International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:5, No:9, 2011

Revised PLWAP Tree with Non-frequent Items for Mining Sequential Pattern

R. Vishnu Priya, A. Vadivel

Abstract**—**Sequential pattern mining is a challenging task in data mining area with large applications. One among those applications is mining patterns from weblog. Recent times, weblog is highly dynamic and some of them may become absolute over time. In addition, users may frequently change the threshold value during the data mining process until acquiring required output or mining interesting rules. Some of the recently proposed algorithms for mining weblog, build the tree with two scans and always consume large time and space. In this paper, we build Revised PLWAP with Non-frequent Items (RePLNI-tree) with single scan for all items. While mining sequential patterns, the links related to the non-frequent items are not considered. Hence, it is not required to delete or maintain the information of nodes while revising the tree for mining updated transactions. The algorithm supports both incremental and interactive mining. It is not required to re-compute the patterns each time, while weblog is updated or minimum support changed. The performance of the proposed tree is better, even the size of incremental database is more than 50% of existing one. For evaluation purpose, we have used the benchmark weblog dataset and found that the performance of proposed tree is encouraging compared to some of the recently proposed approaches.

Keywords**—**Sequential pattern mining; Weblog; Frequent and Non-frequent items; Incremental and Interactive mining

I. Introduction

THE frequent patterns present in database of any categories can be used for extracting interesting patterns for knowledge discovery. Among various well-known data mining techniques, the association rule mining is considered as one of the popular problems and the primary step is to identify the patterns. In sequential database, the items in sequences exist in certain order and items may reoccur many times in same sequence. Mining this characteristic of sequence for finding any specific orders of events satisfying minimum support exist. Sequential data is used in many applications such as customer shopping sequences, web click stream, biological sequence, business organization, super market, financial data, medical data, scientific data, analysis of DNA sequence, system performance and telecommunication network. Our proposed work is based on mining of sequential patterns from weblog data. Sequences in weblog database consist of pair of attributes namely user id and access information. The events in access information are single and in case of general sequential database, it contains set of events, e.g. for web access sequential is {100, <a, b, c>}. The knowledge discovered from web access log are relationship among different user access, direct marketing in e-commerce, information for server performance and useful for restructuring a website infrastructure.

R. Vishnu Priya is with the Department of Computer Applications, National Institute of Technology, Tamilnadu, India (e-mail: vissaru@yahoo.co.in).
A. Vadivel is with the Department of Computer Applications, National Institute of Technology, Tamilnadu, India (e-mail: vadi@nitt.edu).

Further, the data related to web are highly dynamic and new sequences are inserted/ deleted/ updated at any time in the weblog. In addition, the users may frequently change threshold value to acquire required output, which result in reconstruction of prefix tree many times.

In this paper, we propose a RePLNI-tree structure, which scan the web access log only once for mining the sequential patterns and supports both incremental and interactive mining. The advantage of this approach is that it reduces the processing time, provides effective mining from highly dynamic weblog and users may frequently change threshold.

We organize the rest of the paper as follows. We discuss the related work in the next section. In Section III, the procedure for construction of the proposed tree is presented. We summarize the pattern mining with additional functionalities of the tree in Section IV. We present the experimental result in Section V and conclude the paper in the last section of the paper.

II. Related Work

The two famous approaches, which mines sequential pattern from the database are candidate generation-and-test approach and non candidate generation-and-test approach. The algorithms related to the candidate generation-and–test approach is Apriori based algorithms called AprioriAll proposed by Agarwal et al [1], which is the first technique that mine sequential patterns from the database. The sequential patterns are mined using Level-wise search. The sequential pattern of length *'k'* candidate set are generated from the previously generated *(k+1)* patterns. The supports of each candidate are collected by scanning the database once. While the support of candidate pattern is less than a user-specified threshold, they are pruned and the remaining sets are used to find the next level of patterns. This process is terminated while there are no candidate sets with length *(k+1)*, which can be generated from the sequential patterns of *'k'*. Later, many extensive studies on the improvements of Apriori algorithm has been proposed include GSP [13], PSP [6], Gsequence [12], SPADE [16] and graph traversal algorithm [7]. However, these algorithms require multiple scan of large database and large memory space to handle candidate sets. This can be avoided by compressing the database effectively. The non candidate-and-test approach overcome those drawback using the data structure like graphs, tree and database projection to mine sequential patterns. Some of the algorithms are suffix tree [15], Prefix Span [10] etc., Algorithm mine patterns from weblog database include WAP-tree [11] and PLWAP-tree [3]. WAP-tree compress the weblog data into highly compact prefix tree structure for mining sequential patterns and the tree is built with two scans. During the first scan, set of all frequent events are collected. In the second scan of the weblog, a WAP-tree is built only for the frequent events of each

