Chapter 4
FOL-Mine – First Occurrence List Mine

A novel Web Access Pattern Mining Algorithm, FOL-Mine is proposed in this chapter. The FOL-Mine algorithm generates access patterns by suffix building in the projected data base of each frequent event and eliminates the need for construction of pattern tree. A data structure, First Occurrence List (FOL), is introduced for efficient handling of suffix building. Rebuilding of projection databases is completely eliminated in the new method. Experimental analysis of the proposed algorithm reveals significant performance gain over other web access pattern mining algorithms.

4.1 Introduction

Web usage mining, also known as Web log mining, discover interesting and frequent user access patterns from the web browsing details stored in server web logs, proxy server logs or browser logs [Kosala and Blockreel, 2000]. Web log mining has become very critical for effective Web Site Management, creating Adaptive Web Sites, business and support services, personalization, network traffic flow analysis and so on [Srivastava et al., 2000].

Sequential pattern mining is an important data mining tool used for pattern retrieval. Sequential pattern mining can also be used to capture frequent navigational paths among user trails. Web Access Pattern (WAP) is a sequential pattern in a large set of pieces of web logs. A web log is a sequence of pairs: user-id and access information. For the purpose of study of sequential pattern mining, preprocessing [Pei et al., 2000] is applied to the original log file and Web Access Sequence Database (WASD) is generated.

Sequential Pattern Mining Algorithms mainly differ in two ways: (i) the way in which candidate sequences are generated and stored and (ii) the way in which support counting is done. Based on this, there are mainly two heuristics in Sequential Pattern Mining - apriori based and pattern growth based methods.

Apriori based methods use a generate-join procedure to generate candidate sequences and then these sequences are tested for support. So they face the problem of generation of explosive number of candidates and tedious support counting while mining large sequence databases having numerous and/or long patterns.

The pattern growth-methods emerged in the early 2000s, as a solution to the problem of generate-and-test. The key idea is to avoid the candidate generation step altogether, and to focus the search on a restricted portion of the initial database. Pattern Growth methods grow frequent patterns by mining increasingly smaller projection databases. Data structures used for the representation of database and search space partitioning play an important role in the efficiency of Pattern Growth Methods.

Pattern Growth Methods generate patterns by growing the already mined frequent sequences. There are two methods for growing frequent sequences, prefix building and suffix building. Prefix building is a bottom up approach where the possibility of extending an already mined frequent pattern by prefixing a frequent event to it is explored. Whereas, in the suffix building approach, extension of already mined frequent pattern is attempted by suffixing the frequent events. FreeSpan by Han et al. [Han et al., 2000] and PrefixSpan by Pei et al. [Pei et al., 2004] are the two important proposals in this area with the latter being the most influential. As Pattern Growth methods grow frequent patterns by mining increasingly smaller projection databases, they are faster than apriori-based algorithms [Han and Kamber, 2007]. Thus, in Pattern Growth methods, as the frequent pattern grows, size of projection database gets increasingly smaller and this makes them faster than apriori-based algorithms [Han and Kamber, 2007].

In 2000, Pei et al. introduced a Web Access Pattern Tree (WAP-Tree) based method, WAP-Mine, for finding out the complete set of Web Access Patterns [Pei et al., 2000]. It was proved to be an efficient pattern growth method. Subsequent to that, many modifications were suggested to the creation and mining of WAP-Tree for improving the performance [Lu and Ezeife, 2003; Pearson, and Tang, 2008; Tang, Turkia and Gallivan, 2007; Zhou, Hui, and Fong, 2004; Zhou, Hui, and Fong, 2006].

In this chapter, FOL-Mine (First Occurrence List Mine), a new pattern growth based web access pattern mining algorithm is proposed. FOL-Mine generates patterns by suffix building. Highly efficient linked structure is used in the method to hold the Web Access Sequences and then to mine the access patterns. Concept of first occurrence of

symbols in First Occurrence forest Mine (FOF-Mine) [Pearson and Tang, 2008] is employed in the proposed method for improving performance. Occurrence of an event in a sequence is a first-occurrence if it appears for the first time in the sequence. Efficiency of the process of finding out the first occurrences of a symbol is crucial in improving the performance of pattern growth based access pattern mining. In the proposed FOL-Mine, a linked list with header node, stores the first occurrences of each event in a very compact and efficient way.

All WAP-Tree based methods involve two database scan, first to find out the frequent items and second to create tree using frequent sub-sequences. The proposed method needs only one data base scan. WAP-Tree based methods either use complicated linkages of tree nodes or construct intermediate trees for projection data bases for finding out the first occurrences. In the proposed method this is managed very efficiently by a simple linked structure.

4.2 The Background

The concepts and algorithms which forms the basis of the proposed algorithm are discussed below.

4.2.1 Sequential Pattern Mining

Sequential Pattern is a sequence of itemsets that frequently occurred in a specific order. Sequential Pattern Mining finds out the relationships between occurrences of sequential events, and also the specific order of the occurrences. Sequential Pattern Mining is used in a great spectrum of areas. In computational biology, sequential pattern mining is used to analyze the mutation patterns of different amino acids. Business organizations use sequential pattern mining to study customer behaviors. Sequential Pattern Mining is also used in system performance analysis and telecommunication network analysis.

A detailed study of Sequential Pattern Mining and related theory is given in *Section 3.2*

4.2.2 Web Access Pattern Mining

As a web access sequence contains page view information of a user in a session in a time stamp order, sequential pattern mining can be adapted for mining the frequent access patterns in the set of web access sequences, WASD. Given a Web access

sequence database WASD and a support threshold ξ, Web Access Pattern Mining mines the complete set of ξ-patterns of WASD. [Pei et al, 2000].

Section 3.3 illustrated the terminologies and theory behind Web Access Pattern Mining.

4.2.3 WAP-Tree Based Mining of Web Access Pattern

Pei et al. [Pei et al, 2000] proposed a compressed data structure known as Web Access Pattern Tree (WAP-tree) to hold web access sequences and WAP-Mine (Web Access Pattern Tree Mining) algorithm for mining web access patterns using the WAP-Tree. WAP-Tree facilitates support counting elegantly and enables the sharing of tree branch by common prefixes. It was proved to outperform all its earlier counterparts.

A detailed discussion of WAP-Mine algorithm and its significant modifications are given in *Section 3.4*.

