*2010 3rd International Conference on Computer and Electrical Engineering (ICCEE 2010)*
***IPCSIT vol. 53 (2012) © (2012) IACSIT Press, Singapore***
*DOI: 10.7763/IPCSIT.2012.V53.No.2.72*

# Research on Construction Method of Association Rule Mining Based on the Hash-tree

Chen Feng+, Jia Yuanhua,Niu Zhonghai and Yi Huixin

School of Traffic and Transportation Beijing Jiaotong University

1416Room,3#Building,18#Yard,Jiaoda Dong Road, Haidian District

Beijing, P.R.China;

**Abstract—**Association rule is one of the fundamental methods of data mining, the hash-tree is an important data structure that used to mining frequent itemset by homo-Apriori algorithms. Because the hash-tree can't be built successfully one-time, this paper proposes static and dynamic two hash-tree construction methods. Both methods can not only determine the hash table size, hash functions and the number of itemset in the leaf nodes, but also guarantee to build a hash-tree successfully one-time in any case and prove the correctness of the proposed method.

**Keywords-**association rule; hash-tree; frequent itemset; data mining; database;Apriori algorithm

## 1. Introduction

Data mining is an effective way to solve the data-rich and knowledge-poor problem. Its essence is a process of extracting implicit, unknown and potentially useful information from the database, which is recognized as a new field of database research. Promoting the rapid development of data mining is a problem of decision support faced by large retail organizations. The development of bar code technology has made the supermarket to collect and store huge amounts of sales data. Data mining analyze large amounts of data, obtain the relationships, rules, trends, and other patterns, and support to make decisions.

Association rule mining is one of the most active research methods of data mining, which was put forward by Agrawal[1-2], etc. The initial motive is directed at the shopping basket analysis issues, its goal is mining associations rules from a large number of goods. According to the rules supermarkets can arrange for the placement of goods, and improve the effectiveness of supermarkets. These rules describe the customer behavior patterns, can be used to guide businesses to scientifically arrange purchase, inventory and shelf design, etc.

The research, finding frequent itemsets / mode, is the core of association rules[2-4], sequential patterns[5], causality[6], the largest model[7], multi-dimensional patterns[8]. It has become hot spots in the field of data mining, and there are many effective mining algorithms[4,5,9]. In these mining algorithms, most of them like the Apriori algorithm[2] approach for frequent itemset mining and update[10]. The common feature of homo-Apriori algorithm: In order to identify all frequent k-itemsets that contain k ($k > 1$) items, first, generate candidate k-itemsets that contains frequent k-itemsets, and save it in a hash-tree, then scan the database and determines each candidate k-itemset frequent in the library, and finally obtain frequent k-itemsets from candidate itemsets under the pre-specified minimum frequent threshold concentration.

+ Corresponding author.
*E-mail address*: 08114212@bjtu.edu.cn

Before mining process, in order to ensure the hash-tree constructed successfully and mining process go smoothly, we should determine the appropriate hash function, hash table size and the number of items stored by leaf nodes. However, as far as I know, there have been no studies addressing this important issue. In this paper, there are static and dynamic two hash- tree construction methods to solve the question. Both methods ensure the successful construction of the hash-tree one-time, and determine the hash table size, hash functions and the simplification of the number of itemset in the leaf nodes. Static method is simple to use, while the dynamic method use effective mapping approach, which significantly saves the cost of system resources when building a hash-tree.

In the following section 2, descript the mining of frequent itemsets, then introduce the generation of candidate itemsets and the construction of the hash-tree in Apriori- algorithm[2], and analyze the problem. In the section 3 details methods of static and dynamic achievements. Finally, conclusions are given in Section 4.

