IBIMA Publishing
Communications of the IBIMA
http://www.ibimapublishing.com/journals/CIBIMA/cibima.html
Vol. 2011 (2011), Article ID 652178, 8 pages
DOI: 10.5171/2011.6512178

A New Perfect Hashing and Pruning Algorithm for Mining Association Rule

Hassan Najadat[1], Amani Shatnawi[2] and Ghadeer Obiedat[2]

[1]Computer Information Systems Department, Jordan University of Science and Technology, Irbid, Jordan

[2]Computer Science Department, Jordan University of Science and Technology, Irbid, Jordan

Abstract

This paper presents a new hashing algorithm in discovering association rules among large data itemsets. Our approach scans the database once utilizing an enhanced version of priori algorithm, Direct Hashing and Pruning algorithm (DHP). The algorithm computes the frequency of each k-itemsets and discovers set of rules from frequent k-itemsets. Once the expert in the application domain provides the minimum support, the pruning phase is utilized to minimize the number of k-itemsets generated after completing the scanning of specific size database. The required data structure is built to implement the hash table. The analysis shows that the new algorithm does not suffer from the collisions, which lead to high accuracy.

Keywords: Association Rule Mining, Direct Hashing, Basket Market Analysis.

Introduction

Association Rule Mining (ARM) is one of the most popular data mining paradigms that can be simply defined as the way of finding the interesting and useful rules from a large transaction data set as shown in Han et. al. (2006), Fang et al., (2001), and Ming-cheng et. al. (2008). To define the main concepts in ARM, a file consists set of transactions, T, that each t1 includes all items, $I = \{I_1, I_2, I_3, \ldots, I_n\}$, purchased by each customer. A transaction, *t,* is said to contain set of items, *A, if* and only if $A \subseteq t$. An association rule is presented in the form $A \rightarrow B$, where both A and B are set of items and $A \cap B = \emptyset$. There is a support threshold, s, provided by an expert where each a rule *holds* in transaction set D with support s, such that s% of transactions in *D* contains AUB.

TID	Items
1	a , b, c
2	a, c
3	b,d
4	b,c,d

Figure 1: Transactions Data Set.

The support is computing by dividing the number of transactions that contain all the items in both A and B over the total number of transactions. Figure 1 provides a set of transactions with each unique id. T1 represents the set of items, {a,b,c}, in the basket. The frequency of item a is 2 with support 2/4, while the frequency of c is 3 with support 3/4. The rule A$\rightarrow$B has confidence c in transaction set D, where c% of transactions in D containing A that also contain B as provided by Flank (2004) and Xiaoxun et. al. (2008).That is,

Support (A$\rightarrow$B) = P (AUB)
Confidence (A$\rightarrow$B) = Support (AUB) / Support (A)

k-itemset contains k items and the frequency of an itemset is the number of transactions that contain the itemset in work by Anthony et al., (2008) and . The problem of mining association rules involves the generation of all association rule that have support and confidence greater than or equal to the user-specified *minimum support* and *minimum confidence* as stated by özel and güvenir (2001).

In general, association rule mining can be viewed as a process of two-steps. The first step is to find frequent itemset that is above the threshold. The second step is to generate rules from frequent itemsets. For all frequent itemset, *f,* the process finds all nonempty subset of *f* .For every subset x, the process generates rules on the form x $\rightarrow$(f – x). Then, it takes the rule if the ratio of support (f – x) to support (x) is greater than or equal to minimum confidence as shown in work of ÖZEL AND GÜVENIR (2001).

Since the most important task is to generate frequent itemsets, it has been one of the most popular research topic in data mining field. Several algorithms such as AIS, STEM, A priori, Direct hashing and pruning (DHP) have been developed by Jang et al., (1995). This paper provides a comprehensive view of minimizing the time of extracting the frequent itemsets.

The rest of paper is organized as follows: section 2 presents the main related work, section 3 provides the new perfect hashing algorithm using with DH algorithm, section 4 provided an enhancement of the new approach. Generating ARM using New Perfect Hashing and Pruning Algorithm is discussed in section 5, then we conclude in section 6.