4.3 FOL-Mine – First Occurrence List Mine

4.3.1 Motivation

The compact WAP-Tree structure and the corresponding WAP-Mine algorithm [Pei et al., 2000] performed better than all other earlier apriori methods. Studies conducted by many other researchers brought forward significant improvements to the WAP-Mine method [Lu and Ezeife, 2003; Pearson, and Tang, 2008; Tang, Turkia and Gallivan, 2007; Zhou, Hui, and Fong, 2004; Zhou, Hui, and Fong, 2006]. All these WAP-Tree based algorithms require two database scans; one for finding out the frequent items and the other for creating the aggregate tree using the frequent sub-sequences. Moreover, in spite of its compactness, the size of node and complexity of building up the WAP-Tree are disadvantageous [Zhou, Hui and Fong, 2004].

CSB-Mine algorithm is a web access pattern mining method that does not use WAP-tree. It uses a database structure called Conditional Sequence Base (CSB) to store the frequent access sequences. Using this database, intermediate projection databases are generated. The performance analysis shows that CSB-Mine outperforms WAP-Mine algorithm especially when support becomes smaller. CSB-Mine algorithm is more scalable when the input database size becomes larger [Zhou, Hui and Fong, 2006]. Motivated from this, in the proposed FOL-Mine, we have adopted the idea of eliminating WAP-Tree.

CSB-Mine has got certain drawbacks. Sub conditional sequence base are generated in each recursive call of CSB-Mine. Also, to find whether a symbol can be appended as a part of a pattern, it uses a procedure that tests whether all sequences in the sub conditional sequence base can be combined into a single sequence. All these processes are space and time consuming.

All WAP-Tree based mining algorithms except WAP-Mine employ prefix building to generate patterns recursively. They use various techniques to locate the First Occurrences of each symbol so as to build patterns. Efficiency of the process of finding out the first occurrences of each symbol becomes crucial in the mining process. In FOF-Mine [Pearson and Tang, 2008] the authors give the idea of collecting the first occurrences together through the concept of Forest of First Occurrences. This concept is used in the proposed algorithm to improve the efficiency. This avoids the reconstruction of projection databases and thus saves space and time considerably.

In the proposed FOL-Mine, a linked list with header node, stores the first occurrences of each event in a very compact and efficient way. Header node stores the count of first occurrences which gives the total support. The proposed algorithm avoids the unnecessary generations of intermediate projected databases and tedious support counting.

FOL-Mine does not use any of the functions used either in CSB-Mine [Zhou, Hui and Fong, 2006] or in FOF-Mine [Pearson and Tang, 2008].

4.3.2 The Proposed Algorithm

The proposed method involves three major components: *Main*, *FOL-Mine* and *Construct-FOL*. The *Main* algorithm reads in Web Access Sequences from the sequence database, prepares it for mining by loading it into the data structure and by setting the variables. *FOL-Mine* is the recursive mining algorithm used to mine all access patterns using the technique of suffix building. *Construct-FOL* is the algorithm used to build the first occurrence positions of the concerned frequent item in the projected database.

4.3.2.1 Algorithm: Main

Main algorithm reads access sequences one by one from Sequence Database into the linked list and registers the start address in an array of pointers, Headlist[*m*]. While scanning the access sequences, new symbols encountered are entered into the itemlist,

the list of items, with count as 1, whereas for already existing symbols corresponding count is incremented.

Node structure of the linked list where the web access sequences are stored is,

struct node {symbol *item*; next **node* ;}.

The start address of each linked list is stored in an array of pointers to structure node.

When all sequences are read into the structure, list of frequent events are formed by selecting those items in the itemlist with count greater than the absolute support. Main algorithm then calls the recursive FOL-Mine function to generate the set of access patterns. The main algorithm of the proposed method is given in Figure 4.1 and the flowchart is given in Figure 4.2.

Main algorithm to mine the complete set of Web Access Patterns that satisfy a predefined support threshold from Web Access Database, WASD

Algorithm *Main*;

Input:

1. WASD, the Web Access Sequence Database.
2. Support=Support threshold.

Output:

F=the set of sequential accesspatterns

Method:

1. m=0; L=null;
2. While eof (WASD)
    i. Read an access Sequence from file
    ii. Construct linked list
    ii register the start address in *Headlist*[m]
    iii. Update *itemset* and support
    iii. Increment m.
   End while.
3. η =support*m; // absolute support
4. Generate the set of frequent event, $\sum$
5. FOL-Mine (ε, L, η)
6. Return.

