Improved PrefixSpan Algorithm for Efficient Processing of Large Data

Pratik Saraf
M.E. Scholar
Computer Engineering Department,
Thakur College of Engineering and Technology, Mumbai
pratiksaraf308@gmail.com

Sheetal Rathi
Assistant Professor
Computer Engineering Department,
Thakur College of Engineering and Technology, Mumbai
sheetal.rathi@thakureducation.org

Abstract:- PrefixSpan (Prefix-projected Sequential pattern mining) algorithm is very well known algorithm for sequential data mining. It extracts the sequential patterns through pattern growth method. The algorithm performs very well for small datasets. As the size of datasets increases the overall time for finding the sequential patterns also get increased. The efficiency of PrefixSpan algorithm gets reduced while processing the large data. The cost of constructing the projected dataset is also huge which ultimately affect the memory utilization.

The paper provides an improvised PrefixSpan algorithm which overcomes the problems faced by existing PrefixSpan thus reduces the time complexity and enhances the memory utilization. The improvised PrefixSpan algorithm takes only 1/4th time to that of existing PrefixSpan algorithm in order to find sequences from large sequential data which ultimately reduces the time complexity. Parallel processing through multi-threading enabled the algorithm to find sequential pattern in such quick time. The memory utilization is also enhanced in improvised PrefixSpan algorithm through GZIP. GZIP is lossless compression technique that performs very well on text-based contents. It achieves compression rates of as high as 70-90% for larger files. It is helpful to reduce the construction cost of projected dataset. In this paper comparative analysis of results of existing PrefixSpan algorithm and improvised PrefixSpan algorithm in terms of time complexity and memory utilization is done. The results are drawn on basis of two threshold values that is minimum support and maximum prefix length.

Keywords:- *PrefixSpan Algorithm, Minimum support, Maximum prefix length, Time complexity, Memory utilization.*

*****

I. INTRODUCTION

The sequential pattern mining problem was first addressed by Agrawal and Srikant [1995] [1, 2]. They said that, for a given sequential database, in which each sequence consists of a list of transactions. All these transactions are ordered by transaction time and each transaction is a set of items. Sequential pattern mining is made in order to discover all sequential patterns based on user-defined minimum support. The support of a pattern is calculated through the number of data-sequences that the pattern contains.

Sequential Pattern Mining is a well known data mining technique which consists of finding sub-sequences and patterns which are appearing in a given set of sequence very often. The PrefixSpan algorithm which is proposed by Jian

Pei et al. widely used to find the sequential patterns. It avoids the huge candidate sequence generation thus improvise the execution time and memory utilization. But the data is not restricted to a limited size now as enormous amount of data is generated daily. Though PrefixSpan is efficient algorithm to find sequential patterns but it faces problem when the large data is provided as input. The time complexity increases and memory utilization is also poor. These problems are handled by improvised PrefixSpan algorithm is provided. Paper gives brief of the improvisation made in existing PrefixSpan algorithm.

The first section of the paper gives evolution of various sequential pattern mining algorithms. The second section deals with the objective of work. The third section gives the brief about the steps for executing the PrefixSpan algorithm and the results of execution on various datasets (C16D200k, C16D100k and C21D36k) are drawn in terms of time complexity and memory utilization. In fourth section, the improvisation in existing PrefixSpan is discussed along with the improvised results. Last section deals with the conclusion and future scope for the work.

II. RELATED WORK

Sequential pattern mining [1, 2] was first introduced by Agrawal and Srikant in 1995, and three algorithms as AprioriSome, AprioriAll and DynamicSome [1] are proposed by them. Then different parameters such as time constraints, sliding window time, and user-defined systematic, are used so as to generalise the definition of sequential pattern mining and proposed an Apriori-based, improved algorithm as GSP (Generalized Sequential Patterns). Zaki brought up SPADE [3] algorithm which was based on the equivalence of classes. It was simply the expansion of vertical data format sequential pattern mining method. Later pattern growth method come into exists. Two pattern growth algorithms were proposed by Han, which included FreeSpan [4] and PrefixSpan [5]. Compared with projected databases and subsequence connections, PrefixSpan is more efficient than FreeSpan. SPMIP (Sequential Pattern mining based on Improved PrefixSpan) algorithm [6] by LIU Pei-yu et.al and BLSPM (bi-level Sequential Pattern mining) algorithm [7] by Lian Dong and Wang hong are proposed which overcome the problem of constructing huge projected dataset in PrefixSpan algorithm.