## 2. Related research

### 2.1. Mining frequent itemset

$I=\{i_0, i_1, \ldots, i_{m-1}\}$ is a collection contains $m$ different items. $T$ is a transaction that is composed by different items in $I$. $D$ is a database contains different transactions. Itemset is a collection formed by different items, while K-itemset is a collection formed different items. Transaction $T$ contains itemset $X$, if and only if $X \subseteq T$. Frequency of occurrence for itemset $X$ in $D$, $\delta_x$, define as the number of $X$ in $D$, that is , the number of transactions in $D$ that contains Itemset $X$. We pre-specified a minimum frequent threshold $\xi$. If $\delta_x \geq \xi$, claim itemset $X$ is frequent itemset or frequent $k$-itemset when it has $k$ items. All the frequent $k$-itemsets in $D$ denote $F_k$.

With the given minimum frequent threshold $\xi$ and database $D$, mining frequent itemsets is to find out the itemsets which frequency of occurrence is greater than or equal to $\xi$. To simplify the discussion, in the case that does not lead to ambiguity, items in I are expressed by the subscript. For example, $\{i_2, i_8\}$ is *{2, 8}*. In addition, in the study of frequent itemsets, some itemsets have the same items but different sequence because they have the same frequency of occurrence, such as $\delta_{\{2,8\}}=\delta_{\{8,2\}}$. In this paper we only think about the ascending order.

### 2.2. Generation approach of candidate itemsets

All the homo-Apriori algorithms use "generation-test candidate itemset" repeatedly to find out the frequent itemsets in database D. Let's take Apriori algorithm for example.

When the first time (cycle 1) it scan database D, count each item's frequency of occurrence and obtain all the frequent 1-itemsets, F1. Next, use F1 as the seed itemset, generate the candidate itemset(C2) that include all the frequent 2-itemsets in the cycle 2. Then scan database D to gain the frequency of occurrence for the candidate itemset(C2) . After that we can have F2. Cycle like this until there is no candidate itemsets generated or frequent itemsets received. $C_k$ ($k>1$) generate through the connection and cut off processing on $F_{k-1\,p}$ and $F_{k-1\,q}$. Its generation is as follows [2],first connection step:

```
Insert into $C_k$
Select p.item$_1$, p.item$_2$, ..., p.item$_{k-1}$, q.item$_{k-1}$
From $F_{k-1\,p}$, $F_{k-1\,q}$
Where p.item$_1$ = q.item$_1$, ..., p.item$_{k-2}$ = q.item$_{k-2}$, p.item$_{k-1}$<q.item$_{k-1}$
```

Generate $C_k$. Then remove all the candidate itemsets that include non-frequent ($k-1$)-itemset from $C_k$.

### 2.3. Construction and structure of the hash-tree

Except cycle 1, in each cycle, Apriori algorithm store a hash-tree in a candidate itemset, which can help us accelerate the search speed and calculation frequency of occurrence. A hash-tree is composed by the root node, internal nodes and leaf nodes. The depth of the root node is 1. All the candidate itemsets save in leaf nodes, and each leaf node can contain several candidate itemsets. Each internal node in the tree contains a hash table, and internal node in depth of d belongs to the hash table, each point to an internal node or leaf node in depth of d+1. In the initial of building a hash tree, all the nodes are the leaf nodes. When insert a candidate k-itemset X={$x_1$, $x_2$, …, $x_k$} into the hash-tree, from the root node down until find out a leaf node. In the internal node in depth of d, determine the internal node or leaf node in depth of d+1, through hashing the d item $x_d$ （$1 \le d \le k$） in the itemset by hash function. When the number of the candidate itemsets, which are inserted into a leaf node, exceeds a threshold, the leaf node is converted to an internal node. Figure 1 shows 6 candidate 3-itemsets inserted into the tree. In the figure, N ($x_d$) is the subscript of item $x_d$ in the collection I. For example, insert the candidate itemset {$i_1$,$i_3$,$i_4$}(collection {1,3,4}). Owing to f($i_1$)=1,the item 1 in the root node point to a internal node, then the second hash item gain f($i_3$)=3. Item 3, the internal node in depth of 2, point void address, the candidate itemset can be stored in it. So the itemset {1,3,4} is inserted into the leaf node pointed by item 3 instead of hash $i_4$ .

