Graph based Approach and Clustering of Patterns (GACP) for Sequential Pattern Mining

Ashish Patel [1], Amisha Patel [2]

[1] Department of Computer Engineering/Information Technology
Shri S'ad Vidya Mandal Institute of Technology
Bharuch, Gujarat – India

Abstract The sequential pattern mining generates the sequential patterns. It can be used as the input of another program for retrieving the information from the large collection of data. It requires a large amount of memory as well as numerous I/O operations. Multistage operations reduce the efficiency of the algorithm. The given GACP is based on graph representation and avoids recursively reconstructing intermediate trees during the mining process. The algorithm also eliminates the need of repeatedly scanning the database. A graph used in GACP is a data structure accessed starting at its first node called root and each node of a graph is either a leaf or an interior node. An interior node has one or more child nodes, thus from the root to any node in the graph defines a sequence. After construction of the graph the pruning technique called clustering is used to retrieve the records from the graph. The algorithm can be used to mine the database using compact memory based data structures and cleaver pruning methods.

Index Terms-GACP, data mining, sequential data mining, clustering

I. I. INTRODUCTION

Data mining is a relatively new research area that extracts knowledge which is hidden in the database and hence very useful in information retrieval. Frequent pattern mining from sequential data is one of the most important tasks. Frequent patterns are required in satellite images, customer databases, telecommunication systems, frequent buying patterns etc. Agrawal R. and Shrikant R. have first found out some algorithms for mining frequent pattern from a large collection of data sequences [1]. They have used support for analyzing the percentage of data sequences containing the pattern. Later Agrawal R. and Shrikant R. have used some constraints like minimum and maximum gap between adjacent elements of a pattern [2]. Gradually in the field of computing storage and processing devices are become boundless and have allowed the users to store and process huge collection of data.

Example of such collection includes web site usage analysis, medical reports, science and engineering databases etc. They have drawn the attention of a number of researchers in the field of data mining. Mainly the collected data is in sequential form, hence arises the scope for different techniques for exploring sequential patterns.

The goal is to find trends across large number of transactions that can be used to understand and exploit sequential patterns. Given a Sequence Database, the problem to find frequently occurring Sequential patterns on the basis of minimum support provided. Here a brief study of Generalized Sequential Pattern and Web access pattern mine is done which is much more efficient than the candidate generation based algorithms. But it required much space to store the intermediate trees which are generated during the process. So a new algorithm GACP is proposed to make the mining more efficient in terms of storage and time. GACP uses the concept of graph traversal by constructing the graph in one database scan only. The constructed graph then can be used by the algorithm to find the sequential patterns or order list of events from the database. The algorithm uses clustering techniques to prune the paths of the graph.

II. BACKGROUND

The sequential pattern mining problem was first introduced by Agrawal and Srikant[1]. Given a set of sequences, where each sequence consists of a list of elements and each element consists of a set of items, and given a user-specified min support threshold, sequential pattern mining is to find all of the frequent

subsequences, i.e., the subsequences whose occurrence frequency in the set of sequences is no less than min support. A typical Apriori-like sequential pattern mining method, such as GSP, adopts a multiple-pass, candidate generation-and-test approach. The first scan finds all of the frequent items which form the set of single item frequent sequences. Each subsequent pass starts with a seed set of sequential patterns, which is the set of sequential patterns found in the previous pass. This seed set is used to generate new potential patterns, called candidate sequences. Each candidate sequence contains one more item than a seed sequential pattern, where each element in the pattern may contain one item or multiple items. The number of items in a sequence is called the length of the sequence. So, all the candidate sequences in a pass will have the same length. The scan of the database in one pass finds the support for each candidate sequence.

Table 1. Sequence database

Candidate-3 sequences	Candidate -4 sequences	
	After join	After pruning
< (1,2) (3) >	< (1,2) (3,4) >	< (1,2) (3,4) >
< (1,2) (4) >	< (1,2) (3) (5) >	
< (1) (3,4) >		
< (1,3) (5) >		
< (2) (3,4) >		
< (2) (3) (5) >		

All the candidates whose support in the database is no less than min support from the set of the newly found sequential patterns. This set then becomes the seed set for the next pass. The algorithm terminates when no new sequential pattern is found in a pass, or when no candidate sequence can be generated.

