FP-Split SPADE-An Algorithm for Finding Sequential Patterns

Pragya Goel
M.Tech Student
Department of Computer Science & Applications
Kurukshetra University, Haryana,India

Rajender Nath
Professor
Department of Computer Science & Applications
Kurukshetra University, Haryana,India

Kartik
Student
Kurukshetra Institute of Technology & Management
Kurukshetra, Haryana, India

ABSTRACT
Sequential Pattern Mining (SPM) is one of the key areas in Web Usage Mining (WUM) with broad applications such as analyzing customer behavior from weblog files. The current algorithms in this area can be classified into two broad areas, namely, apriori-based and pattern-growth based. Apriori based algorithms for mining sequential patterns need to scan the database many times as they focus on candidate generation and test approach. A lot of research has been done so far, but even the best apriori based algorithm for SPM in terms of number of database scans is SPADE that scans the database three times for discovering sequential patterns. Pattern growth based algorithms avoid the candidate generation step and the best pattern growth algorithm known so far is Prefix Span that needs to scan the database at least twice. In this paper, a novel algorithm for SPM is proposed called FP-Split SPADE that reduced the database scan to only one by creating an FP-Split tree and applying SPADE algorithm on the tree instead on sequence database that greatly improved the efficiency of mining sequential patterns.

Keywords
Sequential Pattern Mining, Web Mining, SPADE, Apriori, FP-Split tree

1. INTRODUCTION
Web Mining is one of the main areas of Data Mining and is defined as the application of data mining techniques to either web log files or contents of the web documents or to the web document's hyperlink structure in order to extract from them the unknown knowledge and potentially valuable patterns. Web Mining is of three types- Web Usage Mining, Web Structure Mining and Web Content Mining. This paper deals with Web Usage Mining (WUM) that identifies the hidden patterns of the visitors that accessed the website by manipulating the web log files data and sequential pattern mining (SPM) that discovers frequent subsequences as patterns by analyzing sequence database. This research paper highlights the goal of SPM in web log data i.e. to discover user's frequent sequential patterns while navigating a website. SPM is an active area of research since its introduction by Agrawal and Srikant in [2]. The SPM problem is concerned with inter transaction patterns as opposed to association rule mining that considered only intra transaction patterns. Thus, SPM also takes care of the time order and can be seen as a generalized model of association rule mining due to which more candidates are generated and hence can be seen as an extension of association rule mining. Many algorithms are proposed for mining sequential patterns on the basis of Apriori heuristic that was first put forward by Agrawal and Srikant in association mining [1] according to which all subsequences of a frequent sequence must also be frequent. Though these Apriori based algorithms performed well, but they must scan the database frequently. The scanning cost becomes non trivial as the length of each sequence grows. Zaki proposed SPADE [4] that was based on Apriori heuristic but it needed to scan the database three times. Algorithm Free Span [3] integrated the mining of frequent sequences with frequent patterns and used projected database that confined the search and growth of subsequent fragments. It also needed to scan the database three times. PrefixSpan [5] was an improved version of FreeSpan and its main idea was that instead of projecting sequence databases, only the prefix subsequences were examined and only their corresponding postfix subsequences were projected into the projected databases. It scans the database twice leading to efficient processing. However the above methods still needs to scan the database at least twice. An algorithm using FP tree [7] was proposed for mining frequent items but the FP-tree construction still needed to scan the database twice. FP-split tree algorithm [8] was an improvement of the FP- tree algorithm as it scanned the database just once.

In this paper, a new algorithm called FP-Split SPADE is proposed for mining sequential patterns by using FP-Split tree. Compared to the previous algorithms that needs to scan the database many times, our new algorithm scans the database only once by constructing FP-Split tree from the web log file and then SPADE algorithm is applied on the FP-Split tree for discovering all frequent sequences.

The rest of the paper is organized as follows: Section 2 reviews the concept of SPADE and FP-Split tree as the basis of the proposed algorithm. In Section 3, the new algorithm is presented, section 4 presents the experimental evaluation of the proposed algorithm and comparison of the new algorithm with the previous one and finally section 5 concludes the paper.

2. RELATED WORK
The problem of mining sequential patterns was first introduced in [2] in which three algorithms were presented viz. AprioriAll, AprioriSome, DynamcSome. All the three algorithms were based on Apriori approach [1] according to which all subsequences of a frequent sequence must also be frequent. Apriori based algorithms are further classified into two types based on database format-horizontal database format and vertical database format. Using vertical database format provided the benefit of generating patterns and calculating their support count without performing costly database scans. A number of Apriori based algorithms were

based on vertical database format from which SPADE algorithm was most efficient in terms of number of database scans since it required only three scans of the database to generate all frequent sequences.

