A FP-Tree Based Algorithm for mining knowledge point association

Bingtao Hu and Xiaoming Ju

No.3663 North Zhongshan Rd. Shanghai, *East China Normal University* ,China

No.3663 North Zhongshan Rd. Shanghai, *East China Normal University* ,China

hbttao@163.com, xmju@sei.ecnu.edu.cn

Keywords: Knowledge point association, DFP-tree, multiple minimum support, frequent pattern

Abstract. In online educational courses, knowledge point is the basic unit used for the organization and dissemination of information. Mining the association of knowledge points can help to improve the navigation of course learning. However, it is difficult for the existing mining algorithms to balance the performance and accuracy of mining the association of knowledge points. This paper proposes a FP-tree based algorithm used for mining the association of knowledge points. With the multiple minimum support strategy, this algorithm modifies the setting mode of each item's minimum support and improves the construction method of FP-tree to adapt to the dynamic database mining. The experiment shows that the algorithm improved the accuracy of the mining results and ensured a good performance.

Introduction

According to the global education industry report from GSV (Global Silicon Valley), the market size of global online education is expected to increase to $ 255.5 billion in 2017. More and more people start to learn knowledge by online education. Knowledge point (defined in reference [3]) is the basic unit used for the organization and dissemination of information in online education. Mining the association of knowledge points is of great significance for improving the navigation and recommendation of courses. Nowadays most researches about the association of knowledge points use traditional data mining algorithms to analyze the users' behavior data directly, without improvement and optimization combined with the features of the association of knowledge points. Hence, the performance and accuracy can't be obtained at the same time in these algorithms. Therefore, it is necessary to find a specialized algorithm used for mining the sets of frequent knowledge points. In this paper, by combining with the features of the association of knowledge points, a new algorithm will be proposed. This algorithm simplifies the setting mode of each item's minimum support in the multiple minimum support strategy and improves the construction method of FP-tree to adapt to the dynamic database mining. Furthermore, a new data structure named frequent conditions pattern filter list is introduced into the algorithm, with the help of the list, the algorithm will have a higher performance in the situation where the minimum support changes continually. In the experiment, original data collected from SLMS (Sophia Learning Manage System) is 70 students' online learning record data of the Android App Development Course. By comparison of the results, the algorithm is better adapted to mining the association of knowledge points.

Related work

In the research area of mining the association of knowledge points, most researches process and analyze the users' behavior data to get the association of knowledge points by traditional data mining algorithms. Therefore, with the improvement and innovation of association rule mining algorithm, the research for association of knowledge points also progresses constantly. Liu. B et al. proposed MSapriori algorithm which allows user to specify multiple minimum item supports. This algorithm solved rare item problem, but the performance of the algorithm wasn't enhanced obviously. J. Han et al. proposed FP-growth algorithm which uses FP-tree to find the set of frequent pattern, and FP-tree is

an extended prefix-tree for storing compressed and crucial information about frequent patterns. FP-growth algorithm is much more efficient than Apriori algorithm. Hu, Y. et al. proposed an extended FP-growth algorithm combined with multiple minimum support model. However, each minimum support of item need to be specified by user. Yang, H. et al. proposed a multi-support mining algorithm based on knowledge points. In this algorithm, each minimum support of item will be calculated from the two aspects which include learning and testing for the corresponding knowledge point. But the calculation method of minimum support is too complex and the algorithm is not very efficient.

A FP-Tree-Based Algorithm for mining the sets of frequent knowledge points.

Compression of transaction database. FP-tree (defined in reference [2]) is a specific data structure used for storing the compressed frequent patterns in transaction database. After building the FP-tree for the transaction database, all of the frequent pattern can be obtained from FP-tree by FP-growth algorithm. But FP-tree only stores the information of frequent item. Thus FP-growth algorithm is only used for the static database. If the transaction database changes, the FP-tree needs to be built again.

This paper presents an improved FP-tree data structure named DFP-tree (Dynamic Frequent Pattern tree) which can be better adapted to the dynamic database mining. Instead of only storing the information of frequent item, all the transaction database will be compressed and stored into DFP-tree during the building process of DFP-tree. According to Fig. 1, this paper introduce the building process of DFP-tree through an example.

