ISSN 2320-9984

Volume 3 Issue 1 | March 2015

ISSN: 2320-9984 (Online)

International Journal of Modern Engineering & Management Research

Website: www.ijmemr.org

Optimisation for Sequential Pattern Mining in Progressive Database

Preeti Bharatrao Gawande
M.E. Research Scholar
Department of Computer Science Engineering,
Prof. Ram Meghe Institute of Technology & Research,
Badnera, Amravati (M.H.) [INDIA]
Email: p6gawande@gmail.com

Prof. Nitesh M. Tarbani
Assistant Professor
Department of Computer Science Engineering,
Prof. Ram Meghe Institute of Technology & Research,
Badnera, Amravati (M.H.) [INDIA].
Email: nmtarbani@mitra.ac.in

Abstract*—Data Mining is an analytic process designed to explore data (usually large amounts of data - typically business or market related - also known as "big data") in search of consistent patterns and/or systematic relationships between variables, and then to validate the findings by applying the detected patterns to new subsets of data.*

Most large retail organizations uses database mining, faces the problem of decision support. Development of bar-code technology has made able retail organizations to collect and store massive amounts of sales data. A record in such data typically consists of the transaction date and the items bought in the transaction. A sequence database consists of sequences of ordered elements or events, recorded with or without a concrete notion of time. There are many applications involving sequence data.

In this paper we have presented an application of data mining on grocery shop database. This warehoused database is mined using PISA algorithms whose results can be utilized for decision making and comparing the performance of this algorithm with other algorithms like GSP+, SPAM+, and DirApp. Programs are coded in C++.

1. INTRODUCTION

Database mining is a decision support problem. Data mining is the process of extracting interesting patterns from huge data such as relational database, data warehouse etc. Data mining is becoming an increasingly important tool to transform these data into information. Data mining is often carried out only on samples of data. The mining process will be ineffective if the samples are not a good representation of the larger body of data.

A sequence database consists of sequences of ordered elements or events, recorded with or without a concrete notion of time[1]. Typical examples include customer shopping sequences, Web click streams, biological sequences, sequences of events in science and engineering, and in natural and social developments.

1.1 Sequential Pattern Mining

Sequence Pattern Miningis the mining of frequently occurring ordered events or subsequences as patterns [9]. Also telecommunications and other businesses may use sequential patterns for targeted marketing, customer retention and many other tasks. Other areas in which sequential patterns can be applied include Web access pattern analysis, weather prediction, production processes, and network intrusion detection analysis. Most studies of sequential pattern mining concentrate on categorical patterns [9].

The sequential pattern mining problem was first introduced by Agrawal and Srikant in 1995[1] based on their study of customer purchase sequences, as follows: “Given a set of sequences, where each sequence consist of a list of events (or element) and each event consists of set of items, and given a user specified minimum support threshold of min_sup, sequential pattern mining finds all the frequent subsequences, that is, the subsequences whose occurrence frequency in the set of sequences is no less than min_sup."

1.2 Progressive Database

There have been many recent studies on the mining of sequential patterns in a static database, which do not fully explore the effect of deleting old data from the sequences in the database. When sequential patterns are generated, the newly arriving patterns may not be identified as frequent sequential patterns due to the existence of old data and sequences [7]. Even, the obsolete sequential patterns that are not frequent recently may stay in the reported results. Generally, users are more interested in the recent data than the old ones. To capture the dynamic nature of data addition and deletion, we propose a general model of sequential pattern mining with a progressive database while the data in the database may be static, inserted, or deleted.

1.3 Progressive Sequential Pattern Mining

The incremental mining algorithms do not consider the deletion of the obsolete data from the sequence database. Thus it is not applicable to a progressive database. However, if a certain sequence does not have any newly arriving elements, this sequence will still stay in the database and undesirably contribute to the number of sequences in the sequence database. Therefore, when new sequential patterns are generated, the new patterns which appear frequently in the recent sequences may not be considered as frequent sequential patterns because number of sequences in the sequence database is never reduced. In view of this, the infrequent sequential patterns whose timestamps are obsolete should be removed [7].

Here we are using an algorithm PISA[2], which stands for Progressive mIning of Sequential pAtterns, corresponding to the mining in a progressive database. PISA takes the concept of period of interest (POI) into consideration. POI is a sliding window, whose length is a user specified time interval, continuously advancing as the time goes by. The sequences having elements whose timestamps fall into this POI, contribute to the number of sequences in the sequential database for current sequential patterns. On the other hand, the sequences having only elements with timestamps older than POI should be pruned away from the sequence database immediately and will not contribute to the sequence thereafter [7].

