Efficient Identification of Common Subsequences from Big Data Streams Using Sliding Window Technique

Adi Alhudhaif
Department of Computer Science, The George Washington University, Washington, DC. 20052, USA
Department of Computer Science, Salman Bin Abdulaziz University,
Al Kharj, The Kingdom of Saudi Arabia

***Abstract*—We propose an efficient Frequent Sequence Stream algorithm for identifying the top k most frequent subsequences over big data streams. Our Sequence Stream algorithm gains its efficiency by its time complexity of linear time and very limited space complexity. With a pre-specified subsequence window size S and the k value, in very high probabilities, the Sequence Stream algorithm retrieve the top k most frequent subsequences of size S. The Stream Sequence algorithm also provides a high accuracy of the estimation of the number of occurrences of each promoted subsequence. Our experiments indicate several factors that influence the result accuracy of the Sequence Stream algorithm: stream size, subsequence size S and frequency of the subsequence.**

Keywords—Frequent subsequence; Stream processing; Periodic pattern; Pattern recognition; Big data processing

I. Introduction

Due to the new data collection methods, vast amount of data is produced [1]. This data-increasing trend is associated with business needs, geographical research works, social media networks and etc. and these result in “Big Data”. Big Data situation relates to the problem of dealing with very large amounts of data [2]. It presents a qualitatively different state of affairs for the organization of information processing, namely, this organization cannot utilize all the data explicitly. Data processing is one of the important challenges and many studies have been made on this topic [3]. Big Data support to build several worldwide social network connections, which integrate human beings with the accelerated progress of communication. Because of big data, entrepreneurs could make wise decision based on consumers’ behaviors. Recently, the use of big data has a key role in improving competitiveness in all kinds of fields. The big data stream contains very large amounts of information. The stream data processing is to understand data and to retrieve useful information from it. Various methods are designed to deal with big data [4][5][6]. The challenges include accuracy, efficiency and availability.

Frequent sequence mining finds sets of data elements that occur together frequently in many subsequences. Frequent sequence mining, which retrieve the most frequent subsequences from a stream of a very large sequence. It gained a great deal of attention in the field of data mining due to its great value in many applications, such as: trend prediction, stock market, DNA sequence analysis (Bioinformatics), using history of side affects or symptoms to predict valuable medical information, web user analysis, finding language or linguistic sequences from natural language text.

In this paper, we introduce a novel technique for mining the top k frequent subsequences over large stream of big data with a pre-defined subsequence size S, in the fashion of stream processing. The algorithm provides very high probabilities for retrieving the most frequent subsequences in leaner time using very limited space and memory locations.

II. Frequent Sequence Mining in Stream Processing

Finding most frequent sequences is considered as one of the most heavily studied data mining task since its introduction in work [7] and is of wide scientific interest [8][9][10][11]. Subsequences are valuable kind of data that occur more often in domains such as: information security, artificial intelligent, machine learning, education, medical, financial and many other fields. As for medical field, extracting frequent subsequences from very large DNA sequences is a key step for understanding biological processes as basic as the RNA transcription [12].

Stream processing uses different methods comparing to traditional datasets computing, it requires relatively smaller respond time with dealing huge amount of data. In computer science, the streaming algorithms are designing for processing data streams in the way of limited time and limited memory. It was first introduced in 1999 [13] [14], and then spread to all kinds of angles in computer science, such as database, networking and machine learning. Now the big data society comes to study stream algorithms when large amounts of data can be operated continuously regardless of storage and access distribution, meanwhile respond quickly to new information. In reality, stock market data is a typical stream data. The data contains real-time price, transaction and other financial information. Traders usually receive and analyze data streams to make decisions by advanced systems.

We focus on the process of massive stream by optimal processing algorithm to extract meaningful value from large sequence of big data. This is done by retrieving (on-the-fly [15][16]) the most frequent subsequences over large stream of big data with the concerns of time-consumption and space-consumption.

III. Frequent Sequence Stream Algorithm

The Frequent Sequence Stream algorithm (FSS), was inspired during the development of Multi-Buffer based algorithm in work [5]. Multi-Buffer based algorithm was proposed to extract the top k most frequent elements over large stream of Big Data. FSS works in window sliding technique and window size is a pre-defined value of S. In addition, FSS holds multi sequence candidates (SeqCan) that hold common sequences of size S. For a Big data stream $u = u_1,...,u_n \in \Sigma^*$ we consider all the subsequences $u_i u_{i+1} ... u_{i+S-1} \sqsubseteq u$, where $1 \leqslant S \leqslant n$ and $1 \leqslant i_1 < i_2 < \cdots < i_n \leqslant n$, and goal is to find the top k most frequent sequences of size S and an approximate counter that reflects each subsequence occurrence.

