A Near Optimal Approach for Top-K Frequent Itemset Mining

Shorya Agrawal[1] and Nirved K. Pandey[2]

[1] *Research Scholar, RGPV Bhopal, India*
[2] *Director, GEC Gwalior, India*
agrawal_shorya@rediffmail.com, nirved_pandey@yahoo.co.in

Abstract

Mining of top-k items is much useful to a user than mining transactions for minimum support threshold. User may only provide expected minimum support after careful scanning of transaction. Still experience and expertise would be required. However user can much more easily project expected number of items to be included as per requirements. For this purpose, some approaches have been suggested but they rely on FP Tree modification. We have implemented another efficient technique for mining frequent itemsets from web logs. This technique is termed as WRDSP for Web Access Pattern Relative Dot Sequence Path. In this paper, we demonstrate this technique for finding frequent itemsets in case of transactions naming it as RDSP. After that we show, how this technique may be suitably modified for mining top-k itemsets. This technique scores over existing efficient techniques, which had been used in recent times. In this technique, each transaction updates the existing graph created by previous transactions, modifying the RDSP value associated with the link. The unique feature of the created RDSP graph is that it contains nodes equal to total number of items only. This significantly reduces the processing time and memory space required for ARM. The technique works optimally for small and moderate size database. Large databases give rise to enhanced RDSP, which are cumbersome in updating. Still, saving in number of access of database and efficient handling of generated RDSP graph achieved by the proposed technique make it a strong candidate for determining top-k itemsets.

Keywords: *Relative Dotted Sequence Path (RDSP), Frequent Pattern FP, Association Rule Mining (ARM)*

1. Introduction

Association rule mining is important ingredient of data mining [1]. There are many techniques used for finding frequent itemsets [2, 3]. The Apriori algorithm performs a breadth-first search. Apriori algorithm is a layered search iterative method based on frequency set theory, The core idea is searching for (k+1) item sets through the k item sets, resulting in finding the relationship between database project, in order to form the rules. This algorithm includes two steps. The first step is to identify all the frequent item sets, eliminating items having support degree not less than the minimum support degree, which the user specifies; The second step is generation of association rules using frequent itemsets found in first step.

In practical applications, Apriori algorithm has some shortcomings. Multiple scanning of database raises time complexity of the algorithm. If the database is very large, it may produce numerous candidates, causing next stage process to become time costlier, which further increases the time complexity. Therefore, in order to improve the efficiency of the Apriori algorithm, a new efficient mining algorithm without candidate set was proposed. Many

variants of the Apriori algorithm have been developed, such as AprioriTid, AprioriHybrid, direct hashing and pruning (DHP), dynamic itemset counting (DIC), Partition algorithm, etc [2, 4]. Apriori algorithm based approaches encounter the problem that multiple scans of the database are required in order to determine; which candidates are actually frequent.

Due to the shortcomings of Apriori algorithm another algorithm known as FP tree algorithm was proposed [5]. This approach follows divide and conquer strategy and produces frequent set from FP tree. The highly condensed data structure: FP tree benefits FP-Growth with better performance than the Apriori-like algorithms. It is about an order of magnitude faster than Apriori algorithm. FP tree data structure is developed for storing patterns from the transaction database. FP-Growth requires two database scans for FP tree construction. During first scan, it finds set of ordered frequent items. A transactional pattern base is constructed during this first scan. During second scan, FP tree is constructed. This Transactional pattern base is substantially smaller in size than the transactional database without loss of any information required for building the FP tree. FP tree is constructed by scanning the transactional pattern base instead of transactional database. This is done by generating first conditional pattern from FP tree. From conditional pattern finally, frequent patterns are extracted.

FP-Growth too has some deficiencies. It needs to recursively create huge amounts of conditional pattern bases and corresponding conditional FP trees during mining process. When the dataset is huge, both the memory usage and computational cost are expensive. FP tree is normally smaller in size in comparison with original database. Sometimes even the FP tree itself cannot meet the memory requirement. Because of the huge storage requirement having limited speed of the sequential FP-Growth, parallel algorithm becomes essential for large scale data warehouse mining. Most previous studies [6, 7] parallelized the FP-Growth algorithm in a shared memory system.