transaction. Finally, intermediate WAP-tree is constructed for mining sequential patterns. This process of intermediate tree construction is continued till the WAP-tree has only one branch or is empty. Based on the above discussions, it is observed that the number of intermediate tree construction increases the mining time considerably and it is required to reduce the same for mining sequential patterns efficiently.

PLWAP-tree mine sequential patterns without constructing intermediate trees and this result in lesser mining time. The PLWAP-tree construction is similar to WAP-tree and the only additional change is the assignment of position codes to each node of the prefix tree for quickly defining the ancestor and descendant of any node. This approach use the prefix sequential search technique for mining, which work better compared to WAP-tree. However, the PLWAP-tree does not support the additional functionality such as incremental mining. Since, the information related to the weblog data are changing overtime, this approach may not be suitable for mining sequential patterns.

In 1997, Wang proposed a suffix tree approach, which construct a suffix tree with '*n*' branches for a sequential with '*n*' events (i.e.) events in S is mapped to a tree in breadth manner, the mining of sequential patterns are performed in post-order fashion. Due to the fact of constructing '*m*' suffix trees for the '*m*' sequential records, this algorithm consumes large memory space. Parthasarathy et al. proposed ISM algorithm [9], which is related to Apriori like algorithm. This algorithm divides the sequences into two sets such as Frequent Set (FS) and Negative Border (NB). FS consist of all frequent events and NB denotes all the non-frequent events. This approach is performed in two phase and the supports of elements in NB and FS sets are updated in phase one. In phase II, NB and FS beyond is done. While non frequent events become frequent events after updating transactions, this algorithm need to rescan the entire database and may increase the execution time for mining the patterns. Kao et. [17] designed the GSP+ algorithm and it also rescan the entire database while non frequent events become frequent, once transactions are updated. In order to generate sequential patterns, this algorithm requires level wise candidate generation, pruning, in which Prefix Span sequential mining technique is used for mining the updated database. MFS+ algorithm [5] is GSP based incremental sequential algorithm. It estimates Maximal Frequent Sequence Set (MFSS). In each iteration, MFSSs is eliminated in the list by using the same pruning rules used in the GSP+. It confirms either the item is frequent or not without rescanning the unchanged database. In our approach the entire old database is not scanned while the small items become large, during which this algorithm scan the whole database. IncSpan algorithm proposed by Cheung et al. [2], use the concept in Prefix Span sequential mining technique. It stores the patterns, whose support is less than minimum support but greater than tolerance support called Semi Frequent Sequences (SFS). Once transactions are updated (db+), only db+ are scanned to recognize the FS, SFS and six types of changed patterns. If the patterns are frequent in both old and updated database or SFS in old database but frequent in updated database or SF in both old and updated database, then the algorithm doesn't rescan the database, but calculate the support from existing frequent patterns. While the patterns are new in updated database, the algorithm scan the incremental database and mine the pattern using prefix span technique and database projection are performed. If the patterns are non-frequent in old database, however frequent or SF in updated database, the entire database is scanned and the database projection is performed. In last case, the algorithm consumes more time for rescanning the entire database and requires more space for performing intermediate database projections and buffering both frequent and semi-frequent patterns. Ezeife et al. [4] have introduced RePL4UP algorithm, which is similar to the construction of PLWAP-tree but while adding new transactions, existing tree has to be revised for retrieving the patterns. While the transaction are updated, RePL4UP algorithm receive seven inputs such as size of database |DB|, incremental database (U), minimum support (s), old tree ($RePLUP^{DB}$), small code profile ($S_{code}{}^{DB}$) (i.e.) S_{code} consist of non frequent events along with position code, old frequent patterns (FP^{DB}) and old candidate list C_1(candidate 1-items), F_1 (frequent 1-items) and S_1 (small 1-items). Updated database is scanned to obtain the candidate lists like C_1^{db}, F_1^{db}, S_1^{db} and S^{db}. Set union and intersection operations are performed on old and new candidate lists to retrieve the candidate lists for entire database such as C'_1, F'_1, and S'_1. The details in F'_1, S'_1**,** F_1 and S_1 are used to classify the items in C'_1 into six item categories such as frequent-frequent, frequent-small, small-frequent, small-small, new-frequent and new-small. This enables the information about the items to be inserted/ deleted to get the current tree. The items in frequent-small set are deleted from $RePL4UP^{DB}$ tree and the deletion starts from the items in header table to each node label on the tree until its final leaf node is deleted. Every deleted node as its position code is saved in S_{code} profile. The remaining details of the nodes such as counts and position codes in the tree are adjusted. Next, items in small-frequent are inserted into RePL4UP tree using the S_{code} profile and the patterns are mined from the modified branches of the updated $RePL4UP^{DB}$ tree to obtain the revised frequent pattern ($ReFP^{DB}$). The tree is built for the incremental database called $RePL4UP^{db}$ for obtaining the new frequent patterns (FP^{db}). The union FP^{DB}-$ReFP^{DB}$, $ReFP^{DB}$ and FP^{db} will retrieve the sequential patterns. Finally, the tree is ready for the next round by updating $RePL4UP^{DB}$ tree with the frequent sequential transactions from incremental database. The efficiency of the algorithm based on time and space are low due to the insertion/ deletion of items. This algorithm takes higher time and works well only if the size of incremental database 50% lesser than the existing one and performance went down for higher size database.