III. OBJECTIVE OF THE WORK

The existing PrefixSpan algorithm is run on various datasets. The sizes of datasets are increased gradually so as to check the execution of algorithm from small datasets to large datasets. Various parameters like memory utilization, time complexity and size of projected dataset are set as benchmark for evaluating the results derived by algorithm on different datasets. Finally the improvisation is done in existing PrefixSpan through parallel processing and compression technique (gzip). The results of both existing PrefixSpan and improvised PrefixSpan are compared.

IV. PREFIXSPAN ALGORITHM

A pattern-growth method based on projection is used in PrefixSpan algorithm [5] for mining sequential patterns. The basic idea behind this method is, rather than projecting sequence databases by evaluating the frequent occurrences of sub-sequences, the projection is made on frequent prefix. This helps to reduce the processing time which ultimately increases the algorithm efficiency.

Jian Pei et al. proposed a novel algorithm called PrefixSpan (Prefix-projected Sequential Pattern Mining) algorithm [5] which works on projection of database and sequential pattern growth. The divide and search space technique is implemented by PrefixSpan. Algorithm mines sequential patterns through following steps;

i. Find length-1 sequential patterns. The given sequence S is scanned to get item (prefix) that occurred frequently in S. For the number of time that item occurs is equal to length-l of that item. Length-l is given by notation <pattern> : <count>.
ii. Divide search space. Based on the prefix that derived from first step, the whole sequential pattern set is partitioned in this phase.
iii. Find subsets of sequential patterns. The projected databases are constructed and sequential patterns are mined from these databases. Only local frequent sequences [8], [9] are explored in projected databases so as to expand the sequential patterns. The cost for constructing projected database is quite high. Bi-level projection and pseudo-projection methods are used to reduce this cost which ultimately increases the algorithm's efficiency.

The PrefixSpan has following advantages:

a. No candidate generation.
b. The frequency of local items only countable.
c. Divide-and-conquer search methodology is used.
d. It is superior to GSP as well as FreeSpan.

But still there is need to improvise the PrefixSpan algorithm so as to reduce the cost for creating projected databases as well as to reduce the scanning time of projected databases.

V. RESULTS OF EXISTING PREFIXSPAN ALGORITHM

The results are drawn by executing the PrefixSpan algorithm on different datasets. Minimum support (minsup) and Maximum prefix length (MPL) are the two parameters which are specified initially, on basis of which the sequential patterns are generated. In previous research only the minimum support values are considered to get the sequential patterns through PrefixSpan algorithm. As the algorithm is tested on large datasets the additional parameter that is maximum prefix length is provided at start of execution and results are drawn based on these two parameters.

The different datasets C16D200k and C16D100k are developed using synthetic dataset generator. One more dataset C21D36k which is conversion of Bible into sequence database is also used to draw the results of algorithm execution. The C stands for average number of item sets per sequence and D stands for number of sequences in the labels of datasets.

Minimum support (MS) and maximum prefix length (MPL) values are set initially on the basis of which the sequential patterns are generated from sequential datasets. The performance of PrefixSpan algorithm on different datasets is evaluated by two parameters that are time complexity and memory utilization. The values of time complexity and memory utilization vary according to different datasets on which the algorithm is run.

The different values of minimum support (MS) and maximum prefix length (MPL) are provided initially for the execution of PrefixSpan algorithm on different datasets (C16D200k, C16D100k and C21D36k) and the results are drawn in terms of time complexity and memory utilization.

C16D200k dataset has two hundred thousand transactions with sixteen average numbers of items per sequence. The existing PrefixSpan algorithm is run by varying the value of minimum support from 0.5 to 0.7 while the maximum prefix length ranges from 1 to 5. The time complexity for 0.5 minimum support is very high than that of 0.6 and 0.7 minimum support. The memory utilization for all minimum support (0.5 to 0.7) does not shows that much variation and thus quite similar.