SPADE (Sequential Pattern Discovery using Equivalence classes) algorithm was proposed in [4] for mining sequential patterns based on vertical database format. SPADE decomposed the original problem into smaller sub problems and used efficient lattice search techniques to solve them independently in main memory. Only three database scans were required to discover all the sequences. The major performance improvement was due to the use of ID lists for each candidate due to which the support count was calculated from its ID list, thus reducing the cost of scanning. The authors decoupled the problem decomposition from pattern search that reduced both computational and input output costs. Experiments showed that SPADE was twice as fast as GSP, the reason being the use of more efficient support counting method based on id list structure.

An algorithm called GO-SPADE was proposed in [6] that extended SPADE to incorporate generalized occurrences. The motivation behind GO-SPADE was that many sequential databases could contain repetition of items that caused performance degradation in the traditional SPADE approach. The authors introduced the concept of generalized occurrences which were compact representations of several occurrences of a pattern and described corresponding primitive operators to manipulate them. Using such a representation reduced the size of ID lists significantly if large number of consecutive occurrences appeared in the database. The authors claimed that this approach not only reduced the memory space used during the process of extraction but also significantly reduced the join cost and therefore, the overall execution time.

A novel algorithm bitSPADE was presented in [9] that combined the best features of SPADE, one of the most memory efficient algorithm and SPAM, the fastest algorithm. The authors used the concept of semi vertical database using bitmap representation of SPAM and combined this semi vertical database with SPADE's lattice decomposition into independent equivalence classes that allowed fast and efficient enumeration of frequent sequences. A new pruning strategy was also presented that could be applied independently to each equivalence class.

pSPADE algorithm was proposed in [10] for mining sequential patterns using personalized support threshold value. As each user behavior was unique; using one minimum support value for all users might affect the pattern generation. This research introduced a personalized minimum support threshold for each web users using their Median item support value for reducing this problem. For selecting the most appropriate minimum support value for each user, the process of generating personalized minimum support was done by manipulating the mathematical median formula. The support for each item or pages was counted, sorted and then the median or middle item value was selected based on the formula presented. After generating the personalized minimum support, the technique was then implemented in the SPADE. pSPADE performance was highest on the discovery of user's origin.

For enhancing the efficiency of frequent itemsets generation, the FP-tree and FP-growth algorithms were proposed [7]. FP-tree was an extended prefix tree structure for storing compressed and crucial information about frequent patterns. FP-growth algorithm was based on the FP-tree that exploited the set of frequent itemsets without candidate itemsets generation and the FP-tree construction algorithm scanned the database only twice. Thus efficiency of mining was achieved by compressing a large database into highly condensed smaller data structure that avoids costly and repeated database scans. A divide and conquer based partitioning method was also used that decomposed the mining task into a set of smaller tasks, hence dramatically reducing the search space.

The FP-growth algorithm which was based on the FP-tree to generate frequent itemsets was time-efficient. But the authors ignored the fact that FP-tree construction could be time consuming. Thus, to improve the process of FP-tree construction, a fast algorithm called FP-split was proposed [8]. The algorithm scanned the database only once for generating equivalence classes of frequent items, then the equivalence classes were sorted in descending order for constructing the FP-split tree. This algorithm was experimentally found efficient and scalable than the previous algorithms.

3. FP-SPLIT SPADE PROPOSED ALGORITHM

As discussed in the related work, some problems were found in the existing SPADE algorithm in terms of time and efficiency such as the number of database scans required. The work done so far to improve the existing SPADE algorithm has not tried to reduce the number of database scans which is an important thread to work on. All the existing algorithms related to SPADE [4][6][9][10] requires three database scans to find all the frequent sequences To address this problem, a novel and efficient algorithm called FP-Split SPADE is proposed for mining frequent sequences that reduces the database scan to only one by creating an FP-Split tree. The proposed algorithm is as shown in Fig 1. The algorithm takes as input minimum support threshold minsup as specified by the user and dataset D.

