Volume 11- Number 1 – Winter 2019 (36 -44)

Fuzzy Sequential Pattern Mining over Quantitative Streams

Omid Shakeri **Manoochehr Kelarestaghi*** **Farshad Eshghi** **Ahmad Ganjtabesh**

Electrical & Computer Engineering Dept.
Kharazmi University, Tehran, Iran
{omid.shakeri, kelarestaghi, farshade, std_agt}@khu.ac.ir

Received: 5 May 2018 - Accepted: 3 November 2018

Abstract -- **Sequential pattern mining is an interesting data mining problem with many real-world applications. Though new applications introduce a new form of data called data stream, no study has been reported on mining sequential patterns from the quantitative data stream. This paper presents a novel algorithm, for mining quantitative streams. The proposed algorithm can mine exact set of fuzzy sequential patterns in sliding window and gap constraints entailing the most recent transactions in a data stream. In addition, the proposed algorithm can also mine non-quantitative or transaction-based sequential patterns over a data stream. Numerical results show the running time and the memory usage of the proposed algorithm in the case of quantitative and customer-transaction-based sequence counting are proportional to the size of the sliding window and gap constraints.**

Keywords - Data Stream, Fuzzy Sequential Pattern Mining, Gap Constraint, Sliding Window.

I. INTRODUCTION

The development of database systems and the availability of massive data caused data mining to be a necessary process to extract understandable and usable high-level knowledge. Sequential pattern mining is among the most important studies in the data mining field with many real-world applications such as customer behavior analysis, DNA sequence analysis, and intrusion detection.

The sequential pattern mining problem was first introduced by Agrawal, and Srikant: “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 frequent subsequences, i.e., the subsequences whose occurrence frequency in the set of sequences is no less than min_support” [1, 2]. Many algorithms have been proposed to mine sequential patterns in sequential databases like Apriori-All [1] and GSP [3], or PrefixSPAN [2].

In recent years, emerging applications have introduced a new form of data called Data Stream. A data stream is an unbounded sequence in which new elements are generated consecutively. Any mining method for data streams must consider two critical constraints. The first constraint is the memory usage constraint meaning all the stream data cannot be stored in memory. The second constraint says that each stream component can be looked at once, without performing any blocking operation. Because of these constraints, traditional data mining methods developed for static databases like PrefixSPAN cannot be applied for mining data streams [4].

The existent sequential pattern mining algorithms for data streams like SS-MB and SS-BE [4], PLWAP-Based Algorithms [5] and the algorithm proposed in [6], use the batch processing to store the last n transactions in memory. The batching process performs mining in each batch and neglects the relationship between transactions in subsequent batches which prevents mining sequential patterns whose items are distributed over some subsequent batches. Also, these algorithms do not consider the quantitative sequential databases, which necessarily lead to another loss of

[*] Corresponding Author

information when they use the binary representation of numerical data [7, 8]. Moreover, these algorithms do not account for the user-specified constraints in order to mine patterns more efficiently; and also some of them are not exact and they mine approximate sequential patterns as they cannot mine some sequences [9].

The proposed method in this paper uses a new time-based moving sliding window and time gap which are defined as user constraints. These enable the proposed method to mine the following patterns:

- Complex sequential patterns which cannot be mined by batch-based algorithms,
- The exact set of constrained sequential patterns over a data stream.

It is well-worth mentioning that the proposed method fuzzifies the input data to face the quantitative data stream, and representing them as fuzzy categorical items, which enables the proposed method to output fuzzy sequential patterns in terms of fuzzy items and itemsets.

The remaining of this paper is organized as follows. Section II introduces the basic concepts and the problem of sequential pattern mining in static and stream databases. Section III introduces the constrained sequential pattern mining. Section IV presents the proposed algorithm. In Section V, some experiments are conducted to analyze the proposed, and the conclusion is stated in Section VI.

