Journal of Engineering Sciences, Assiut University, Vol. 34, No. 1, pp. 79-87, January 2006

FAST ALGORITHM FOR MINING ASSOCIATION RULES

M. H. Margahny and A. Shakour
Faculty of Science, Computer Science Department, Assuit University, Assiut, Egypt
Email : *marghany@acc.aum.edu.eg*

(Received August 31, 2005 Accepted November 22, 2005)

One of the important problems in data mining is discovering association rules from databases of transactions where each transaction consists of a set of items. The most time consuming operation in this discovery process is the computation of the frequency of the occurrences of interesting subset of items (called candidates) in the database of transactions.

Can one develop a method that may avoid or reduce candidate generation and test and utilize some novel data structures to reduce the cost in frequent pattern mining ? This is the motivation of my study.

A fast algorithm has been proposed for solving this problem. Our algorithm use the "TreeMap" which is a structure in Java language. Also we present "Arraylist" technique that greatly reduces the need to traverse the database. Moreover we present experimental results which show our structure outperforms all existing available algorithms in all common data mining problems.

KEYWORDS : *data mining, association rules, TreeMap, ArrayList.*

1. INTRODUCTION

The rapid development of computer technology, especially increased capacities and decreased costs of storage media, has led businesses to store huge amounts of external and internal information in large databases at low cost. Mining useful information and helpful knowledge from these large databases has thus evolved into an important research area [1–3]. Among them association rule mining has been one of the most popular data-mining subjects, which can be simply defined as finding interesting rules from large collections of data.

Association rule mining has a wide range of applicability such Market basket analysis, Medical diagnosis/ research, Website navigation analysis, Homeland security and so on.

Association rules are used to identify relationships among a set of items in database. These relationships are not based on inherent properties of the data themselves (as with functional dependencies), but rather based on co-occurrence of the data items. Association rules, first introduced in [4]. The subsequent paper [5] is considered as one of the most important contributions to the subject. It's main

algorithm, Apriori, not only influenced the association rule mining community, but it affected other data mining fields as well.

Association rule and frequent itemset mining became a widely researched area, and hence faster and faster algorithms have been presented. Numerous of them are Apriori based algorithms or Apriori modifications. Those who adapted Apriori as a basic search strategy, tended to adapt the whole set of procedures and data structures as well [6–9]. Since the scheme of this important algorithm was not only used in basic association rules mining, but also in other data mining fields (hierarchical association rules [9–11], association rules maintenance [12–15], sequential pattern mining [16,17], episode mining [18] and functional dependency discovery [19,20]).

Frequent pattern mining techniques can also be extended to solve many other problems, such as iceberg-cube computation [21] and classification [22]. Thus the effective and efficient frequent pattern mining is an important and interesting research problem.

The reset of this paper is organized as follows. In section (2) we give a formal definition of association rules and the problem definition. Section (3) introduces our development. Experimental results are shown in section (4). Section (5) contains conclusions.

2. ASSOCIATION RULE PROBLEM

Frequent itemset mining came from efforts to discover useful patterns in customers' transaction databases. A customers' transaction database is a sequence of transactions ($T=\{t_1,t_2,\ldots t_n\}$), where each transaction is an itemset ($t_i \subseteq T$). An itemset with k elements is called a k-itemset. The support of an itemset X in T denoted as support(X), is the number of those transactions that contain X, i.e. support(X)= $\left|\{t_i : X \subseteq t_j\}\right|$ .An itemset is frequently if its support is greater than a support threshold, originally denoted by min_supp. The frequent itemset mining problem is to find all frequent itemset in a given transaction database.

The algorithms were judged for three main tasks: all frequent itemsets mining, closed frequent itemset mining, and maximal frequent itemset mining.

A frequent itemset is called closed if there is no superset that has the same support (i.e., is contained in the same number of transactions). Closed itemsets capture all information about the frequent itemsets, because from them the support of any frequent itemset can be determined.

A frequent itemset is called maximal if there no superset that is frequent. Maximal itemsets define the boundary between frequent and infrequent sets in the subset lattice. Any frequent itemset is often also called free itemset to distinguish it from closed and maximal ones.

Obviously, the collection of maximal frequent itemset is a subset of the collection of closed frequent itemset which is a subset of the collection of all frequent itemsets, the supports of all their subsets is not available, while this might be necessary for some applications such as association rules. On the other hand, the closed frequent itemsets from a lossless representation of all frequent itemsets since the support of those itemsets that are not closed is uniquely determined by the closed frequent itemsets [23].