Using k Sequence candidates, the FSS algorithm can be stated as following: store the first new arrival of sequence ($u_i u_{i+1} ... u_{i+S-1}$) to SeqCan#1 and set Weight (w) to β and Counter (c) to 1. Keep comparing the incoming new sequence with the previously stored sequences candidates (SeqCan). If the new incoming new sequence equals to one of the sequence candidates in (SeqCan), increase its associated counter by 1 and increase its weight by β.

Otherwise, assign this new subsequence to any sequence candidates that has an associated weight equal to zero and set that weight to β and its counter to 1. At the case of no weight equals to zero, decrease the weight (w) with minimum value by 1. By the end of this stream, output stored sequences (Candidates) and their associated counters (c).

For example; when k = 3 the algorithm FSS can be described as follows:

```
Repeat
    Get next sequence using sliding window of size S
    Seq = $u_i u_{i+1} ... u_{i+S-1}$
    if ( w1 ≠ 0 and SeqCan1 = Seq ):
        w1=w1+ β, c1=c1+1
    Else_if ( w2 ≠ 0 and SeqCan2 = Seq):
        w2=w2+ β, c2=c2+1
    Else_if ( w3 ≠ 0 and SeqCan3 = Seq):
        w3=w3+ β, c3=c3+1
    Else _if (SeqCan1 ≠ Seq and w1 ≠ 0) and (SeqCan2 ≠ Seq
    and w2 ≠ 0) and (SeqCan3 ≠ Seq and w3 ≠ 0):
        Minimum [w1,w2,w3] =  Minimum[w1,w2,w3] - 1
    Else_if (w1 = 0 ):
        w1 = β , c1 = 1 , CanSeq1 = Seq
    Else_if (w2 = 0 ):
        w2 = β , c2 = 1 , CanSeq2 = Seq
    Else_if (w1 = 0 ):
        w3 = β , c3 = 1 , CanSeq3 = Seq
    Move window by 1( i = i + 1).
Until no more sequences
```

Moreover, the output of the FSS algorithm will be k pairs (candidate, counter). The focus of this algorithm is to improve the probability that one of the k pairs contains the most frequent sequence of size S, and enhance the accuracy of estimating its frequency. The FSS algorithm is able to select up to k – 1 top frequent sequences in the data stream. For example, when k = 3 and an input of random sequences with two top occurrence of frequency 12% and 15%, they would be selected efficiently by using three sequence candidates (SeqCan) or more.

IV. Experimints

For every single stream file with determined sequence frequncy we generated many iterations using The Fisher-Yates shuffle algorithm [17][18]. Generating pseudo-random numbers was done using both generator functions in Python's library Lib/random.py and the random number library in C that takes variable seeds such as: current system time to generate pseudo-random numbers. Then, according to the most frequent frequence.

We performed and examined Frequent Sequence Stream (FSS) algorithm using the big data stream under a common implementation framework to test their performance as accurately as possible. The algorithm was implemented using both C and Python, and compiled using gcc on Cygwin 1.7.25 for C code, and Python 2.7.5 for python code. We ran Python experiments on 2.6GHz dual-core Intel Core i5 with 8GB of RAM running OS X 10.9.2. Experiments of algorithms in C were ran on Intel 4th generation core i5 using 8GB of RAM running Microsoft Windows Server 2012. We did not observe notewothy differences between two compilers.

A. The Calculation of Sequence Frequency

In big data stream of size *n*, and a pre-defined subsequence size S. A subsequence X has a frequency of 100% when the number of occurrence of sequence X is $\lfloor n/S \rfloor$. For example: stream of size 100,000 elements and a subsequence size of 7, the subsequence X has a frequency of 15% when it occurs 2,142 times ($\lfloor (n/S)* 0.15 \rfloor$).

B. Results

Using stream sizes 30,000, 100,000 and 1,000,000 Fig. 1 shows the probabilities of retrieving the most frequent subsequences of size 3, with low frequencies: 5%, 4%, 3%, 2% and 1%.