Refer a sequence with k items as a k-sequence. (If an item occurs multiple times in different elements of a sequence, each occurrence contributes to the value of k). Let Lk denote the set of all frequent k-sequences, and Ck the set of candidate k-sequences. Given Lk-1, the set of all frequent (k-1)-sequences, we want to generate a superset of the set of all frequent k-sequences. Let first define the notion of a contiguous subsequence.

Definition: Given a sequence s = < s1,s2,.....sn > and a subsequence c, c is a contiguous subsequence of s if any of the following conditions hold:

1. c is derived from s by dropping an item from either s1 or sn.
2. c is derived from s by dropping an item from an element si which has at least 2 items.
3. c is a contiguous subsequence of c', and c' is a contiguous subsequence of s.

For example, consider the sequence s = < (1, 2) (3, 4) (5) (6) >. The sequences < (2) (3, 4) (5) >, < (1, 2) (3) (5) (6) > and < (3) (5) > are some of the contiguous subsequences of s. (see figure 1) However, <(1, 2) (3, 4) (6)> and < (1) (5) (6) > are not. Likewise the operation is carried out and one can find sequential pattern from the database.

Another algorithm is of WAP-tree[3], which stands for web access pattern tree. The main steps involved in this technique are summarized next. The WAP-tree stores the web log data in a prefix tree format similar to the frequent pattern tree (FP-tree) for non-sequential data.

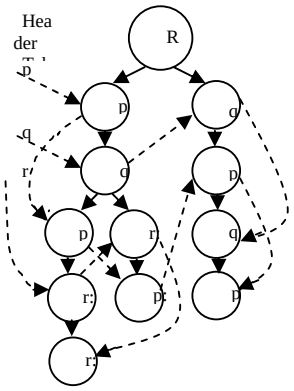
Table 2. Sequence database for WAP-tree

TID	Web access sequence	Frequent Subsequence
100	pqspr	pqpr
200	tptqrp	pqrp
300	opqupt	qpqp
400	puqprur	pqprr

The algorithm first scans the web log once to find all frequent individual events which is shown in table 2. Second, it scans the web log again to construct a WAP-tree over the set of frequent individual events of each transaction. Third, it finds the conditional suffix patterns. In the fourth step, it constructs the intermediate conditional WAP-tree using the pattern found in previous step. Finally, it goes back to repeat the previous steps until the constructed conditional WAP-tree has only one branch or is empty.

Thus, with the WAP-tree algorithm, finding all frequent events in the web log entails constructing the WAP-tree and mining the access patterns from the WAP tree. The web log access sequence database in Table 2 is used

to show how to construct the WAP-tree and do WAP-tree mining. Suppose the minimum support threshold is set at 75%, which means an access sequence, *s* should have a count of 3 out of 4 records in our example, to be considered frequent. Constructing WAP-tree, entails first scanning database once, to obtain events that are frequent. When constructing the WAP-tree, the non-frequent part of every sequence is discarded. Only the frequent sub-sequences are used as input. For example, in Table 2, the list of all events is p, q, r, s, t, u and the support of p is 4, q is 4, r is 3, s is 1, t is 2, and u is 2.With the minimum support of 3, only p, q, r are frequent events. Thus, all non-frequent events (like *s, t, u*) are deleted from each transaction sequence to obtain the frequent subsequence. With the frequent sequence in each transaction, the WAP-tree algorithm first stores the frequent items as header nodes so that these header nodes will be used to link all nodes of their type in the WAP-tree in the order the nodes are inserted. When constructing the WAP tree, a virtual root (Root) is first inserted. Then, each frequent sequence in the transaction is used to construct a branch from the root to a leaf node of the tree. The complete tree for given database is shown in fig. 1.

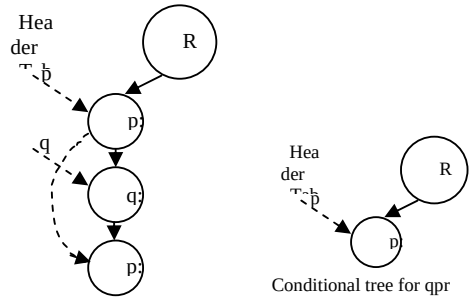


Complete Tree

figure:1 Tree constructed by WAP

After this algorithm first computes prefix sequence of the base r or the conditional sequence base of *c as*: pqp:2; pq:1; pqpr:1; pqp:-1. The conditional sequence list of a suffix event is obtained by following the header link of the event and reading the path from the root to each node (excluding the node).