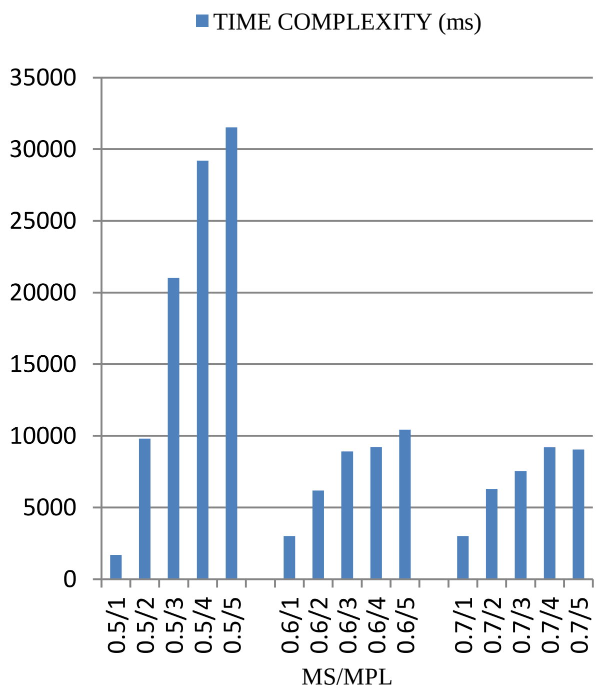

Fig 1: Time complexity Bar Graph for C16D200k

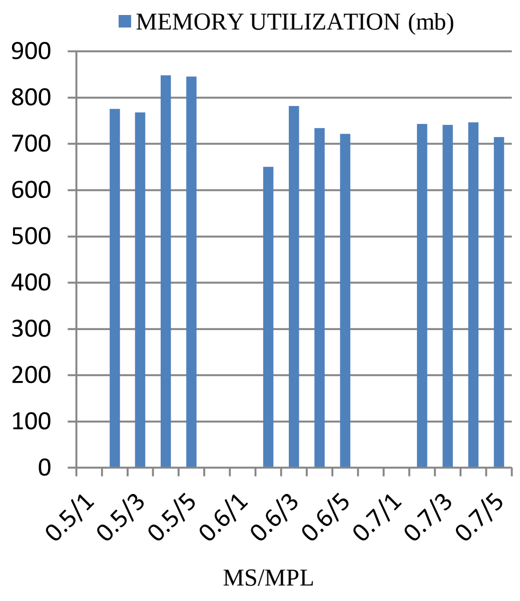

Fig 2: Memory Utilization Bar Graph for C16D200k

C16D100k dataset has one hundred thousand transactions. It is a sequential dataset that contains sixteen numbers of items per sequences. The algorithm is tested on C16D100k dataset by varying the minimum support and maximum prefix length values and the final results are drawn in terms of time complexity and memory utilization.

