Volume 2, Issue 4, April 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Versatile Tree Construction Support-Ordered Trie Itemset in Rapid Association Rule Mining

P Neelima[1]
2nd M.Tech, Dept of CSE
CREC, Tirupati
neelima.pannem@gmail.com

J.Nagamuneiah[2]
Assoc.Professor,Dept of CSE
CREC, Tirupati
nagamuni513@gmail.com

Abstract—**A new algorithm called Rapid Association Rule Mining (RARM) is proposed to further push the speed barrier so that association rule mining can be performed more efficiently in electronic commerce. To achieve large speed-ups even at low support, thresholds, RARM constructs a new data structure called Support-Ordered Trie Item set (SOTrieIT). This trie-like tree structure stores the support counts of all 1-itemsets and 2-itemsets in the database. RARM uses SOTrieIT to quickly discover large 1-itemsets and 2-itemsets without scanning the database. The need to generate candidate 1-itemsets and 2-itemsets constitutes the main bottleneck in large item set generation. Therefore, by eliminating this step, RARM achieves significant speed-ups even though subsequently, it also applies the Apriori algorithm to obtain larger-sized item sets.**

Keywords— **Association Rule Mining,Electronic commerce,SOTrieIT,Frequent Itemsets**

I. Introduction

1. Association Rule

The basic premise of an association is to find all association such that the presence of one set of items in a transaction implies the other items. There are numerous applications of data mining which fit in to this framework.

The classic example from which the problem gets its name is the supermarket. In this context, the problem is to analyze customers' buying habits by finding associations between the different items that customers play in their shopping baskets. The discovery of such associations can help the retailer develop marketing strategies, by gaining insight into the matters like 'which items are most frequently purchased by customers'. It also helps in inventory management, sale promotion strategies etc.

1.1 DEFINITION

Let **A**= $\{i_1, i_2 \ldots i_m\}$ be a set of items. Let $\mathbf{t_1}$ the transaction database, be a set of transactions, where each transaction **t** is a set of items, then an association rule is an expression of the form $X \rightarrow Y$, where X and Y are subsets of A and $X\rightarrow Y$ holds with confidence T, if T% of transactions in T that support X also support Y. The rule $X\rightarrow Y$ has support σ in the transaction set T if σ% of transactions in T supports $X\cup Y$.

The intuitive meaning of such a rule is that a transaction of the database which contains X tends to contain Y. Given a set of transactions, T, the problem of mining association rules is to discover all rules that have support and confidence greater than or equal to user-specified minimum support and minimum confidence, respectively.

1.2 EXAMPLE

Let T consist of 5, 00,000 transactions; 20,000 transactions of these contain diapers; 30,000 transactions contain beer; 10,000 transactions contain both diapers and beers. Support measures how often beer and diapers occur together as a percentage of the total transactions, 2% in this case (10,000/ 50,00,000). Confidence measures how much a particular item is dependent on another. Since transactions contain diapers and 10,000 also contain beer, when people buy diapers, they also buy beer 50% of the time.

13 ITEM SET

An item set that contains k-items is a k-item set.

1.3.1 SUPPORT

A transaction T is said to support an item I1, if I is present in T. T is said to support a subset of items, if T supports each item in the subset. An itemset is said to have support S, if S% of transactions in the database support that itemset.

1.4 FREQUENT ITEMSET

An item set whose support is greater than user specified support is known as frequent item set.

1.5 MINING OF ASSOCIATION RULES

The problem of mining association rules can be decomposed into two sub problems

- Find all sets of items (item sets) whose support are greater than or equal to the user specified minimum support. Such item sets are called frequent item sets.
- Use the frequent item sets to generate the desired results. The general idea is that if, say ABCD and AB are frequent item sets, and then we can determine if the rule $AB \rightarrow CD$ holds by checking the following inequality.

Support (ABCD)/Support (AB) is greater than or equal to confidence.