It is imperative from the above discussion that, the RePL4UP- tree algorithm consumes space for maintaining the details relating to candidate lists and support for all old, updated and revised. It also preserves the particulars of deleted nodes for later use. While revising the tree for updated transactions, the operations such as union/ intersection are performed to generate the new candidate lists. Additionally the candidate list is categorized into six item categories and adjustment of pointers for newly inserted/ deleted of nodes in the tree. These steps incur very high execution time and hence the tree should be constructed with overcoming all the above

mentioned drawbacks. In this paper, we have proposed the RePLNI-tree, which scan the database only once and the time for sequential pattern mining is also found to be low, since there is no additional operation and insertion/ deletion of nodes are performed. In addition the algorithm requires less space, since, the details of candidate lists and S_{code} are unnecessary to our work. This algorithm supports incremental and interactive mining as well.

III. The Proposed Tree

The proposed approach captures sequence information in the weblog to construct the tree along with header table for mining sequence patterns. Each event in the access sequence is inserted as a branch into the tree based on the events arranged in each sequence. This course of action will construct the tree with single scan and reduce the time to build tree. The order of events are not rearranged to follow the sequence pattern strategy (i.e.) an order exist between the item. In addition, the proposed tree supports both incremental and interactive mining, hence, the non-frequent events are not pruned. The construction strategy of the proposed tree is illustrated with suitable example given in sample table. In Table I(a), the TID is the transaction id and the list of all events is considered under the attribute of web access sequence.

TABLE I
SAMPLE TRANSACTIONS (a) EXISTING (b)INCREMENTAL DATABASE

TID	Web Access Sequences
1	a d b a c
2	a e d c a c e
3	d a d a
4	a f d a c f c
5	a d e g f h

(a)

TID	Web Access Sequences
6	d a d a e f
7	e a g f h

(b)

Initially, an empty node is created with null as the label. The first sequence, which is <a d b a c>, scanned once to insert as a branch. Since there is no child for root node, this sequence is directly inserted as the branch of root along with its occurrences. While insertion, the count of the corresponding item in the header table is updated and unique position code is assigned to each node for recognizing the ancestor/descendant relationships between nodes of the tree. Let the inserted branch be <a:1 d:1 b:1 a:1 c:1>. The next sequence is scanned once and the first event in sequence is checked with children of root node. If events are common, merge with the existing branch with increment of count and the process continue to check with the next event. Otherwise, events can't share with existing branch, hence, it will be inserted as a new branch/ path. In this case, a=a thus, merge with existing branch and count is incremented. Since there are no other common events in the sequence, the remaining events are added as a new path to 'a'. This process is continued till all the sequence is inserted as the branch into the tree. The insertion of rest of the transactions is performed in similar fashion. In Fig. 1, we depict the header table and proposed tree with position code.