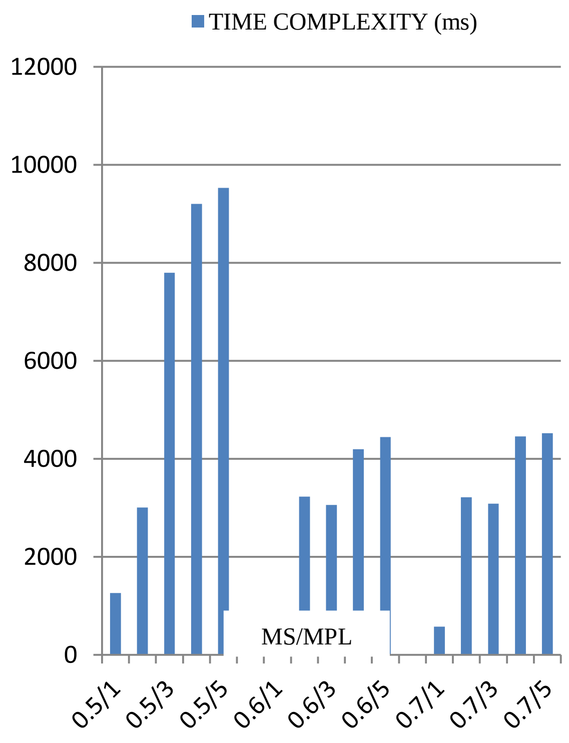

Fig 3: Time complexity Bar Graph for C16D100k

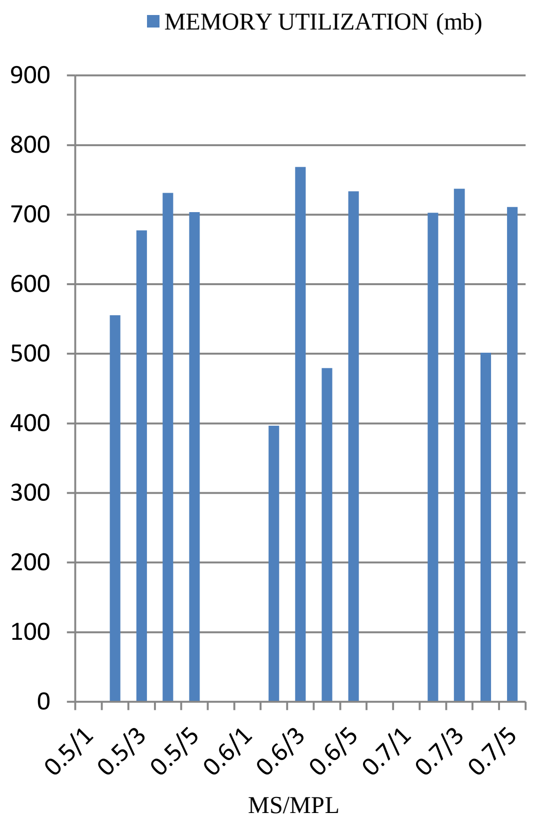

Fig 4: Memory Utilization Bar Graph for C16D100k

C21D36k dataset is conversion of conversion of bible into sequential dataset. It contains thirty six thousand sequential transaction with twenty one average number of item sets per sequence. The PrefixSpan algorithm is executed on C21D36k. The performance of existing PrefixSpan algorithm is evaluated in terms of time complexity and memory utilization by varying the minimum support and maximum prefix length values.

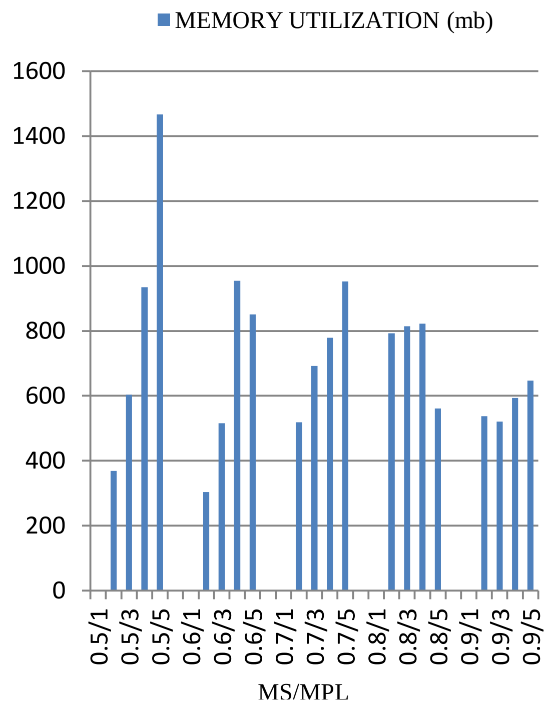

Fig 5: Memory Utilization Bar Graph for C21D36k [10]

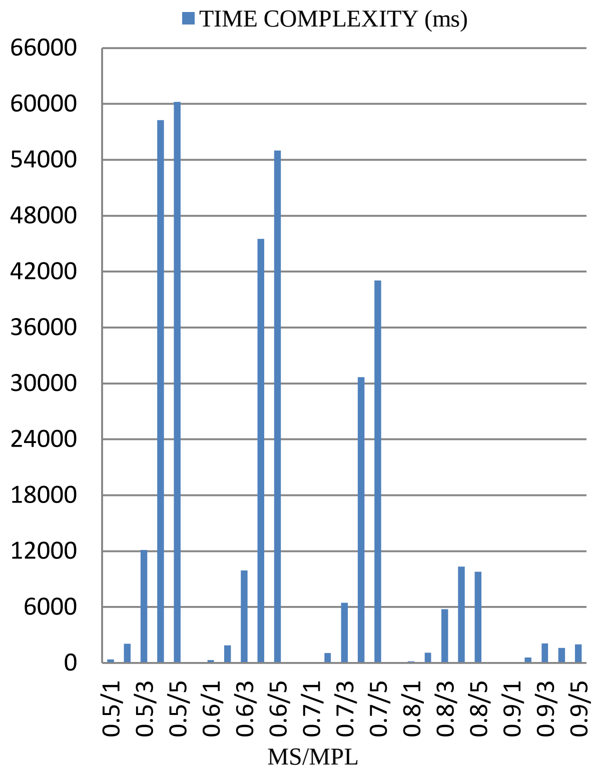

Fig 6: Time Complexity Bar Graph for C21D36k [10]