The discovery of association rules is solely dependant on the frequent sets. Thus, a majority of algorithms are concerned with efficiently determining the set of frequent item sets in a given transaction database. The problem is essentially to compute the frequency of occurrences of each item set in the database. Since the total number of item sets is exponentially in terms of the number of items, it is not possible to count the frequencies of these sets by reading the database in just in one pass. The number of counters is too many to be maintained in single pass. As a result, having multiple passes for generating all the frequent item sets is unavoidable. Thus, different algorithms for the discovery of association rules aims at reducing the number of passes by generating candidate sets, which are likely to be frequent sets. In other words these algorithms attempt to eliminate, as early as possible all those sets which can be estimated to be infrequent sets. The algorithms differ from one another in a method of handling the candidate sets and the method of reducing the number of database passes.
Rapid Association Rule Mining (RARM) is one such algorithm to generate large l-item sets and 2-itemsets quickly without scanning the database and without candidate 2-itemset generation.

II. PROBLEM DESCRIPTION

The problem of mining association rules can be formally described as follows: Let I = $\{a_1, a_2, \ldots\ldots, a_m\}$ be a set of literals called items. Let ***D*** be a database of transactions, where each transaction ***T*** contains a set of items such that $T \subseteq I$. An ***item set*** is a set of items and a K-item set is an item set that contains exactly k items. For a given item set $X \subseteq I$ and a given transaction ***T***, ***T*** contains X if and only if $X \subseteq T$. Let σx be the ***support count*** of an item set X, which is the number of transactions in ***D*** that contain X. Let s be the ***support threshold*** and |***D***| be the number of transactions in ***D.*** An item set X is large or frequent if $\sigma_x >= |D|.\ s\%$.
An **association** rule is an implication of the form X→.Y, Where $X \subseteq I$, $Y \subseteq I$ and $X \cap Y = 0$. The association rule X → Y holds in the database ***D*** with ***confidence C%*** if no less than C% of the transactions in ***D*** that contain X also contain Y. The association rule X → Y has ***support 8%*** in ***D*** if $\sigma_{X \cup Y} = |D|.\ s\%$. For a given pair of confidence and support thresholds, the problem of mining association rules is to discover all rules that have confidence and support greater than the corresponding thresholds. For example, in a computer hardware shop, the association rule Scanner → Printer means that whenever customers buy scanners, they also buy printers c% of the time and this trend occurs s% of the time. This problem can be decomposed into two sub-problems as discussed below:

1. Finding of large item sets
2. Generation of association rules from large item sets

Most researchers addressed the first sub-problem only because it is more computationally expensive. Moreover, once the large item sets are identified, the corresponding association rules can be generated in a simple and straightforward manner. Hence, only the first sub-problem will be addressed in our project.

2.1 Our Approach

In our project, a new algorithm called Rapid Association Rule Mining (RARM) is proposed to further push the speed barrier so that association rule mining can be performed more efficiently in electronic commerce. To achieve large speed-ups even at low support, thresholds, RARM constructs a new data structure called Support-Ordered Trie Itemset (SOTrieIT). This trie-like tree structure stores the support counts of all 1-itemsets and 2-itemsets in the database. All transactions that arrive are pre-processed; all 1-itemsets and 2-itemsets are extracted from each transaction. The extracted information is used to update the SOTrieIT. This structure is then sorted according to the support counts of each node in descending order.

RARM uses SOTrieIT to quickly discover large 1-itemsets and 2-itemsets without scanning the database. The need to generate candidate 1-itemsets and 2-itemsets constitutes the main bottleneck in large item set generation. Therefore, by eliminating this step, RARM achieves significant speed-ups even though subsequently, it also applies the Apriori algorithm to obtain larger-sized item sets.

In computer science and data mining, Apriori is a classic algorithm for learning association rules. Apriori is designed to operate on databases containing transactions (for example, collections of items bought by customers, or details of a website frequentation). Other algorithms are designed for finding association rules in data having no transactions (Winepi and Minepi), or having no timestamps (DNA sequencing).

As is common in association rule mining, given a set of item sets (for instance, sets of retail transactions each listing individual items purchased), the algorithm attempts to find subsets which are common to at least a minimum number C (the cutoff, or confidence threshold) of the item sets. Apriori uses a "bottom up" approach, where frequent subsets are extended one item at a time (a step known as candidate generation, and groups of candidates are tested against the data. The algorithm terminates when no further successful extensions are found.