Algorithm 1(DFP-tree construction)
Input: transaction database DB.
Output: Its DFP-tree.
Method:
1. For each transaction in DB to get the actual support of each item in DB.
2. Create the root of a DFP-tree T, and label it as “null”.
3. For each transaction t in DB do the following:
   (1). Sort all items in t according to their actual support in decreasing order.
   (2). Call insert-tree([p|P],T).

Algorithm 2(DFP-tree dynamic data processing)
Input: newly added transaction data NDB, the DFP-tree of previous DB.
Output: updated DFP-tree
Method:
1. For each transaction in NDB to get the actual support of each item in NDB.
2. Update the support of each item and sort according to their actual support in decreasing order.
3. Check whether the index of each item in DB changed.
   If all of the indexes didn’t change:
      For each transaction t in NDB do the following.
         (1). Sort all items in t according to their actual support in decreasing order.
         (2). Call insert-tree ([p|P], T).
   Else call Algorithm 1 to rebuild DFP-tree.

Function insert-tree([p|P], T) (p is the first element and P is the rest in sorted items [p|P].)
While (P is not null){
   If T has a child N whose item-name = p.item-name then N.count++;
   else create a new node N, and N.count = 1;
      let N.parent-link link to T;
      let N.node-link link to the node which has the same item-name.
}

Fig. 1 The algorithms for DFP-tree construction.

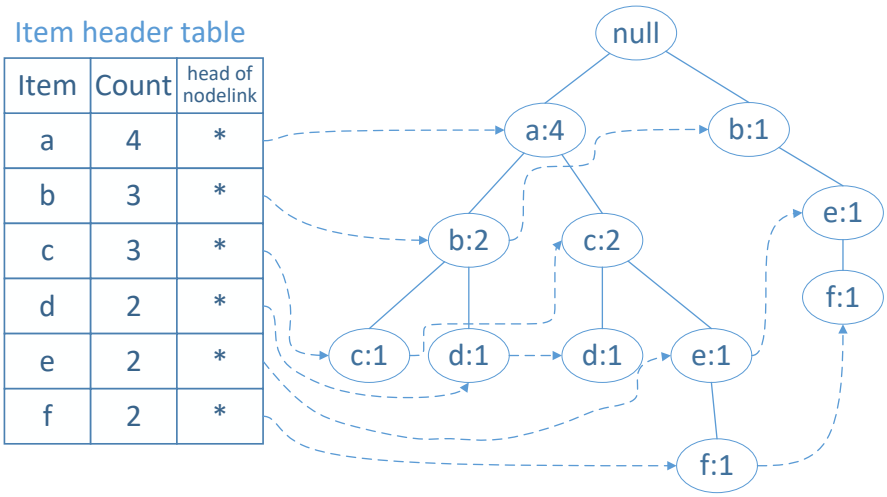

Fig. 2 The DFP-tree built from the example.

Fig. 1 shows the algorithm for building the DFP-tree. The transaction database is {(c,f,e,a),(d,a,b),(f,b,e),(c,d,a),(b,c,a)}. Firstly, the algorithm scan the transaction database to get the actual support of each item. Secondly, all the transactions is scanned again to sort the items according to their actual support in decreasing order. Thirdly, each sorted transaction will be inserted into the DFP-tree. Finally, the DFP-tree like Fig. 2 will be established.

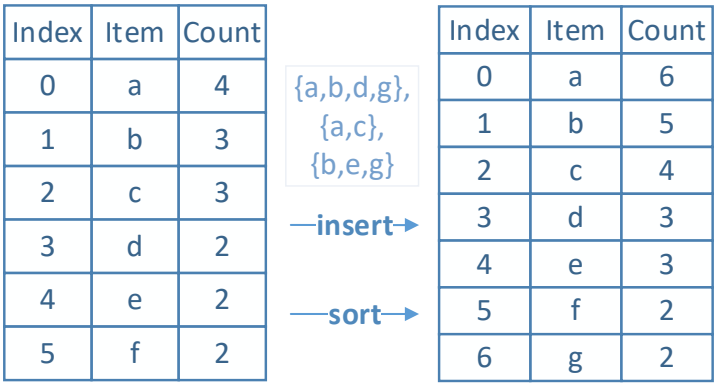

Fig. 3 New transaction is inserted into transaction database.