Related Work

Association Rule Mining that uses Hash-Based Algorithm to filter the unnecessary items can be found in an effective hash-based for mining association rule in works by Jang et al., (1995), John and Soon (2002), and Han et. al. (2006). It gives a description of Direct Hashing and Pruning algorithm (DHP), which reduces the number of items in each pass iteratively. DHP determines the size of hash table to distribute items through the table. Jang et al., (1995) provides a detailed algorithm for mining association rules using perfect hashing and database pruning is provided. DHP is considered an enhancement of the efficiency of apriori algorithm.

These algorithms generate candidate k+1-itemsets from large k-itemsets by counting the occurrence of candidate k+1-itemsets in the dataset. DHP utilizes a hashing technique to filter the unnecessary itemsets to generate next candidate itemsets. The set of large k-itemsets, L_k, is used to generate a set of candidate k+1-itemsets, C_{k+1}, by joining L_k with itself on k-1 denoted by, $L_k * L_k$, to find the common items for next pass. Increasing number of items in the C_{k+1} will increase the processing cost of finding the $L_{k+1.}$

Scanning all databases and testing each transaction to determine L_k from C_k is very expansive process. DHP algorithm constructs smaller size C_k than A priori algorithm. therefore it is more faster in counting C_k from database to determine L_k as mentioned in Özel and Güvenir (2001).The size of L_k decreases rapidly as k increases. A smaller L_k will lead to smaller $C_{k+1,}$ so lower corresponding processing cost. DHP reduces the corresponding processing cost of determining L_k from C_k by reducing the number of itemsets to be explored in C_k in initial iteration significantly. DHP algorithm has two major features; making efficient generation of large itemsets and reducing transaction database size in effective way as stated in Jang et al., (1995).

In DHP algorithm, we find support count of C_k by scanning the database. The algorithm also accumulates information about candidate k+1-itemsets. That means all possible k+1 subset of items of each transaction after pruning item are less than min_support from hash table. Each entry in hash table consists of number of items that have been hashed to this entry. Thus far this table will be used to determine C_{k+1}-itemsets from L_k as A priori algorithm. Each bucket in the hash table consists of number to present how many itemset have been hashed to this bucket. A bit vector can be constructed. If the number of corresponding entry of the hash table is greater than or equal to s, set the value of a bit vector to one.

The hash function is a black box which produces an address every time you drop in a key. H(k) transforms key, k, into the correct address, that is used to store and retrieve set of records. Figure 2 shows a representation of the data structure to provide a direct access to the returned value of a hash function.

For many cases, the address generated by the hash function is a random value and depends in the architecture of the table. A collision is occurred when two different keys are transformed to the same address. In fact, it is impossible to hold two records in the same space address.

To remedy the problem, there should be a hashing algorithm to spread the records over the available address. Hence, finding a hash table with no collision is the main issue such as a perfect hash table in work of Steven (1994). The Efficiency of a hash function is measured of how efficiently the hash function produces a value for elements within a set of data. The complexity of a hash function is big o of one. The main characteristic of a hash function includes simple, quick, and stable.

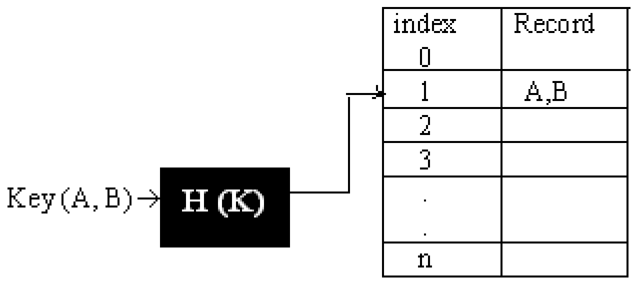


Figure 2: Hashing Function Example.

New Hashing Algorithm Approach

In this section, we describe a new hashing algorithm, which finds the hash index of a K-itemsets. The new hashing algorithm provides a unique hash value for each K-itemsets. New hashing function can be used by the hashing and pruning algorithms like DHP algorithm. The main idea of the new hash algorithm is to divide the hash table H_k virtually in iterative way.