A related method termed as QFP based ARM, was presented in [8]. Through scanning the database only once, these algorithm can convert a transaction database into a QFP tree after data preprocessing, and then do the association rule mining of the tree. This algorithm performs better than FP-Growth algorithm, and retains the complete information for mining frequent patterns. It does not destroy the long pattern of any transaction and significantly reduce the non-relevant information. Previous approaches in this direction were [9] and [10].

Another notable approach was FP-split algorithm [11]. It works in two main steps. The first step is to scan transaction database once for generating equivalence classes of frequent items. The second step is to sort these equivalence classes of frequent items in descending order so as to construct the FP-split tree. Previous attempts in this direction were [12].

There have been many other attempts upon improving the time complexity of ARM. So far no attempt may claim that it has achieved optimum performance. RDSP based approach to mine association rules is another such fresh attempt. We claim that for moderate sized database, this approach achieves near optimum results.

For finding frequent itemset, minimum support provided by the user is used for elimination of items that are below a threshold. This causes lesser number of items to be catered for, which helps in generating pertinent graph as well as analyzing it for generation of frequent items set. So far some approaches have been put forward [13, 14, 15, 16]. We have examined veracity of WRDSP in context of web logs. In this paper, after demonstrating proposed approach for mining frequent itemset in transactions, this paper briefly shows how RDSP may be modified for finding top-k itemsets.

The next sections are organized as follows. RDSP approach has been discussed in section 2. Section 3 deals with performance analysis. It compares RDSP based ARM and FP-Growth

tree and provides brief projection about how RDSP may be modified in case of mining top-k itemsets. Remaining sections deal with conclusion and references.

2. The Relative Dotted Sequence Path (RDSP) BASED ARM

In this section, first we describe technique. RDSP stands for Relative Dotted Sequence Path. The method for calculation of frequent item set named as RDSP based ARM is demonstrated with the help of an example. Algorithm runs in three major steps. These steps are as follows:

1. Each transaction is transformed according to descending frequency order.

2. RDSP graph is created, where each link is labeled with its RDSP value.

3. Frequent Item Sets are determined from RDSP graph generated in step 2.

First step is also employed in FP tree creation hence is not elaborated. Second step creates RDSP graph, which contains all nodes that have qualified to belong to the group that have support above threshold level. Creation of RDSP graph begins with creation of a starting node also known as null node along with all other qualified nodes. After this, nodes are linked with each other and with the starting node as per transformed frequent items in each transaction. The label of each link is its RDSP. Each transaction is now considered. Items are linked from null node one by one. First item of the each transaction is always connected with null node. RDSP path is created or updated with each item based on RDSP prefix path as mentioned earlier. In this way, graph has all necessary links that are labeled as per their RDSP values. The illustration of first and second major steps is done using following running example.

2.1. Running Example of RDSP Calculation

Table 2-1: Sample Database

	Items Transaction	**(Ordered) frequent items**
100	d,a,e,g	a,d,e
200	b,a	a,b
300	h,a,c,b,e	a,b,c,e
400	a,b,c,d	a,b,c,d
500	a,c,b,f,d,i	a,b,,c,d
600	b,a,c	a,b,c
700	f,a,b,i,c,d	a,b,c,d
800	a,e,c,b	a,b,c,e
900	j	

Assume that the minimum support threshold is 50%. As first step, algorithm makes the first pass through the database and finds singleton itemsets (items) with enough support as follows.

a: 8, b: 7, c: 6, d: 4, e: 3

Item ‘a’ and ‘b’ have the maximum frequency and items ‘c’,‘d’ and ‘e’ have the minimum frequency in the given transaction. All other items, whose support was below threshold value,

have been dropped. In the second pass, algorithm transforms items in each transaction in descending order of determined frequency.

In second step, RDSP graph is created. Graph has starting node also known as null node along with other nodes i.e. a, b, c, d and e. Each transaction is considered one by one. Each transaction updates an existing RDSP count or/and determines RDSP value of a newly created link.