Through our study to find patterns problem we can divide algorithms into two types: algorithms respectively with and without candidate generation. Any Aprioi-like instance belongs to the first type. Eclat [11] may also be considered as an instance of this type. The FP-growth algorithm [24] is the best–known instance of the second type. Comparing the two types, the first type needs several database scans. Obviously the second type performs better than the first.

Table 1 summarize and provides a means to briefly compare the three algorithms. We include in this table the maximum number of scans and data structures proposed.

Table 1: Comparisons of algorithms.

Algorithm	Scan	Data structure
Apriori	M+1	HasTable&Tree
Eclat	M+1	HasTable&Tree
FP-growth	2	Prefix-tree

3. PROPOSED ALGORITHM

The efficiency of frequent itemset mining algorithms is determined mainly by three factors: the way candidates are generated, the data structure that is used and the implementation details [25]. In this section we will show these points in our algorithm.

3.1. Data Structure Description

A central data structure of my algorithm is *TreeMap*, which is a collection in Java Languages. *TreeMap* is a sorted collection. It stores key/value pairs. You can find a value if you provide the key. Keys must be unique. You cannot to store two values with the same key. You insert elements into the collection in any order. When you iterate through the collection, the elements are automatically presented in sorted order. Every time an element is added to a tree, it is placed into its proper sorting position. Therefore, the iterator always visits the elements in sorted order.

If the tree contain n elements, then an average of $\log_2 n$ comparisons are required to find the correct position for new element. For example, if the tree already contains 1,000 elements, then adding a new element requires 10 comparisons. If you invoke the remove method of the iterator, you actually remove the key and its associated value from the map.

A large *TreeMap* can be constructed by scanning the database where each different itemsets is mapped to different locations in the *TreeMap*, then the entries of the *TreeMap* gives the actual count of each itemset in the database. In that case, we don not have any extra the occurrences of each itemset.

By using this collection my algorithm runs as follows, during the first pass of our algorithm, each item in the database is mapped to different location in the *TreeMap*. The “Put” method of the *TreeMap* adds a new entry. If an entry of item does not exist in the *TreeMap* construct a new *ArrayList* and add transaction number to it, otherwise add current transaction number.

After the first pass, the *TreeMap* contains all elements of the database as the keys and it’s transaction number as it’s values, which is the exact number of occurrences of each item in the database. By making one pass only over the *TreeMap* and check its

values we can generate the frequent 1-itemsets (L_1). Hence *TreeMap* can be used instead of the original file of database.

In our algorithm we use the property that the transaction that does not contain any frequent k-itemset is useless in subsequent scans. In order that in subsequent passes, the algorithm prunes the database that represent the *TreeMap* at the present time by discarding the transactions, which have no items from frequent itemsets, and it also trims the items that are not frequent from the transactions.

3.2. How to Generate Candidate 2-Itemsets and Their Frequencies?

After getting L_1 from the first pass, the *TreeMap* will include all the information about the database such as the keys that represents the items and the values of the transactions numbers. Then the candidate 2-itemsets will be generated from comparing the *ArrayLists* and choosing the common elements between both of them. This will lead us to get a new *ArrayList* of the candidate 2-itemsets (C_2). In addition, L_2 will be generated using the same procedure for getting L_1. The process continues until no new L_k is found.

3.3. Example

3.3.1. Problem data

An example with a transactional data D contents a list of 4 transactions and minsup=0.5.

D

TID	List of items
1	1 2 4 6
2	2
3	3 4 5
4	2 5 6 7

3.3.2. Solution Procedure

1-Scan DB to construct TreeLarge Map

**TreeLarge Map has two parameters:*

P1 : key =➔ items

P2 : value=➔ ArrayList

The elements of ArrayList (the transaction number that contains the item)

Treelarge Map

Key (item)	Value (List of items)
1	ArList1(1)
2	ArList2(1,2,4)
3	ArList3(3)
4	ArList4(1,3)
5	ArList5(3,4)
6	ArList6(1,4)
7	ArList7(4)

2-Loop to construct L_1(Large 1-itemset) **….. L_k** (Large k-itemset) :

Directly from Treelarge Map can construct **L_1….. L_k** :
by give the size of ArrayList and compare it with the minsup

Treelarge Map

Key (item)	Value (List of items)
2	ArList2(1,2,4)
4	ArList4(1,3)
5	ArList5(3,4)
6	ArList6(1,4)

