FP_TopK(DB, k)
1. Call FTPP-tree-construction(DB, min_sup, k) to build FTPP-tree.
2. Determine the Node-list of Tabk(top-rank-k frequent 1-patterns).
3. Find 1-patterns in Tabk denote the set of these 1-patterns as TR1.
4. For(j = 2; TRj-1≠∅; j++).
```

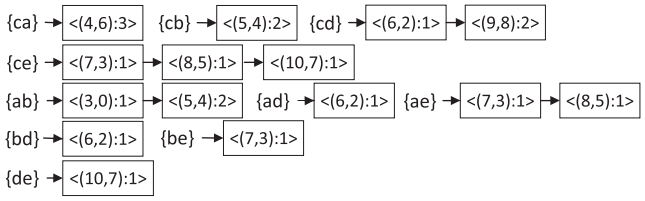


Figure 5. Node-lists for Candidate 2-patterns.

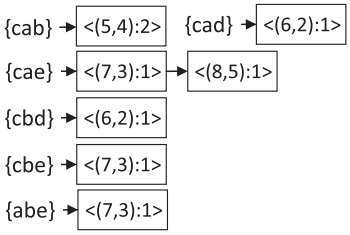


Figure 6. Node-lists for Candidate 3-patterns.

Table 4. Tab_4 after 2-patterns Inserted.

k	Sup(X)	Patterns
1	5	c
2	4	a
3	3	b,d,e,ca,cd,ce,*ab*
4	2	cb,ae

Table 5. Final Results for Tab_4.

k	Sup(X)	Patterns
1	5	c
2	4	a
3	3	b,d,e,ba,dc,ec,ca
4	2	ae,cb,cae,cab

Table 6. Characteristics of the Experimental Datasets.

Data-set	Average transaction length	Item number	Transaction number
Connect	43	130	67,557
Mushroom	23	119	8124
Retail	10.3	16,470	88,162
T25I10D100 K	25	990	99,822

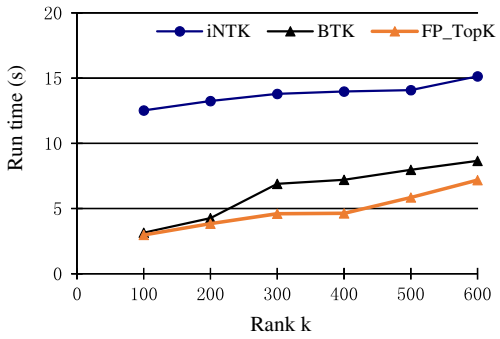


Figure 7. Run Time on Connect Data-set.

5. CR_j = **Candidate_gen**(TR_{j-1}).
6. For each C∈CR_j, C is generated by P_1(∈TR_{j-1}) and P_2(∈TR_{j-1}).
7. C.Node-list = Node-list-intersection(P_1.Node-list, P_2.Node-list).
8. If Sup(C) is equal to the support of the patterns in any entry of Tab_k.
9. Insert C with its support into the same rank of Tab_k together with other patterns who have the same support.
10. If Sup(C) is larger than min_sup and smaller than the support of the patterns in any entry of Tab_k and the number of the entries is less than k.
11. Insert C into the Tab_k as the last rank pattern.
12. If C∈Tab_k then TR_j = TR_j∪{C}.

Procedure **Candidate_gen**(TR_{j-1})

1. CR_j = ∅.
2. For each C_u∈TR_{j-1}.
3. For each C_v∈TR_{j-1}(v > u).
4. If (C_u[1] = C_v[1]^C_u[2] = C_v[2]^⋯^C_u[j−2] = C_v[j−2]^C_u[j−1]≠C_v[j−1]).
5. C = C_u[1]C_u[2]C_u[j−2]C_u[j−1]C_v[j−1].
6. CR_j = CR_j∪{C}.
7. Return CR_j.

(1) Scan the database, find the top-rank-k frequent 1-patterns and insert them into top-rank-k frequent table Tab_k.
(2) Sort the transactions according to the rank of the 1-patterns in Tab_k and form FTPP-tree.
(3) Scan the FTPP-tree and generate Node-lists of all 1-patterns in Tab_k.
(4) For each t-pattern X and another t-pattern Y in Tab_k, FP_TopK finds all candidate (t+1)-patterns by combining X with Y, satisfying X[1] = Y[1]^X[2] = Y[2]^⋯^X[t−2] = Y[t−2]^X[t−1]≠Y[t−1]), X.NL[i].pre < Y.NL[j].pre and X.NL[i].post > Y.NL[j].post.