Figure 4.1: Algorithm Main

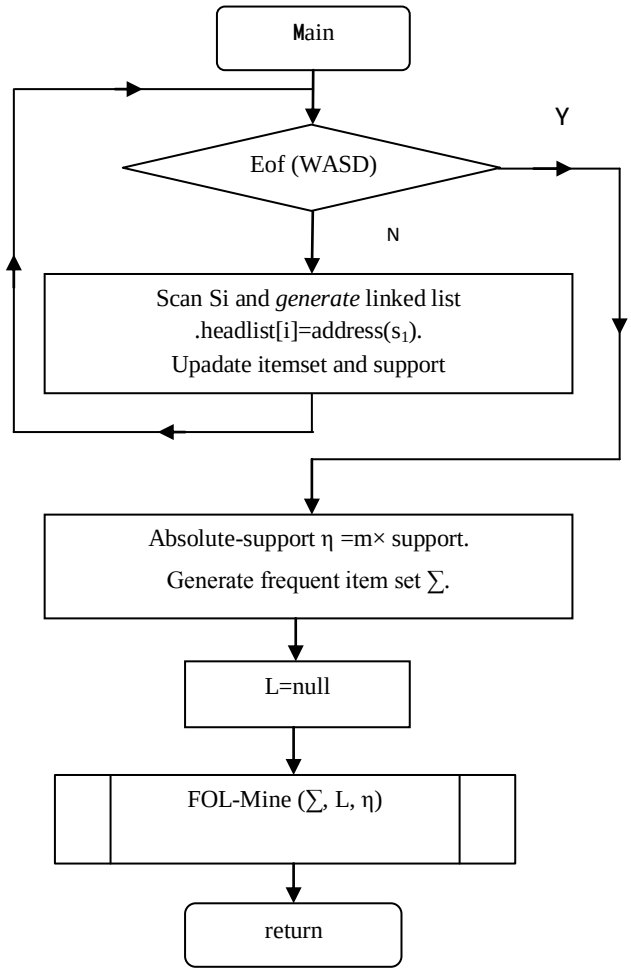

Figure 4.2: Flowchart for the Algorithm: *Main*

4.3.2.2 Algorithm: FOL-Mine

The recursive mining algorithm *FOL-Mine* works as follows: - The algorithm scans the set of frequent item. Assume that ***a*** is the current symbol. The algorithm uses the function *Construct-FOL* to generate the list of first occurrence position (FOL) *L* of ***a*** in the database. Detailed algorithm for *Construct-FOL* is given in Figure 4.6. Flowchart for the algorithm of *Construct-FOL* is shown in Figure 4.7. During the first call, function *Construct-FOL* scans the database itself to find out the first occurrence positions of the current symbol in various Access Sequences. In the subsequent calls it uses the First Occurrence List (FOL) generated in the previous call. The header of FOL contains the total number of occurrences, that is the support of ***a*** either in the database or in the projected databases as the case may be. If the support is greater than the absolute support, the symbol is attached to the previously generated pattern and the recursive procedure is carried out until no more possibilities of expansion exists. If the

support is not satisfied, the recursive call is terminated. This process is continued with all frequent symbols and terminated when no more patterns to be generated. Figure 4.3 gives the algorithm for *FOL-Mine* and Figure 4.4 depicts its flowchart.

The Recursive mining algorithm to build pattern by suffix building

---

Algorithm FOL-Mine (pattern q, FOL L, η)

1. for each $\boldsymbol{a} \in \sum$ do

   i. La← Construct-FOL (L, ***a***)

   ii. if support of ***a*** > η

   F←F U{ q .*a* }

   F'=F UFOL-Mine(q .***a*** , La, η)

   end if

   iii. delete La

   end for

2. return

Figure 4.3: Algorithm: *FOL-Mine*

4.3.2.3 Algorithm Construct-FOL

FOL is a linked structure used to store the First Occurrence Positions of a symbol. It has a header node containing label and total number of occurrences. Other nodes contain the sequence number and position of each of its First Occurrence. Figure 4.5 shows the structure of FOL.

The header of the list holds the support count of the current symbol in the data base. In the initial call the database is the original database itself, but in the subsequent calls it is the projected database. First step of the algorithm initializes the count to zero. When the call to the *Construct-FOL* is made for the first time, the input list L will be empty. If it is not empty, it means that First Occurrences of the current symbol in the projected database is to be found out. So the content of FOL acts as the starting positions of the search for first occurrences. At the end of the search the header node is updated with number of occurrence of the current symbol. The list *FOL* is returned to the calling function for further processing.

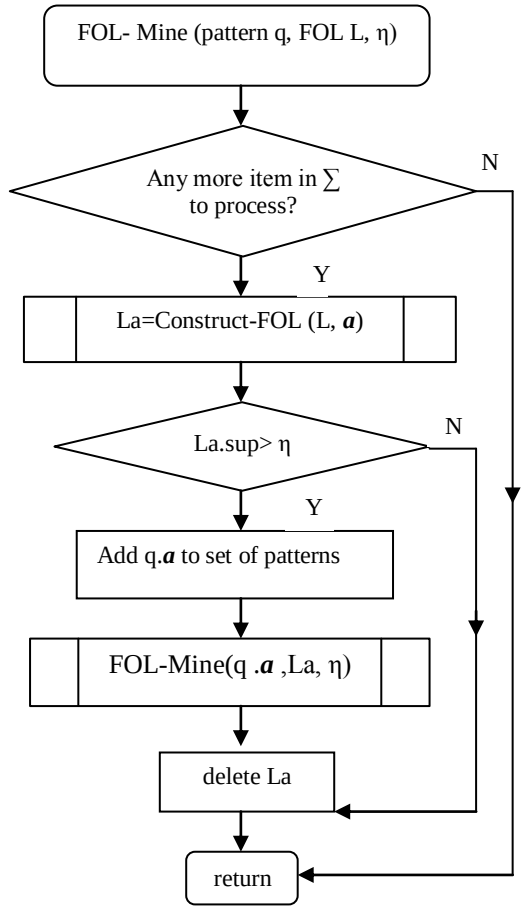

Figure 4.4: Flowchart of Algorithm *FOL-Mine*

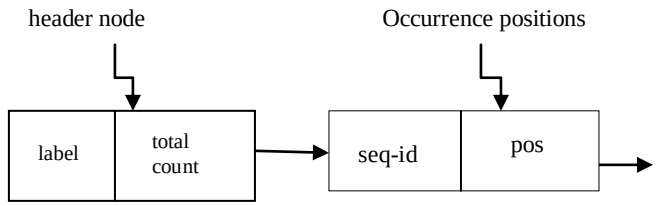

Figure 4.5: Structure of FOL

Algorithm for constructing first occurrence list

Algorithm: *Construct-FOL* (L, ***a***)

// *Algorithm for generating the first occurrence list FOL* //

1. Count_a=0

2. La = create a header node with label ***a***

3. if L is not empty

// FOL is made using FOL in the previous call//

for each item in L

find the next occurrence of item ***a***

If (exists)

create a node and attach to La

increment Count_a.

end for.

else

for each web access sequence

find the first occurrence of item *a*.

if (exists)

create a node and attach to La

increment Count_a.

end for.

End if.

4. update header node with Count_a

5. return La

Figure 4.6: Algorithm: *Construct-FOL*

Table 4.1: A Database of Web Access Sequences

User ID	Web Access Sequence
100	*abdac*
200	*eaebcac*
300	*babfaec*
400	*afbacfc*