⇩

Treelarge Map

Key (item)	Value (List of items)
2	ArList2(1,2,4)
6	ArList6(1,4)

L_3= Null

L_1

1-itemset	Support
2	0.75
4	0.5
5	0.5
6	0.5

⇩

L_2

2-itemset	Support
2,6	0.5

⇩

4. EXPERIMENTAL EVALUATION

In this section, we present a performance comparison of our development with FP-growth and the classical frequent pattern-mining algorithm Apriori. All the experiments are performed on a 1.7 GHZ Pentium PC machine with 128 M.B main memory, running on WindowsXP operating system. All programs were developed under the java compiler, version 1.3. For verifying the usability of our algorithm, we used three of the test datasets made of available to the Workshop on Frequent Itemset Mining Implementations(FIMI'04) [26].

We report experimental results on one synthetic datasets and two real data sets. The synthetic dataset is T10I4D100K with 1K items. In this data set, the average maximal potentially frequent itemset size are set to 10 and 4, respectively, while the number of transactions in the dataset is set to 100K. It is sparse dataset. The frequent itemsets are short and not numerous.

We use Chess and Mushroom as real data sets. The test datasets and some of their properties are described in Table 2.

Table 2: Test dataset description.

Dataset	(T)	I	ATL
T10I4D100K	100 000	1000	10
Chess	3196	75	37
Mushroom	8124	119	23

T= Numbers of transactions
I = Numbers of items
ATL=Average transactions length

The performance of Apriori, FP-growth, our algorithm and these results are shown in figures (1-Chess, 2-Mushroom, 3-T10I4D100K). The X-axis in these graphs represent the support threshold values while the Y-axis represents the response times of the algorithms being evaluated.

In these graphs, we see that the response times of all algorithms increase exponentially as the support threshold is reduced. This is only to be expected since the number of itemsets in the output, the frequent itemsets increases exponentially with decrease in the support threshold.

We also see that there is a considerable gap in the performance of Apriori and FP-growth with respect to our algorithm.

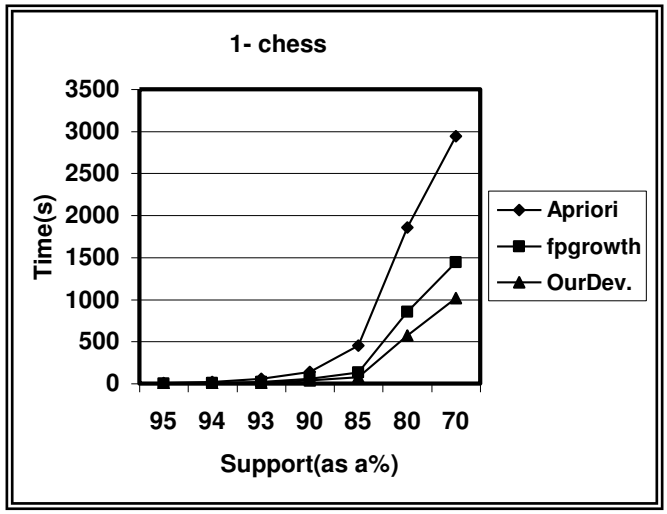

Figure 1: Chess database.

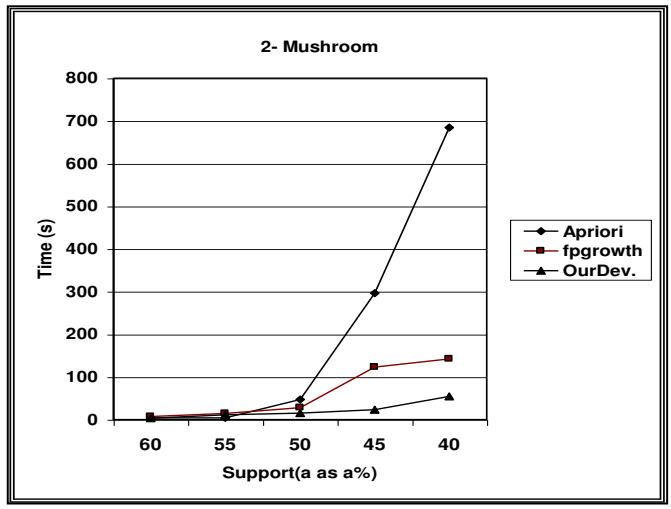

Figure 2: Mushroom database.