III. APRIORI ALGORITHM

Apriori uses breadth-first search and a hash tree structure to count candidate item sets efficiently. It generates candidate item sets of length *k* from item sets of length *k* − 1. Then it prunes the candidates which have an infrequent sub pattern. According to the downward closure lemma, the candidate set contains all frequent *k*-length item sets. After that, it scans the transaction database to determine frequent item sets among the candidates. For determining frequent items quickly, the algorithm uses a hash tree to store candidate item sets. This hash tree has item sets at the leaves and hash tables at internal nodes (Zaki, 99). Note that this is not the same kind of hash tree used in for instance p2p systems

Apriori, while historically significant, suffers from a number of inefficiencies or trade-offs, which have spawned other algorithms. Candidate generation generates large numbers of subsets (the algorithm attempts to load up the candidate set with as many as possible before each scan). Bottom-up subset exploration (essentially a breadth-first traversal of the subset lattice) finds any maximal subset S only after all $2^{|S|} - 1$ of its proper subsets.

3.1Example

This example suggests the process of selecting or generating a list of likely ordered serial candidate item sets. The techniques goal is to construct a set of *k* node ordered serial item sets from *k* − 1 length item sets. For example, with *k* = 4, suppose there are two such sets of length *k* − 1...

$A \rightarrow B \rightarrow C,$

and

$A \rightarrow B \rightarrow D,$

two candidate item sets are generated, namely

$A \rightarrow B \rightarrow C \rightarrow D$

and

$A \rightarrow B \rightarrow D \rightarrow C.$

```
Algorithm
Apriori $(T, \varepsilon)$
$L_1 \leftarrow \{$ Large 1-itemsets}
$k \leftarrow 2$
While $L_{k-1} \neq \varnothing$
$C_k \leftarrow$ Generate (Lk − 1)
for transactions $t \in T$
$C_t \leftarrow$ Subset (Ck, t)
for candidates $c \in C_t$
$\mathrm{count}[c] \leftarrow \mathrm{count}[c] + 1$
$L_k \leftarrow \{c \in C_k | \ \mathrm{count}[c] \geq \varepsilon\}$
$k \leftarrow k + 1$
return $\bigcup_k L_k$
```

3.2 Data Structure

A **trie** is a tree structure whose organization is based on a hey space decomposition. In key space decomposition, the key range is equally subdivided and the splitting position within the key range for each node is predefined. An ***alphabet trie*** is a trie used to store a dictionary of words.

3.3 A Complete TrieIT

Given a database ***D*** of transactions, we store it as a forest of lexicographically-ordered tree nodes known **as Trie Itemset** (TrieIT) so that the first sub-problem of association rule mining-finding large itemsets can be done efficiently. Let the set of items I = {a_1,a_2,.... ,a_N} in the database be ordered so that for any two items a $_i$∈I, a_j ∈ I (1 ≤i, j≤ N), ai < aj if and only if i < j. Likewise, every transaction ***T*** ∈ ***D*** is also ordered with respect to the ordering in I.

3.3.1 DEFINITION

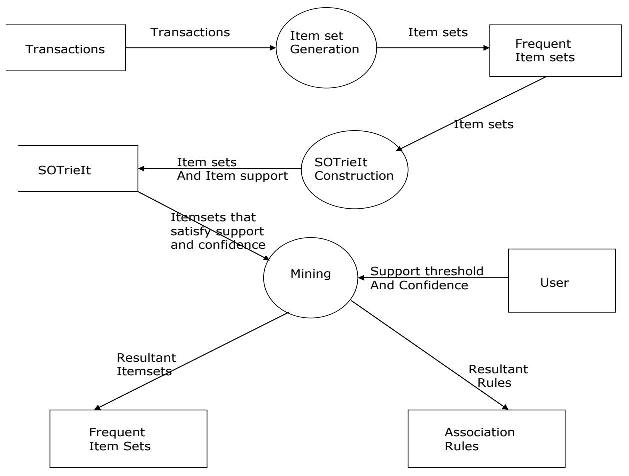
A complete TrieIT is a set of tree nodes such that every tree node w is a .2-tuple (w $_1$, w,) where w $_1$ ∈ I is the label of the node and w_C is a support count. Since every tree node corresponds to some item ai E I, for brevity, we also use w_i to refer to a tree node that corresponds to ai ∈ I. The following conditions hold:

1. Let, C (w;) be the ordered set of children nodes *of* w_i. *If* C (w_i*)* # 0, *then C (wi) c* {w_{i+l}, w_{i+2} , ... ,w_N*}.*

2. Given a node w_i, let w_k, w $_{k+l,...,}$$w_{i-l}$(l≤k≤i-1) be the set *of nodes* on the path from the root *to the* parent *of wi, then* w_c *(the* count *of wi) is a count of the* item set{a_k,a_{K+1},,. . . , ai} in the database**.** Hence, the support count *of* any k-itemset can be obtained by following a set *of* nodes *to* a depth *of* k**.** Each complete TrieIT W_i corresponds *to* some ai w ∈ ***I*** such that the root node has label ai. Then D is stored as a *set of* complete TrieITs denoted by W where

W ⊆ {w_1, w2,..., W_N}. I

In order to improve the efficiency of finding large item sets using complete TrieITs, every transaction ***T*** ∈ ***D*** is inserted into ***W*** as follows: Every ordered itemset X ∈ ***P (T) (P (T)*** is the power set of ***T)*** increases the count by one of node w_j of TrieIT $\mathbf{W_i}$ where ai and aj are the first and last items of X respectively. If TrieIT $\boldsymbol{W_i}$ or node wj does not exist, it is created with an initial count of one. Thus, each ***T*** ∈ ***D*** updates the support counts of all its sub-itemsets (from its powerset) in the corresponding TrieITs of ***W.*** Hence, no database scanning is required subsequently as the support counts are already stored in the TrieITs.



IV. IMPLEMENTATION ISSUES

4.1 Support-Ordered Trie Item set

This approach builds on the ideas presented in the paper on DHP . In that paper, it is discovered that generation of large 2-itemsets is the main performance bottleneck. Using a hash table, DHP is able to improve performance significantly by reducing the size of the candidate 2-itemsets. Similarly, this approach seeks to find a data structure that allows for quick generation of large 2-itemsets without the heavy processing and memory requirements of the previous two structures. The solution is a 2-level support-ordered tree which is called a **SOTrieIT** (Support-Ordered Trie Itemset).

It is unnecessary and inefficient to go beyond two levels because the memory requirements will far outweigh the computation savings.

4.1.1 DEFINITION

A **SOTrieIT** is a complete TrieIT with a depth of 1; i.e., it consists only of a root node wi and some child nodes. Moreover**,** all nodes in the forest of SOTrieIT are sorted according to their support counts in descending order from the left.

4.2 ALGORITHM RARM

In this section, the RARM algorithm that uses a SOTrieIT is described.

4.2.1 Pre-processing

For every transaction that arrives, l-itemsets and 2-itemsets are first extracted from it. For each itemset, the SOTrieIT, denoted by Y, will be traversed in order to locate the node that stores its support count. Support counts of 1-itemsets and 2-itemsets are stored in first-level and second-level nodes respectively. Therefore, this traversal requires at most two redirections which make it very fast. Y will then be sorted level-wise from left to right according to the support counts of the nodes in descending order. If such a node does not exist, it will be created and inserted into Y accordingly. Similarly, Y is then sorted after such an insertion.

<u>Pre-processing Algorithm</u>

```
1.  Let Y be a set of SOTrieITs
2.  for ($k = 1$; $k \leq 2$; k++) do begin
3.  Obtain all k-item sets of the transaction and
    Store them in Ck
4.  for each item set $X \in C_k$ do begin
5.  Traverse Y to locate nodes along the path that
    represents X
6.  if such a set of nodes exists in Y then
7.  Increment the support count of the leaf node
8.  Sort updated node among siblings according to
    its new support count in descending order
9.  Else
10. Create a new set of nodes with support counts of
    1 that represent a path to X
11. Insert them into Y according to their support
    counts in descending order from the left
12. end if
13. end for
14. end for
```

4.2.2 Mining of large item sets

The SOTrieIT, Y, is first traversed to discover large l-itemsets and 2-itemsets. In our approach, depth-first search is used, starting from the leftmost first-level node. As Y is sorted according to support counts, the traversal can be stopped the moment a node is found not to satisfy the minimum support threshold. After large 1-itemsets and 2- itemsets are found, the algorithm proceeds to discover other larger itemsets using the Apriori algorithm.