(5) Each candidate t-pattern whose support is equal to any entry of Tab_k is inserted to the entry. If the support is larger than min_sup and less than the smallest support of the Tab_k, and the number of entries in Tab_k is no more than k, then the t-pattern will be inserted into the last entry of Tab_k. The supports of the t-patterns cannot be larger than any entries of Tab_k, because the support of their supper patterns must be larger or equal to theirs.
(6) Repeat steps 4 and 5 until no new candidate patterns can be generated.

3.3. An Illustrative Example

Let rank = 4, min_sup = 2. The process of mining top-rank-4 frequent patterns from the database DB in Table 1 is as follows:

(1) Find top-rank-4 frequent patterns and form the top-rank-4 frequent table Tab_4. The result is shown in Table 2 without the last row.
(2) Sort the transactions in DB and form the FTPP-tree as shown in Table 3 and Figure 1.
(3) Extract Node-list from FTPP-tree as shown in Figure 2.
(4) Candidates are generated as shown in Figure 3 and Figure 4. Candidate 2-patterns and their Node-lists are listed as shown in Figure 5.

Sup(ca) = 3, Sup(cd) = 3, Sup(ce) = 3, Sup(*ab*) = 3. Sup(cb) = 2, Sup(ae) = 2.

Sup(de) = 1, Sup(ad) = 1,Sup(bd) = 1, Sup(be) = 1.

2-patterns de, ad and bd are infrequent, because their supports are smaller than min_sup. 2-patterns ca, cd, ce, *ab* are inserted into the third rank entry of Tab_4. 2-patterns cb, ae are inserted into the fourth rank entry of Tab_4 as shown in Table 4.

Node-lists for candidate 3-patterns are shown in Figure 6.

Sup(cab) = 2, Sup(cae) = 2. Sup(cad) = 1, Sup(cbd) = 1, Sup(cbe) = 1, Sup(abe) = 1.

3-patterns cad, cbd and cbe are infrequent, because their supports are smaller than min_sup. 3-patterns cab, cae are inserted into the fourth rank entry of Tab_4. Table 5 is the final results.

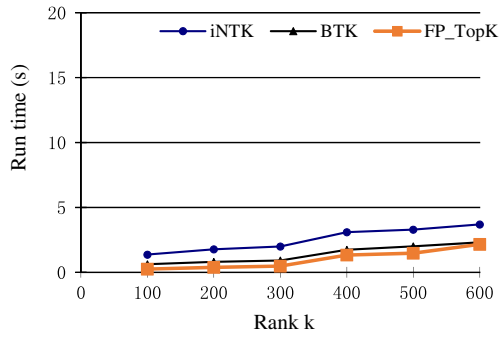


Figure 8. Run Time on Mushroom Data-set.

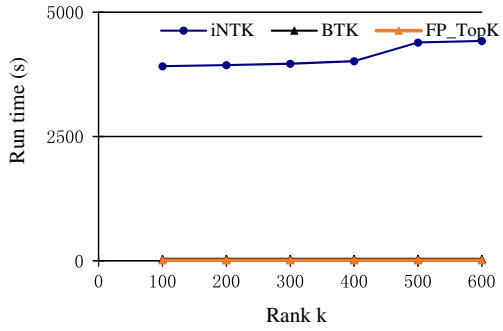


Figure 9. Run Time on Retail Data-set.

4. Experiment Results

Experiments are performed on a PC with Intel(R) Core(TM) 3.6 GHz CPU and 16G main memory, running on Windows 8. We evaluate the run time of FP_TopK algorithm, and compare it with iNTK and BTK using different rank k thresholds. Minimal support threshold is used in FP_TopK and BTK, but not in iNTK. There are three real datasets; Connect, Mushroom and Retail downloaded from https://fimi.ua.ac.be/data/ and a synthetic data-set T25I10D100 K generated by the IBM data generator. To test the algorithms in the same coding environment, all the programs are written in C++ using Visual Studio 2013. Table 6 shows the characteristics of these datasets, including the average transaction length, the item number and the transaction number.

The run time comparison of FP_TopK with iNTK and BTK is shown in Figures 7–10. It is noted that, here run time means the total execution time, from input to output. The min_sup is set to 5%, which is as small as possible not to affect the mining results of top-rank-k patterns (Retail data-set is affected). Then run time of the two algorithms can be fairly compared.

The experimental results show that FP_TopK outperforms iNTK more and BTK less for all the values of k. This is because FP_TopK can save time in every link than iNTK. At the beginning fewer nodes are built in the FTPP-tree than PPC-tree. This can lead to less Node-list creation, calculation and frequent top-rank-k table insertion. Although iNTK uses subsume indexes to avoid an unnecessary connection, subsume index generation requires a cost of time and the cost is even worse for small value of rank k or sparse datasets. BTK is similar to iNTK as they use the same subsume index strategy, but it also uses the min_sup to do early pruning, so its performance is better than iNTK and close to FP_TopK, when the value of rank k is small or the data-set is sparse, iNTK is not effective and BTK is more or less interrupted.