The following constraints should be satisfied:

(1) the dataset must be presented as continuous numeric data set,

(2) items in the itemset is sorted in descending order that means if k-itemset is $(I_k, I_{k-1}, I_{k-2},\ldots,I_2, I_1)$ then $I_k < I_{k-1} < I_{k-2} < \ldots < I_2 < I_1$.

(3) Suppose that *n* is the number of elements in the data set, and r is the smallest number in the data set. The number of distinct k-itemsets will be equal to:

$$\text{Number of k-itemsets} = \binom{n}{k} \quad \ldots\ldots\ldots(1)$$

Equation 1 is used to determine the size of the hash table (H_k). The number of different possible for the k-itemsets is equal to:

$$\# \text{ of diff. possible choice} = \begin{pmatrix} n-r \\ k-1 \end{pmatrix} \ldots\ldots (2)$$

For example, assume there are 10 elements (1, 2, 3, 4, 5, 6, 7, 8, 9, 10). To create 2-itemsets, there are 45 possible combinations. Figure 3 shows all possible combinations in 10 elements. Suppose that we have n items, k-itemsets, and H_k table with size as described in equation (1). In the first iteration, we divided the table H_k into (n-k+1) parts, i.e., see figure: 3 we have 9 parts, each part has all k-itemsets that starts with specific item. In the next iteration take each part as a new H_{k-1} table and divide it starting with first item of parent part +1, i.e., if we are in the next iteration and in first part then (1+1) will be the first number inside table and end with (n-(k-1)+1). Repeat the division process until we get 1-itemsets, take the index of entire table (after final iteration) by the value of 1-itemsets. Because the elements are sorted in the k-itemset, the interred corresponding index will be computed by difference between last 2-item-1.

1,2								
1,3	2,3							
1,4	2,4	3,4						
1,5	2,5	3,5	4,5					
1,6	2,6	3,6	4,6	5,6				
1,7	2,7	3,7	4,7	5,7	6,7			
1,8	2,8	3,8	4,8	5,8	6,8	7,8		
1,9	2,9	3,9	4,9	5,9	6,9	7,9	8,9	
1,10	2,10	3,10	4,10	5,10	6,10	7,10	8,10	9,10

Figure 3: All Possible Combinations of 2-itemsets

The following is an abstract example that describes the process of indexing k-itemsets (X_k, X_{k-1}, X_{k-2}... X2 X_1) as shown in figure 4.

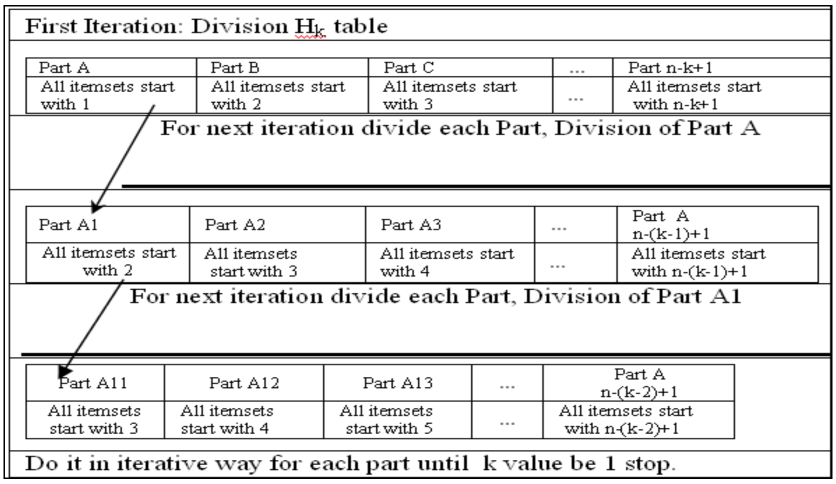

First Iteration: Division Hk table
Part A
All itemsets start with 1
Part B
All itemsets start with 2
Part C
All itemsets start with 3
...
Part n-k+1
All itemsets start with n-k+1
For next iteration divide each Part, Division of Part A
Part A1
All itemsets start with 2
Part A2
All itemsets start with 3
Part A3
All itemsets start with 4
...
Part A n-(k-1)+1
All itemsets start with n-(k-1)+1
For next iteration divide each Part, Division of Part A1
Part A11
All itemsets start with 3
Part A12
All itemsets start with 4
Part A13
All itemsets start with 5
...
Part A n-(k-2)+1
All itemsets start with n-(k-2)+1
Do it in iterative way for each part until k value be 1 stop.