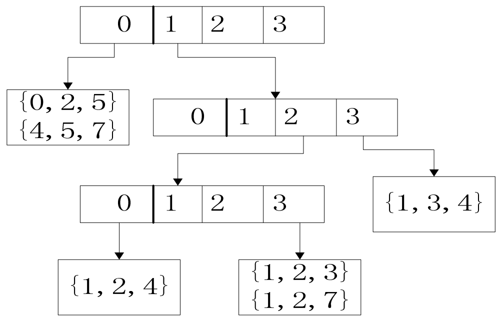

Fig.1. Sample hash tree

Hash function：$f(x_d) = \mathcal{N}(x_d)$;
Hash table size is 4,the number of itemsets in the leaf node
is I,I={0,1,2,3},the number of all possible 2-itemsets is 16.

### 2.4. Problems

In general,the maximum tree depth can not exceed k for k-itemset hash-tree. Otherwise, itemset inserting will fail, because in the k-itemset there are at most k items that are used to determine branches of hash table with different depth. Take figure 1 for example, if we insert (1,6,7) into the hash-tree, the construction will fail. For the final positioning of the leaf node has stored the (1,2,3) and (1,2,7), the number of itemsets has reached the threshold laid down in 2 and tree depth also has reached the limit value , so it cannot be converted to an internal node. This situation is called insertion overflow of a candidate itemset. If insertion overflow occurs, it cannot guarantee that the frequent itemsets mining process success. This paper gives two solutions in the next section.

## 3. Construction approachs of the hash-tree

### 3.1. Construction of the static hash-tree

In the process of mining frequent itemsets, different minimum frequent thresholds will lead to the generation of different candidate itemsets. To ensure the process of data mining goes smoothly, the construction of the hash-tree should adopt the worst situation following. Make sure that all the itemsets exist should be inserted into the hash-tree. And there is a most intuitive and simple approach,each item in *I* should has a specific non-repetition of the hash value. The arbitrary two different itemsets in I are impossible to have the same hash value in the internal node in the same depth, and they will be stored in different leaf nodes. As theorem 1 following:

Theorem 1: There is a itemset *I*, $I=\{i_0, i_1,\ldots, i_{m-1}\}$, which is made up by *m* different item. There are k different k-itemsets in *I*, $X=\{x_1, x_2,\ldots, x_k\}$. Build the hash-tree of k-itemsets by principle of Apriori algorithm. The size of hash table is m , hash function is $f(x_d) = \mathcal{N}(x_d)$,$x_d$ is the d item in k-itemset and $\mathcal{N}(x_d)$ is the subscript of $x_d$ in *I*. So each leaf node in the hash-tree can only store a k-itemset to ensure no overflow when build a hash-tree.

Proof: Assume that overflow happened in the process of building the hash-tree, there must be two different k-itemsets, X={$x_1$, $x_2$,…, $x_k$} and Y={$y_1$, $y_2$,…, $y_k$,},which are inserted into the same leaf nodes. According to the approach of building hash-tree, in the internal node that has the same depth with that from the leaf node to the root one, the hash values of X and Y in the internal node , f($x_d$) and f($y_d$), must be equal: N ($x_d$)=N ($y_d$). Accordingly, xd and yd are the same item in I. Therefore,X and Y are the same k-itemset. It is contrary to the assumptions of that X and Y are different k-itemsets. So the proposition is proved.

Figure 2 give us an example. I={0, 1, 2, 3}. All the 2-itemsets are inserted into the hash-tree built through the approach in Theorem 1, with which we can prove the correctness of Theorem 1.