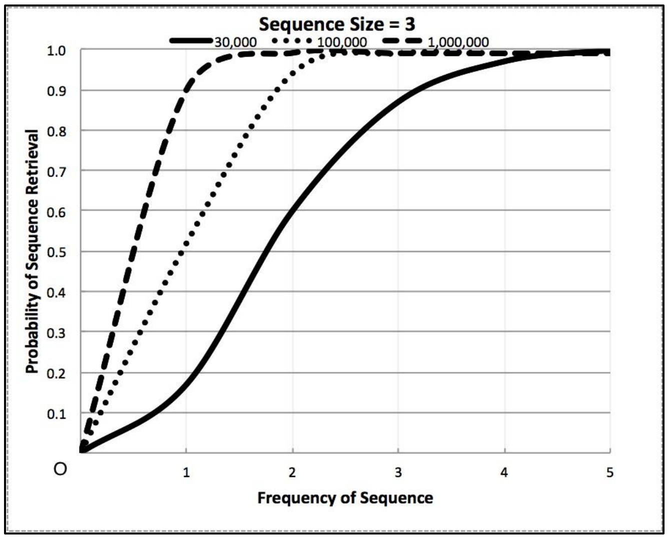

Fig. 1. Probabilities of extracting most frequent subsequences of size 3.

Fig. 2 represents the probabilities for retrieving most frequent subsequences of size 7 with low frequencies using various stream sizes.

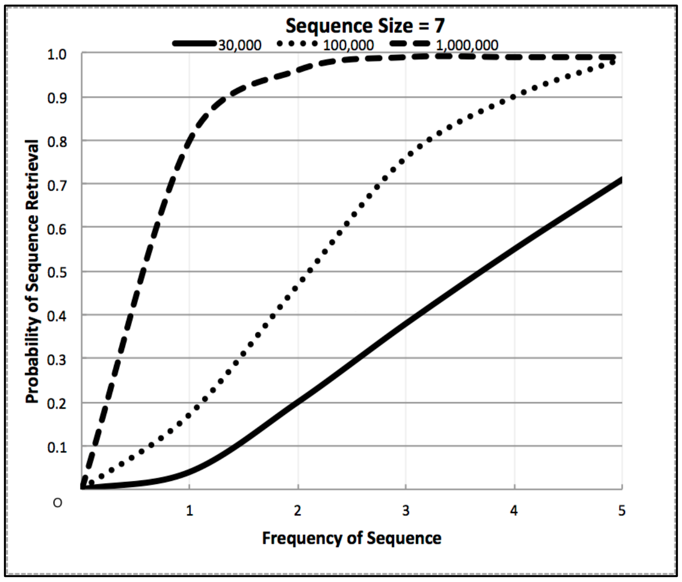

Fig. 2. Probabilities of extracting subsequences of size 7 using various stream sizes

Moreover, in Fig. 3 we increased the subsequence size to be matched to 15 using various stream sizes: 30,000, 100,000 and 1,000,000.

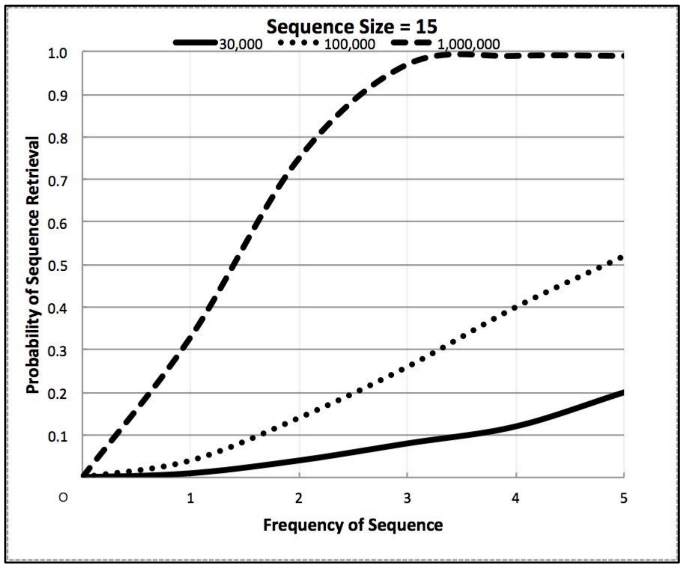

Fig. 3. Probabilities for extracting subsequences of size 15 within various stream sizes.

For larger subsequence frequencies like 10%, Fig. 4 shows the probabilities for retrieving subsequences of various sizes versus various stream sizes.

Counters that represent the number of a subsequence occurrence are part of the FSS algorithm. For a stream size of 100,000, and subsequences of low frequencies, Fig. 5 shows the accuracy of counter values returned by the FSS algorithm compared to the real number of occurrences within the big data stream.