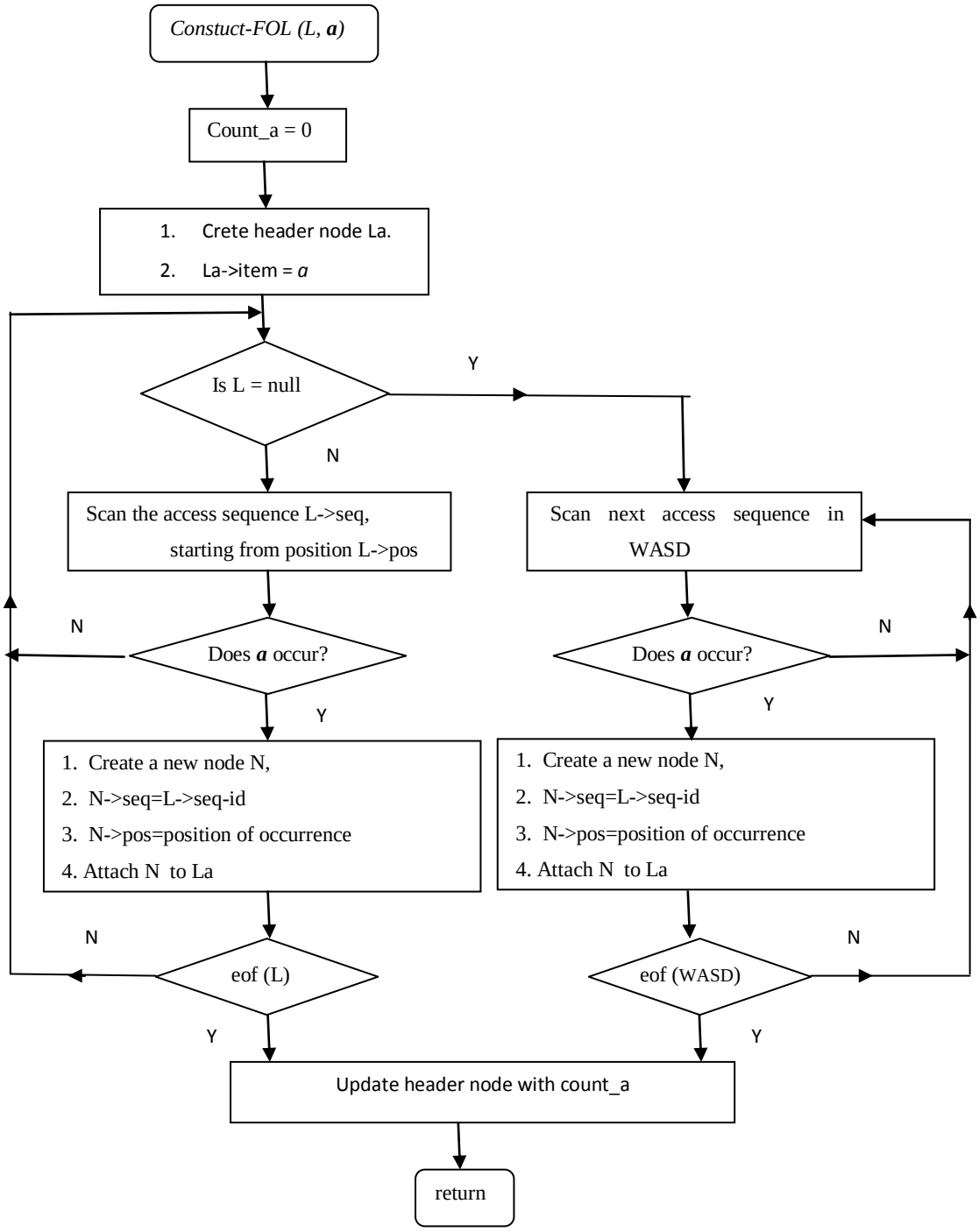

Figure 4.7: Flowchart for Algorithm: *Construct-FOL*

4.3.3 Illustration of the Proposed Algorithm with an Example

For illustration purpose a simple, preprocessed, web access sequence database with the set of access events E = {***a, b, c, d, e, f***} which is shown in Table 4.1 is considered. This database is same as that in Table 3.3, but herein frequent subsequences are not generated.

The algorithm works as follows:

Step-1 initializes m, the number of access sequences in WASD and the initial FOL. Step-2 retrieves elements of a sequence from the disk file and forms the linked list. The start address of the linked list is registered in an array of pointers, Head list. List of elements and their support are updated each time a sequence is read. Once all sequences are brought into the linked list, the format of the linked structure would be as shown in Figure 4.8.

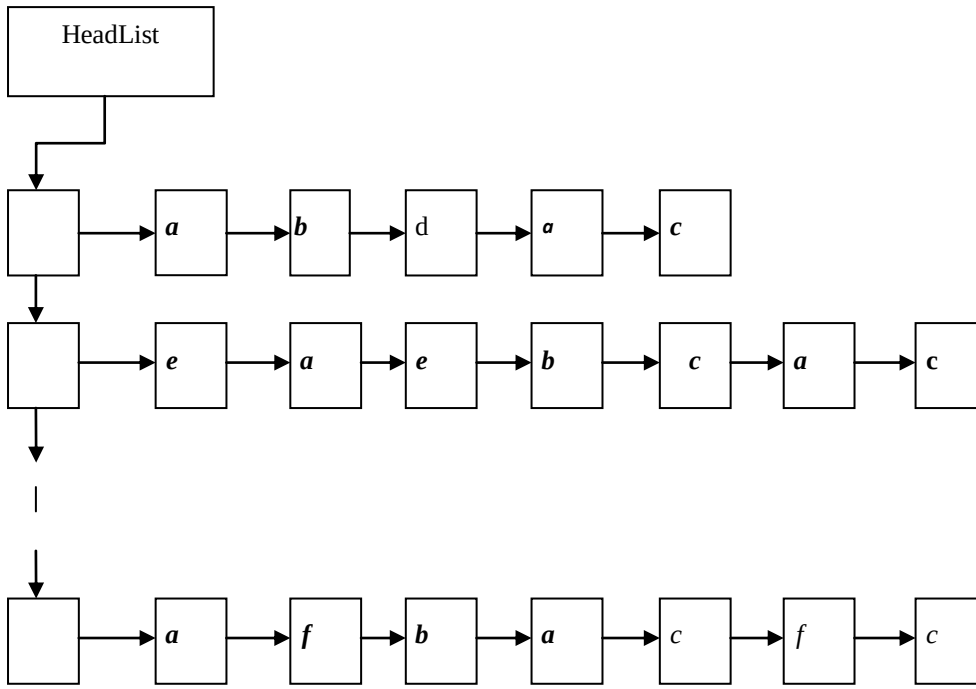

Figure 4.8: Structure of Linked List of Access Sequences

After reading all access sequences into the linked structure, *m* holds the total number of access sequences in WASD. Step 3 calculates the absolute support, η. In the example *m*=4. Let the support be 0.75. So, the absolute support η is 4×0.75=3. In step 4, events that have support greater than or equal to 3 are identified as set of frequent event ∑. In this example ∑ = {***a, b, c***}. Step 5 of algorithm calls the recursive mining function *FOL-Mine*() and generates the access patterns.