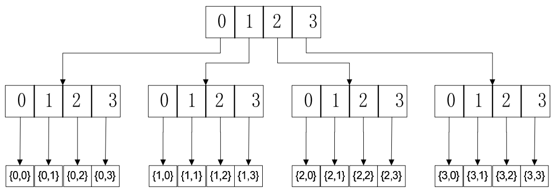

Fig.2. The example of all 2-itemset inserted into hash-tree

Hash function：f($x_d$)= N ($x_d$)；

The size of the hash table is 4, the number of items in a leaf node is 1, I={0,1,2,3},the number of all the possible 2-itemsets is 16.

In Theorem 1, there is no limit of the order of the items in itemset X, which belong the normal situation. According to Theorem 1, we can reason as follows:

Reasoning 1：Based on the Theorem 1, if we define the k-itemset stored in hash-tree as ascending k-itemset composed by different items in I, insertion overflow won't appear through the same construction approach.

As the size of the hash table in this method is determined by the number of items in database, it is called static construction of hash-tree. There are many advantages of this method. It can ensure that the hash-tree is built successfully one-time with no overflow. Leaf nodes store a small number of items, which make selection process more efficiently. The size of hash table and the hash function can be determined simply. However, it waste a lot of system resources to build a hash tree. The items in the candidate itemsets are only a part of that in I, and the number will reduce drastically with increase of the length of frequent itemsets. For example a database contains 1000 different items, if only 10 items are in the candidate itemset in a cycle, the hash table will have 99% of void items. As follows, a dynamic method will proposed to solve the problem.

## 3.2. Construction of the dynamic hash-tree

From the discussion of the static method, only those items which are from the candidate itemsets $C_k$ hashing into hash table are used. When we build a hash-tree, if the size of the hash table isn't less than the number of items in $C_k$ in which items will be stored into the hash-tree by the suitable approach, according to Theorem 1 and Reasoning 1, we can ensure to build the hash-tree successfully one-time.

Analyzing the generation of the candidate itemsets in section II.B, we can see that $C_k$ is generated by the connection and pruning process on $F_{k-1}$ and the number of items in $C_k$ is less than that in $F_{k-1}$. Therefore we can use the number of items in $F_{k-1}$ as the size of the hash table, and get the value when we get $F_{k-1}$. To ensure that the items in $F_{k-1}$ are hashed in different item in the hash table, we'll use the following mapping approach for building the hash-tree:

- Array the different items in $F_{k-1}$ according to the ascending order of the subscript in I. And set the new number from 0 to re-start with integer. For example,$F_2$ ={$i_3$, $i_5$,},{ $i_3$, $i_7$},{ $i_3$, $i_9$},{ $i_5$, $i_7$}, { $i_7$, $i_9$}}will be arrayed as：$i_3$, $i_4$, $i_5$, $i_7$, $i_9$ and set as 0, 1, 2, 3, 4.
- We can use a mapping table show the mapping relation of them. The mapping table of above simple as follows:

| Items in $F_{k-1}$ | $i_3$ | $i_4$ | $i_5$ | $i_7$ | $i_9$ |
|---|---|---|---|---|---|
| New number | 0 | 1 | 2 | 3 | 4 |

After completion of the mapping process, when building the hash-tree, we replace items in candidate by new numbers and insert into the hash-tree. When the transaction compute out the frequency of occurrence of the candidate itemset in the hash-tree, the related items should replace by new number and insert into the hash-tree to compute. When continue the step above, the items, which appear in the mapping table, will be ignored, for they are useless on the computing of the frequency of occurrence. Finally, after the computing and the output of $F_{k-}$, then use the mapping table to map back the original item.

According to Reasoning 1, we build a tree by the method above, which can guarantee no overflow. This method can reduce system resource, because there are few void items in hash table. Like the example above, there are 1000 items in *I*. Compared to the static method, when we build an internal node, the space will be allocated 5 units instead of 1000, only 5‰.