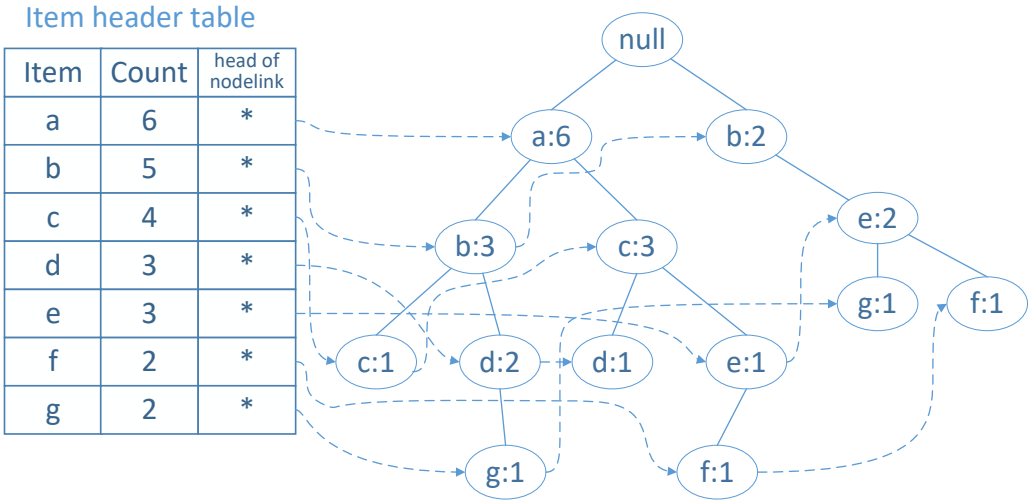

Fig. 4 The DFP-tree after transaction database changed.

As shown in Fig. 3, the support of each item is updated and sorted according to their actual support in decreasing order again when the size of transaction database increases. If all of the items' indexes didn't change like Fig. 3, the incremental data of transactions will be sorted and inserted into DFP-tree directly, otherwise, the DFP-tree will be built again. In the end, the new DFP-tree is shown like Fig. 4. After building the DFP-tree for the transaction database, it's time to get all the frequent pattern from DFP-tree by DFP-growth algorithm.

Mining frequent pattern with multiple minimum support strategy. In multiple minimum support strategy, for each item, it is not an easy job to set minimum support which mainly depends on experience. Thus, as is shown in Definition 1, this paper modifies the setting mode for minimum combined with the features of the association of knowledge points. Related definitions are as follow:

Definition 1. Let $I=\{a_1,a_2,\ldots,a_n\}$be a set of items, $MIS(a_i)$ $(1\leq i\leq n)$ denote the minimum support of a_i, $count(a_i)$ denote the actual support of a_i and r is the global support threshold, then $MIS(a_i)$ is presented by the following expression:

$$MIS(a_i) = count(a_i)*r. \quad (1)$$

Definition 2. Let $I=\{a_1,a_2,\ldots,a_n\}$be a set of items, $MIS(a_i)$ $(1\leqslant i\leqslant n)$ denote the minimum support of a_i and MIS(A) denote the minimum support of item-set $A=\{a_1,a_2,\ldots,a_k\}(1<k<n)$, then MIS(A) is presented by the following expression:

$$MIS(A) = \min[MIS(a_1), MIS(a_2),\ldots, MIS(a_k)]. \quad (2)$$

Definition 3. The pattern p is the a_i 's conditional pattern if a_i belongs to p and satisfies expression:

$$MIS(a_i) = MIS(p). \quad (3)$$

Definition 4. The pattern p is the frequent conditional pattern of a_i if p is the conditional pattern of a_i and p is the frequent pattern.

Definition 5. Frequent condition pattern filter list is defined as follows.

(1) It consist of each item's frequent conditional pattern linked list and item header table which contains all items in transaction database.

(2) Each node in frequent conditional pattern linked list consists of frequent-pattern-name, count and node-link, where frequent-pattern-name labels which pattern this node present, count records the number of transaction including this pattern, and node-link links to the next frequent conditional pattern of this item.

(3) Each entry in the item header table consist of item-name, count which is the actual support of this item, head and tail of frequent conditional pattern linked list.