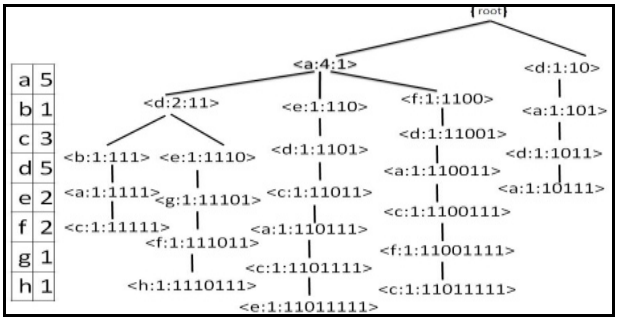


Fig. 1 RePLNI-tree

In order to mine sequential patterns efficiently, the proposed tree must be traversed fast. To facilitate the tree traversal, the proposed tree maintains a header table, where each event is pointing to the first occurred node and nodes with same name are linked in pre-order fashion. Due to the lack of clarity, the links are not depicted in the Fig. 1 and the pre-order traversal is done in the tree.

A. *Algorithm for tree construction.*

Input: Weblog DB, Total No of transactions N.
Output: RePLNI-tree along with header table .
Method:

```
1.  Read DB, N.
2.  Create the root of a tree T and label it as
    "null".
3.  for each transaction t_j till the end of DB
4.   Scan the transaction once.
5.    for each event e_i in t_j
6.      If T as a child C and C.name= e_i.name then
7.        C.support +=1.
8.      else
9.        create a new_node e_i;
10.       e_i support=1;
11.       e_i predecessor_link be linked to T;
12.       e_i successor_link be linked to rest of the
    event e_i in t_j
13.     End if
14.   Increment the count of corresponding event in
    the header table
15.   end for
16. end for
17. Use preorder technique to link the events with
    same label.
```

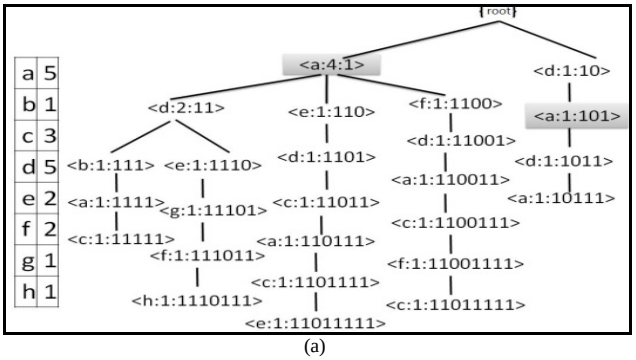
IV. Sequential Patterns Mining And Functionalities of Proposed Tree

A. *Sequential Pattern Mining*

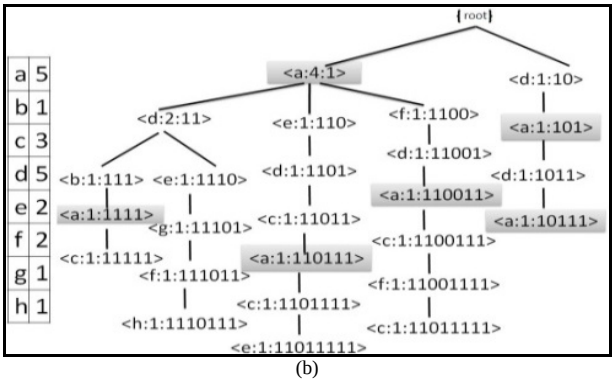
One of the important tasks in association rule mining is to discover the sequential patterns. In this section, we illustrate the sequential pattern mining from the proposed tree using prefix sequential search technique used in PLWAP tree [2]. From Fig. 1, we observe that the proposed tree and header table contains both frequent and non-frequent events. All the events are linked in preorder fashion to mine the sequential patterns. In our approach, while mining the patterns, only frequent events in header table (i.e.) those events satisfy the minimum support is taken into consideration and non-frequent events are discarded virtually. It means, all events and its relative links exist physically, however, while mining, non frequent events are not considered. We can mine the patterns