Algorithm: FP-Split SPADE (minsup, D)

//minsup=Minimum support threshold
// D= Dataset

Begin

1. Collect the dataset
2. Preprocess the dataset till the entries in it are exhausted and store it in database.
   2.1 Perform data cleaning
   //by removing the records with keywords (.jpg, .png, .gif, .bmp, .doc, .docx, .pdf, .xls, .txt, .css, .ico, spider, crawler) in URLs
   2.2 Perform data filtering
   // keep only unique page URLs
3. Construct FP-Split tree by performing a single database scan
   3.1 Create equivalence class of unique pages computed in step 2.2
   3.2 Create frequent-1 pages removing those that did not satisfy the minsup
   3.3 Construct FP-Split tree according to the rules
4. Apply SPADE algorithm on FP-Split tree with no more database scans required.
   4.1 Create frequent-1 and frequent-2 pages
   4.2 Create prefix based equivalence class for frequent-2 pages
   4.3 Generalize the generation of rest of the frequent sequences.

End

Fig 1: FP-Split SPADE algorithm

The detailed steps of the proposed algorithm are explained below with respect to web log file collected from http://www.uietkuk.org.

Step1: Preprocessing
Algorithm: Preprocessing of the web log file

Input: Raw web log file L

file_extension= file extension of requested uristem field

status_code= HTTP status field

method_name= HTTP access method field

```
for every entry in L do
    if status_code €/ [200,299] or method_name €/
    [GET or POST] or file_extension €
    [pdf,doc,docx,jpg,gif,png,bmp,css,txt,ico,xls
     spider, crawler] then
        delete this entry from L
    end
 transfer the remaining entries from L to database
end
```

Output: Cleaned web log file in database DB

Explanation: First of all, web log files are collected. Log files contain noisy and ambiguous data which may affect the result of overall mining process. So, to improve the quality of weblog data, preprocessing needs to be done by removing the entries that are not necessary for the mining process. For the purpose of this research, image files (.gif, .jpeg, png, .bmp etc), document files (.doc, .pdf, .txt, .xlsetc), failed requests etc are removed from the web log files and the cleaned weblog file is stored in a database.

Step 2: Create FP-Split tree

Algorithm: Creation of FP-Split tree from cleaned web log file

Input: Cleaned web log file in database DB

EC_P- Equivalence class of page p

P_{id} -Page id

S_{id} -Session id in which page p is accessed

E_{id} -Event id to denote the sequence

Sup_p-Support of page p

minsup- Minimum support threshold

header_list- header table

F1- Frequent 1 pages

(i) Create EC_p with P_{id} on one side along with S_{id} and E_{id} on the other side.

i.e. $EC_{P=}\{P_{id}:\{S_{id}:E_{id}\}\}$

(ii) for each page p do
$Sup_{p=}|EC_P|$
if Sup_p < minsup
delete the page from DB
end

F1{p:sup_p<minsup}

(iii) Create header_list with F1 on one side and its occurrence in FP-Split tree on the other side. Save each F1 in one node or split into number of nodes according to following rules:
Let n- new node to be added
p- specific node in the tree
root- dummy node

- if (p is root && p.link_child==null) then
  p.link_child<-n
  else compare(p.link_child.list,n.list)
  endif
- if(n.list $\subseteq$ p.list && p.link_child==null) then
  p.link_child<-n
  else compare(p.link_child.list,n.list)
  endif
- if(n.list$\cap$p.list==Ø && p.link_sibling==null) then
  p.link_child<-n
  else compare(p.link_child.list,n.list)
  endif

- if(p.list∩n.list≠Ø && p.list-n.list≠Ø) then
  call split(n) and return two nodes n1 and n2.

Output: FP-Split tree

Explanation: Once the database is ready, an FP-split tree [8] is created through the following steps with the slight modification of including event identifier EID in the tree to capture the sequence:

(i) Create equivalence class of pages
Scan the database once to create equivalence class EC of pages. Equivalence class of a page denotes the list of sessions in which that page has been accessed along with the event id to denote the sequence of pages.

(ii) Create frequent 1 pages
Calculate support of each page. The support of page p refers to the number of records contained in equivalence class of p i.e. EC_p. After calculating support for all pages, delete those pages whose supports are below the predefined minimum support threshold.

(iii) Construct FP-Split tree
After generating frequent pages, the equivalence class of pages is then converted to nodes for the construction of FP-split tree. A header table is also built so that each page can point to its first occurrence in FP-split tree. There are two entries for each page in the header table: one to store frequent page and other to link to the occurrence of the associated page in the FP-split tree. Each frequent page can be saved in one node or split into a number of nodes.

Step 3: Apply SPADE on FP-Split tree
Algorithm: SPADE

Input: DB, FP-Split tree

F1-Frequent 1 pages

F2-Frequent 2 pages

S_{id} -Session id in which page p is accessed

P_{id} -Page id

E_{id} -Event id to denote the sequence

Prefixequi- Prefix based equivalence class

(i) F1 is taken from Step 2.