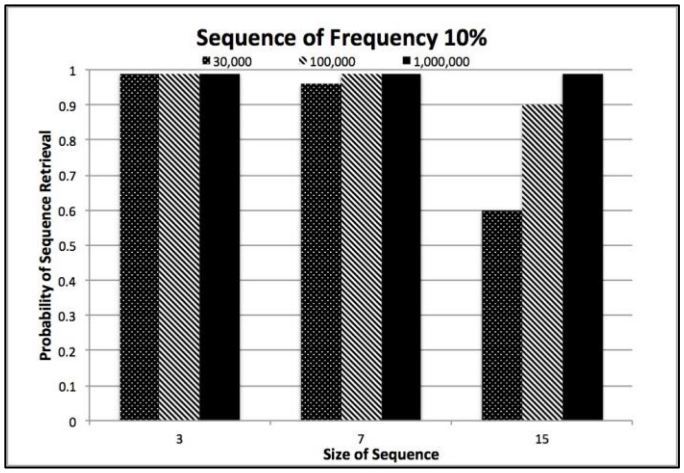

Fig. 4. Probabilities for retrieving subsequences of various stream and subsequences sizes

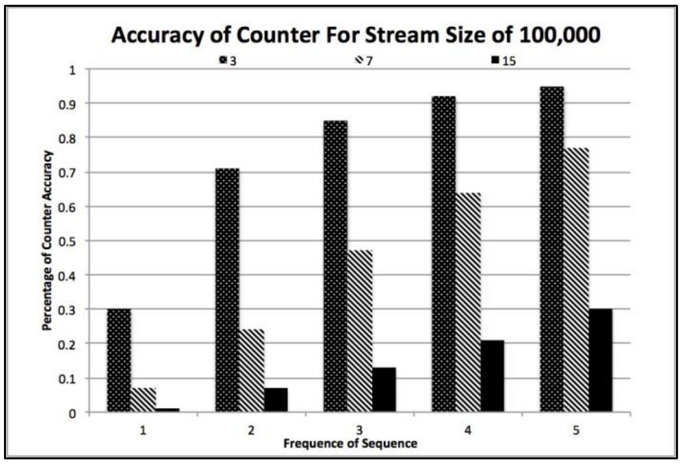

Fig. 5. Accuracy of counter values using various subsequences sizes and stream size of 100,000.

For a stream size of 1,000,000 and subsequences of low frequencies, Fig. 6 shows the accuracy of counter values returned by the FSS algorithm compared to the real number of occurrences.

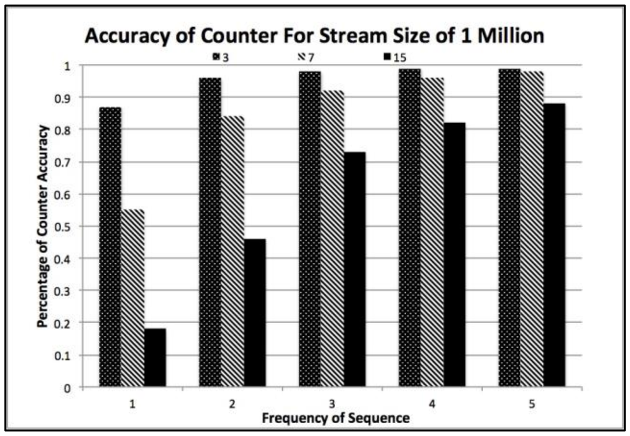

Fig. 6. Accuracy of counter values using various subsequences sizes and stream size of 100,000.

V. COMMENTS/FUTURE RESEARCH

During experiments many values for β were tested and verified. The best value for β turned out to be S^2. As for Multi-Buffer based algorithm in work [5], stream sizes has an impact on the performance of the FSS algorithm as shown in Fig. 1, Fig. 2 and Fig. 3. The bigger the stream size the better the results.

Moreover, we observed that the subsequence sizes have an influence on the output results of the FSS algorithm. The smaller subsequences' sizes to be processed, the better accuracy of the results to be promoted. To explain this, for a stream size of 100,000, 1% frequency of a subsequence size 3 is 333 times, 1% frequency of a subsequence size 7 is 142 times, and a 1% frequency of subsequence size 15 is 66 times.

By tracking changes of these sequence candidates and their associated weights, we find that the entire process can be divided into two important stages: stable stage and unstable stage explained in work [5]. For subsequences of high frequencies such as %10 and more, probabilities for retrieving those frequencies are very high compared to low frequencies.

As for counter values, Fig. 5 and Fig. 6 show that the stream size factor influence the accuracy of the counter value returned by the FSS algorithm. Counter accuracy increases when the stream size increases. In addition, frequencies of subsequence also impact the accuracy of the returned counter vales.

