Appropriate Item Partition for Improving the Mining Performance

Tzung-Pei Hong[1,2], Jheng-Nan Huang[1], Kawuu W. Lin[3] and Wen-Yang Lin[1]
[1]Department of Computer Science and Information Engineering
National University of Kaohsiung
[2]Department of Computer Science and Engineering
National Sun Yat-sen University
[3]Department of Computer Science and Information Engineering
National Kaohsiung University of Applied Sciences

Abstract

Along with the progress of information techniques and the increase of information need, some databases in the real world grow very quickly and their sizes become very huge. If the FP-Growth procedure is directly executed on these databases to mine association rules, the computer memory may not allow all nodes of a FP-tree generated from a huge database. In this paper, a sophisticated mining approach with a flexible partition of items is proposed to effectively derive association rules under the constraint of memory limitation. The experimental results show that the proposed approach can make the mining process under the memory limitation always feasible.

1 Introduction

In the past, many algorithms for mining association rules from transactions were proposed [1][5][6][7][8][9][10][11][12], most of which were based on the Apriori algorithm [1], which generated and tested candidate itemsets level by level. This may cause iterative database scan and high computational cost. Han et al. thus proposed the Frequent-Pattern-tree (FP-tree) structure for efficiently mining association rules without generation of candidate itemsets [2]. The FP-tree [2] was used to compress a database into a tree structure which stored only large items. It was condensed and complete for finding all the frequent patterns. The construction process was executed tuple by tuple, from the first transaction to the last one. After that, a recursive mining procedure called FP-Growth was executed to derive frequent patterns from the FP-tree. They showed the approach could have a better performance than Apriori.

However, the FP-Growth procedure will be inefficient because of the high page fault rate in the mining process. Han et al. then proposed the concept of database projection [4] to solve the memory problem. Their approach used “data copy” to generate a set of independent databases, with the number of domain items in each database smaller than a threshold β, which is indicated by users. The approach needs many I/O operations and extra disk space in building independent databases.

Nan et al. then proposed the concept of grouping domain items [3] to solve the memory problem. The sizes of the groups might, however, be very unbalanced, thus it is possible for a group to be still too large to be handled due to the memory limitation. The paper thus focuses on solving or easing off the mining problems incurred from memory limitation. A sophisticated mining approach with a flexible partition of items is proposed to effectively derive association rules under the constraint of memory limitation. This approach is based on the branch-and-bound search strategy to find the best partition under the criterion that the cross-group itemsets should be the least. The domain items that appear in a transaction database are divided into a set of groups under the constraint that the number of items in each group cannot exceed a threshold. The purpose of the phase is to assure the processing can satisfy the memory limitation. The groups in the partition may thus be independent or dependent according to the given data. The proposed approach can make the mining process under the memory limitation always feasible.

The remaining parts of this paper are organized as follows. Some related researches are reviewed in Section 2. The proposed method for dividing a big group into a set of smaller dependent groups is detailed in Section 3. The experimental results are described in Section 4. The conclusions and future work are finally given in Section 5.

2 Literature Survey

Han et al. [2] proposed a tree based data structure named *FP-tree* and the corresponding mining algorithm named *FP-Growth* for discovering frequent patterns. The algorithm requires two scans on a database to complete the mining task. The first scan is to calculate the support of each item. In the meantime, it also creates a header table recording the item name, its corresponding support and the first node-link linking to the first node in the FP-tree carrying the same item name. The items of the header table are sorted descendingly by the support. In the second scan, for each transaction, the items with support smaller than the threshold are filtered out and the remaining items are sorted in descending order by the support value. The sorted items of each transaction are inserted to the tree, namely FP-tree. The structure of a FP-tree consists of a root node labelled as null, a set of item-prefix subtrees as the children of root, and a header table. The structure of the nodes of FP-tree is as <item-name, count (support), node-link>, in which the item-name is the item name used for identification, the count is the number of transactions reaching this node by the same path from root, and the node-link is a pointer linking to the next node in the FP-tree with the same item name. To insert a transaction, *P*, into the FP-tree, *T*, we check whether *T* has a child, *n*, such that *n.item-name* is identical to the item-name of the first element of *P*. If the node exists, the count of *n* is increased by 1. Otherwise, it creates a new node, *m*, with the same item name as *n*. Meanwhile, the count of *m* is set to 1, the parent link is set to *T*, and its node-link is set to the nodes with the same item-name via the node-link structure. We recursively perform the insertion in order for each item in *P* until each item is inserted into the FP-tree. After the FP-tree is constructed, FP-growth is used for discovering the frequent patterns. An item of the header table is selected to construct the conditional FP-tree by inserting all of the prefix paths of the item, which can be retrieved by node-link structure in header table. The name of the item is called the conditional pattern base. Then the FP-growth is executed recursively and the conditional pattern base is cascaded by new one in each recursion until the conditional FP-tree contains only a single path or is an empty tree. The frequent patterns can be easily generated by the cascaded conditional pattern base and the FP-tree. After processing each item in header table, all of the frequent patterns are obtained.