Algorithm 3 (DFP-growth)
Input: DFP-tree, global support threshold r.
Output: all the frequent pattern and its support, frequent condition pattern filter list FCPF-list.
Method:
For each item i in item header table from bottom to top do the following:
(1).calculate to get item i's support MIS(i) according to the value of r.
(2).find all the paths in the DFP-tree including i by node-link.
(3).generate patterns p represented by the paths with support = i.support.
(4).build i's conditional DFP-tree T by patterns p and construct i's conditional pattern base.
(5).If DFP-tree T is not null: call conditional-DFP-growth (T, MIS(i)).

Algorithm 4 (conditional-DFP-growth)
Input: conditional-DFP-tree, MIS(i)
Output: all the conditional frequent patterns for i and its support
Method:
For each item t in item header table from bottom to top do the following:
(1).If t's support less than MIS(i) : continue the next iteration;

else: record the corresponding pattern and support to FCPF-list.
(2).find all the paths in the DFP-tree including t by node-link.
(3).generate patterns p represented by the paths with support = t.support.
(4).build i's conditional DFP-tree T by patterns p and construct t's conditional pattern base.
(5).If DFP-tree T is not null: call conditional-DFP-growth (T, MIS(i)).

Algorithm 5 (Multiple-r-DFP-growth)
Input: DFP-tree, global support threshold r's range, pace
Output: all the frequent patterns of different r
Method:
1. call DFP-growth(DFP-tree, the minimum value of r) to get FCPF-list.
2. let temp-r be equal to the minimum value of r.
3. While(temp-r < the maximum of r){
   traverse the FCPF-list to get the frequent patterns when the global support threshold is temp-r;
   temp-r = temp-r + pace;
   }

Fig. 5 The DFP-growth algorithm.

DFP-growth algorithm builds conditional DFP-tree recursively to obtain all the frequent patterns at last. Fig. 6 shows the process of finding item d's the frequent conditional pattern. First of all, following the node-link of d, two paths which end at node d can be found in DFP-tree: a-b-d and a-c-d. Then let the count of each node in the two paths be equal to node d, respectively. After adding these two paths, the conditional DFP-tree for item d is shown in Fig. 6(a). According to Definition-2 and Definition-4, whether an item is frequent in d's conditional DFP-tree is judged by checking whether its support exceeds *MIS*（d）.In this example, the value of r is 0.5 and the support of d is 3, according to Definition-1, MIS(d) = 3*0.5=1.5. In the conditional DFP-tree of d, the count of (a, b, c) are (3, 2, 1), respectively. Hence, item a, b is frequent in conditional DFP-tree, in other word, patters a-d and b-d are d's frequent conditional patterns. Then the algorithm build a-d and b-d's conditional pattern base and conditional DFP-tree, respectively. There is no item in a-d's conditional pattern base, then the algorithm only build the conditional DFP-tree for b-d shown in Fig. 6(b).