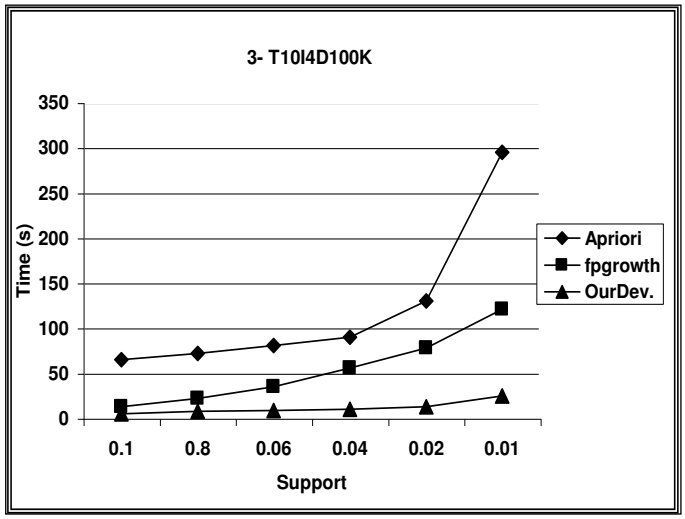

Figure 3: T10I4D100k database.

5. CONCLUSION AND FUTURE WORK

Determining frequent objects is one of the most important fields in data mining. It is well known that the way candidates are defined has great effect on running time and memory need, and this is the reason for the large number of algorithms. It is also clear that the applied data structure also influences efficiency parameters. In this paper we presented an implementation that solved frequent itemset mining problem.

In our approach, *TreeMap* store not only candidates, but frequent itemsets as well. By the way, the algorithm has the following advantages:

1. Candidate generation becomes easy and fast.
2. Association rules are produced much faster, since retrieving a support of an itemset is quicker.
3. Just one data structure has to be implemented, hence the code is simpler and easier to maintain.
4. Using unique *TreeMap* to store L_k and C_k reduce the usage of memory.
5. Dealing with *TreeMap* is faster than dealing with database file.
6. The original file isn't influenced by the prunning process where its role ends as soon as *TreeMap* is constructed.
7. The volume of the resulting database decreases each time we produce L_k, because prunning items is not frequent.
8. Constructed *TreeMap* contains all frequent information of the database, hence mining the database becomes mining the *Treeap*.
9. It provides features I haven't seen in any other implementations of the Apriori-generating algorithm. The most major one is that the frequent itemset generated are printed in alphabetical order. This makes easier for the user to find rules on as specific product.

With the success of our algorithm, it is interesting to re-examine and explore many related problems, extensions and application, such as iceberg cube computation, classification and clustering.

Our conclusion is that our development is a simple, practical, straight forward and fast algorithm for finding all frequent itemsets.

REFERENCES

[1] R. Agrawal, T. Imielinksi and A. Swami, Database Mining: a performance perspective, IEE Transactions on knowledge and Data Engineering, 1993.

[2] M. S. Chen, J.Han and P.S. Yu. Data Mining : An overview from a database perspective, IEE Transactions on Knowledge and Data Engineering 1996.

[3] C.-Y. Wang, T.-P. Hong and S.–S. Tseng. Maintenance of discovered sequential patterns for record deletion. Intell. Data Anal. pp. 399-410, February 2002.

[4] R. Agrawal, T. Imielinski, and A.Sawmi. Mining association rules between sets of items in large databases. In proc. of the ACM SIGMOD Conference on Management of Data, pages 207-216, 1993.

[5] R. Agrawal and R.Srikant. Fast algorithms for mining association rules. In Proc. of Intl. Conf. On Very Large Databases (VLDB), Sept. 1994.

[6] J. S. Park, M-S. Chen, and P.S.YU. An effective hash based algorithm for mining association rules. In M.J. Carey and D.A. Schneider, editors, Proceedings of the 1995 ACM SIG-MOD International Conference on Management of Data, pages 175-186, San Jose, California, 22-25. 1995.

[7] S. Brin, R. Motwani, J.D.Vllman, and S.Tsur. Dynamic itemset counting and implication rules for market basket data. SIGMOD Record (ACM Special Interest Group on Management of Data), 26(2): 255, 1997.

[8] H. Toivonen. Sampling large databases for association rules. In the VLDB Journal, pages 134-145,1996.

[9] A. Sarasere,E.Omiecinsky,and S.Navathe. An efficient algorithm for mining association rules in large databases. In Proc. 21St International Conference on Very Large Databases (VLDB) , Zurich, Switzerland, Also Catch Technical Report No. GIT-CC-95-04, 1995.