Figure 4: Hash Table Divisions

The start index for each part and the size taken are computed as shown in figure 5.
S_i can be computed using equation (2), In general,

$$\sum_{i=1}^{X_k-1} S_i \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(3)$$

$$\text{where} \quad S_i = \begin{pmatrix} n-i \\ k-1 \end{pmatrix}$$

K-Itemsets start with	Start Index	Size Taken
Part A (start with 1)	$S_0 = 0$	S_1
Part B (start with 2)	S_1	S_2
Part C (start with 3)	S_1+S_2	S_3
. . .		
Last Part (start with n-k+1)	$\sum S_i$ (i=1 to m-1)	Sm (m=X_k)

Figure 5: Size And Start Index of the First Iteration

(K-1)Itemsets start with	Start Index	Size Taken
Part A1 (start with 2)	$S_0 + (S_1'=0)$	S_2'
Part A2 (start with 3)	$S_0 + S_2'$	S_3'
Part A3 (start with 4)	$S_0 + S_2' + S_3'$	S_4'
. . .		
Last Part (start with n-(k-1)+1)	$S_0 + \sum S_i'$ ($i=X_k$+1to m-1)	S_m' ($m'=X_{k-1}$)

Figurer 6: Size and Start Index of Next Iteration

S_i' can be computed using equation (2). In general

$$\sum_{i=1}^{X_k-1} S_i + \sum_{i=X_k+1}^{X_{k-1}-1} S_i' \ldots\ldots\ldots\ldots\ldots\ldots (4)$$

where $$Si' = \binom{n-i}{k-2}$$

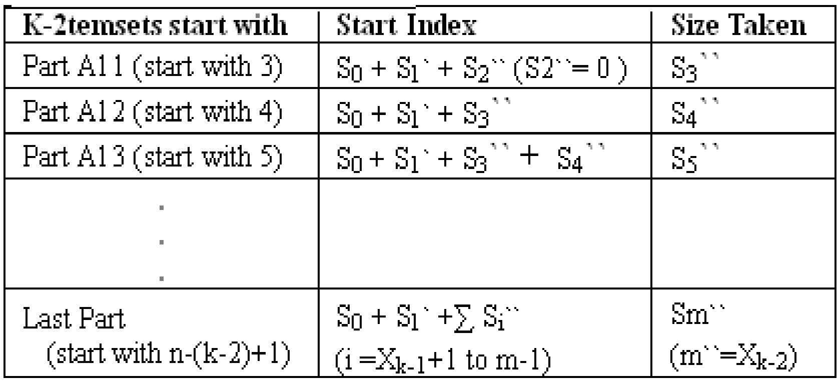
Si" can be computed using equation (2). In general

$$\sum_{i=1}^{X_k-1} S_i + \sum_{i=X_k+1}^{X_{k-1}-1} S_i' + \sum_{i=X_{k-1}+1}^{X_{k-2}-1} S_i'' \ldots\ldots\ldots\ldots (5)$$

where is $$S_i'' = \binom{n-i}{k-3}$$

From the equation 5, a general formula for any iteration can be as follows:

$$\sum_{i=1}^{X_k-1} S_i + \sum_{i=X_k+1}^{X_{k-1}-1} S_i' + \sum_{i=X_{k-1}+1}^{X_{k-2}-1} S_i'' + \ldots + \sum_{i=X_3+1}^{X_2-1} S_i''' \ldots\ldots\ldots\ldots (6)$$