The recursive algorithm FOL-Mine starts execution by considering the first frequent event ***a***. Initial value of pattern is set to ε. First call to FOL-Mine (ε, null, η), generates the first occurrence list *FOL*, i.e. the first occurrence position of event ***a*** in each sequence of D using algorithm *Construct-FOL* (L, ***a***).

If an event ***a*** does not occur in a sequence there will not be any node for that sequence. This saves space and the *First Occurrence List* is shown in Figure 4.9.

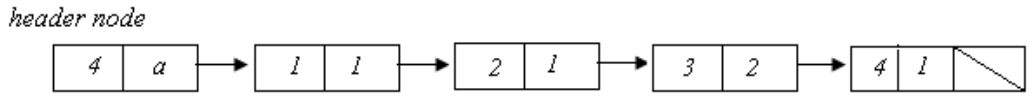

Figure 4.9: First Occurrence List *FOL(L, **a**)*

Support of ***a*** is available in header node. Since the support of ***a*** > η, ε .***a*** = ***a*** is added to *F* according to step ii in algorithm *FOL-Mine*. Thus the current set of pattern F is {***a***}. Next, *FOL-Mine(**a**, L, η)* is called. *FOL-Mine()* then calls *Construct FOL (L, **aa**)*. It generates the first occurrence list of frequent events in the *a*-projection database. Thus the second call builds *FOL* as in Figure 4.10.

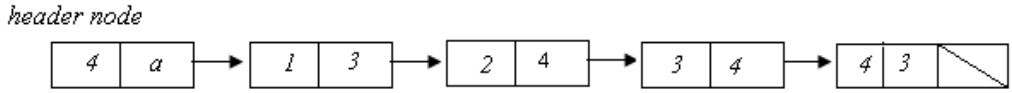

Figure 4.10: First Occurrence List FOL (L, ***aa***)

Since the support of *a* is 4 which is greater than η , ***a.a*** = ***aa*** is added to *F* according to step 1.ii of algorithm FOl-Mine. So, F= {***a, aa***}. Next, FOL-Mine is called with new pattern ***aa***, new FOL list and η. *Construct* FOL (*L*, ***aaa***) is now called which generates the first occurrence list of frequent events in the ***aa***-projection database. This call builds *FOL* as in Figure 4.11.

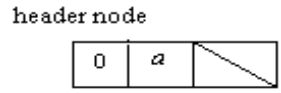

Figure 4.11: First Occurrence List FOL (L, ***aaa***)

Now, according to the header node event ***a*** does not meet the required minimum support and it returns to previous recursive call after freeing the memory by deleting the current FOL. So, according to the loop in step1 of algorithm *FOL-Mine (pattern q, FOL L, η)*, it proceeds with the next event ***b*** in ∑ using FOL in Figure 4.10 and builds FOL as in Figure 4.12.

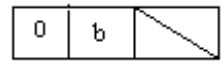

Figure 4.12: First Occurrence List FOL (L, ***aab***)

As ***b*** also does not meet the support and the call returns and initiate a new call with next event ***c*** in ∑. New *FOL* generated (Figure 4.13) shows ***c*** has enough support and thus pattern ***aa.c***= ***aac*** is added to *F*. That is, F= {***a, aa, aac***}.

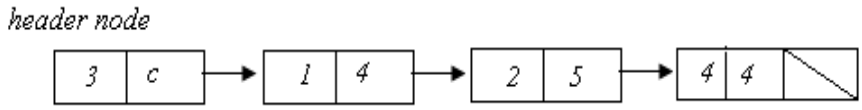

Figure 4.13: First Occurrence List FOL (L, ***aac***)

Now the set of frequent pattern is over and thus that call terminates and returns to complete the call using FOL in Figure 4.9.

The recursive calling of FOL-Mine terminates when no more patterns are to be generated.

4.4 Performance Evaluation

Frequent sequential pattern mining algorithms fall into two categories; apriori based and pattern growth based. Many studies reveal that the pattern growth methods outperform apriori ones. As *FOL-Mine* is a pattern growth method, comparison with other pattern growth based algorithms is enough to prove its efficiency.

WAP-Mine and other WAP-Tree based methods are the prominent proposals in pattern growth approach. In [Lu and Ezeife, 2003] authors proved *PLWAP-Mine* performs better than the *WAP-Mine* method in [Pei et al., 2000]. The authors of *FLWAP-Mine* [Tang, Turkia and Gallivan, 2007] claim the better performance of *FLWAP-Mine* over *PLWAP-Mine* in their performance evaluation. Experimental results in [Pearson and Tang, 2008], claim that the execution time and memory requirement is less for FOF-Mine than FLWAP [Pearson and Tang, 2008]. Thus, it is clear that FOF-Mine performs better than all other algorithms.

CSB-Mine is a proposal that adopts concepts of pattern growth but avoids the use of complicated tree structures [Zhou, Hui and Fong, 2006]. Authors of CSB-Mine claimed that the algorithm performs better than WAP-Mine at lower supports. But In *CSB-Mine* intermediate sub conditional sequence bases are generated in each recursive call. This takes up a significant amount of time and space. But, in *FOL-Mine* once the sequences are represented in a liked format, rest of the mining is done using it. No intermediate reconstruction is required.

CSB-Mine includes two more procedure to carry out the mining. During each call it has to build an event queue. Algorithm Single Sequence Test is used to verify whether all sequences in sub conditional sequence bases can be combined into a single sequence. If so, the mining of that sub conditional sequence bases will be terminated. This single sequence will be used to form a part of the final sequential access patterns. Otherwise, sub conditional sequence base is constructed and mining is done recursively. This also increases the execution time of *CSB-Mine* significantly. From the discussion it is evident that *FOL-Mine* will perform better than *CSB-Mine.*

Now, to prove *FOL-Mine* to be a better performer, it is enough to compare it with FOF-Mine.

Time and Memory are the two major concerns in sequential pattern mining. So, these two factors are considered for performance evaluation of the proposed algorithm.

4.4.1 Test Environment and Datasets

All tests were performed on a 1.79Ghz AMD Sempron(tm) machine with 512 MB RAM and running Microsoft Windows XP Professional version 2002. Both *FOF-Mine* and *FOL-Mine* are implemented in Microsoft visual C++ 6.0.