VI. IMPROVISATION IN EXSITING PREFIXSPAN

The existing PrefixSpan algorithm faces the problem of huge projected dataset cost which results into inefficient memory utilization. The time complexity is also high while running the algorithm on large datasets. In order to overcome these problems improvisation is done in existing PrefixSpan to process the large data and improvement is done through;

1) Parallel processing and
2) Compression Technique (GZIP)

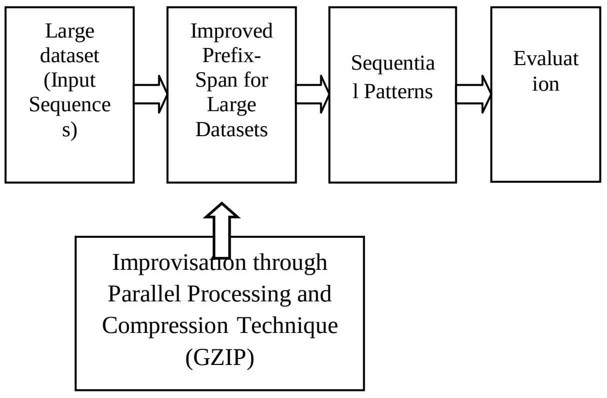
Large dataset (Input Sequences)
Improved Prefix-Span for Large Datasets
Sequential Patterns
Evaluation
Improvisation through Parallel Processing and Compression Technique (GZIP)

Fig 7: Improved PrefixSpan Architecture

VI.1 Parallel Processing

In order to achieve efficient and fast data processing for large datasets the parallelization comes into play. The parallel processing can be achieved through the simultaneous use of more than one CPU or processor core to execute a program or multiple computational threads. In proposed system the parallelization is done through multi-thread programming which enable to process the large data in time efficient manner.

The existing PrefixSpan algorithm first derives the projected dataset based on corresponding prefixes. Then various sequential patterns are derived based on projected dataset. At last the appropriate sequences are derived based on the threshold values (Minimum support and Maximum prefix length) which are provided at the start of the algorithm's execution.

In order to improved PrefixSpan these task are assigned to different threads. That means one thread will read all the data once, second thread will find the projected dataset, third tread will derive sequential patterns. Another improvisation is implementing compression which is also assigned to another thread. Through multiple threads there is no need to wait for completion of one process rather all processes can work parallel. This finally reduces the overall time complexity and thus improvised the existing PrefixSpan algorithm.

VI.2 Compression Technique (GZIP)

GZIP is based on the DEFLATE algorithm. It is a combination of LZ77 and Huffman coding. The LZ77 is used to eliminate the duplicate string while the Huffman coding deals with the bit reduction. GZIP first locates all similar strings within a given text file and then replaces those strings temporarily in order to reduce the overall size of the file. GZIP is a generic compressor which enables it to apply to any stream of bytes.

It also has feature to remember some of the previously seen content and it helps to find and replace duplicate data fragments in an efficient way. GZIP performs very well on text-based contents. It often achieves compression rates of as high as 70-90% for larger files. These features of GZIP are very much useful in order to improvise the existing PrefixSpan. As GZIP has such high compression rate on large datasets it ultimately enhances the memory utilization.

VII. RESULTS OF IMPROVISED PREFIXSPAN ALGORITHM

The existing PrefixSpan algorithm is improvised through parallel processing and compression which results into time and space efficient processing of large data. The improvised PrefixSpan is run on C16D200k, C16D100k and C21D36k datasets. The results show that the time complexity is minimized significantly while the memory utilization is also reduced.

For dataset C16D200k, the improvised PrefixSpan algorithm is executed by varying the values of minimum support (MS) and maximum prefix length (MPL). For existing PrefixSpan algorithm when run on C16D200k dataset, the time complexity varies from 1658 milliseconds to 10437 milliseconds.

But for improvised PrefixSpan algorithm, time complexity is greatly reduced and it varies from 263 milliseconds to 407 milliseconds. Memory utilization is increased in improvised PrefixSpan only for 0.5 minimum supports and when maximum prefix length is set to 1. For other minimum support values (0.6 and 0.7) the space utilization is reduced through GZIP compression technique.