2. PISA ALGORITHM

The main concept of PISA is to progressively update the information of each sequence and each candidate sequential pattern in the database. Using PS Tree it stores all sequences from one POI to another. PS-Tree is the core part of the algorithm PISA. It contains the information of all sequences in a progressive database and helps PISA to generate frequent sequential patterns in each POI. There are two types of nodes in the PS-Tree: root node and common node. Root node, contains a list of common nodes as its children. Each common node stores its node label (element of the sequence) and a sequence list (list of sequence IDs to represent the sequences containing this element). Each sequence ID in the sequence list is marked by a corresponding timestamp.

Whenever there are a series of elements appearing in the same sequence, there will be a series of nodes labeled by each element, respectively, with the same sequence IDs in their sequence lists. Then, the first node will be connected to the root node and the second node representing the following element will be connected to the first node. The other nodes will be connected analogously. In such a way, the path from root node to any other node will represent the candidate sequential pattern appearing in this sequence. The appearing timestamp for each candidate sequential pattern will be marked in the node labeled by the last element. If there is another sequence having the same pattern, the sequence ID will be inserted into the sequence lists of the same node labeled by these elements on the path. On the other hand, if an element appearing in a sequence is obsolete, the corresponding sequence ID will be removed from the sequence list of the node. In addition, if a node has no sequence in the sequence list, it will be pruned away from PS-tree. Thus, there are only up-to-date candidate sequential patterns available in PS-tree.

The main concept of PISA is to progressively update the information of each sequence and each candidate sequential pattern in the database. Using PS Tree it stores all sequences from one POI to another. PS-Tree is the core part of the algorithm PISA. It contains the information of all sequences in a progressive database and helps PISA to generate frequent sequential patterns in each POI. There are two types of nodes in the PS-Tree: root node and common node. Root node, contains a list of common nodes as its children. Each common node stores its node label (element of the sequence) and a sequence list (list of sequence IDs to represent the sequences containing this element). Each sequence ID in the sequence list is marked by a corresponding timestamp.

Whenever there are a series of elements appearing in the same sequence, there will be a series of nodes labeled by each element, respectively, with the same sequence IDs in their sequence lists. Then, the first node will be connected to the root node and the second node representing the following element will be connected to the first node. The other nodes will be connected analogously. In such a way, the path from root node to any other node will represent the candidate sequential pattern appearing in this sequence. The appearing timestamp for each candidate sequential pattern will be marked in the node labeled by the last element. If there is another sequence having the same pattern, the sequence ID will be inserted into the sequence lists of the same node labeled by these elements on the path. On the other hand, if an element appearing in a sequence is obsolete, the corresponding sequence ID will be removed from the sequence list of the node. In addition, if a node has no sequence in the sequence list, it will be pruned away from PS-tree. Thus, there are only up-to-date candidate sequential patterns available in PS-tree.

3. COMPARISON OF THE PERFORMANCE OF THE ALGORITHMS

In this section, we conduct several experiments to evaluate the performance of the proposed algorithm and the effects of input parameters. The only existing work that can deal with the progressive database still applies static mining algorithm to remind each sub database. We implement the simple conceivable algorithm, DirApp, as well. GSP+, SPAM+, and PISA are all coded in C++, and the experiments are executed on a computer with Pentium 4, 3-GHz CPU and 2-Gbyte RAM. First, we will describe the method to generate the synthetic data sets. Then, we show the performance improvement of PISA over GSP+, SPAM+, and DirApp. The execution time of fast version of PISA are also included. To give more insights into the proposed algorithm, we will investigate the effects of some parameters. To investigate the searching space of DirApp and PISA, we calculate the maximum memory usage of each algorithm. We will show the trend of memory usage.

3.1 Experiment Design

The synthetic data sets are generated in a way similar to the IBM data generator designed for testing sequential pattern mining algorithms. Several parameters can be assigned to produce different synthetic data sets. We use Netflix Prize data as the testing workload, which contains 17,770 different items and 480,169 users. The previous works about incremental sequential pattern mining append the newly arriving elements of all sequences directly to the end of the original sequence database. They do not concern themselves with the POI, but instead, take the whole database of all elements into consideration. In our work, the obsolete elements which exceed the POI will be deleted from the sequence database. For this reason, each element should be designated an arriving timestamp. The items in this interval are combined as an element at a timestamp. Thus we transform the format of the generated data sets. First, we divide the target data set into n timestamps. According to the input parameter POI, the first m timestamps are viewed as the original database and the rest of elements in the data set are received by the system incrementally.