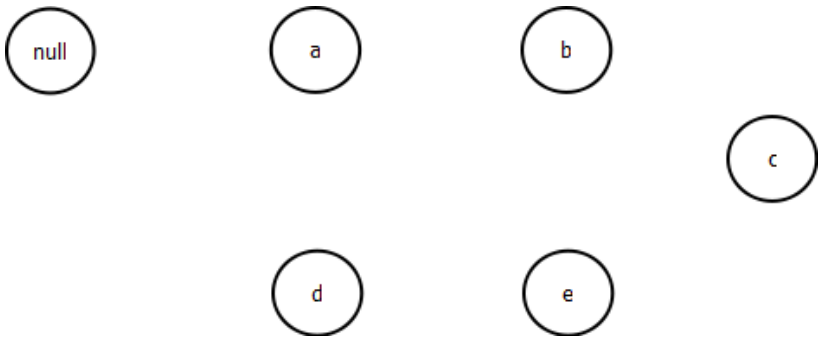

Fig. 2-1 Items with Null Node

In present case for tid = 100, a link is created from null to a with RDSP 1(1). First 1 indicates the path name and Second (1) in parenthesis indicates count value of item a. The count denotes no. of occurrences of this item. After this links are similarly created from a to d with RDSP = 1.1(1) and finally from d to e with RDSP = 1.1.1(1).

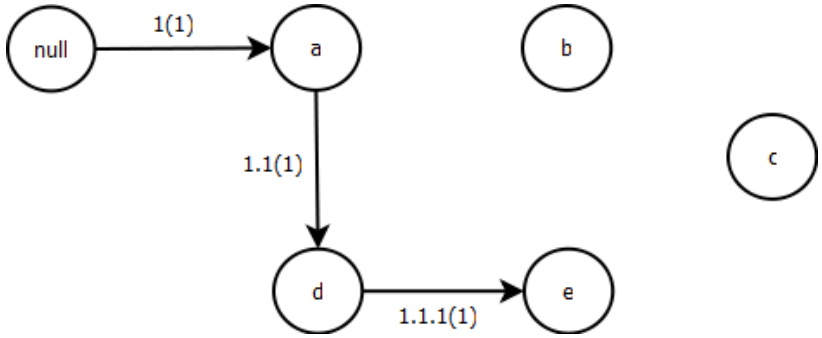

Fig. 2-2 Graph after First Transaction

Next transaction with tid = 200 has transformed frequent items as a,b. Item ‘a’ has already been connected with starting node null. Hence its count is incremented by 1, which updates RDSP of link as 1(2). Item b is to be connected from item a for the first time. So far item a has one connection only, which is from item d. Prefix till item a so far is 1. Hence RDSP of link ab is 1.2(1).

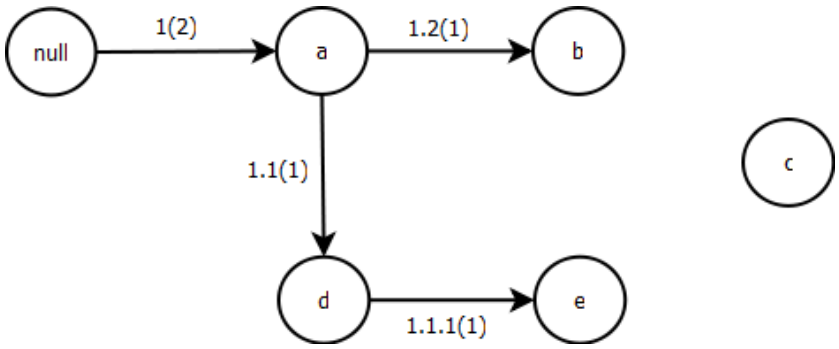

Fig. 2-3 Graph after Second Transaction

For tid = 300, RDSP of link null to a is updated as 1(3). Similarly RDSP of link ab is updated as 1.2(2). Item c is to be connected from item b for the first time. Prefix till item b so far is 1.2. Hence RDSP of link bc is 1.2.1(1). Similarly RDSP of link ce is 1.2.1.1(1).