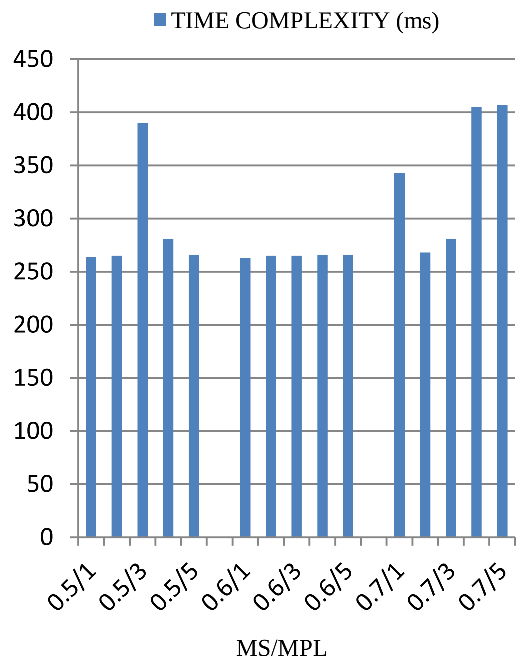

Fig 8: Time complexity Bar Graph for C16D200k of Improvised PrefixSpan

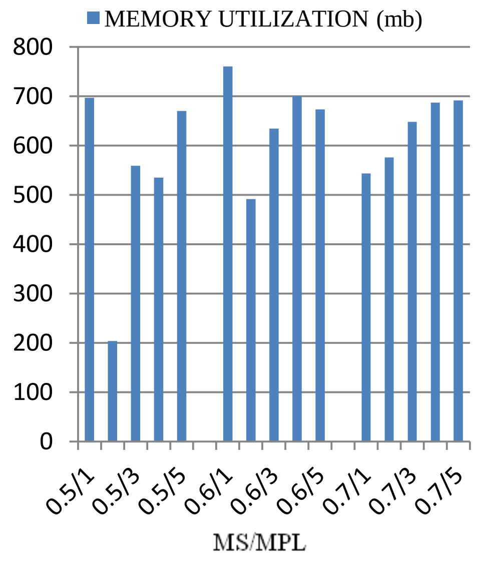

Fig 9: Memory Utilization Bar Graph for C16D200k of Improvised PrefixSpan

The improvised PrefixSpan algorithm is executed on dataset C16D100k by varying the values of minimum support (0.5 to 0.7) and maximum prefix length (1 to 5). When existing PrefixSpan algorithm is run on C16D100k dataset, the time complexity varies from 359 milliseconds to 9532 milliseconds.

After improvising PrefixSpan through parallel processing time complexity is significantly reduced and it varies from 265 milliseconds to 484 milliseconds. In case of memory utilization, it is increased in improvised PrefixSpan only for 0.5 minimum supports and for 1 maximum prefix length. For other minimum support values (0.6 and 0.7) the memory utilization is enhanced through GZIP compression.