The length of POI is inevitably smaller than n, and the overall timestamps must be longer than the maximum number of elements that one sequence produces. The first run of the experiments mines the first POI from the beginning m timestamps of the data set ðm ¼ POI Þ. After that, we shift the POI one timestamp forward for the following runs. In this way, the elements in the up-to-date timestamp stand for the incremental part of the sequence database, and the obsolete elements are deleted. As for GSP+ and SPAM+, we retrieve the elements of corresponding m timestamps in the data set in each run and feed them into the system. Then, GSP+ and SPAM+ remine the input elements of m timestamps in that iteration. Because the remining process takes excessive time, we move POI t timestamps ðt mÞ forward instead of shifting POI only one timestamp forward in every run for better execution time of GSP+ and SPAM+. Therefore, the sequential patterns in the skipped POIs cannot be generated by GSP+ and SPAM+. In contrast, while performing even more efficiently, algorithms DirApp and PISA are still able to produce sequential patterns for every POI when processing newly arriving elements with multiple timestamps at the same time, showing the progressive advantage over the competitors. The experiment shows the cumulative time of continuous runs of the algorithms. Except the experiment, every point on the figure in the other experiments has the total execution time of 16 runs. That is, there is a total of 40 timestamps and POI is set as 10. In addition to the first run of the first POI, there are other runs of incremental sub databases (two timestamps forward) fed into the system contributing the reported time of every point. If there is no specific description, the minimum support threshold is set to 0.02 and the number of different items is set to 1,000. Because there is no special trend of execution time or memory usage on the number of different items, we do not include the result of this experiment in this paper. As for real data sets, we randomly choose successive 120 days for the performance evaluation. A timestamp is set as 3 days in order to obtain sufficient frequent sequential patterns. Therefore, there is a total of 40 timestamps and POI is set as 10, which meets the same environment as synthetic data sets. In this way, the new data sets contain more than 5,000 sequences and 2,000 different items.

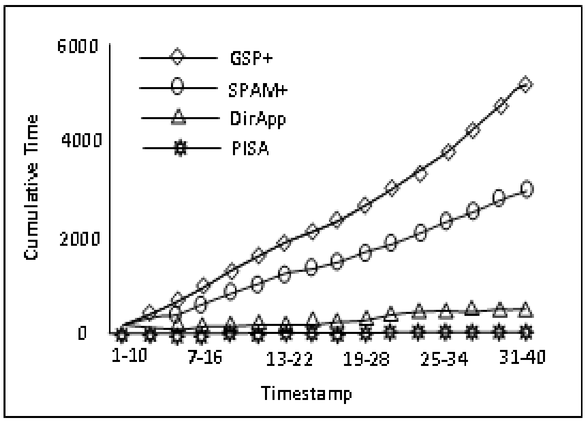


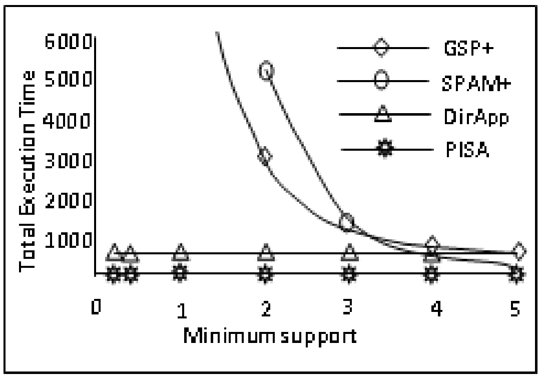
Figure 1 : Cumulative execution time.

3.2 Cumulative Execution Time

The algorithm PISA is a progressive algorithm to handle the situation that the POI advances over time. Figure 1 shows the superiority in terms of cumulative execution time of PISA over GSP+, SPAM+, and DirApp. We record the execution time of all algorithms at each timestamp from the beginning to the end. Then, we show the accumulated time from the beginning timestamp to the current one as cumulative execution time in Figure 1. The last point of each algorithm represents the total execution time needed by the algorithm for processing the whole data set. We see that as the first point of each series shown in Figure 1, PISA consumes shorter execution time than the competitors in each single run. This is because only one scan of the PS tree is needed by Pisa, which combines the sequences having the same pattern together. However, GSP+ has to scan sub database multiple times to check the occurrence frequencies of candidate sequential patterns. SPAM+ has to scan a big lexicographic sequence tree of all the items and the candidate sequential patterns. As for DirApp, the candidate sets of all sequences involve a lot of computation time. Furthermore, as the POI advances over time, PISA and DirApp need to process only new elements that lie at arriving timestamps, but GSP+ and SPAM+ have to rerun the mining process on the whole sub database. Therefore, the cumulative execution time of PISA and DirApp shows more superiority against GSP+ and SPAM+. But, with the help of PS-tree, PISA is more efficient than DirApp while they update the sequences by more than 10 times.