The conditional search of r is now finished. The search for frequent patterns that have the suffix of other header frequent events (starting with suffix base |q and then |p) are also mined the same way the mining for patterns with suffix r is done which is shown in fig. 2.



Conditional tree for r

Conditional tree for qpr

figure:2 Conditional Sequence tree for r and pqr

After mining the whole tree, discovered frequent pattern set is: {r, qpr, pqpr, pr, pqr, qr, qb, pq, p, pp, qp, pqp}.

III. RELATED WORK

Since its introduction, sequential pattern mining [1] has become an important data mining task, and it has been used in a broad range of applications, including the analyses of customer purchase behavior, disease treatments, Web access patterns, DNA sequences, and many more. The problem is to find all sequential patterns with higher or equal support to a predefined minimum support threshold in a data sequence database. The

support of a sequential pattern is the number, or percentage, of data sequences in the database that contain that pattern. Different techniques and algorithms have been proposed to improve the efficiency of this task [1, 2, 5, 6, 10].

The sequential patterns generated by the sequential pattern mining program can be used as the input of another program to do a specific data mining task, for example frequent patterns can be used to generate association rules. It has been recognized that by decreasing the minimum support, the number of frequent sequential patterns can grow rapidly. This large number of frequent sequential patterns can reduce the efficiency, and effectiveness of the mining task. The efficiency is reduced, because of the large number of patterns generated in the first stage needed to be processed in the later stages of the mining task. The effectiveness is also reduced, because users have to go through a large number of elements in the result set to find useful information.

In recent years business and scientific research has seen an explosion in the amount of data collected [4]. Supermarket chains routinely collect terabytes of data on purchases their customers make. Several scientific fields like astronomy or particle physics now have to deal with databases in the range of petabytes. This data is useless unless it can be analyzed which, due to its huge size, is a very difficult task. It has been noted [7] that as processor speed doubles every 18 month (according to Moore's law), the amount of data stored by companies doubles every year, so increase in computing power will not provide a solution and a new methodology is required. This resulted in the creation of the data mining field. Early efforts in data mining concentrated on modifying Machine Learning algorithms to scale up better with the size of the datasets. The first major algorithm developed specifically for large datasets came with the introduction in 1993 by Rakesh Agrawal et al. [1] of the association rule mining problem. In [1] the Apriori algorithm has been presented which is capable of finding all association rules satisfying certain criteria even in very large datasets. The paper has been followed by hundreds of publications further improving the algorithm, and applying it in other areas such as clustering.

Today sequential pattern mining has proved to be a very promising. The application of sequential pattern mining are in areas like Medical treatment, science & engineering processes, telephone calling patterns, bioscience. Sequential pattern mining Web usage mining for automatic discovery of user access patterns from web servers. It is used by an e-commerce company, this means detecting future customers likely to make a large number of purchases, or predicting which online visitors will click on what commercials or banners based on observation of prior visitors who have behaved both positively and negatively to the advertisement banners. So it very necessary that the algorithm used to find the sequential patterns should efficient in terms of space and time complexity. There are many algorithms proposed before such as Apriori, AprioriAll, AprioriSome [9] by R. Agrawal and R. Srikant. The GSP (Generalized Sequential Patterns) [2] algorithm is 20 times faster than the Apriori algorithm. The Graph Traversal mining [8], uses a simple unweighted graph to store web sequences and a graph traversal algorithm similar to Apriori algorithm to traverse the graph in order to compute the k-candidate set from the (k-1)-candidate sequences without performing the Apriori-gen join. The FP-tree structure [12] constructs a tree to store the frequent items and uses this tree to mine the frequent patterns. FreeSpan [13] and PrefixSpan [11] are other pattern growth methods which deal with the pattern mining.