3 Item Partition by Tree Search

The Proposed Algorithm

INPUT:

1. A set of *n* transactions in a database with a set of *m* items $\{I_1, I_2, \ldots, I_m\}$ named DIL (Domain Item List);
2. A minimum support threshold named min_support.
3. A number threshold β for constraining the number of items in each group of a partition.

OUTPUT:

1. A proper partition *P* from the DIL with the item number in each group equal to or less than β.
2. The association relations between each big group and its refined sub-groups.

The proposed item-partition algorithm:

PHASE 1:

STEP 1: Generate all the 2-itemsets from the given items and calculate their counts.
STEP 2: If the count of an itemset is larger than the threshold, min_support, then put it in the set of frequent 2-itemsets (FI).
STEP 3: Initially set the partition *P* to have *m* groups, with each consisting of only one item in DIL.
STEP 4: The two groups with the two items in a frequent 2-itemset will be merged together if they belong to different groups for dependency consideration.
STEP 5: Repeat the above step (STEP 4) until there is no frequent 2-itemsets or only one group in the partition.
STEP 6: Output the partition into Phase 2 for possible finer division.

PHASE 2:

STEP 7: If in the partition there is at least one big group (with the item number larger than the number threshold β), do the next step; otherwise, exit the algorithm and output the partition.
STEP 8: Use the “Refine-partition” procedure, which is based on the branch-and -bound strategy, to divide each big group into a set of small groups (with their item numbers equal to or smaller than β).
STEP 9: Set the association relations between each big group and its refined sub-groups for the usage of later mining.
STEP 10: Output the final partition and the association relations between each big group and its refined sub-groups.

Note that after step 10, a proper partition with no or little inter-group dependency can be found. The partition may then be used to improve the efficiency of data mining under some situations. For example, mining association rules under memory limitation in a good application of it. Next, the “Refine-Partition” procedure, which divides each big group into a set of small groups, is introduced. Basically, it is based on the branch-and-bound strategy. It is stated as follows.

For each big group, the procedure first finds how many sub-groups are enough for it. It can be easily done by dividing the item number of the original group over the item number threshold β and finding the ceiling of the result. This will cause the minimum number of sub-groups for the group. This is also what we desire since the sub-groups will depend on each other and the minimum number of subgroups can reduce the computation time for the later mining process. Then, the procedure will try to find the sub-groups with less inter-group dependency and with the item number of each equal to or less than the item number threshold β.

The inter-group dependency is measured by the number of infrequent 2-itemsets in each group. It is a relative measure. That is, if the number of infrequent 2-itemsets in a group is small, then the number of frequent 2-itemsets is large and the group is much self-contained. Thus, the total of the numbers of the infrequent 2-itemsets in all the groups is used to measure the fitness of the partition. The minimum the value is, the better the partition. Since the total number of frequent 2-itemsets is known, it means that the number of inter-group 2-itemsets in this way is the minimum among all the possible partitions. Note that only 2-itemsets are used here to measure the goodness of partition. Itemsets with more items can also be used to measure but they will raise the complexity of the real implementation and the computation time. The procedure is based on the branch-and-bound strategy. The content of each node includes two parts, the decided part and the undecided part. The sub-groups which have been decided in a search branch are put in the decided part. All the remaining items are then put in the undecided part. After each search level, one more sub-group will be decided and moved from the undecided part to the decided part. The details of the procedure are written below.