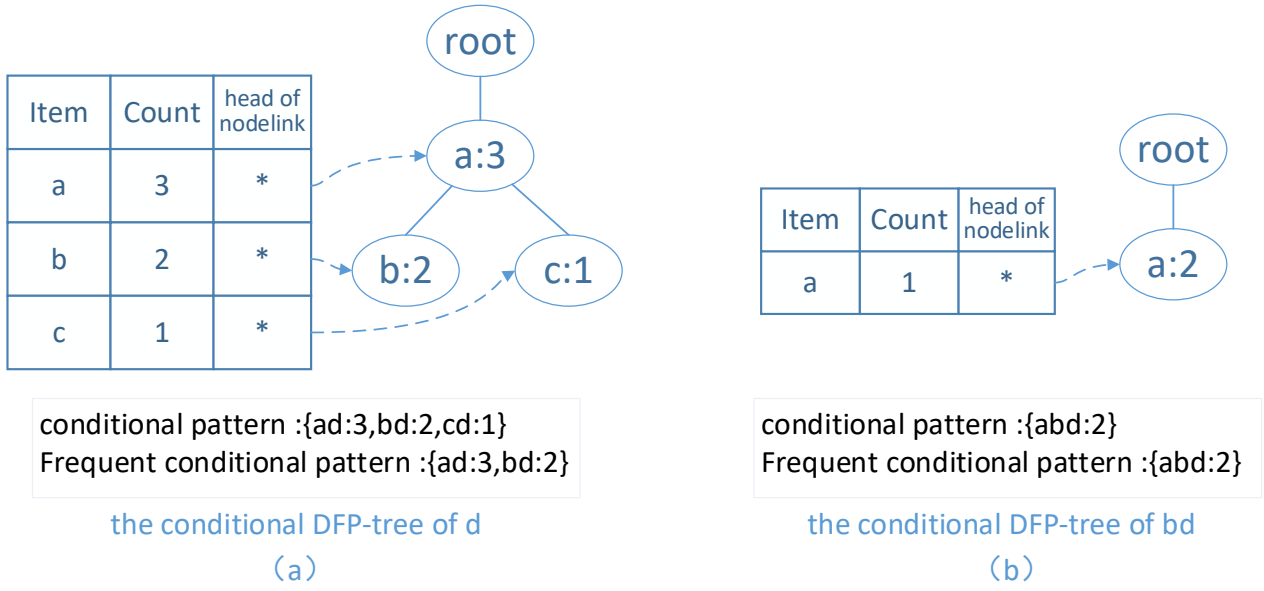

Fig. 6 The recursion procedure of item d's conditional DFP-tree

The algorithm will not terminated until all conditional pattern bases of d contain no items. Repeatedly finishing this recursive procedure for all the items in the header table, all the conditional pattern bases and frequent conditional pattern can be obtained.

As is demonstrated in Fig. 5, all the frequent conditional patterns and their support of each item are obtained and recorded in the frequent condition pattern filter list FCPF-list. If there are two values of global support thresholds r1 and r2, and r1 < r2, then r2's frequent patterns is the subset of r1's frequent patterns. Therefore, all the frequent patterns of a larger value of r can be filtered to obtain from the FCPF-list of a smaller value of r.

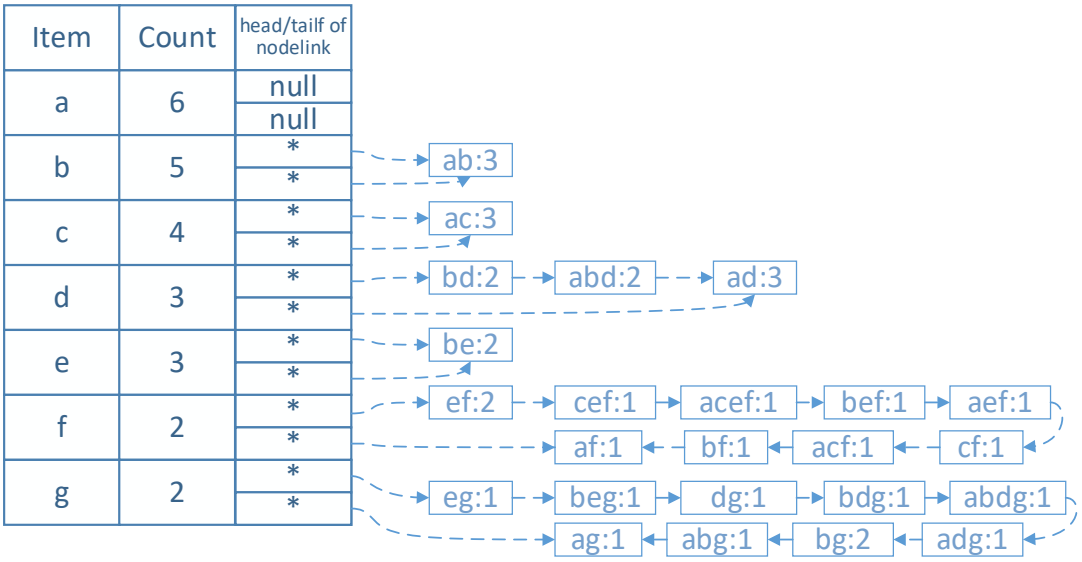

Fig. 7 Frequent condition pattern filter list

In Fig. 5 Algorithm 5, user is required to provide the value range of r and pace which denote the incremental quantity every time. For example, if 0.4<r<0.6 and pace = 0.05, then the FCPF-list can be built as shown in Fig.7 where the value of r is 0.4. Then the frequent patterns of different values including 0.45, 0.50, 0.55 and 0.60 for r can be filtered to obtain from FCPF-list. User can get the appropriate value of r and the corresponding frequent pattern according to actual need.

Experiment

This section compares the performance of DFP-tree algorithm with other two algorithms including Apriori algorithm and FP-tree algorithm. All the original data is collected from SLMS (Sophia Learning Manage System) .All experiment are performed on a Intel Core i5 3.2G PC with 8G main memory, running on Windows 7 Professional. All the programs are written in Matlab R2015a.