<u>Mining Algorithm</u>.

```
1.  Let $N^{P}_{Q}$ be the qth child node of parent node p.
2.  Let $NC_p$, be the number of child nodes under node
    1
3.  Let $I_n$, be the item set represented by node n.
4.  for (x=1; $x \leq NC_{Root.}$; x++) do begin
5.  Let $X = N_x^{Root.}$
6.  if ox $\geq$ |D| . s% then begin
7.  Add $I_x$ to $L_1$.
8.  for (y=l; y <= $NC_x$; y++) do begin
9.  if $\sigma_{Ny}^{x}$>=|D| . s% then
10. Add $I_{Ny}^{x}$ to $L_2$.
11. end for
12. end if
13. end for
14. Run Apriori from its third iteration to find the rest
    of the large item sets from 3-itemsets onwards.
```

V. CONCLUSION

The rising popularity of electronic commerce presents new challenges to association rule mining. Due to the easy availability of huge amount of transactional data, there is a greater need for faster and scalable algorithms to exploit this knowledgebase. We have proposed a new algorithm called RARM which uses an efficient trie-like structure known as the SOTrieIT. By eliminating the need for candidate item set and 2-itemset generation, RARM is able to achieve significant speed-ups. Experiments have shown that RARM is much faster than Apriori and FP-growth. It also maintains its sharp edge at various support thresholds and is scalable. Though there are additional pre-processing and storage requirements, they are both insignificant and worthwhile considering the immense speed-up they can help achieve. The SOTrieIT is a simple and yet highly dynamic structure which can be put to greater use in association rule mining. Due to the dynamic nature of the Internet, current algorithms are inadequate to handle web-based databases. The SOTrieIT may be a good tool to impart dynamism into static algorithms. Hence, methods and algorithms to exploit the SOTrieIT will be explored in future work.

VI. REFERENCES

1. C. C. Aggarwal and P. S. Yu. Online generation of association rules. In ***Proc.*** 14th ***Int. Conf.*** **on** ***Data*** Engineering, pages 402-411, Orlando, Florida, USA, 1998.
2. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In ***Proc. 20th*** Int. ***Conf.*** on Very Large Databases, pages 487-499, Santiago, Chile, 1994. M. Berry and G. Linoff. ***Data*** mining ***techniques: for*** marketing, sales, and ***customer*** support. Wiley Computer Pub., New York, 1997.
3. A. Berson, S. Smith, and K. Thearling. Building data mining applications for ***CRM.*** McGraw-Hill, New York, 2000.
4. D. W. Cheung, J. Han, V. T. Ng, and C. Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. In Proc. ***12th Ink Conf.*** on Data Engineering, pages **106-114,** New Orleans, Louisiana, 1996.
5. D. W. Cheung, S. D. Lee, and B. Kao. A general incremental technique for maintaining discovered association rules. In ***Proc. 5th*** Znt. Conf. ***on Database*** Systems ***for Advanced Applications,*** pages 185- 194, Melbourne, Australia, 1997.
6. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In ***Proc. ACM SIGMOD*** Int. Con& on Management ***of Data,*** pages 1-12, Dallas, Texas, 2000.
7. C. Hidber. Online association rule mining. In Proc. ***ACM SIGMOD Int. Conf.*** on Management ***of*** Data, pages 145-154, Philadelphia, Pennsylvania, USA, 1999.
8. J. S. Park, M. S. Chen, and P. S. Yu. Using a hash-based method with transaction trimming for mining association rules. ***IEEE Tkans.*** on ***Knowledge*** and ***Data*** Engineering, 9(5):813%324, Sept. 1997. M. R. Sanders. Global ecommerce ***Approaches Hypergrowth.*** http://www.forrester.nl/ER/Research/Brief/Excerpt/0, 1317, 9229,FF.html, April 2000.
9. N. L. Sarda and N. V. Srinivas. An adaptive algorithm for incremental mining of association rules. In ***Proc. 9th Irk*** Conf. on ***Database*** and ***Expert Systems,*** pages 240-245, Vienna, Austria, 1998.