Data sets used for the experiments are T25I10D10K and T10I4D100k, T40I10D100Kand msnbc. The first three synthetic datasets are generated using the publicly available synthetic data generation program of the IBM Quest data mining project at http://www.almaden.ibm.com/cs/quest/, which has been used in most sequential pattern mining studies [Lu and Ezeife, 2003; Pearson, and Tang, 2008; Tang, Turkia and Gallivan, 2007; Zhou, Hui, and Fong, 2004; Zhou, Hui, and Fong, 2006]. T25I10D10K is a 948 KB data base with 10000 sequences and T10I4D100k is of 3.83 MB with 1 lakh sequences. The data base T40I10D100K is 15116KB in size containing 1lakh sequences.

The data set *msnbc.com* comes from Internet Information Server (IIS) logs for msnbc.com and news-related portions of msn.com for the entire day of September, 28, 1999 (Pacific Standard Time). It is available in UCI Machine learning repository [http://archive.ics.uci.edu/ml]. It contains 10 lakh sequences and of size 11.9 MB.

4.4.2 Testing of Accuracy

After implementing the proposed method *FOL-Mine* and *FOF-Mine* algorithms, first verification performed was to prove the accuracy of the proposed method. For this, the proposed *FOL-Mine* was first run on the sample database in Table 4.1. This generated the very same set of patterns as the authors of [Pei et al., 2000] claimed in their paper. Then the algorithm was run on a small database of 1000 access sequences, and the resulted set of patterns are verified manually. Set of patterns generated by both *FOL-Mine* and *FOF-Mine* were found to be the same. Moreover, the number of patterns generated by both algorithms on all test data sets is same. This proves the accuracy of the proposed method.

4.4.3 Execution Time Comparison

The execution time of the two algorithms was thoroughly tested. Codes are added in the original implementation to measure the total execution time. Experiments were repeated by varying the support threshold. In all data sets *FOL-Mine* algorithm is shown to outperform *FOF-Mine* especially when the support value becomes smaller.

Comparison of Execution time is conducted using the four datasets. Figure.4.14, Figure.4.15, Figure.4.16 and Figure 4.17 show the comparison of execution time of *FOL-Mine* and FOF-Mine on databases T25I10D10K, T10I4D100K, T40I10D100K and msnbc respectively. Results of the comparison are also given in Table 4.2, Table 4.3, Table 4.4 and Table 4.5.

Table 4.2: Execution Time Trend and Number of Patterns at Various Minimum Supports for T25I10D10K

support	Execution time (sec)		Number of Patterns
	FOL-Mine	*FOF-Mine*	
0.35	41	25	255
0.03	51	36	309
0.02	66	80	536
0.015	87	112	882
0.01	146	214	3300
0.0095	184	272	6893
0.009	225	321	11164
0.0085	270	355	13370

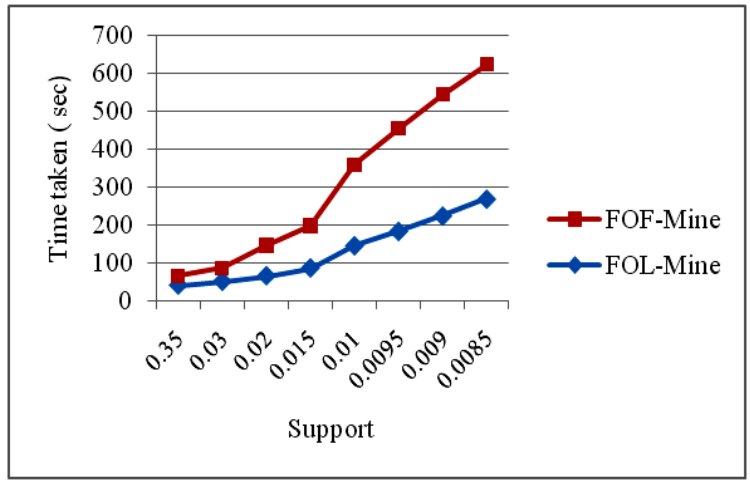

Figure 4.14: Execution Time at Various Minimum Supports for T25I10D10K

Table 4.3: Execution Time Trend and Number of Patterns at Various Minimum Supports for T10I4D100K

support	Execution time (sec)		Number of Patterns
	FOL-Mine	*FOF-Mine*	
0.05	4	4	10
0.03	11	6	60
0.02	40	37	155
0.015	73	79	237
0.01	144	174	385
0.0095	152	186	404
0.009	160	197	421

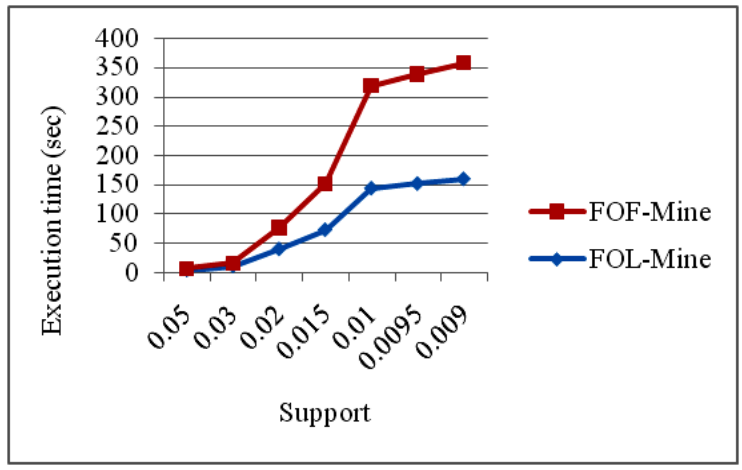

Figure 4.15: Execution Time Trend at Various Minimum Supports for T10I4D100K

Table 4.4: Execution Time Trend and Number of Patterns at Various Minimum Supports for T40I10D100K

support	Execution Time (sec)		Number of Patterns
	FOL-Mine	*FOF-Mine*	
0.095	134	460	92
0.09	224	485	110
0.085	232	517	120
0.08	257	596	137
0.075	322	695	157
0.07	451	831	183
0.065	465	965	208