[10] Y. F. Jiawei Han. Discovery of multiple-level association rules from large databases. In Proc. of the 21St International Conference on Very Large Databases (VLDB), Zurich, Switzerland, 1995.

[11] Y. Fu. Discovery of multiple-level rules from large databases, 1996.

[12] D.W-L. Cheung, J.Han, V.Ng, and C.Y.Wong. Maintenance of discovered association rules in large databases : An incremental updating technique. In ICDE, pages 106-114,1996.

[13] D. W-L. Cheung, S.D.Lee, and B.Kao. A general incremental technique for maintaining discovered association rules. In Database Systems for advanced Applications, pages 185-194, 1997.

[14] S. Thomas, S.Bodadola, K.Alsabti, and S.Ranka. An efficient algorithm for incremental updation of association rules in large databases. In Proc. KDD'97, Page 263-266, 1997.

[15] N. F. Ayan, A.U. Tansel, and M.E.Arkm. An efficient algorithm to update large itemsets with early prunning. In Knoweldge discovery and Data Mining, pages 287-291,1999.

[16] R. Agrawal and R. Srikant. Mining sequential patterns. In P.S.Yu and A.L.P. Chen, editors, Proc.11the Int. Conf. Data engineering. ICDE, pages 3-14. IEEE pages, 6-10, 1995.
[17] R.Srikant and R.Agrawal. Mining sequential patterns. Generalizations and performance improvements. Technical report, IBM Alamden Research Center, San Jose, California, 1995.
[18] H. Mannila, H. Toivonen, and A.I.Verkamo. Discovering frequent episodes in sequences. In proceedings of the First International Conference on knowledge Discovery and Data Mining, pages 210-215. AAAI pages, 1995.
[19] Y. Huhtala, J. Karkkainen, P. Pokka, and H. Toivonen. TANE : An efficient algorithm for discovering functional and approximate dependencies. The computer Journal, 42(2) : 100-111, 1999.
[20] Y. Huhtala, J. Kinen, P. Pokka, and H. Toivonen. Efficient discovery of functional and approximate dependencies using partitions. In ICDE, pages392-401, 1998.
[21] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and ice-berg cubes. In Proc. 1999 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'99), pages 359-370, Philadelphia, PA, June 1999.
[22] B. Liu, W.Hsu, and Y.Ma. Integrating classification and association rule mining. In Proc. 1998 Int. Conf. Knowledge Discovery and Data Mining (KDD'98), pages 80-86, New York, NY, Aug 1998.
[23] B. Goethals. Efficient Frequent pattern mining. PhD thesis, transactional University of Limburg, Belgium, 2002.
[24] Han, J., Pei , J. ,and Yin ,y. . "Mining Frequent Patterns without Candidate Generation : A Frequent-Pattern Tree Approach". In Proc. ACM-SIG MOD Int. Conf. Management of Data(SIG MOD'04) , pp. 53-87, 2004.
[25] F. Bodon. A Fast Apriori Implementation. Proc.1st IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI2003,Melbourne,FL). CEUR Workshop Pioceedings 90, A acheme, Germany2003.
http://www.ceur-ws.org/vol-90/
[26] Frequent Itemset Mining Implementations (FIMI'03) WorkShop website, http://fimi.cs.helsinki.fi, 2004.

خوارزم جديدة لتعدين القواعد الترابطية للبيانات

قواعد ترابط العناصر الموجودة في قاعدة بيانات كبيرة من أهم المشاكل في تعدين البيانات والشيء المهم في إتمام هذه العملية هو حساب التكرارات للعناصر المختلفة الموجودة في قاعدة البيانات.

ولذلك فكرنا في تقديم طريقة جديدة لحساب هذه التكرارات في وقت قليل بالمقارنة ببعض الطرق المعروفة في هذا المجال ونبنى الفكرة على استخدم إحدى الخوارزميات والتي تسمى **"TreeMap, ArrayList"** وهى عبارة عن تركيبات بيانية في لغة الجافا (إحدى لغات الحاسب) .

قمنا بإجراء بعض التجارب العملية على بعض قواعد البيانات المعروفة سلفاً ولقد تم مقارنة النتائج مع طرق معروفة واستطعنا استنباط أن الطريقة المقترحة تعطى نتائج طيبة مقارنة مع الطرق الأخرى.