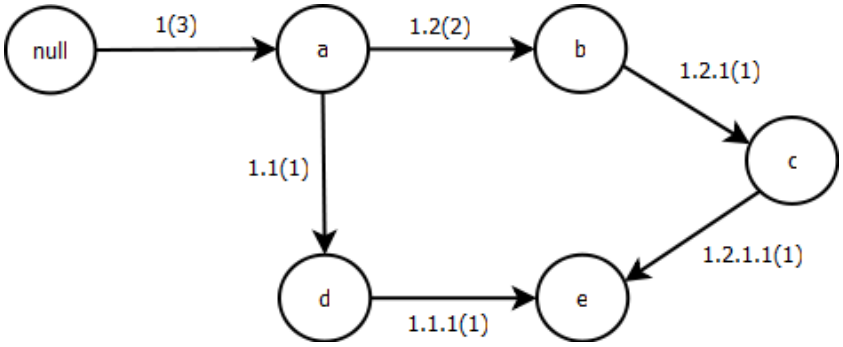

Fig. 2-4 Graph after Third Transaction

For tid = 400, transformed frequent set consists of item a, b, c and d. RDSP of link null to a is updated as 1(4). Similarly RDSP of link ab is updated as 1.2(3) and link bc is updated as 1.2.1(2). Item d is to be connected from item c for the first time. Prefix till item c so far is 1.2.1. Hence RDSP of link cd is 1.2.1.2(1) on the basis of similar reasoning.

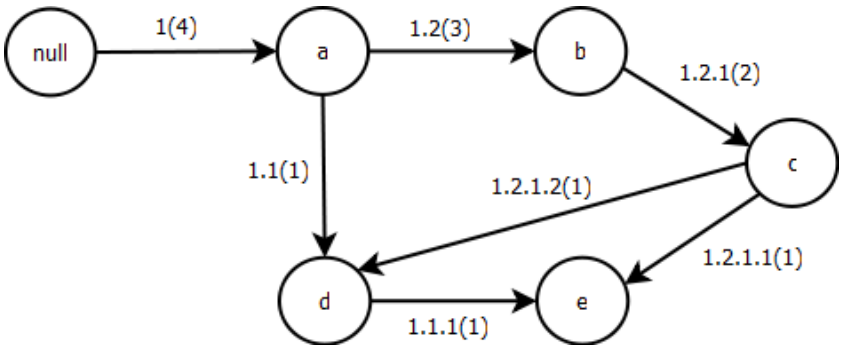

Fig. 2-5 Graph after Forth Transaction

For tid = 500, transformed frequent items are a, b, c and d, which are same as tid = 400. Hence only count values of RDSP are updated as shown below.

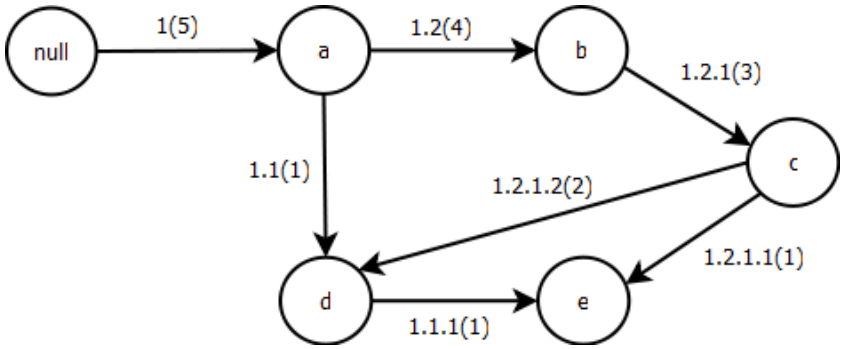

Fig. 2-6 Graph after Fifth Transaction

For tid = 600, transformed frequent set consists of item a, b and c. RDSP of link null to a is updated as 1(6). Similarly RDSP of link ab is updated as 1.2(5) and link bc is updated as 1.2.1(4) on the basis of similar reasoning.

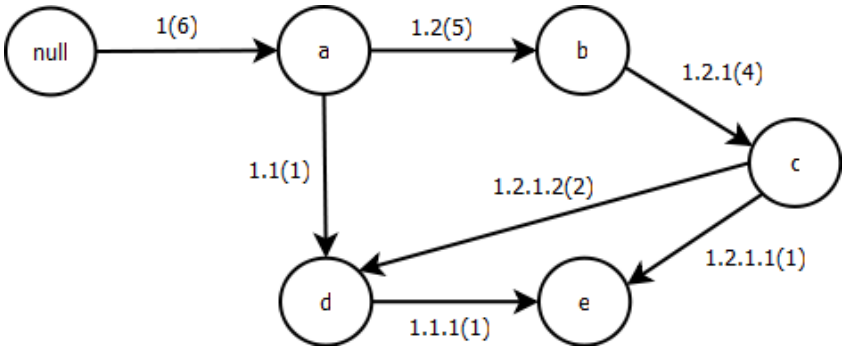

Fig. 2-7 Graph after Sixth Transaction

For tid = 700, transformed frequent items are a, b, c and d, which are same as tid = 500. Hence only count values of RDSP are updated as shown below.