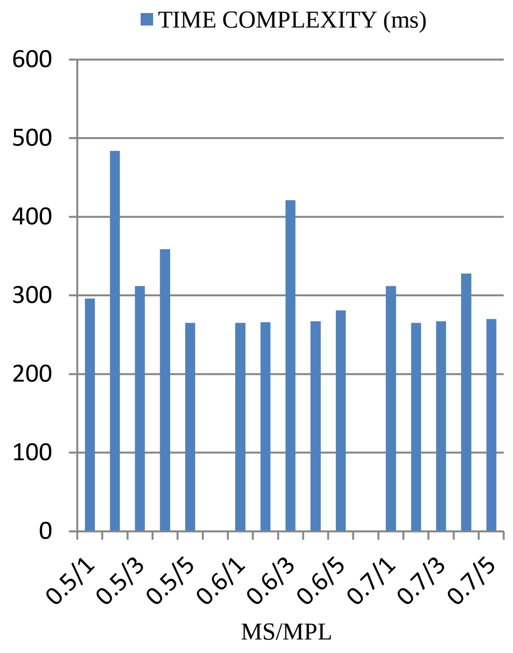

Fig 10: Time Complexity Bar Graph for C16D100k of Improvised PrefixSpan

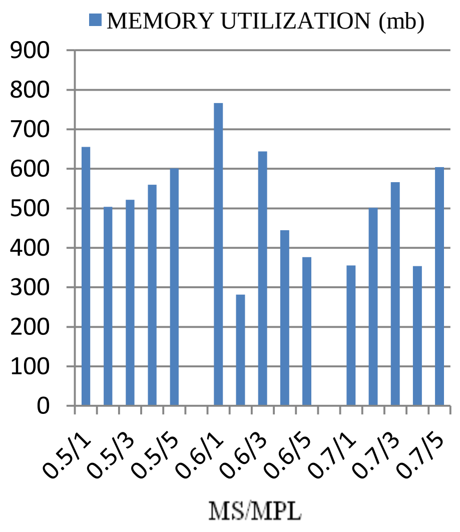

Fig 11: Memory Utilization Bar Graph for C16D100k of Improvised PrefixSpan

The improvised PrefixSpan algorithm is also tested on dataset C21D36k which is conversion of bible into sequence dataset. The values of minimum support (0.5 to 0.9) and maximum prefix length (1 to 5) are set to a threshold value and are changed regularly. When existing PrefixSpan algorithm is run on C21D36k dataset, the time complexity varies from 374 milliseconds to 363847 milliseconds.

When the existing PrefixSpan algorithm is improvised through multi-threads based parallel processing, time complexity is significantly reduced and it varies from 260 milliseconds to 375 milliseconds. In case of memory utilization, it is increased in improvised PrefixSpan only for 0.5 minimum supports and for 1 maximum prefix length. For other minimum support values (0.6 to 0.9) the efficient memory utilization is achieved through compression (GZIP).

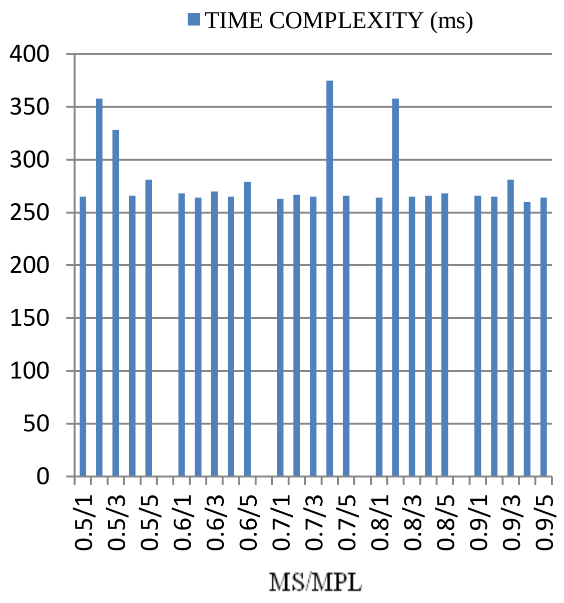

Fig 12: Time Complexity Bar Graph for C21D36k of Improvised PrefixSpan

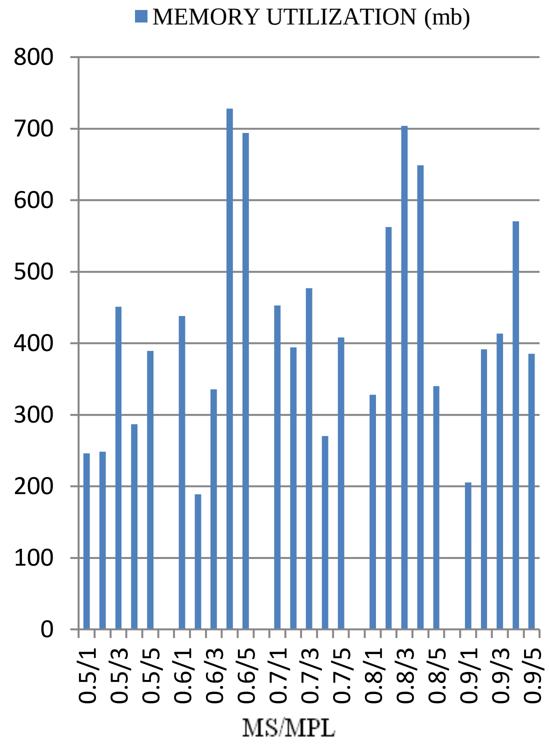

Fig 13: Memory Utilization Bar Graph for C21D36k of Improvised PrefixSpan

VIII. CONCLUSION

First the performance of existing PrefixSpan algorithm is evaluated by running the algorithm on different datasets (C16D200k, C16D100k and C21D36k). The two parameters minimum support and maximum prefix length are set at beginnig of the algorithm's execution. The sequences which having value more than minimum support (provided at start) are extracted from sequential datasets. When the pattern occurs in sequential dataset is divided by the total number of sequences in the database, the minimum support is calculated. As the PrefixSpan algorithm is run on large datasets, the maximum prefix pattern value plays important role to get sequential output. It is used to specify the length of the sequence to be there in output. For getting the sequential output based on minimum support and maximum prefix length,. The two parameters time complexity and memory utilization are set as the benchmark for performance evaluation when PrefixSpan algorithm is run on different datsets based on various values of minimum support and maximum prefix length. Both the parameters vary from one dataset to other. These results are plotted in bar graph and are useful in order to analyse the performance of existing algorithm.