FP_TopK performs well in both dense and sparse data-sets. It prunes infrequent patterns in the early stage, and more

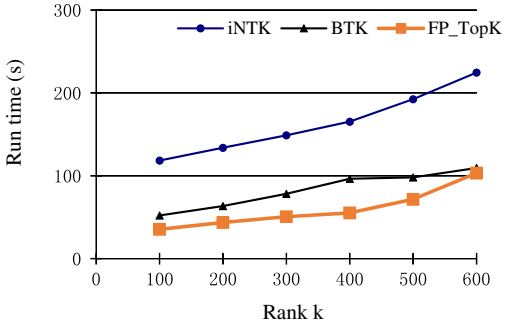


Figure 10. Run Time on T25I10D100 K Data-set.

Table 7. Number of Returned Item Sets for Each Real Data-set(min_sup=5%, rank=100).

AlgorithmData-set	FP_TopK(L_1/Tab_{100})	iNTK(L_1/Tab_{100})	BTK(L_1/Tab_{100})
Connect	86/100	100/100	86/100
Mushroom	73/100	100/100	73/100
Retail	6/16	100/100	6/16

infrequent patterns will be pruned in the later process for sparse datasets. Among the four datasets, the two real datasets; Connect and Mushroom are very dense, another real data-set; Retail and the synthetic data-set T25I10D100 K are much sparser. In Figure 9, it is irregular that iNTK runs much slower than FP_TopK. This is, because the data characteristic of Retail data-set is too sparse. Even the min_sup is small, a lot of the infrequent patterns are pruned by FP_TopK at the beginning, however, these infrequent patterns contain many elements in the subsume indexes, and most of the run time in iNTK is for generating the subsume indexes. The numbers of frequent 1-patterns and top-rank-100 frequent patterns for each real data-set and algorithm are shown in Table 7. For FP_TopK and BTK, the pattern contained in L_1 is smaller than Tab_{100} that means less patterns are needed to be frequency-checked and less candidates need to be generated. For iNTK, lots of unwanted patterns and their growth patterns will first enter and then leave the Tab_{100}. That is why the efficiency of iNTK is the lowest.

It seems that if the value of rank k is small, the patterns with small support usually have less chance to enter the top-rank-k table. In fact, the support is large or small is relative. If there are too many small supports, then patterns with poor quality will be inserted into the table. Min_sup is necessary to guarantee that each pattern in the top-rank-k table is meaningful. Whether it affects the mining result or not it can be effectively helpful to save run time.

5. Conclusion

This paper presents an algorithm called FP_TopK for fast mining top-rank-k frequent patterns based on Node-list data structure. FP_TopK prunes the infrequent items and builds FTPP-tree with fewer nodes. This further avoids unnecessary Node-list creation and frequent top-rank-k table insertion, which makes the run time and memory usage reduced. FP_TopK is effective for both dense and sparse datasets. The minimal support threshold can satisfy the frequency requirement and guarantee the quality of the patterns in the top-rank-k table. Even if it is small enough not to affect the top-rank-k patterns, it is helpful for efficiency improvement. After analyzing the experimental results, FP_TopK is proven to be efficient.

Acknowledgment

The authors are grateful to valuable comments and suggestions of the reviewers.

Disclosure Statement

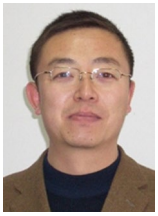
No potential conflict of interest was reported by the authors.

Funding

This work is supported by the National Natural Science Foundation of China under Grant No. 61472341, No.61572420, and the Natural Science Foundation of Hebei Province P.R.China under Grant No. F2014203152, and No.F2016203330. It is also supported by China Scholarship Council.

Notes on Contributors

Qian Wang received Ph.D. degree in Computer Software and Theory from Yanshan University, China, in June 2016. Since 2016, she has been a lecturer at the College of Information Science and Engineering, Yanshan University, China. Her current research interests include data mining, software security.



Jiadong Ren received Ph.D. degree in Computer Application Technology from Harbin Institute of Technology, China, in December 1999. Since 2005, he has been a professor at the College of Information Science and Engineering, Yanshan University, China. His current research interests include data mining, software security.

Darryl N Davis received Ph.D. degree in Wolfson Image Analysis Unit from Victoria University of Manchester, United Kingdom, in June 1991. Since 1999, he has been a senior lecturer at the School of Engineering and Computer Science, University of Hull, United Kingdom. His current research interests include data mining, cognition and affect, agents and artificial life.