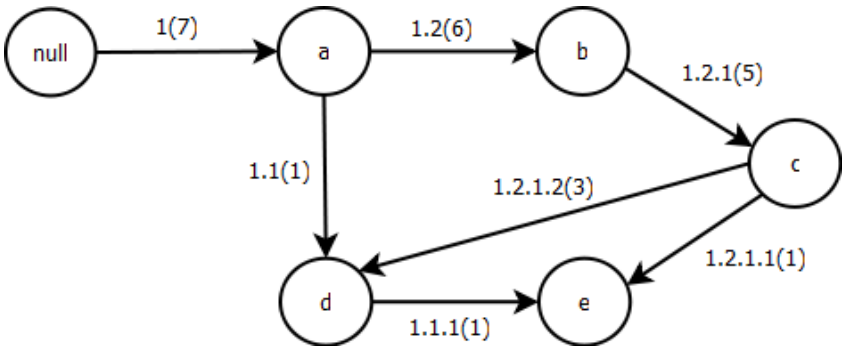

Fig. 2-8 Graph after Seventh Transaction

For tid = 800, transformed frequent items are a, b, c and e, which are same as tid = 300. Hence only count values of RDSP are updated as shown below.

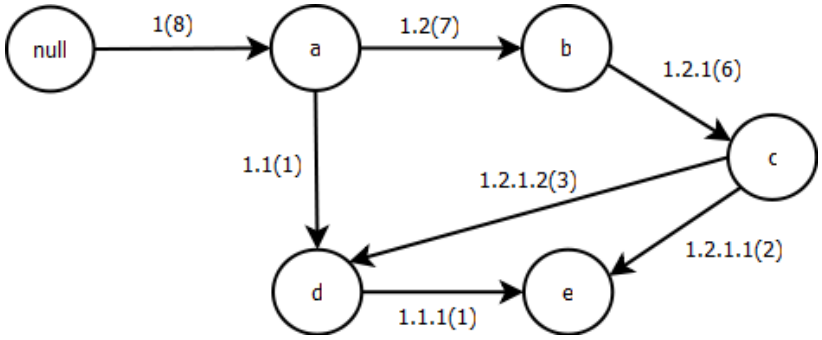

Fig. 2-9 Graph after Last Transaction

Third and final step results in generation of conditional frequent pattern by reading RDSP-graph. This is achieved in 3 steps. Initially the items are arranged from least frequent items to most frequent items. This information is available from first major step. In this running example, items are e, d, c and b. Step 1: Path is traced from e to root (null), which is available using RDSP graph generated in second major step. As shown in fig. 2-9. There are two paths from e to the root. First link joins e from c. RDSP of this link is 1.2.1.1(2). The prefix of last dotted value is 1.2.1 which is provided by link that joins c to b. Note that this prefix is not same as RDSP value of link dc, which is 1.2.1.2. Hence the path towards root is e, c, b so far. This process is repeated. At a prefix of the link is 1.2, which is provided by link joining b to a. Hence path from e is extended as e, c, b, a. Similarly the second path from e to root becomes e, d, a. For items forming first path i.e. e, c, b, a count values are initialized to the values of the RDSP of last item. Thus for item e associativity from remaining items c, b, a is initialized to the RDSP count of link e to c, which is 2. Similarly associativity of item c from item b, a is initialized to 2, associativity from b to a is initialized to 2. The last item a is associated with null.

Counts are final for each of the association and need not be updated. This is a significant improvement over FP tree based ARM. In FP tree, while finding associativity, calculation at each link may be repeated several times. This happens as items themselves are repeated because one tree is formed for each path. In RDSP based approach, each link traced so far will never be required to be considered for next associativity calculation. So far the associativity is as follows.

e → cba(2)

d →

c → ba(2)

b → a(2)

Taking up, second link from e to root i.e. e, d, a, the associativity of each item is similarly updated to obtain given set.

e → cba(2), da(1)

d → a(1)

c → ba(2)

b → a(2)