IV. THE GACP ALGORITHM AND CLUSTERING OF PATTERNS

The new approach is based on graph representation, and avoids recursively re-constructing intermediate WAP-trees during mining of the WAP tree for frequent patterns. The GACP algorithm is not using the candidate generate and test methodology so it avoids the need of repeatedly scanning the intermediate results like GSP. A graph is a data structure accessed starting at its first node and each node of a tree is either a leaf or an interior node. A leaf is an item with no child. An interior node has one or more child nodes and is called the parent of its child nodes. All children of the same node are siblings. Like WAP-tree mining, every frequent sequence in the database can be represented on a branch of a graph. Thus, from the root to any node in the graph defines a frequent sequence. For any node labeled ei in the graph, all nodes in the path from root of the graph to this node (itself excluded) form a prefix sequence of ei . The count of this node ei is called the count of the prefix sequence. Any node in the prefix sequence of ei is an ancestor of ei. On the other hand, the nodes from ei (itself excluded) to leaves form the suffix sequences of ei. Any node in the suffix sequence is a descendant of ei . The suffix sequence of ei is not unique. Normally, there are several children of ei in the tree, and each branch from a child to a leaf node will represent a suffix sequence and these suffix branches of ei are called the suffix trees of ei .

1. The GACP data structure, similar to WAP-tree, is used to store access sequences in the database, and the corresponding counts of frequent events compactly, so that the tedious support counting is avoided during mining.

2. Starting with the first node, each transaction will create a branch for a specific pattern.

3. Every time if the branch starting with the same item is available, the GACP algorithm follows the same path and increments the count associated with each node, as shown in the above figure.

4. An efficient recursive algorithm is proposed to enumerate sequential patterns from the graph. The philosophy of this mining algorithm is prefix sequence search rather than suffix search done by WAP-tree algorithm. Instead of searching common suffix pattern (that is, obtain the frequent pattern starting with the last event in the sequence and keep extending the suffix subsequence until the entire sequence is found) as WAP tree mining does, the GACP searches common prefix pattern in the graph (by finding the first event in the sequence and extending this prefix subsequence until the entire sequence is found).

Table:3 Sample access sequence database

TID	Access sequence	Frequent Subsequence
100	abdac	abac
200	eaebcac	abcac
300	babfaec	babac
400	afbacfc	abacc

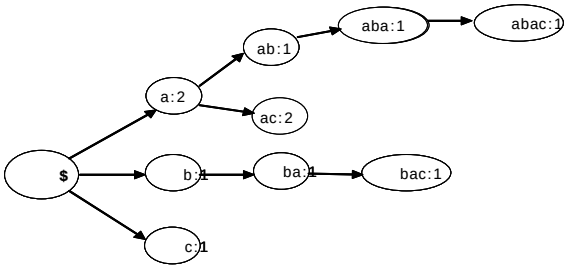


figure:3 Graph for transaction abcac

Start with $ and total number of unique items a,b and c, the first Transaction is abac and the corresponding sequential patterns are abac, bac, ac and c. So first branch is created with a:1, ab:1, aba:1, abac:1 as shown in the fig. 3.

The second Transaction is abcac and the corresponding sequential patterns are abcac, bcac, cac, ac and c. With the increment of each node count we will get Figure shown below. Every Time node count for each second referenced node is incremented. For example node count for node ac is now 2.

After inserting babac and abacc we have the complete graph. From this graph we can easily find out the required frequent sequences. Here each node has a counter which shows us the frequency associated with that pattern, e.g. abac:3. The procedure for retrieving the desired frequency again start with graph traversal, starting with the initial node $, and following the subsequent paths as shown in the fig. 4.

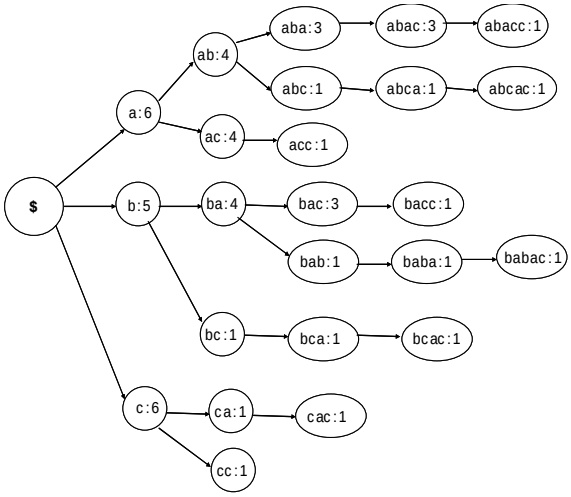


figure:4 Graph for example database

A simplest approach is to cluster the patterns is to group them by their end states. An automaton representation for each cluster can be found from the automaton representations by reversing the edges of the underlying graph and computing which nodes are reachable from the corresponding final state. This can be done in time linear in the number of transitions. There are several possibilities how the clustering can be visualized by the automata. For example, the nodes can be colored based on which clusters they belong. If a state belongs to several clusters, its color can be a mixture of the colors of the clusters.