(ii) F2 is computed by performing vertical to horizontal transformation of the DB with S_{id} on one side and (P_{id}, E_{id}) pair on the other side i.e. {S_{id} :{ P_{id},:E_{id} }} and then pairing of pages is done for each S_{id} removing pairs with sup< minsup.

(iii) Create prefixequi for F2. Two sequences are in same prefixequi if they share a common k length prefix.

(iv) for each prefixequi

generalize the generation of frequent

sequences.

Output: Frequent sequences

Explanation: After FP-Split tree is created, SPADE algorithm [4] is applied on it to generate frequent sequences. SPADE algorithm, works as follows:

(i) First of all, frequent 1 sequences need to be computed that can be done without any database scan since we have already computed it while constructing FP-Split tree.

(ii) After computing frequent 1 sequence, frequent 2 sequences are computed by performing vertical to horizontal transformation of the database with session ID on one side and (page, EID) pair on the other side and then pairing of the pages is done for each session ID. After that, those pairs are removed that did not satisfy the support count thus creating a list of frequent 2 sequences.

(iii) After that prefix based equivalence class is created by decomposing the original search space into smaller pieces that can be processed independently in main memory. The two sequences are in the same equivalence class if they share a common k-length prefix.

(iv) After creating prefix based equivalence classes, we generalize the generation of frequent sequences.

(v) Finally frequent k length sequence that is accessed the most is obtained.

4. EXPERIMENTAL EVALUATION OF THE PROPOSED ALGORITHM

In order to evaluate the proposed algorithm, the, existing SPADE algorithm and proposed FP-Split SPADE algorithm were implemented using Java programming language and Netbeans IDE. All experiments were performed on a PC with Intel Core i3 CPU @ 1.70 GHz, 4GB RAM and 32 bit Windows 7 Home Premium operating system. The experiments were performed on dataset collected from a reputed college's website http://www.uietkuk.org.

The proposed algorithm was evaluated on the following parameters- size of dataset and support count.

4.1 Evaluation of Algorithm by varying size of dataset

Datasets having varying dataset size (100,150,200,250,300,350,400) were generated for the purpose of time comparison of the two algorithms and support count was fixed at 5.

Table 1: Variation of time taken (in sec) with the dataset size

NUMBER OF RECORDS	TIME(in sec) FP-SPLIT SPADE	TIME(in sec) SPADE
100	4.052	49.671
150	10.537	59.241
200	28.488	70.440
250	46.122	80.243
300	67.337	92.564
350	110.221	120.334
400	175.201	210.290

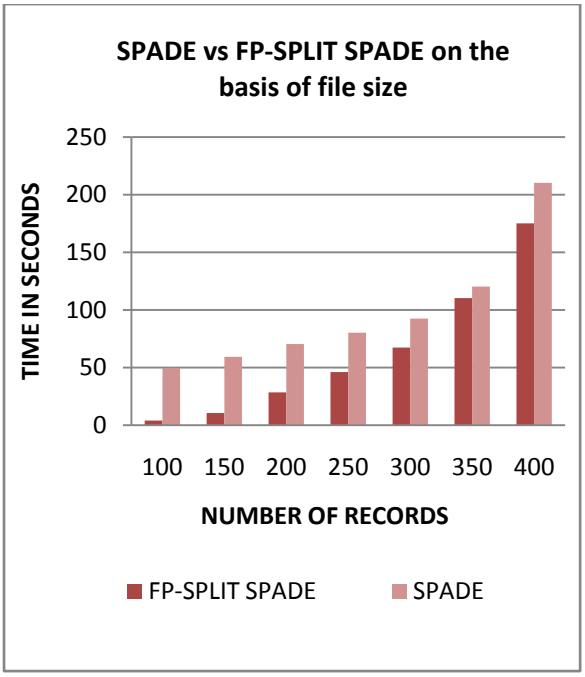

Fig 2: Performance comparison on the basis of dataset size

Table 1 shows how the time taken by the two algorithms varies with the dataset size.

The graph in Fig 2 shows that as the dataset size increases, the time taken by the algorithms also increases. But the proposed algorithm FP-Split SPADE always takes lesser time than the traditional SPADE algorithm.

4.2 Evaluation of Algorithm by varying support count

For the purpose of time comparison on the basis of support count specified by the user, the two algorithms were run for different support counts (2,3,4,5,6,7,8,9,10) and the number of records in the database was fixed at 80.