So far items encountered are e, c, b and a. From item e, both links ec and ed have been considered. In the list of remaining items, next explored least frequent item is item d. From item m links da and dc traverse towards starting node. Link da has already been considered. To emphasize the merit over FP tree algorithm, link da will not be considered anymore. Link dc will now be considered. This will update the list of items not considered so far. This process continues until all items of the list are considered. Link dc results in new associativity relation d, c, b, a. The updated associativity status becomes:

e → cba(2), da(1)

d → a(1), cba(3)

c → ba(5)

b → a(5)

Process is repeated at link cba, making the final status as follows.

e → cba(2), da(1)

d → a(1), cba(3)

c → ba(6)

b → a(6)

Process is repeated at link ba, making the final status as follows.

e → cba(2), da(1)

d → a(1), cba(3)

c → ba(6)

b → a(7)

Step 2: Frequency of each item is counted with each least frequent item. The frequencies of these items are added. In the case of running example, items d, c, b and a are associated with item e. The frequencies of association of these items are 1, 2, 2 and 3 respectively. The frequency of item a is summed up as 3 from association e, c, b, a with frequency 2 and association e, d, a with frequency 1 . As threshold value is 3 or more, only item a is retained in final association status. This process is repeated for all items. Final associativity status is as follows.

e → a(3)

d → a(4), c(3), b(3)

c → b(6), a(6)

b → a(7)

Step 3: For each association, all combinations of the item that includes the item being considered itself are generated. In the running example, for association e → a(3), single frequent item sets (FIS): ea(3) is generated. For association d → a(4), c(3), b(3), FIS: ad(4), cd(3), bd(3), acd(3), abd(3), cbd(3) and acbd(3) are generated. Similarly for item c, FIS:

bc(6), ac(6) and bac(6) are generated and for item b, FIS ab(7) is generated. This achieves the final goal.

2.2. RDSP Algorithm The creation of the proposed graph may be described as follows:

```
RDSP_ARM (D, S)
{
   Freq ← frequency of each item from D without items
            with min_support
   D ← itemset in non-increasing order by Freq.
   G ← starting node
   For all Tran ϵ D
         Create RDSP_graph (Tran, G, Freq)
   End For
   res_link ← null
   curr_node ← no. of Freq.item
   While curr_node!= 0
         curr_ref ← Freq[curr_node].ref
      While curr_ref != null
          pre_ref ← curr_ref. previous_ref
          If curr_ref.RDSP.prefix = prefix_ref.RDSP
          Then
              res_link.curr_ref.link_item ← pre_ref.item
              ►olc is a output link count
              If curr_ref.olc = 0
              Then
                   pre_ref.olc --
                   curr_ref.previous_ref ← null
                   pre_ref.next_ref ← null
              End if
              curr_ref ← pre_ref
               For next_ref ϵ current node to last node
                  append(pre_ref.item, res_link.next_ref
                              .link_item)
               End For
          End if
       End While
       curr_node --
  End While
  fre_link ← Frequency of each item's is counted in res_link
  Print frequent set is created of fre_link
}

RDSP_graph (Tran, G, Freq)
{
   ►Tran is a row of each transaction
    prev_ref ← G
    prev_prefix ← null
     link ← each item's in Tran
```

```
►link is queue having each of the transaction
curr_node ← 0
While link != 0
  If  is not existing in Freq.ref for link[curr_node]
    curr_ref ← Create new node of link[curr_node]
  End If
  R1 ← prev_ref.next_ref
  R2 ← curr_ref
  If R2!=R1 OR (R2=R1 AND prev_prefix !=
    curr_ref.RDSP.prefix)
  Then
     path ← prev_prefix.(prev_ ref.olc +1)
    ►olc is a output link count
    prev_ref.olc ← prev_ref.olc+1
    curr_ref.olc ← 0
    curr_ref.RDSP ← path
    prev_prefix ← path
    prev_ref.next_ref ← curr_ref
    curr_ref.previous_ref ← prev_ref
    prev_ref ← curr_ref
    curr_ref.Icount ← 1
    Freq.link[curr_node].ref ← curr_ref
  Else
    prev_ref ← curr_ref
    prev_prefix ← curr_ref.RDSP
    curr_ref.Icount ← curr_ref.Icount+1
  End If
curr_node ← curr_node +1
End While

}
```

3. The Performance Analysis

In this section, RDSP based ARM is compared with FP-Growth technique. FP tree growth algorithm has been briefly discussed in section 2. RDSP based ARM was described in previous section. At the end of this section, we project how RDSP may be adapted for mining top-k itemsets.