The "Refine-Partition" procedure

INPUT: A big group (with the item number larger than the number threshold β).

OUTPUT: A set of small groups (with the item number equal to or larger than the number threshold β) with the minimum total of the numbers of the infrequent 2-itemsets in all the groups.

STEP 1: Initially set the upper bound as an infinite value.

STEP 2: Set the score of each infrequent 2-itemset generated from the big group is 1 and that of each frequent one is 0.

STEP 3: Initially set the decided part of the root node of the search tree as null and the undecided part as the originally big group to be refined with its lower bound being 0. The root node is the currently only one unexplored node.

STEP 4: Generate the child nodes from the unexplored node. Each child node is formed by moving one possible sub-group from the undecided part to the decided part. The sub-group must contain the first item in the originally undecided part. The possible item number of the subgroup is between the ceiling and the floor of the average item number of a sub-group from the group to be divided.

STEP 5: Calculate the lower bound of each child node by the following substeps.

SUBSTEP 5.1: Calculate the scores (S_{decide}) of the decided part by the summation of the scores of the 2-itemsets covered in the decided part.

SUBSTEP 5.2: Calculate the number n of the 2-itemsets generated from the final refinement of the undecided part as:

$$n = \left\lfloor \frac{R}{\beta} \right\rfloor \times C(\beta, 2) + C((R \bmod \beta), 2),$$

where R is the item number in the current undecided part and β is the item number threshold defined in the above algorithm.

SUBSTEP 5.3: Calculate the lower bound ($S_{unecide}$) of the undecided part as the summation of the lowest n scores among the possible 2-itemsets generated from the current undecided part.

SUBSTEP 5.4: Calculate the lower bound of the node as the summation of S_{decide} and $S_{undecide}$.

STEP 6: If the lower bound of a node is equal to or larger than the kept upper bound, then stop the search from the node.

STEP 7: Choose the node with the minimum lower bound among all the unexplored nodes. If several nodes have the same minimum lower bound, choose the one with the deepest level among them.

STEP 8: If the child chosen has been a feasible partition, its lower bound becomes its actual fitness value. Compare the value with the previously kept upper bound. If the new value is smaller than the previous one, replace the upper bound with the current one.

STEP 9: Repeat STEPs 4 to 8 until there are no un-explored nodes.

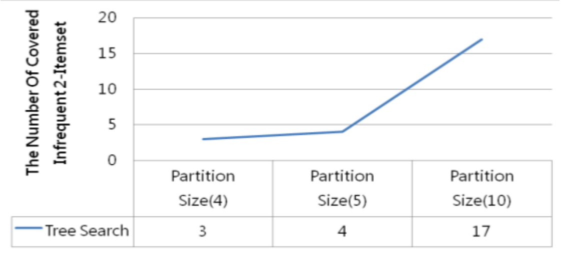
Note that in Step 4, the sub-group to be newly formed must contain the first item in the originally undecided part. The purpose is to avoid generating redundant branches, which are of the same partition but with different group orders.

4 Experimental Results

To evaluate the performance of the proposed algorithm, we use IBM's Quest Synthetic Data Generator to generate the workload data for mining. The experiments were conducted on a PC equipped with an AMD Athlon 64 Processor 3200+ and 2GB of available RAM. The program is written in Java and runs on Windows 7 platform.

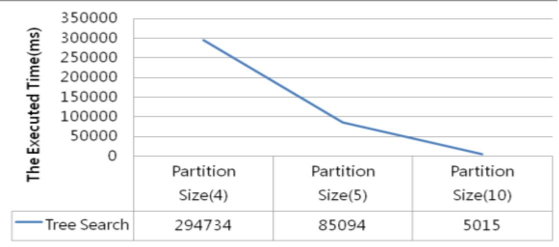
In the following experiments, we investigate the performance of the proposed algorithm in terms of execution time by varying the number of partition size to 4, 5 and 10. In addition, we will also observe the number of covered infrequent 2-itemset. The number of transactions, minimum support and number of items of the dataset are 100, 30% and 20, respectively.