Now suppose we want to retrieve the entire frequent item with minimum support=3 we will get the result shown in fig.5. The entries with support greater than 3 will be there in the set of clustered item. The GACP algorithm eliminates the need to store numerous intermediate trees during mining. GACP also eliminates the need to store and scan intermediate conditional pattern bases for re-constructing intermediate trees. In a Tree based approach we have to store the header table, this is not required in GACP. In candidate generation and test method, large numbers of candidates are generated during mining process. This is also note required in the given algorithm.

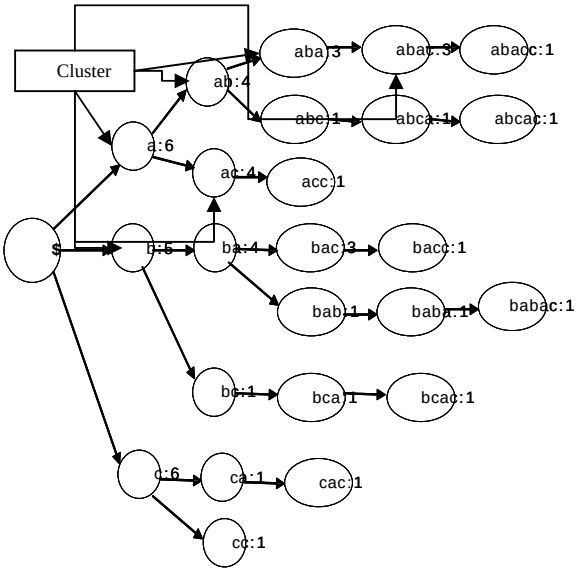


figure:5 Clustering of the graph for retrieving the desired frequency (Support = 3)

Since only the graph is stored, it drastically cuts off huge memory access costs, which may include disk I/O cost in a virtual memory environment, especially when mining very long sequences with millions of records. This algorithm also eliminates the need to store and scan intermediate conditional pattern bases for re-constructing intermediate Web access pattern trees. This algorithm uses clustering of nodes to store all events in the same suffix path closely together in the linkage, making the search process more efficient.

A. The Algorithm - GACP

Input: Access database D and minimum support δ
(0< δ<=1)
Output: Sequential patterns in D
Begin:
1. Scan D to find frequent subsequences in each transaction with respect to δ
2. With the first scan also find out total number of unique items in D.
3. Initialize Count associated with each node of the graph with 0
4. Create a first node Φ of the graph G
5. for each frequent sequence RK in D
6. {
7. Create a stack for RK

```
8.  Check Item IN Є RK
9.    If (IN is a new node) then
10.        Create a new branch BI from Φ
11.   Else if (IN Є BI) then
12.       Insert IN in BI→Last
13.       Increment Count from i=1 to i=Last-1
14.   Else
15.       Create a new branch BI from Φ
16. }
17. Check (Stack)
18. If (Stack != Empty ())
19. {
20.   Stack.pop();
21.   Go to step 7
22. }
23. Else
24.  Go to step 6
25. Scan last node of each branch
26. Cluster the nodes with same frequency FC
27. Assign a new node CI for each FC
28. Directly retrieve frequency FMAX with
    1 <=I <= FMAX
29. End
```

The algorithm scans the access sequence database first time to obtain the support of all events in the item set. All items that have a support greater than or equal to the minimum support are frequent. Each node in a graph has two fields: node label and node count. The root of the graph is a special virtual node with an empty label and count 0. Every other node is labeled by an item in the item set. Then it scans the database a second time to obtain the frequent sequences in each transaction. The non-frequent events in each sequence are deleted from the sequence. The algorithm starts building the graph starting from the root node. From the frequent sequence itemsets, it will check the items one by one. Now for each frequent sequence it creates a stack (which is used to store the entire sequence) for the sequence. Then the first item is compared with the root, if it is a new node, new branch is created and the node count stores the value. Second, if the item of the sequence is not a new node then create a new node and store that item as well as the count field. This process is repeated for each item of the stack. If the stack is empty, algorithm will check the new sequence from the database. This way complete graph is created. After construction of the graph, the algorithm searches the graph for finding out the frequency of each node which is there with its node count. Clustering technique can be applied then to store each and every frequency from the graph. Extra nodes are created for this frequency. The backward pointers of the frequency nodes will give the path for traversing the graph for required frequency.