Data preprocessing. The original data is 70 students' online learning record data in the Android App Development Course. The learning knowledge point's record consists of five columns: log_id, user_id, knowledge_point_id, session_id and create_time. The log_id, user_id, knowledge_point_id denote the serial number of record, user and knowledge point, respectively. Session_id denotes the user's login exclusively. Create_time denotes the time of creating this record. Thus, a record indicates that a user logins and learns a knowledge point. And all the records are divided into different groups by user_id and session_id. In a group, all the records carry the same user_id and session_id, in other word, one group indicates all the knowledge points which are learned by the same user at the same login. In transaction database, one knowledge point is regarded as one item, and one user learns different knowledge points at one login is regarded as one transaction. Therefore, after dividing all the records into different groups by user_id and session_id, the transaction database can be built.

The experiment results contrast. As is shown in Fig. 8, the running time of three algorithms increase with the growth of transactions' quantity. The support threshold of the three algorithms is equal to 0.1. It is obvious that the DFP-growth algorithm performs like FP-growth algorithm and much better than Apriori algorithm. Fig. 9 shows that the frequent itemsets' quantity of three algorithms reduces when the value of minsup (minimum support for Apriori/FP-growth algorithm) or

r (global support threshold for DFP-growth algorithm) increases from 0.05 to 0.2. The frequent itemsets' quantity of DFP-growth algorithm are significantly more than that of other two algorithms. Hence, the DFP-growth algorithm can excavate the association of knowledge points deeply. In the specific scene of mining the association of knowledge points, the knowledge point doesn't exist independently. DFP-growth algorithm can help to find the frequent patterns for each knowledge point in a course, but the other algorithm may not be sure to do this. Therefore, DFP-growth algorithm improved the accuracy in mining the association of knowledge points.