related to the frequent events with the help of links between the same labels for traversing prefix sequence and identifying their suffix trees as well as the next prefix patterns efficiently. The sequential patterns are found for all the frequent events from top to bottom in header table. Let us consider the frequent event e_r in the header table, using the links of the event, we can calculate the count of e_r, which is the sum of all counts of first occurred $e_{r's}$ in all the path of current suffix tree. If the sum of the count is greater than the user defined support, this event e_r is appended to the sequential pattern list. Next consider the events below e_r as the roots of the current suffix tree and find the next prefix frequent events e_1 in all path of the current suffix tree. Using the same methodology, we can calculate the counts of e_1, if 'e_1' is frequent then e_1 is appended to the already computed sequential pattern list. Now, consider the events below e_1 as roots and this process of searching is continued for each next prefix event along the path of current suffix tree until there is no suffix trees are taken into consideration for search. This action is performed for all those frequent events in the header table to retrieve the sequential patterns.

Let us illustrate this mining process with the suitable example. The first element in the header table is 'a'. Using the link of 'a', we calculated the count of first occurrences of 'a' in all path of suffix tree as 5 and shown in Fig. 2(a). Hence, 'a' is appended to the sequential pattern list. Now, we have to search for 2-sequences such as {aa, ac, ad}, since, the events b, e, f, g and h are non-frequent and the link related to those events only are considered for mining. The mining process is continued with next suffix tree rooted at {d:2:11 , e:1:110 , f:1:1100, d:1:1011} for finding 2-sequence, which is shown in Fig. 2(a). In order to find 'aa' frequent sequence, we follow the link of 'a' in the current suffix tree. Once the status of 'a' is found, the pattern 'aa' is appended to the list. The events below 'a' is considered as root of next suffix tree, which is depicted in Fig. 2(b). This process is continued until no suffix trees are taken into consideration for search. The only difference for mining the patterns from PLWAP and the proposed tree is that while mining the patterns using the proposed tree, the links related to the non-frequent events are not considered.



(a)



(b)

Fig. 2 Mining Patterns (a) for "a" (b) for "aa"

B. Incremental and Interactive Mining

While new transactions are updated in the given sample transactions, presented in Table I(b), each updated transactions is inserted as branches into tree and the counts corresponding to that events in the header table are updated using the procedure presented in Section 3. The preorder linkage is constructed for the newly inserted branches and the patterns can be retrieved efficiently using the prefix sequence search technique. While the weblog is updated with new transactions, it is not required to scan the entire transactions and initiate the sequential pattern mining process from the beginning. Below, we illustrate the incremental mining with a suitable example. TID's 6-7 are updated in the sample transactions and are shown in Table 1(b). The transactions <d a d a e f > and <e a g f h> are scanned once and inserted as existing branch <d:2:10, a:2:101, d:2:1011, a:2:10111, e:1:101111, f:1:1011111>. New branch <e:1:100, a:1:1001, g:1:10011, f:1:100111, h:1:1001111> in the existing tree (Fig. 1) to built as new tree (Fig. 3). Preorder traversal is used to build the new linked for the newly constructed tree. The frequent events from this newly constructed tree are {a, d, e, f}. Rest of the link related to the non frequent events is virtually discarded.

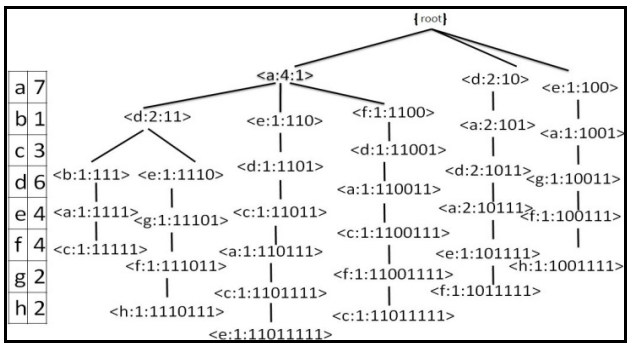


Fig. 3 Newly revised tree for both old and incremental database

Since our proposed tree consists of both frequent and non-frequent events, hence the proposed tree is suitable for interactive mining. If minimum support given by the user is too low or too high, there is a possibility that some of the important patterns may be missed. Interactive mining provides the user an opportunity to interactively alter the minimum support. In interactive mining, the transactional database usually remains unchanged and only minimum support value

is changed. Thus, the tree is constructed only once for a given database and used for mining with various minimum support values. This ensures that the tree construction cost can be reduced to some extent over several runs of the mining process. Similar to other interactive mining algorithms, mining time in the proposed tree is reduced considerably by reusing the sequential patterns mined in the previous round for the current round with different minimum support value.