B. Complexity

GACP creates one node in its graph for each frequent itemset. At the first glance, this seems to be highly compact since tree based algorithms does not ensure that each frequent node will be mapped to only one node in the tree. However, each branch of the tree may store many "hidden" frequent patterns due to the potential generation of many combinations using its prefix paths. Notice that the total number of frequent k-itemsets can be very large in a large database or when the database has quite long frequent itemsets. In the worst case, the sequences in the batch have the same length: m. The GACP algorithm has a time complexity of $O(m^2)$. In the worst case, the clustering algorithm has a time complexity of $O(m^2{:}n^2)$ with n the number of sequences. Actually, in the worst case, GACP is called once for the first sequence, twice for the second, and so on. The complexity is thus $O(n(n+1) / 2)m^2 = O(m^2n^2)$ It is well suited for sequential patterns and the results obtained datasets show its effectiveness. The complexity of the graph construction algorithm for a bath of sequence is $O(nm^2)$.

V. EXPERIMENTAL RESULTS

This experiment uses fixed size database and different minimum support .The datasets and algorithms are tested with minimum supports between 0.8% and 10% against the 60 thousand (60 K) database. From Table 4 and Figure 6, it can be seen that the execution time of every algorithm decreases as the minimum support

increases. This is because when the minimum support increases, the number of candidate sequence decreases. Thus, the algorithms need less time to find the frequent sequences.

Table:4 Execution time for dataset at different minimum supports.

	Runtime in Seconds at different supports				
Algo.	2	3	4	5	10
GSP	1200	1075	810	550	325
WAP	750	510	330	280	150
GACP	230	160	110	95	48

The GSP algorithm always uses less runtime than the WAP and GSP algorithms. GSP has the highest storage cost because it uses the candidate-generate and test methodology. It scans the database several times before getting the results. WAP tree mining incurs higher storage cost (memory or I/O). Even in memory only systems, the cost of storing intermediated trees adds appreciably to the overall execution time of the program.

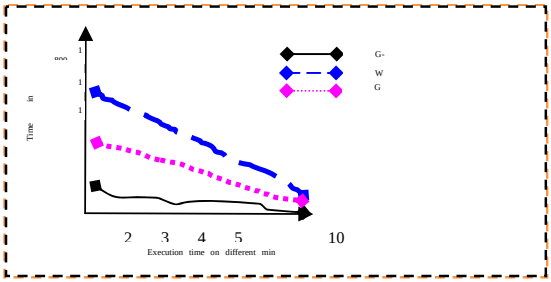


figure:6 Execution time trend for different minimum support

It is however, more realistic to assume that such techniques are run in regular systems available in many environments, which are not memory only, but could be multiple processor systems sharing memories and CPU's with virtual memory support.

As the minimum support threshold decreases, the number of events that meet minimum support increases. This means that WAP-tree becomes larger and longer, and the algorithm needs much more I/O work during mining of WAP tree.

Table:5 Execution times trend with different data sizes.

	Runtime in Seconds at different supports			
Algo.	40k	60k	80k	100k
GSP	83	142	190	226
WAP	55	93	125	159
GACP	27	43	60	89

As minimum support decreases, the execution time difference between GSP, WAP-tree and GACP increases. In this experiment, databases with different sizes from 20 K to 100 K with the fixed minimum support of 7% were used which is shown in table 5 and fig.7.

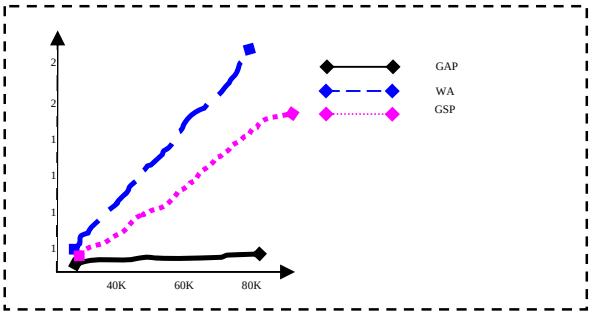


figure:7 Execution time trend for different data sizes.