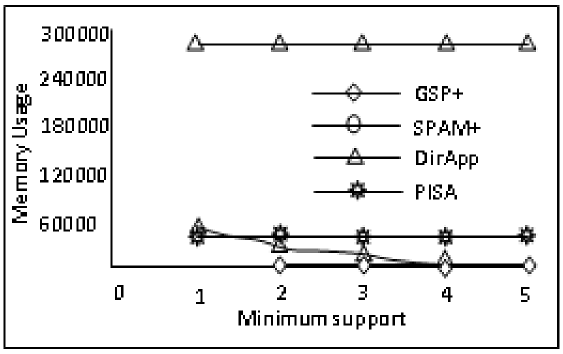
3.3 The Effects of the Input Parameters

In the following experiments, we examine the effects of the input parameters, the minimum support, and the POI. Figure 2a shows the total execution time of all algorithms over different minimum support values. The execution time of GSP+ and SPAM+ grows as the minimum support value reduces. When the minimum support is set to 2 percent, the execution time of GSP+ is more than 106 times of PISA and the execution time of SPAM+ is more than 61 times of PISA. The reason is that as the minimum support value diminishes, the number of candidate sequential patterns generated by GSP+ and SPAM+ increases considerably. Thus, the scanning time of all candidate sequential patterns needed by GSP+ and SPAM+ increases incredibly. On the other hand, the execution time of PISA and DirApp over different minimum support values remains the same.



(a)

Figure 2. (a) Total execution time.



(b)

Figure 2. (b) Memory usage with minimum supports varied.

It is because they generate the same number of candidate sequential pattern irrespective of the minimum support value. Therefore, the processing time of all candidate sequential patterns is the same over different minimum support values.

4. CONCLUSION

The sequential patterns generated by a progressive database over time should be updated accordingly [3]. This project aimed at maintaining the latest sequences, finding the up -to-date sequential patterns and delete obsolete patterns on the fly. The proposed algorithm **PISA** efficiently handles the problem of sequential pattern mining over *progressive data*. It is seen that PISA requires a single scan of the PS-tree, which is a candidate structure for this algorithm. This generalized model was further extended to accept new data and generate the modified PS-tree dynamically. PISA was then modified to generate *weighted* *patterns* depending on the timestamps of occurrence of individual items in the pattern. The enhanced version of PISA now provides a certain amount of *data hiding*. In this, the actual data is hidden from the user by adding certain *spurious values*, such that they do not affect the original outcome of the algorithm.

Thus PISA is a generalized model for sequential pattern mining which works on categorical data. It can further be extended to discover correlations and trends in numerical data in order to widen its scope of application.

REFERENCES:

[1] Agrawal, R. and Srikant, R. Mining sequential patterns. In Eleventh International Conference on Data Engineering, P. S. Yu and A. S. P. Chen, Eds. IEEE Computer Society Press, Taipei, Taiwan, pp. 3-14, 1995.

[2] S. Parthasarathy, M.J. Zaki, M. Ogihara, and S. Dwarkadas, "Incremental and Interactive Sequence Mining," Proc. 8th ACM Int'l Conf. Information and Knowledge Management (CIKM '99), pp. 251-258, 1999.

[3] F. Masseglia, P. Poncelet, and M. Teisseire, "Incremental Mining of Sequential Patterns in Large Databases," *Data and Knowledge Eng*., vol. 46, pp. 97-121, 2003.

[4] A. Balachandran, G.M. Volker, P. Bahl, and P.V. Rangan, "Characterizing User Behavior and Network Performance in a Public Wireless LAN," Proc. ACM SIGMETRICS Int'l Conf. Measurement and Modeling of Computer Systems (SIGMETRICS '02), pp.195- 205, June 2002.

[5] http://en.wikipedia.org/wiki/Java (programming language)

[6] J. Han and M. Kamber. "Data Mining: Concepts and Techniques". Second Edition, 2006.

[7] Jen-Wei Huang, Chi-Yao Tseng, Jian -Chih Ou, and Ming-Syan Chen. "A General Model for Sequential Pattern Mining with a Progressive Database", Knowledge And Data Engineering, vol. 20, no. 9, pp. 1153-1167, September 2008.