V. Experimental Results

In this section, we present the performance of RePLNF-tree for mining sequential patterns. RePL4UP is the recently proposed algorithm outperforming all other incremental sequential algorithms such as Incspan, GSP+, MFS+ and non incremental sequential pattern mining algorithm such as PLWAP, which is stated in [3]. For evaluation, we have used the database available at (http: //www.almaden.ibm.com /software/quest/Resources/index.shtml). The database description is given as T10.N.30.U200K, where T represents average number of items per transaction, N is the number of distinct items and U represent the total number of transactions.

Table II
Execution time for various incremental database size

Algorithms	Execution time in see at different incremental db size on support 1%					
	10%	20%	40%	60%	80%	100%
PLWAP	3.187	3.187	3.187	3.187	3.187	3.187
RePLNF	0.458	0.782	1.093	1.25	1.39	1.516
Our Relative Comparison	6.959	4.075	2.916	2.549	2.293	2.102
Obtained Relative Comparison	6.991	4.083	1.976	1.496	1.101	1.248

Execution time of these algorithms is obtained by altering the incremental database size and without altering the values of both minimum support and updated database. The experiment is performed with incrementing the database sizes of 10, 20, 40, 60, 80, 100% of entire database (U), where |U|=200,000 records and with fixed minimum support of 1%. In the final output, the number of frequent patterns (FPs) and total number of frequent 1-items (F1's) for various incremental database sizes are 134 and 38 respectively. In table II, the first and second rows show the execution time of PLWAP and RePLNI algorithms, which is implemented in C. The third row represents the relative comparison between the execution times of RePLNF and PLWAP algorithms. The relative comparison between the execution time of RePL4UP and PLWAP algorithms are obtained from [3] is specified in fourth row. From Table II, we can notice that the execution time of proposed tree takes lower time compared to PLWAP. For the lower incremental database size the relative comparison time of the proposed approach is same as the relative comparison times obtained from [3], since the operation like insertion/ deletion of the nodes and saving/ retrieving the details of S_{code} and candidate lists are less in RePL4UP algorithm. While, increasing the size of incremental database, the performance enhancement of our approach is notably high compared to comparison from [3]. This is due the statement from [3], that the overhead of building, repairing the old tree and updating the structure for next round mining with the small code. In current world, the size of transactions updated with existing one is more than 50%, however, RePL4UP algorithm works better while the incremental database is less than 50% of exist database. In our work, even though the size of the incremental database is high (i.e.) more than 50%, it will not affect the performance, since, the incremental approach is performed by revising the existing tree (i.e.) as it is only insertion of transactions, which takes less time compared to the work done in RePL4UP algorithm. Thus, from the above statement, we conclude that the proposed tree is better for both time and space.

VI. Conclusion

Mining Sequential pattern from large database is used for knowledge discovery. There are various schemes and method proposed for knowledge discovery both in academic and industry. Among various approaches, tree based approaches are found to be well-known and extracts sequential pattern efficiently. Some of the recently proposed well-known trees are discussed in literature survey consumes large time for patterns extraction. In this paper, we have proposed an algorithm, which captures the entire weblog data with single scan to construct tree along with header table. While mining, the preorder technique is used and the links related to the non-frequent items are not taken into consideration. RePL4UP-tree always consumes more time and space to perform various operations and holds the required details to be used for next round. The proposed tree will not hold/ store the unnecessary information. It takes lesser time for mining sequential pattern compared to RePL4UP. The algorithm overcome the drawback of RePL4UP-tree algorithm, which performs well only if the size of incremental database is 50% lesser than the existing one. We pictorially explained the proposed work with suitable example. Additionally, we have showed experimentally that the proposed tree supports both incremental and interactive mining. The tree is constructed for all the items in weblog irrespective of whether the, items are non-frequent or frequent. While the new transactions are updated, the already existing tree is revised for updated transactions. We have estimated the performance of the proposed tree on benchmark weblog. It is noticed from the experimental result that the time taken by the proposed tree is encouraging.