K-2temsets start with	Start Index	Size Taken
Part A1 1 (start with 3)	$S_0 + S_1` + S_2`` (S2`` = 0)$	$S_3``$
Part A1 2 (start with 4)	$S_0 + S_1` + S_3``$	$S_4``$
Part A1 3 (start with 5)	$S_0 + S_1` + S_3`` + S_4``$	$S_5``$
. . .		
Last Part (start with n-(k-2)+1)	$S_0 + S_1` + \sum S_i``$ $(i = X_{k-1}+1 \text{ to } m-1)$	$Sm``$ $(m`` = X_{k-2})$

Figure 7: Third iteration Computation

The index in the 2-itemsets is computed in final table by taking the difference between the two items in the itemset $(x_1 - x_2)$ $(x_1 > x_2)$ -1 as follows:

$$(x_1 - x_2) - 1 \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (7)$$

The new algorithm's accuracy gains the optimal ratio, where the collision will not occurred. This will be used to count frequency of k-itemsets, which will give reliable result and simple arithmetic operations to compute the actual index of each k-itemsets. The implementation of the algorithm allocates large memory space for each H_k hash table.

NHA Enhancement

The hash table was implemented dynamically by using a specific data structure using r*oot, after,* and *befor*e *Pointers, valid bit array and nodes*. Each node has numeric variable to save the frequency of each k-itemsets and the next and previous pointers.

0	0	1	1	…	0

Figure 8: Valid Bit Array

Valid bit array size is $\left\lceil \frac{N}{K} \right\rceil$

Where N is the number of items. Initially, we have zero value for all different possibilities of k-itemsets. The first k-itemsets will be one which represents the k-itemsets frequency in hash table. The following operations continue the process:

- *Root* and *tail* pointers points to Nil initially,
- Compute first k-itemsets element,
- Create new node, Root and tail point to the new nod
- Change corresponding valid bit to 1.

To generate next k-itemsets, compute index of k-itemsets based on NHA, and then check if it's in the valid bit array. If the value is one, so it is in the list search and increase counter by one. Else, insert it into dynamic array list and increment counter by one. To insert node, if hash value greater than maximum value, add node in the end of list, let tail pointer point to it and change corresponding valid bit to be 1. Else, compute corresponding index of k-itemsets i using the hash function, change valid bit to be one then compute summation of all indices less than computed index i. of valid bit array.

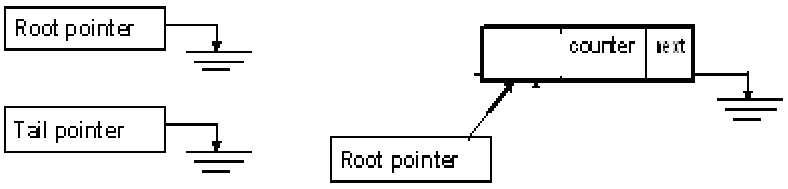


Figure 9: Hash lists Nodes

Data Structure of new Hash Algorithm

The data structure is described as follows:

- Node class or struct consists of next pointer, which points to next adjacent node in dynamic list.

- *Hash class* For each hash table we define an instance of hash class, which includes there pointers such as 1) pointer is used to point the first node in the hash table which implemented as linked list, and 2) the other two pointers before and after are using when add new k-itemsets to the hash table. The following represents the main method of Hash class:

a. *Increment index metod:* to increase the corrosponding index of k-itemset, send index (computed by hash NHA) as parameter. Increment index method computes right position by using search_node method and then calls insert_node method.

b. *Method search_node*: to compute position where to insert node in hash list. It is computed by summation of valid bit array from the beginning to the index parameter.

c. *Insert_node Method:* it has many cases.

Case 1, when hash table(list) is empty, creat new node, let root and tail point to it, increment data variable (counter) by one and finally change valid bit to be one for this itemset.

Case 2: there is itemsets in hash list, we have two choices first one that this itemsets inserted before(valid bit is one) so get its position and increment data by one. Second choice when no node for this itemset, add new node for them, increment data by one and change corrosponding valid bit to be one.

- We define a valid bit array for each hash table with initial value equal to zero. When add new k-itemsets, the corresponding valid bit will be changed to one.

Using the data structure to implement the hash table, it reduces the size of hash table, which will solve the problem of unused space and in the same time will be no collision.