Table 2: Variation of time taken (in sec) with the support count

SUPPORT COUNT	TIME (in sec) SPADE	TIME (in sec) FP-SPLIT SPADE
2	35.256	22.245
3	35.241	13.306
4	35.239	10.905
5	35.230	10.140
6	35.220	8.845
7	35.210	8.127
8	35.208	7.815
9	35.200	6.521
10	35.190	3.401

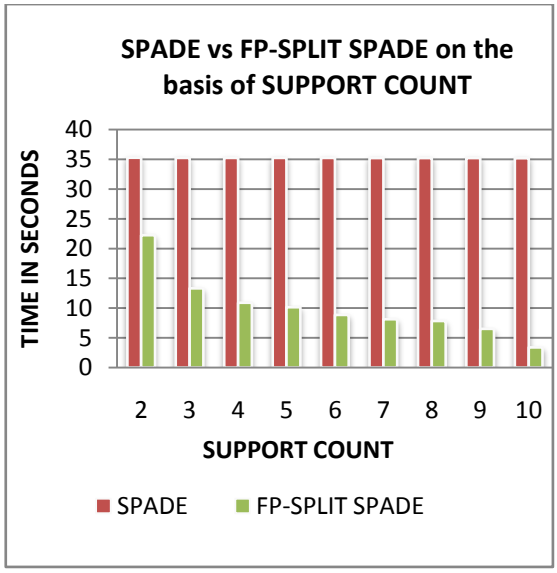

Fig 3: Performance comparison on the basis of support count

Table 2 shows how the time taken by the two algorithms varies with the support count specified by the user. It also shows that as the support count increases from 2 to 10, time taken by the algorithms decreases i.e. there is an inverse relationship between support count and time taken. When support count is high, time taken is low and vice versa. However the time decreases much more slowly in case of traditional SPADE algorithm as compared to the proposed FP-Split SPADE algorithm.

5. CONCLUSION

This paper has proposed a novel algorithm called FP-Split SPADE for Sequential Pattern Mining based on SPADE and FP-Split tree algorithms for reducing the number of database scans. The algorithm has been evaluated on two parameters: varying dataset size and varying support count. It has been observed that as the dataset size increases, the time taken by the algorithm also increases and when the support count is high, time taken is low and vice versa. But, the time taken by existing SPADE algorithm is always more than the proposed algorithm. Thus, the experimental evaluation has shown that the proposed algorithm has reduced the time taken for discovering the sequential patterns and FP-Split SPADE algorithm outperformed the existing SPADE algorithm in both the cases.

6. REFERENCES

[1] Agrawal, R. and Srikant, R. 1994. Fast algorithms for mining association rules, In Proceedings of the 20thInternational Conference on Very Large Databases, VLDB, ACM, 487-499.

[2] Agrawal, R., and Srikant, R. 1995. Mining sequentialpatterns, In Proceedings of the 11th IEEE InternationalConference on Data Engineering, 3-14.

[3] Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., and Hsu, M.-C. 2000. FreeSpan: frequent pattern-projected sequential pattern mining, In Proceedings of the 6th ACM SIGKDD International Conference onKnowledge Discovery and Data Mining, 355-359.

[4] Zaki, M. J. SPADE. 2001. An Efficient algorithm for mining frequent sequences, Machine Learning, 31-60.

[5] Pei, J., Han, J., Moratzavi-Asl, B., Pinto, H., Chen, Q., Dayal and U. PrefixSpan. 2001. Mining Sequential Patterns Efficiently by Prefix- Projected Pattern growth, In Proceedings of the 17th IEEE International on Data Engineering, 215-224.

[6] Leleu, M., Rigotti, C., Boulicaut, J.-F., and Euvrard, G. 2003. GO-SPADE: Mining Sequential Patterns overDatasets with Consecutive Repetitions, In Proceedingsof the 3rd International Conference on MachineLearning and Data Mining in Pattern Recognition, Springer, 293-306.

[7] Han, J., Pei, J., Yin, Y. and Mao R. 2004. MiningFrequent Patterns without candidate generation: AFrequent pattern tree approach, Data Mining and Knowledge Discovery, Springer, 53-87.

[8] Lee, C.F., Shen and T-H. 2005. An FP-Split Method for Fast Association Rules Mining, In Proceedings of the3rd IEEE International Conference on InformationTechnology: Research and Education, 459-463.

[9] Aseervatham, S., Osmani, A., and Viennet, E. 2006. bitSPADE:A Lattice-based Sequential Pattern Mining Algorithm Using Bitmap Representation, InProceedings of the 6th IEEE International Conferenceon Data Mining, 792-797.

[10] Alias, S., and Norwawi, N. M. 2008. pSPADE:Miningsequential pattern using personalized support thresholdvalue. International Symposium on Information Technology, IEEE, 1-8.