### 3.3. Comparison of the two methods

Both the two methods have their own advantages and disadvantages. Their common characteristics are:

- It is simple and practical to determine the size of the hash table and the hash function. Do not need test and adjustment for different mining process.
- In order to make the selection process quickly and efficiently, we can set a minimum value that limit the number of itemsets in the leaf nodes.
- For all cases of the selection process of frequent itemsets, the method can guarantee to build a hash-tree successfully, and avoid the mining processing fail caused by the failure of the construction of the hash-tree.

In addition, each of them has their own different characteristics, made them can apply to different situations of the mining process.

In the static method, the size of hash table is determined by the number of items in the database. When build or process a tree, we can use the subscripts in I simply and quickly. The disadvantage is that too many items in the hash table is avoid, wasting a lot of system resources. Therefore, the method applies to smaller database, in which the number of different items is small. And there are enough system resources in database, make sure we have enough memory for the mining deal with frequent itemsets. This is a method of space-for-time.

Compared to the static method, according to items that are needed in process of data mining, the dynamic method can change the size of hash table. This reduced system resources for building the hash-tree. And improve the efficiency of items in the hash table. But it must create another mapping table corresponding with the hash table, which make additional resources. This is a method of time-for-space. It is suitable for large database, in which the number of different items is large. And there are not enough system resources and available memory for the process of frequent itemsets mining.

## 4. Conclusion and Discussion

With the development of society, data mining is increasingly widespread and profound on the application in the field of finance, sales and service industries. Association rule is a kind of basic approach for data mining, and arouse more and more concern and attention. In the view of problems that cause by the current wide use of homo-Apriori frequent itemsets in hash-tree construction. Two hash-tree construction methods were raised, the static and dynamic, which can guarantee to build a hash-tree simply and successfully one-time. This paper described in detail both static and dynamic methods, and proved the correctness. On the basis of analysis of their respective characteristics, advantages and disadvantages, gave their respective treatment conditions for mining.

## 5. Acknowledgment

The work is supported by Chinese National High-Tech Research and Development Plan ("863 Program", 2007AA11Z213) under the title Dynamic Information Sharing Platform and Coordinated Control Technology Research for Freeway and related Urban Expressway.

## 6. References

[1] R Agrawa,l T Imielinsk,i A Swam.i DatabaseMining:A Performance Perspective[J]. IEEE Trans Knowledge and Data Eng, 1993, (5): 914-925.

[2] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In VLDB’94, pp. 487-499.

[3] X. P. Du, K.Kaneko, and A. Makinouchi. Fast Algorithm to Find Frequent Itemsets for Mining of Association Rules. In IS’00, pp. 408-414.

[4] J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. Proc. 2000 ACM-SIGMOD Int. Conf. on Management of Data, Dallas, TX, May 2000, pp. 1-12.

[5] R. Agrawal and R. Srikant. Mining Sequential Patterns. In ICDE'95, pp. 3-14.

[6] C. Silverstein, S. Brin, R. Motwani, and J. Ullman. Scalable Techniques for Mining Causal Structures. In VLDB'98, pp. 594-605.

[7] R. J. Bayardo. Efficiently Mining Long Patterns from Databases. In SIGMOD'98, pp. 85-93.

[8] M. Kamber, J. Han, and J. Y. Chiang. Metarule-guided Mining of Multi-dimensional Association Rules Using Data Cubes. In KDD'97, pp. 207-210.

[9] J.S. Park, M.S. Chen, and P.S. Yu. An Effective Hash-based Algorithm for Mining Association Rules. Proc. ACM SIGMOD Int. Conf. Management of Data, San Jose, CA, May 1995, pp. 175-186

[10] X. P. Du and A. Makinouchi. Ameliorated Algorithm to Maintain Discovered Frequent Itemsets. Res. Rep. ISEE Kyushu University, Vol.6, No.1, March 2001, pp. 19-24.yushu University, Vol.6, No.1, March 2001, pp. 19-24.