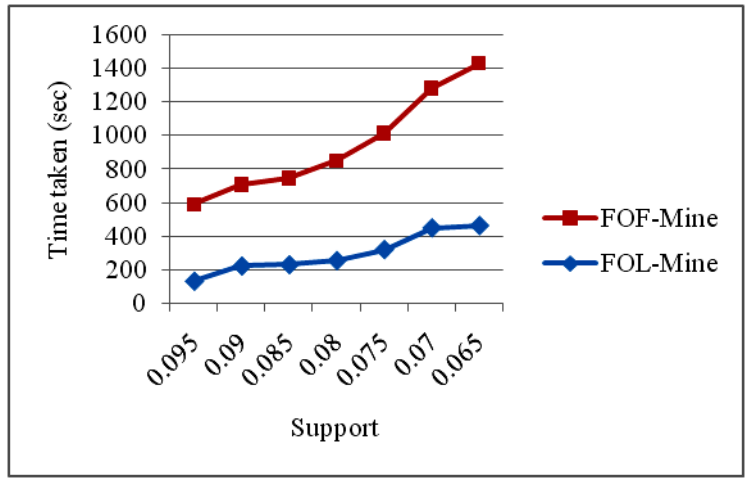

Figure.4.16: Execution Time Trend at Various Minimum Supports for T40I10D100K

Table 4.5: Execution Time trend and Number of patterns at various minimum supports for *msnbc*

support	Execution Time (sec)		Number of Patterns
	FOL-Mine	FOF-Mine	
0.0025	14	19	2009
0.002	22	28	4199
0.001	42	64	16569
0.0009	51	74	20677
0.0008	56	95	32568
0.0007	72	117	45047
0.0006	98	154	52236

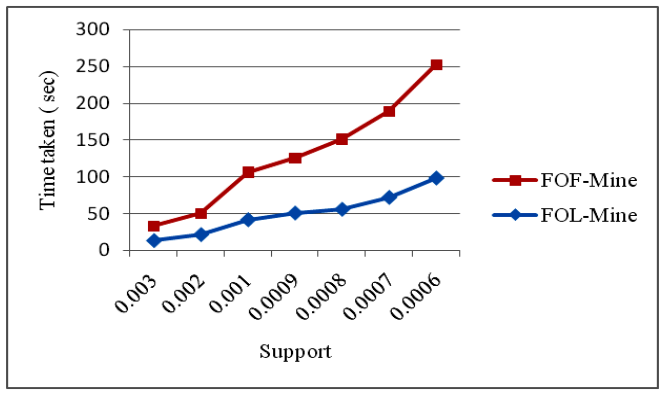

Figure 4.17: Execution Time Trend at Various Minimum Supports for *msnbc*

4.4.4 Comparison of Number of Patterns

Total number of patterns generated during mining increases very rapidly when the support threshold decreases. This criterion also has been brought under study. Relation between the number of patterns generated and the minimum support for various databases are shown in Figure 4.18, Figure 4.19, Figure 4.20 and Figure 4.21. The data is available in tabular format in Table 4.2, Table 4.3, Table 4.4 and Table 4.5.

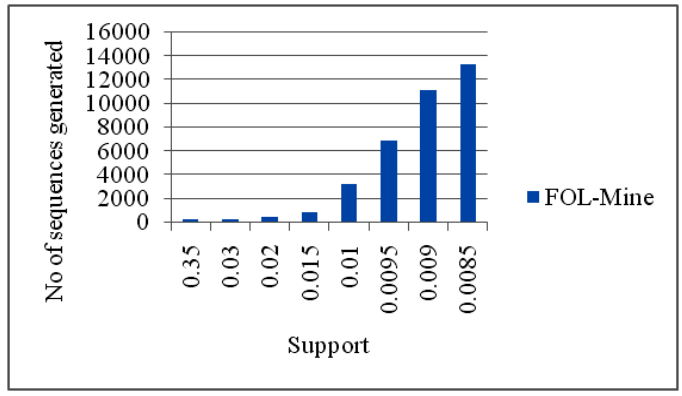

Figure 4.18: Number of Patterns Generated at Various Minimum Supporsts for T25I10D10K

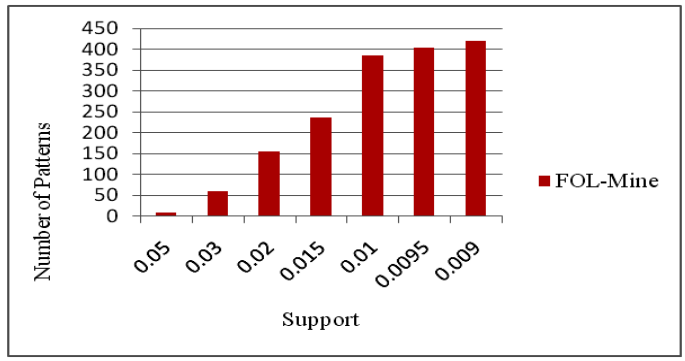

Figure 4.19: Number of Patterns Generated at Various Minimum Supports for T10I4D100K

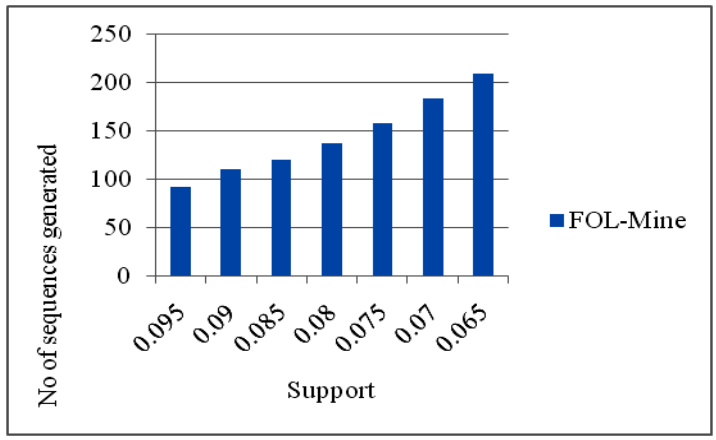

Figure 4.20: Number of Patterns Generated at Various Minimum Supports for T40I10D100K

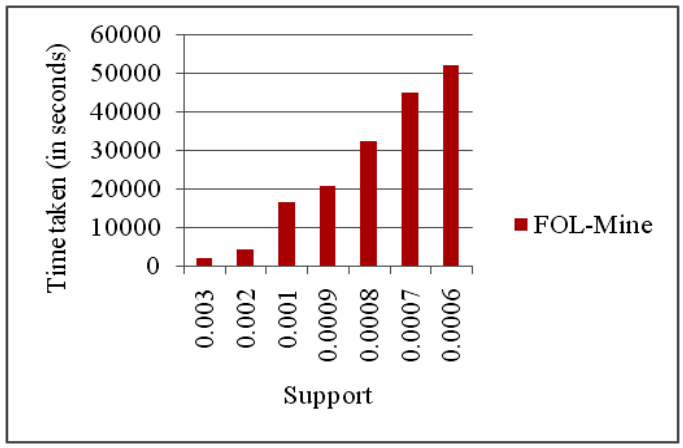

Figure 4.21: Number of Patterns Generated at Various Minimum Supports for *msnbc*

4.4.5 Scale-up Experiments

Scale up experiments evaluates how the performance of the algorithm is related to the size of the database. Experiment is done using the Dataset T10I4D100K. Data size is changed from 20k to 100k. Minimum support is maintained at 0.005. Figure 4.22 and Table 4.6 show the result of the scale-up experiments. From the result it is very clear that *FOL-Mine* has good scalability. Furthermore, *FOL-Mine* shows a better scalability than FOF-Mine.