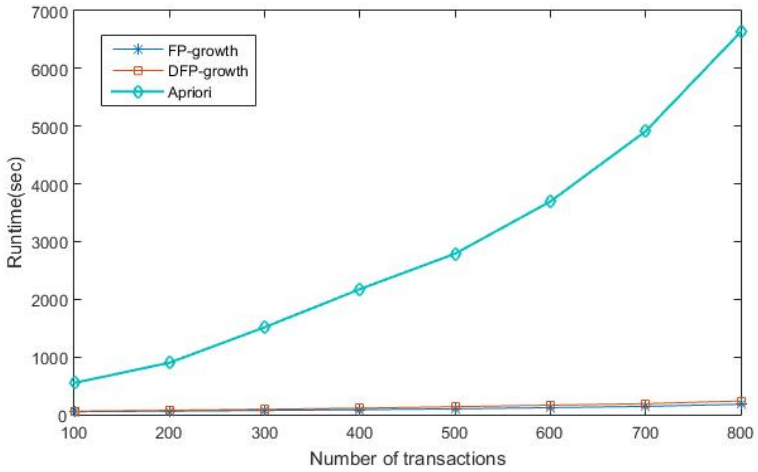

Fig. 8 Algorithms' performance comparison

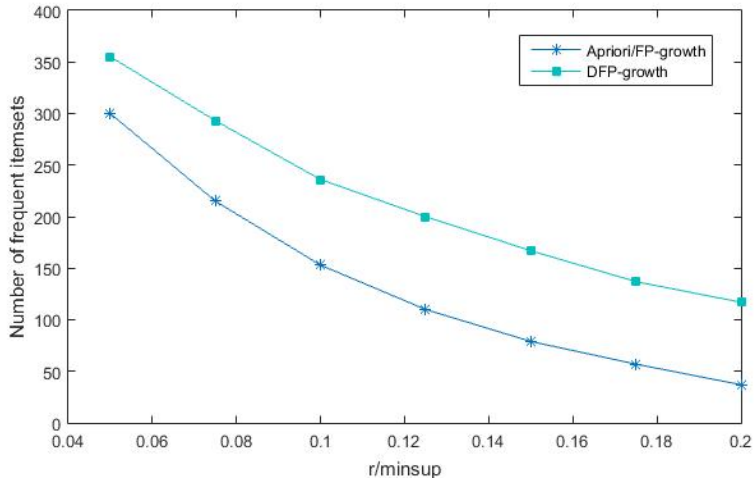

Fig. 9 Number of frequent itemsets comparison

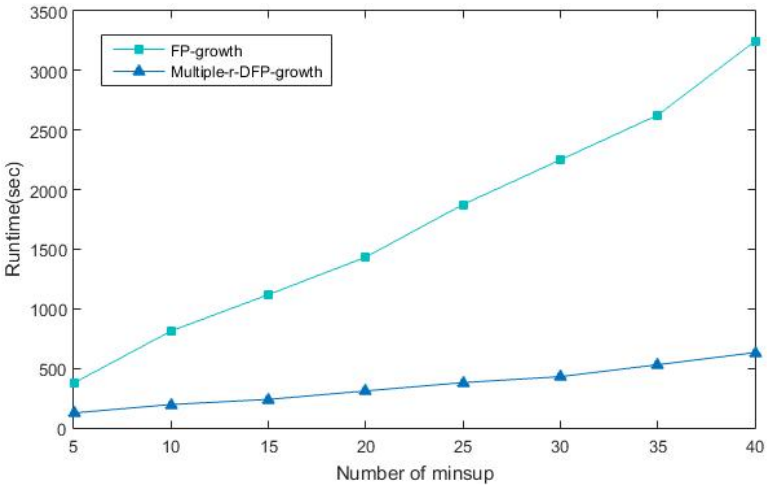

Fig. 10 Algorithms' performance comparison for different number of minsup

In practice, user may need to compare the mining results of different minsup to determine one appropriate value. Fig. 10 shows that the Multiple-r-DFP-growth algorithm has better performance when the size of transaction database is 200 and the number of minsup increases. And the Multiple-r-DFP-growth algorithm is better adapted to practical need.

Summary

This paper proposes an algorithm which modifies the FP-tree and adopts the multiple minimum support strategy. In the application scene of mining the association of knowledge points, the algorithm is better adapted to dynamic database and simplify the setting mode of minimum support for each item. With the help of FCPF-list, the algorithm can obtain the frequent patterns for different values of support threshold efficiently. In practice, user can get the appropriate value of r and the corresponding frequent patterns by comparison of mining results for several different support threshold. Experiments show that the algorithm improved the accuracy of the mining results and ensured a good performance.

Acknowledgements

We'd like to thank Xiaoming Ju for leading the whole project, Yinglu Lin and Shengyuan Shu for building and maintaining the server side and Chen Chen for helping collecting data. This research is supported by the Shanghai University Knowledge Service Platform (No.ZF1323) and Software and Hardware Co-design Engineering Research Center, MoE.

References

[1] Liu B, Wynne H, Ma Y: Mining association rules with multiple minimum supports. *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* .(1999), p337-341.

[2] J. Han, J. Pei, and Y. Yin: Mining frequent patterns without candidate generation. *ACM SIGMOD International Conference on Management of Data*, (2000), p1-12.

[3] Shi Y.B, Zhang S.Y, Xiang C: Study on expression and conection technology of knowledge point in engineering network course. Journal of Zhejiang University. (2003), 37(5), p508-511

[4] Hu, Y. and Y. Chen: Mining association rules with multiple minimum supports: a new mining algorithm and a support tuning mechanism. *Decision Support Systems*.(2006), 42(1), p1-24.

[5] Lin, C., T. Hong, W. Lu: Linguistic data mining with fuzzy FP-trees. *Expert Systems with Applications.* (2010), 37(6), p4560-4567.

[6] Hu, Y., F. Wu, Y. Liao: An efficient tree-based algorithm for mining sequential patterns with multiple minimum supports. *Journal of Systems and Software*.(2013), 86(5), p1224-1238.

[7] Yang, H., Liu G: A multi-support mining algorithm based on Knowledge points. *Computer Applications and Software.* (2014), 7, p169-172

[8] Hu, Y., et al: A novel approach for mining cyclically repeated patterns with multiple minimum supports. *APPLIED SOFT COMPUTING*. (2015), 28, p90-99.

[9] Sahoo, J., A.K. Das , A. Goswami: An efficient approach for mining association rules from high utility itemsets. *Expert Systems with Applications*. (2015), 42(13), p5754-5778.

[10] Popp, J.S., S.R. Goldman: Knowledge building in teacher professional learning communities: Focus of meeting matters. *Teaching and Teacher Education.* (2016), 59, p347-359.