Figure 1 shows the number of covered infrequent 2-itemset when the partition size is set to 4, 5 and 10. Figure 2 shows the execution time when the partition size is set to 4, 5 and 10. Obviously, the required execution time decreases with the increase in partition size dramatically. Note that the required execution time of partition size set to 10 is only 5015 (ms), which is much better than that of partition size being 4, 294734 (ms). This demonstrates the good performance of the proposed algorithm.



	Partition Size(4)	Partition Size(5)	Partition Size(10)
Tree Search	3	4	17

Figure 1. The impact of the number of covered infrequent 2-itemset when varying the partition size.



	Partition Size(4)	Partition Size(5)	Partition Size(10)
Tree Search	294734	85094	5015

Figure 2. The impact of the execution time when varying the partition size.

5 Conclusions and Future Work

The paper focuses on solving or easing off the mining problems incurred from memory limitation. A sophisticated mining approach with a flexible partition of items is proposed to effectively derive association rules under the constraint of memory limitation. This approach is based on the branch-and-bound search strategy to find the best partition under the criterion that the cross-group itemsets should be the least. The domain items that appear in a transaction database are divided into a set of groups under the constraint that the number of items in each group cannot exceed a threshold. The proposed approach can make the mining process under the memory limitation always feasible.

References

[1] R. Agrawal, T. Imielinksi and A. Swami, "Mining association rules between sets of items in large database,“ The ACM SIGMOD Conference, pp. 207-216, 1993.

[2] J. Han, J. Pei, and Y. Yin, ”Mining frequent patterns without candidate generation” The 2000 ACM SIGMOD International Conference on Management of Data, 2000.

[3] A. Nanopoulos, A. N. Papadopoulos, and Y. Manolopoulos, "Mining association rules in very large clustered domains", Information Systems, Vol. 32, pp. 649–669, 2007.

[4] J. Han, J. Pei, Y. Yin, R. Mao, Mining frequent patternswithout candidate generation: a frequent-pattern tree approach, Data Min. Knowl. Discovery 8 (2004) 53–87.

[5] R. Agrawal and R. Srikant, "Fast algorithm for mining association rules," The International Conference on Very Large Data Bases, pp. 487-499, 1994.

[6] R. Agrawal, R. Srikant and Q. Vu, "Mining association rules with item constraints," The Third International Conference on Knowledge Discovery in Databases and Data Mining, pp. 67-73, 1997.

[7] T. Fukuda, Y. Morimoto, S. Morishita and T. Tokuyama, "Mining optimized association rules for numeric attributes," The ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 182-191, 1996.

[8] J. Han and Y. Fu, "Discovery of multiple-level association rules from large database," The Twenty-first International Conference on Very Large Data Bases, pp. 420-431, 1995.

[9] H. Mannila, H. Toivonen, and A.I. Verkamo, "Efficient algorithm for discovering association rules," The AAAI Workshop on Knowledge

Discovery in Databases, pp. 181-192, 1994.
[10] J. S. Park, M. S. Chen, P. S. Yu, "Using a hash-based method with transaction trimming for mining association rules," IEEE Transactions on Knowledge and Data Engineering, Vol. 9, No. 5, pp. 812-825, 1997.
[11] R. Srikant and R. Agrawal, "Mining generalized association rules," The Twenty-first International Conference on Very Large Data Bases, pp. 407-419, 1995.
[12] K. Hu, L. Diao, Y. Lu, and C. Shi, "A heuristic optimal reduct algorithm,“ Lecture Notes in Computer Science, Vol. 1983, Springer, Berlin, 2000, pp. 139-144.
[13] R. Agrawal and R. Srikant, ”Mining sequential patterns," The Eleventh IEEE International Conference on Data Engineering, pp. 3-14, 1995.