For any k-itemsets if we complete more than C% scaning of database, where C% is greater than (100% - min_supp), the k-itemset has probability to be frequent k-itemset if min_supp of it until now plus remaining percantage database (not scanned data) is greater than or equal user minimum_support.

This hashing technique can be used with all enhancment algorithm of a priori like DHP and others. But, all these algorthm need to scan database for many times, so it takes a lot of time to find association rule. The proposed new Perfect hashing and Pruning algorithm for mining association rule scans database only one time.

Generating ARM Using New Perfect Hashing and Pruning Algorithm

The new algorithm is used to scan database only one time to mine association rule based on hashing technique. In the first phase, the algorithm scans the database to compute the frequency of each k-itemset for all k and saves it in the hashe table. In the second phase, it finds all interesting rules.

Phase 1: for each transaction of database, find all k-itemsets k = (1, 2, 3… length of transaction). Then, compute index of k-itemsets and increase corresponding index in H_k by one. Store them in array list. After completing the database scanning, then perform the followings:

(1) delete all non frequent 1-itemset from array list,

(2) in each H_k hash table, delete all non frequent nodes and change corresponding valid bit to zero. In addition, after complete c% of database, prune k-itemset by determining (k-1)-itemsets.

Phase 2: use frequent 1-itemset to find all possible rules.

Conclusion

We provided a perfect hash function with minimum collision. The proposed algorithm does not require many arithmetic operations.

A distribution of itemsets over all hash tables is inexpensive. We defined a new data structure to implement a hash table. The implementation is aimed to reduce the size of normal implementation of any hashing technique that has been used.

Also, using this data structure may rehash the table by deleting all nodes of non frequent itemsets after a scanning a specific percentage of database. The hashing algorithm and its data structure can be used with enhanced algorithm like PHP and DHP. We implemented the new algorithm to generate ARM by scanning database once. The new algorithm is designed based on the new hashing algorithm that does not suffer from the collision, which leads to high accuracy and reliable.

References

Flank, A. (2004). "Multirelational Association Rule Mining," (online) from http://www8.cs.umu.se/education/examina/Rapporter/AntonFlank.pdf [Accessed 20th February 2010].

Han, J., Kamber, M. & Pei, J. (2006). Data Mining: Concepts and Techniques, 2nd edition, Morgan Kaufmann.

Holt, J. D. & Chung, S. M. (2002). "Mining Association Rule Using Inverted Hashing and Pruning," *Elsevier Information Processing Letters*, 83(4), 211 – 220.

Lee, A. J. T., Wang, C., Weng, W., Yi-An C. & Huei-Wen, W. (2008). "An Efficient Algorithm for Mining Closed Inter-transaction Itemsets," *Data & Knowledge Engineering*, 66(1), 68-91.

Liu, F., Lu, Z. & Lu, S. (2001). "Mining Association Rules Using Clustering," *Intelligent Data Analysis*, 5(4), 309-326.

Özel, S. A. & Güvenir, H. A. (2001). "An Algorithm for Mining Association Rules Using Perfect Hashing and Database Pruning," Proceedings of the Tenth Turkish Symposium on Artificial Intelligence and Neural Networks, Gazimagusa, T.R.N.C., June, 257-264.

Park, J. S., Chen, M. & Yu, P. (1995). "An Effective Hash-Based Algorithm for Mining Association Rules," *ACM SIGMOD Record archive*, 24(2), 175 – 186.

Seiden, S. S. & Hirschberg, D. S. (1994). "Finding Succinct Ordered Minimal Perfect Hash Function," *Elsevier Information Processing Letters*, 51(6), 283-288.

Sun, X., Li, M., Wang, H. & Plank, A. (2008). "An Efficient Hash-Based Algorithm for Minimal K-Anonymity," Proceedings of the thirty-first Australasian conference on Computer science, 01 01 January, Wollongong, Australia, 101-107.

Tseng, M., Lin, W. & Jeng, R. (2008). "Incremental Maintenance of Generalized Association Rules under Taxonomy Evolution," *Journal of Information Science*, 34(2),174-195.