When the existing PrefixSpan algorithm is run on C16D200k, C16D100k and C21D36k datasets, the time for finding sequential output is high as these datasets are large datasets. It ultimately increased the overall time complexity of algorithm. As the existing algorithm is tested on large datasets, it faces the problem of huge cost construction for projected datasets. It ultimately leads to inefficient memory utilisation.

In order to overcome these problems of existing PrefixSpan when tested on large datasets, the improvisation is done through parallel processing and compression technique (GZIP). The parallel processing is done through by assigning various processes to corresponding threads which termed as multi-thread programming. It helps to process the large data in time efficient manner which ultimately reduced the overall time complexity to 1/4th of the existing PrefixSpan algorithm.

The efficient memory utilisation is provided by implementing the compression technique GZIP which is termed as lossless compression. GZIP achieved efficient compression through Huffman Coding and LZ77. Huffman coding improvises the entropy encoding while LZ77 replaces the repeated occurrences of data with a single copy of the data existing earlier in the uncompressed data stream. Thus GZIP achieves compression rates of as high as 70-90% for larger files by combining features of Huffman coding and LZ77. This compression ultimately enhances the memory utilization and reduces the overhead construction cost of projected dataset.

IX. FUTURE SCOPE

The PrefixSpan algorithm can further be improved in order to process BIG sequential data. Other constraints can also be added in order to develop the algorithm for concrete application. As for concrete application only limited and specific sequential output will be needed. More precision can be introduced through additional parameters for final sequential outputs.

X. ACKNOWLEDGMENT

The authors are grateful to Dr. R. R. Sedamkar for his exemplary guidance, valuable feedback and constant encouragement for the research. His valuable suggestions were of immense help throughout the research work.

REFERENCES

[1] R Agrawal and R Srikant, 1995. Mining sequential patterns, In Proceedings of 1995 International Conference Data Engineering (ICDE'95), pp. 3- 14, Taipei, Taiwan.

[2] R Agrawal and R Srikant, 1994. Fast algorithms for mining association rules, In Proc. 1994 Int. Conf. Very Large Data Bases (VLDB'94), pp. 487- 499, Santiago, Chile.

[3] M. Zaki, 2001. SPADE: An Efficient Algorithm for Mining Frequent Sequences, Machine Learning, vol. 40, pp. 31- 60.

[4] Han J., Dong G., Mortazavi-Asl B., Chen Q., Dayal U., Hsu M.-C., 2000. Freespan: Frequent pattern-projected sequential pattern mining, In Proceedings 2000 Int. Conf. Knowledge Discovery and Data Mining (KDD'00), pp. 355-359. 2000.

[5] Jian Pei, Jiawei Han, Behzad Mortazavi, Umeshwar Dayal, 2004. Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach, IEEE transactions on knowledge and data engineering, Vol. 16, pp. 1424-1440.

[6] LIU Pei-yu, GONG Wei and JIA Xian, 2011. An Improved PrefixSpan Algorithm Research for Sequential Pattern Mining, In Proceedings 2011 International Symposium, pp. 377-380.

[7] Liang Dong and Wang Hong. 2014. A improved PrefixSpan Algorithm for Sequential Pattern Mining, In Proc. 2014 IEEE International Conference, Vol. 1, pp. 103-108.

[8] Zhou Zhao, Da Yan and Wilfred Ng. 2014. Mining Probabilistically Frequent Sequential Patterns in Large Uncertain Databases, IEEE transactions on knowledge and data engineering, Vol. 26, pp. 1171-1184.

[9] J. Pei, J. Han and W. Wang, 2007. Constraint-based sequential pattern mining: the pattern growth methods, J Intell. Inf. Syst, Vol. 28, No.2, pp. 133 –160.

[10] Pratik Saraf, R. R. Sedamkar and Sheetal Rathi 2015. PrefixSpan Algorithm for Finding Sequential Pattern with Various Constraints, International Journal of Applied Information Systems, Vol. 9, pp. 37- 41.