Yongqiang Cheng received Ph.D. degree in Design and Technology from University of Bradford, United Kingdom, in June 2010. Since 2014, he has been a lecturer at the School of Engineering and Computer Science, University of Hull, United Kingdom. His current research interests include data mining, AI, embedded systems.

References

Agrawal, R., & Srikant, R. (1994, September). Fast algorithms for mining association rules. In *Proceedings 20th international conference very large data bases, VLDB* (Vol. 1215, pp. 487–499)

Awad, B., Ekanayake, J., & Jenkins, N. (2010). Intelligent load control for frequency regulation in microgrids. *Intelligent Automation & Soft Computing, 16*, 303–318.

Chae, Y.M., Kim, H.S., Tark, K.C., Park, H.J., & Ho, S.H. (2003). Analysis of healthcare quality indicator using data mining and decision support system. *Expert Systems with Applications, 24*, 167–172.

Dam, T.L., Li, K., Fournier-Viger, P., & Duong, Q.H. (2016). An efficient algorithm for mining top-rank- k, frequent patterns. *Applied Intelligence, 45*, 96–111.

Deng, Z.H., & Fang, G.D. (2007, August). Mining top-rank-K frequent patterns. In *Machine Learning and Cybernetics* (Vol. 2, pp. 851–856). IEEE.

Deng, Z.H. (2014). Fast mining top-rank-k frequent patterns by using node-lists. *Expert Systems with Applications, 41*, 1763–1768.

Deng, Z.H. (2016). Diffnodesets: An efficient structure for fast mining frequent item sets. *Computer Science, 41*, 214–223.

Fang, G.D., & Deng, Z.H. (2008, October). VTK: Vertical mining of top-rank-k frequent patterns. In *International Conference on Fuzzy Systems and Knowledge Discovery*, (Vol. 2, pp. 620–624). IEEE.

Han, J., Pei, J., & Yin, Y. (2000, May). Mining frequent patterns without candidate generation. In *Acm Sigmod Record* Vol. 29. (No. 2, pp. 1–12).

Huynh-Thi-Le, Q., Le, T., Vo, B., & Le, B. (2015). An efficient and effective algorithm for mining top-rank-k frequent patterns. *Expert Systems with Applications, 42*, 156–164.

Jianyong Wang, J., Han, J., Lu, Y., & Tzvetkov, P. (2005). TFP: an efficient algorithm for mining top-k frequent closed item sets. *IEEE Transactions on Knowledge and Data Engineering, 17*, 652–663.

Le, T., & Vo, B. (2014). MEI: An efficient algorithm for mining erasable item sets. *Engineering Applications of Artificial Intelligence, 27*, 155–166.

Lee, G., & Yun, U. (2016). A new efficient approach for mining uncertain frequent patterns using minimum data structure without false positives. *Future Generation Computer Systems, 68*, 89–110.

Liu, G., Lu, H., Lou, W., Xu, Y., & Yu, J.X. (2004). Efficient mining of frequent patterns using ascending frequency ordered prefix-tree. *Data Mining and Knowledge Discovery, 9*, 249–274.

Liu, G., Lu, H., & Xu Yu, J.X. (2007). CFP-tree: A compact disk-based structure for storing and querying frequent item sets. *Information Systems, 32*, 295–319.

Shenoy, P., Haritsa, J.R., Sudarshan, S., Bhalotia, G., Bawa, M., & Shah, D. (2000). Turbo-charging vertical mining of large databases. In *Acm Sigmod Record* (Vol. 29, No. 2, pp. 22–33).

Sadik, AT. (2008). Premises reduction of rule based expert systems using association rules technique. *International Journal of Soft Computing, 3*, 195–200.

Tanbeer, S.K., Ahmed, C.F., Jeong, B.S., & Lee, Y.K. (2008, October). Efficient frequent pattern mining over data streams. In *Proceedings of the 17th ACM conference on Information and knowledge management* (pp. 1447–1448).

Vo, B., Hong, T.P., & Le, B. (2013). A lattice-based approach for mining most generalization association rules. *Knowledge-Based Systems, 45*, 20–30.

Vo, B., Le, T., Hong, T.P., & Le, B. (2014). An effective approach for maintenance of pre-large-based frequent-item set lattice in incremental mining. *Applied Intelligence, 41*, 759–775.

Vo, B., Pham, S., Le, T., & Deng, Z.H. (2017). A novel approach for mining maximal frequent patterns. *Expert Systems with Applications, 73*, 178–186.

Wei, X., Wang, Y., Li, Z., Zou, T., & Yang, G. (2015). Mining users interest navigation patterns using improved ant colony optimization. *Intelligent Automation & Soft Computing, 21*, 445–454.

Zaki, M.J., & Gouda, K. (2003, August). Fast vertical mining using diffsets. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 326–335).