Above experiments were performed using three sequence candidates (k = 3), we observed a minor enhancement when using more sequences candidates. Moreover, more factors (range of data, number of candidates, number of frequent subsequences) could impact the accuracy of results but this needs to be verified by more experiments. Also, the optimal value of β, its relationship with subsequence's size and subsequence's frequency worth more investigations, and whither has β and number of top frequent subsequences are related.

ACKNOWLEDGMENT

Many thanks for Professor Simon Y Berkovich of Computer Science Department of George Washington University, for his guidance and support.

REFERENCES

[1] Manyika J, Chui M, Brown B, et al. Big data: The next frontier for innovation, competition, and productivity. 2011.

[2] Simon Berkovich, "Physical World as an Internet of Things" COM.Geo '11: Proceedings of the 2nd International Conference on Computing for Geospatial Research & Applications, May 2011, p.66

[3] Berkovich, S., Liao, D.: On Clusterization of Big Data Streams. In: 3rd International Conferenceon Computing for Geospatial Research and Applications, article no. 26. ACM Press,New York (2012)

[4] Zikopoulos P, Eaton C. Understanding big data: Analytics for enterprise class hadoop and streaming data. McGraw-Hill Osborne Media, 2011.

[5] Adi Alhudhaif, Tong Yan and Simon Berkovich. "On the organization of cluster voting with massive distributed streams", in Proceedings of the 5th international Conference on Computing for Geospatial Research & Application, Washington, D.C., 2014.COM.Geo

[6] Adi Alhudhaif, Tong Yan and Simon Berkovich, "A cyber-physical algorithm for selecting a prevalent element from big data streams", GSTF Journal on Computing (JoC) Vol 4 No 1.

[7] R. Agrawal and R. Srikant. Mining sequential patterns. In ICDE'95, pages 3–14.

[8] Eric P. Xing, Michael I. Jordan, Richard M. Karp, and Stuart Russell. A hierarchical Bayesian Markovian model for motifs in biopolymer sequences. In In Proc. of Advances in Neural Information Processing Systems, pages 200–3. MIT Press, 2003

[9] Pavel A. Pevzner and Sing-Hoi Sze. Combinatorial approaches to finding subtle signals in dna sequences. In Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology, pages 269–278. AAAI Press, 2000.

[10] Jean-Marc Fellous, Paul H. E. Tiesinga, Peter J. Thomas, and Terrence J. Sejnowski. Discovering Spike Patterns in Neuronal Responses. J. Neurosci., 24(12):2989–3001, 2004.

[11] Nebojsa Jojic, Vladimir Jojic, Brendan Frey, Christopher Meek, and David Heckerman. Using "epitomes" to model genetic diversity: Rational design of HIV vaccine cocktails. In Y. Weiss, B. Sch¨olkopf, and J. Platt, editors, Advances in Neural Information Pro- cessing Systems 18, pages 587–594. MIT Press, Cambridge, MA, 2006.

[12] P.P. Kuksa and V. Pavlovic, "Efficient discovery of common patterns in sequences over large alphabets", in DIMACS Technical Report, 2009.

[13] Alon, Noga, Yossi Matias, and Mario Szegedy. "The space complexity of approximating the frequency moments." Proceedings of the 28th annual ACM symposium on Theory of computing. ACM, 1996

[14] Babcock, Brian; Babu, Shivnath; Datar, Mayur; Motwani, Rajeev; Widom, Jennifer (2002), "Models and issues in data stream systems", Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2002), pp. 1–16, doi:10.1145/543613.543615.

[15] Duoduo Liao, "Real-Time Solid Voxelization Using Multi-Core Pipelining", The George Washington University, February 2009 http://gradworks.umi.com/3344878.pdf.

[16] Duoduo Liao and Simon Y. Berkovich, "A New Multi-Core Pipelined Architecture for Executing Sequential Programs for Parallel Geospatial Computing", in Proceedings of the 1st international Conference on Computing for Geospatial Research & Application, Washington, D.C., June 21 - 23, 2010. COM.Geo '10, ACM, New York, NY, U.S.A., 2010.

[17] Richard Durstenfeld, Algorithm 235: Random permutation, Communications of the ACM, v.7 n.7, p.420, July 1964.

[18] Fisher, Ronald A.; Yates, Frank (1948) [1938]. Statistical tables for biological, agricultural and medical research (3rd ed.). London: Oliver & Boyd. pp. 26–27.