3.1. ARM with Other Techniques

RDSP based ARM has been compared with well-known algorithms for mining frequent closed itemsets i.e. FP tree [5]. Experiments were performed on a 266-MHz Pentium PC machine with 128 megabytes main memory. Absolute number of runtime are not directly compared with those in some published reports running on the RISC workstations because different machine architectures may differ greatly on the absolute runtime for the same algorithms. Run time implies the total execution time, that is, the period between input and output, instead of CPU time measured in the experiments in some literature. Run time is a more comprehensive measure since it takes the total running time consumed as the measure of cost, whereas CPU time considers only the cost of the CPU resource. Also, all reports on

the runtime of FP-Growth include the time of constructing FP trees from the original databases.

The test comparison is performed by Mushroom. This data set is generated by IBM synthetic data generator, which is a standard transactional database used in FP tree and many other techniques. The scalability of RDSP with FP tree as the support threshold decreases from 10% to 0.1% is shown in Figure 3-1. RDSP scales much better than FP tree because as the support threshold goes down, the number as well as the length of frequent itemset increase exponentially. On the basis of the results it could be affirmed that RDSP algorithm performs much faster than FP tree construction and other approaches.

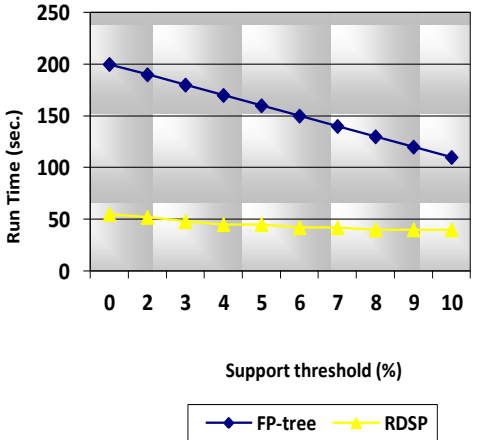

Fig. 3-1 Scalability with Threshold

Scalability with the number of transactions versus running time is taken up next. The support threshold is set to 7%. The results are presented in Figure 3-2, FP-Growth and RDSP algorithms show linear scalability with the number of transactions from 100K to 500K. However, RDSP though much more scalable than FP tree, tends to become comparable to FP split method. As the number of transactions grow up, the difference between the FP tree and RDSP approaches becomes closer.

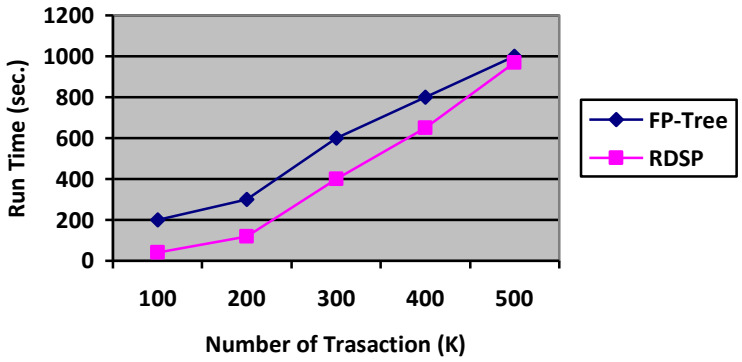

Fig. 3-2 Scalability with Number of Transactions

In top-k mining user provides minimum length of frequent sets along with k value, which

is top number of items that are associated in transactions. RDSP technique is modified to take minimum support value is zero. With this RDSP graph is plotted as usual. User supplied values are used to determine how much to traverse and which paths to discard entirely. In given running example assuming the RDSP graph with zero support, top two items may be found by traversing two links from starting node.

4. Conclusion & Future Scope

In this paper, first a technique for determining association rules using RDSP graph has been demonstrated. The merit of RDSP approach lies in the formation of such a graph in which each item of the transaction has single occurrence. This limits the number of links required. In this way, all association rules may be generated in an optimal way. The resulting performance analysis justifies the projection. However as the size of the database becomes larger from moderate level, performance does not scale in the same proportion. This technique has been used to mine web logs and found better than conventional web access pattern tree. It may be modified to mine top-k items by taking up user's choice for minimum length and top-k items. In future this technique is to be tailored suitably in case of large databases so that its performance remains better than conventional techniques.