Acknowledgment

The work done by Dr. A.Vadivel is supported by research grant from the Department of Science and Technology, India, under Grant SR/FTP/ETA-46/07 dated 25th October, 2007 and DST/TSG/ICT/2009/27 dated 3rd September 2010.

References

[1] R. Agrawal and R. Srikant, "Mining sequential patterns," In: Proceedings of the 11th Int'l conference on data engineering, Taipei, 1995, pp 3–14.

[2] H. Cheung, X. Yan and J. Han, "IncSpan: incremental mining of sequential patterns," In: Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining, Seattle, 2004, pp. 527–532.

[3] C.I. Ezeife, Yi Lu and Yi Liu, "PLWAP sequential mining: open source code," In: Proceedings of the open source data mining workshop on

frequent pattern mining implementations, in conjunction with ACM SIGKDD, Chicago, August 21–24, 2005, pp 26–29.
[4] C.I. Ezeife and Yi Liu, "Fast incremental mining of web sequential patterns with PLWAP tree," Int J Data Mining Knowledge Discovery, Springer Science Publisher, vol. 19, 2009, pp 376–416.
[5] B. Kao, M. Zhang, C-LYi and D.W Cheung, "Efficient algorithms for mining and incremental update of maximal frequent sequences," Int J Data Mining Knowledge Discovery, Springer Science Publisher, vol. 10, 2005, pp 87–116.
[6] F. Masseglia, P. Poncelet and R. Cicchetti, "An efficient algorithm for web usage mining," Netw Inform Syst Journal, vol. 2(5–6), 1999, pp 571–603.
[7] A. Nanopoulos and Y. Manolopoulos, "Mining patterns from graph traversals," Data Knowledge Engineering, vol. 37(3), 2001, pp 243–266.
[8] S. Nguyen, X. Sun and M. Orlowska, "Improvements of incSpan: incremental mining of sequential patterns in large database," In: Proceedings 2000 Pacific-Asia conference on knowledge discovery and data mining (PAKDD'05), 2005, pp 442–451.
[9] S. Parthasarathy, M.J Zaki, M. Ogihara and S. Dwarkadas, "Incremental and interactive sequence mining," In: Proceedings of the 8th international conference on information and knowledge management (CIKM99), Kansas City, pp 251– 258.
[10] J. Pei, J. Han, B. Mortazavi-Asl and H. Pinto, "PrefixSpan: mining sequential patterns efficiently by prefix projected pattern growth. In: The proceedings of the 2001 international conference on data engineering (ICDE '01), pp 215–224.
[11] J. Pei, J. Han, B. Mortazavi-asl and H. Zhu, "Mining access patterns efficiently from web logs," In: proceedings 2000 Pacific-Asia conference on knowledge discovery and data mining (PAKDD'00), 2000, Kyoto, pp 396–407.
[12] M. Spiliopoulou, "The laborious way from data mining to webmining," Journal Computer System Science Eng, Special Issue Semant Web ,vol.14, 1999, pp 113–126.
[13] R. Srikant and R. Agrawal, "Mining generalized association rules," In: Proceedings of the 21st int'l conference on very large databases (VLDB), Zurich,1995.
[14] R. Vishnu Priya, A.Vadivel and R.S. Thakur, "Frequent Pattern Mining Using Modified CP-Tree for Knowledge Discovery," In the proceedings of international conference ADMA'10, Part I, LNCS 6440, 2010, pp. 254–261.
[15] K. Wang, "Discovering patterns from large and dynamic sequential data," J Intell Information System, vol. 9(1), 1997, pp 33–56
[16] M.J Zaki, "SPADE: an efficient algorithm for mining frequent sequences," Machine Learning, vol.42, 2000, pp 31–60.
[17] M. Zhang, B. Kao, D. Cheung and C-L.Yip, "Efficient algorithms for incremental update of frequent sequences," In: Proceedings of the sixth Pacific-Asia conference on knowledge discovery and data mining (PAKDD), 2002, pp 186–197.