The execution speed gain of the proposed *FOL-Mine* algorithm is also brought under study and the result is shown in Figure 4.23 and Table 4.6.

Table 4.6 Execution Time for Various Data Sizes at Minimum Support 0.005 for T10I4D100K

Size	*FOL-mine*	*FOF-mine*	Speed Difference
20k	38	50	12
40k	97	120	23
60k	161	200	39
80k	229	285	56
100k	292	368	76

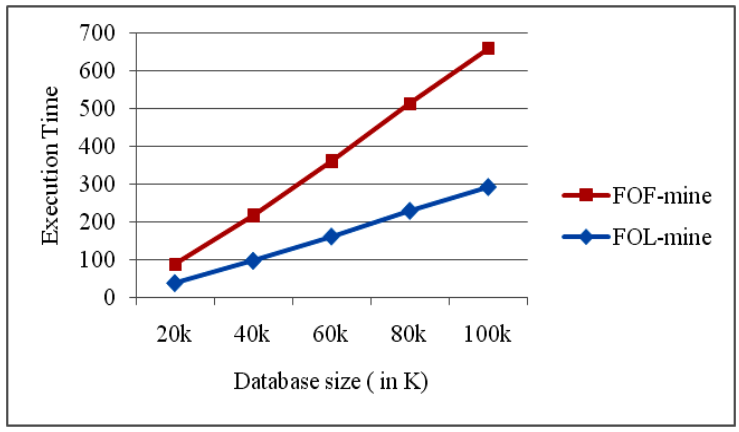

Figure 4.22: Execution Time for Varying Database Size at Minimum Support 0.005 for T10I4D100K

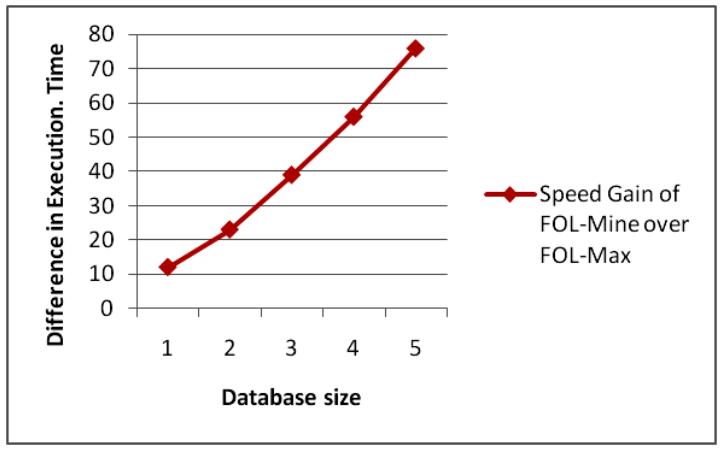

Figure 4.23: Speed Up of *FOL-Mine* over *FOF-Mine* for Varying Database Sizes at Minimum Support 0.005 for T10I4D100K

4.4.6 Memory Considerations

Generally, tree structure is very compact, but the Aggregate tree used in WAP-Tree methods, node structure requires more space. In the linked-database used in FOL-Mine, each node holds only two different fields, item and a pointer as shown in Figure 4.24(i). But each node of aggregate tree in FOF-Mine [Pearson and Tang, 2008] store four different information: item, count, pointer to right-sibling and a pointer to left- child as shown in Figure 4.24(ii).

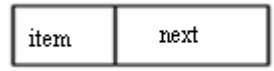

Figure 4.24(i): Node in FOL-Mine

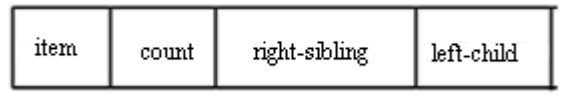

Figure 4.24(ii): Node of FOF-Tree in FOF-Mine

The storage requirement of the tree structure is double that of FOL-Mine. This means that the Pattern-tree structure has got any advantage in terms of storage space only when the number of node in the Pattern-tree is less than half of the number of nodes in the linked-database of FOL-Mine. This happens only when the access sequences in WASD are almost similar.

4.5 Summary

A new pattern growth, web access pattern mining algorithm FOL-Mine, is proposed in this chapter.

The compact WAP-Tree structure and the corresponding WAP-Mine algorithm introduced by Pei et al. performed better than all other earlier apriori methods. Many modifications of the WAP-Tree and WAP-Mine were introduced by various authors to improve its efficiency.

But, there are some shortfalls for WAP-Tree based methods. All WAP-Tree based methods involve two data base scan, first to find out the frequent items and second to create the pattern tree using frequent sub-sequences. This increases the execution time significantly.

Generally, tree structure is very compact and efficient. But the Aggregate tree used in WAP-Tree methods has got some drawbacks, though they efficiently hold the access sequence and facilitate support counting. WAP-Tree based methods either use complicated linkages of tree nodes or construct intermediate trees for projection data bases for finding out the first occurrences. Increase in the complexity of node structure increases the size of the node as the information held at each node is more. Thus both complexities of tree node and intermediate reconstruction of trees lead to more memory requirement.

The proposed method FOL-Mine needs only one data base scan. In the single scan it loads all access sequences in the linked structure and collects all necessary information for support counting. Instead of deleting the infrequent elements from the access sequences as in the earlier WAP-Tree based methods, the proposed method skips them efficiently during pattern mining. This prompts the use of the proposed linked database structure for incremental mining too.

The proposed FOL-Mine avoids the use the complicated node structures. It uses much simple FOL structure to facilitate efficient mining. Correctness of the method is verified using test database. The efficiency of the method is evaluated by comparing the proposed method with earlier method both in terms of speed and memory. Experiments are done on both standard synthetic data sets and real time data sets. The new method outperforms the earlier method significantly. The scale-up experiments exhibits a linear increase in the execution time as the size of data base increases.