VI. CONCLUSION

In this paper, the problem of sequential pattern mining is analyzed. Here after discussing the three approaches it is clear that the GACP approach is more efficient than GSP and WAP Tree approach. This presents a discussion of the advantages and disadvantages of all the three approaches conducted by comparing the performance with help of graph. The GACP algorithm eliminates the need to store numerous intermediate Web access pattern trees during mining. Since only the graph is stored, it drastically cuts off huge memory access costs, which may include disk I/O cost in a virtual memory environment, especially when mining very long sequences with millions of records. This algorithm also eliminates the need to store and scan intermediate conditional pattern bases for re-constructing intermediate Web access pattern trees. This algorithm uses clustering of nodes to store all events in the same suffix path closely together in the linkage, making the search process more efficient.

REFERENCES

[1] Agrawal R., Srikant R.: Mining Sequential Patterns. Proc. of the 11th Int'l Conference on Data Engineering (1995).

[2] Srikant R., Agrawal R.: Mining Sequential Patterns:Generalizations and Performance Improvements. Proc. of the 5th Int'l Conf. on Extending Database Technology (1996).

[3] C. Ezeife and Y. Lu. Mining web log sequential patterns with position coded pre-order linked wap-tree. International Journal of Data Mining and Knowledge Discovery (DMKD) Kluwer Publishers, 10(1):5–38, 2005.

[4] A Dissertation Presented by Szymon Jaroszewicz., "Information-Theoretical and Combinatorial methods in Data Mining", doctor of philosophy December 2003.

[5] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth", Proc. 2001 Int. Conf. on Data Engineering (ICDE'01), Heidelberg, Germany, April 2001.

[6] M. J. Zaki, "SPADE: An Efficient Algorithm for Mining Frequent Sequences", in Machine Learning Journal, special issue on Unsupervised Learning (Doug Fisher, ed.), pages 31-60, Vol. 42 Nos. 1/2, Jan/Feb 2001.

[7] U. Fayyad, N. Rothleder, and P. Bradley. "Enterprise Customer Data Mining for E-Business." Tutorial at the Second SIAM Conference on Data Mining, April 2002.

[8] Yu Hirate, Eigo Iwahashi, and Hayato Yamana, "TF2P-growth: An Efficient Algorithm forMining Frequent Patterns without any Thresholds", International Journal of Data Mining and Knowledge Discovery, Jan 2005.

[9] R.Agrawl and R.Srikant, "Fast Algorithms for Mining Association Rules," In Proc. of VLDB '94, pp. 487-499, Santiago, Chile, Sept. 1994.

[10] Gunopulos, D., Khardon, R., Mannila, H., Saluja, S., Toivonen, H., Sharma, R.s.: Discovering all most specific sentences. ACM Transaction on Database Systems 28 (2003) 140-174

[11] Han. J., Pei, J., Mortazavi-Asl, B. and Pinto, H. "Prefixspan : Mining sequential patterns efficiently by prefix-projected pattern growth." In proceedings of the 001 International Conference on Data Engineering (ICDE 01), p.214-224, 2001.

[12] Han, J., Pei, J., Yin, Y. and Mao, R. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. International Journal of Data Mining and Knowledge Discovery, p.53–87, Jan 2004.

[13] Han. J., Pei, J., Mortazavi-Asl, B. Q. Chen, U. Dayal, and M.-C. Hsu. "Freespan : Frequent pattern-projected sequential pattern mining" In proc. 2000 ACM-SIGMOD Int. Conf. Management of Data, pages 1-12, Dallas, TX, May 2000.

Ashish Patel M.E (Comp.) is associated with engineering education since 2005. His interested areas are data mining, computational theory, wireless networks etc. He has published more than 5 papers in various internation/national journal and conferences.

Amisha Patel. M.E. (Comm. Engg.) is individual researcher. She has achieved her master degree from G.H.Patel College of Engineering, Vallabh Vidyanager, Gujarat.