References

[1] J. Han and M. Kamber, "Data Mining: Concepts and Techniques", 2nd Edition, Morgan Kaufmann Publishers, August 2006.

[2] R. Agrawal, T. Imielinski, and A.N. Swami, "Mining AssociationRules between Sets of Items in Large Databases," Proc. ACM SIGMOD Int'l Conf. Management of Data, pp. 207-216, May 1993.

[3] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules", VLDB (1994), 487-499.

[4] R. Srikant and R. Agrawal, "Mining Sequential Patterns: Generalizations and performance improvements", Proceedings of the Fifth International Conference on Extending Database Technology, (Avignon, France, 1996), Springer-Verlag, vol. 1057, 3-17.

[5] J Han, J Pei and Y Yin, "Mining frequent patterns without candidate generation", Proc ACM-SIGMOD, Dallas, TX, USA, ACM Press May 2000, Vol. 29, No. 2, pp. 1-12.

[6] Osmar R. Zaane, Mohammad El-Hajj, and Paul Lu, "Fast parallel association rule mining without candidacy generation", First IEEE International Conference on Data Mining, San Jose, California, USA, IEEE CS Press, November 2001, pp.665-668.

[7] Li Liu, Eric Li, Yimin Zhang and Zhizhong Tang, "Optimization of frequent itemset mining on multiple-core processor", 33rd International Conference on Very Large Data Bases, Vienna.

[8] Li Juan and Ming De-ting, "Research of an association rule mining algorithm based on FP tree", Intelligent Computing and Intelligent Systems (ICIS), 2010 IEEE International Conference on, Vol 1, pp. 559-563.

[9] G. Grahne, "Efficient Mining of Constrained Correlated Sets[C]", International Conference Data Engineering (ICDE), San Diego, 2000, 512-521.

[10] N. Pasquier, R. Taouil and Y. Bastide, "Generating A Condensed Representation for Association Rules", Journal of Intelligent Information Systems, 2005, 24(1):29-60.

[11] Chin-FengLee and Tsung-HsienShen, "An FP_split method for fast association rules mining", Information Technology: Research and Education, 2005, 3rd International Conference on, pp. 459 – 463.

[12] K. Wang, L. Tang, J. Him, and J. Liu, "Top Down FP-Growth for Association Rule Mining", Proceedings of the 6ih Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, 2002 pp. 334-340.

[13] Jianyong Wang, J. Han, Y. Lu and P. Tzvetkov, "TFP: an efficient algorithm for mining top- k frequent closed itemsets", Knowledge and Data Engineering, IEEE Transactions on, Volume: 17, Issue: 5, 2005, Page(s): 652 – 663.

[14] Lan Yongjie and Qiu Yong, "Efficient Algorithms of Mining Top-k Frequent Closed Itemsets", Electronic Measurement and Instruments, 2007, 8th International Conference on , 2007, Page(s): 2-551 - 2-554.

[15] P. Songram and V. Boonjing, "Mining top-k closed itemsets using best-first search", Computer and Information Technology, 2008, 8th IEEE International Conference on, 2008, Page(s): 77 – 82.

[16] S. Roy and D.K. Bhattacharyya, "Efficient Mining of Top-K Strongly Correlated Item Pairs using One Pass Technique", Advanced Computing and Communications, 2008, 16th International Conference on, 2008, Page(s): 416 – 421.

Authors

Shorya Agrawal completed his Bachelor's Degree in Computer Science from RGPV University, Bhopal, M.P., India in 2009. He is undergoing Master's Degree Program in Information Technology from RGPV University, Bhopal. His field of study is Toc, Data mining, DBMS and Distributed Computing. He has published 5 research papers.

Nirved Pandey completed his Bachelor's Degree in Electrical Engineering from Government Engineering College, Jabalpur, M.P., India in 1988. He completed his Master's Degree in Computer Technology from IIT Delhi in 1996, and doctoral studies (Ph.D. in Computer Science) from ABV-Indian Institute of Information Technology and Management, Gwalior. At present He is serving as Principal, Gwalior Engineering College, Gwalior. He has experience of over 21 years in the field of academics and research. His field of study is analysis and design of algorithms, data mining, and distributed computing. He has published more than 25 research papers.