II. PROBLEM STAEMENT

The problem is to mine the exact set of constrained fuzzy sequential patterns over a quantitative data stream. Let $I = \{I_1, I_2, \ldots, I_p\}$ be the set of all *items*. An *itemset* is a nonempty set of items, like $e_1 = I_1\ I_2$. A concurrent itemset is the one whose items happen concurrently, and are denoted by surrounding parentheses, like $e_2 = (I_4\ I_7)$. A *sequence* is an ordered list of *itemsets*. A sequence s is denoted $< e_1e_2e_3 \ldots e_l >$, where *itemset* e_1 occurs before e_2, and e_2 before e_3, and so on[HYPERLINK \l "Jia06" 10]. Fuzzy itemset is a set of fuzzy items, and it can be donated as a pair of sets (set of items, set of fuzzy sets associated to each item's quantity) or as a list of fuzzy items, like ([I_1, low: 0.6] [I_3, med: 1] [I_6, high: 0.25]), which shows three concurrent items with their fuzzified quantities.

The main challenge in data stream mining is the limited resources of time and memory. Data mining has been studied extensively in static datasets, where data mining algorithms can handle reading the input data several times, like the algorithms proposed in [1, 2, 3]. When the source of data items is a data stream, not all data can be loaded into the memory, and off-line mining algorithms with the static dataset is no longer technically feasible due to the features of data streams [11, 12]. The following constraints are forced due to data stream features model [12]:

- The length of a data stream is potentially infinite, and it would be impossible to store all elements. Thus, only a small part is stored and processed.
- As the elements of a data stream are received fast, they should be processed in real time.

The above constraints limit the amount of memory and the time-per-item that the stream mining algorithm can use.

While the existent algorithms use batch processing to face the above limitations, this paper uses time-based moving sliding window and gap constraints to handle the above limitations as well as limitation of the batch processing for mining sequential patterns over multiple subsequent batches. Also, to face the data loss in the quantitative sequential pattern algorithms over data streams, the proposed method fuzzifies numerical data, and then it mines sequential patterns over the fuzzified data.

III. SLIDING WINDOW & GAP CONSTRAINTS

Mining without user- or expert-specified constraints may generate numerous patterns that are of no interest. Thus, user-specified constraints are incorporated into the mining algorithm to reduce the search space and mine the only patterns that are of interest to the user [10, 13]. In this paper, moving sliding window and gap constraints are used to face the limitations of stream mining.

A Sliding Window or duration constraint is defined only in sequence databases where each transaction has a time-stamp. It requires that the pattern appears frequently in the sequence database such that the time-stamp difference between the first and last transactions in the pattern is shorter than a given constant [13]. Thus, the transactions in a sliding window are assumed to be concurrent.

A gap constraint is defined only in sequence databases where each transaction in every sequence has a timestamp. It requires that the pattern appears frequently in the sequence database such that the timestamp difference between every two adjacent transactions - be longer than a given gap [13, 14].

Sliding window and gap constraints are defined by two membership functions in term of time difference, as shown in Figure 1 [15, 16, 17].

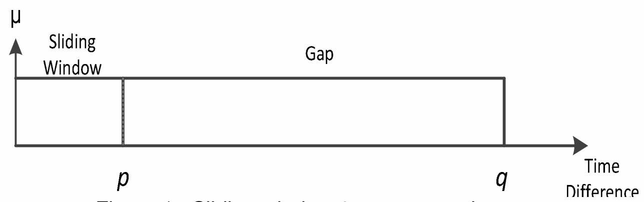


Figure 1: Sliding window & gap constraints

IV. THE PROPOSED METHOD

The proposed method consists of three phases as shown in Figure 2 and the pseudo-code of the proposed method, Constrained Fuzzy Stream Sequence Miner (CFSSM), is shown in Figure 3. In the first phase, data streams are buffered.

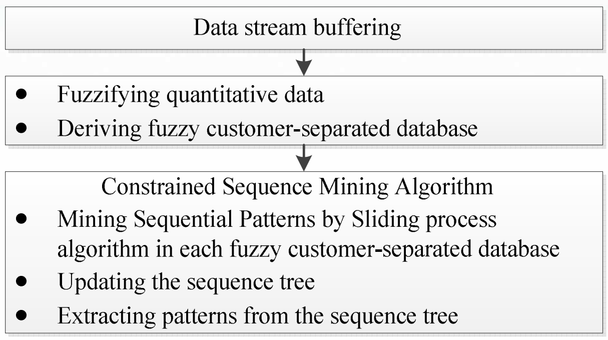

Data stream buffering
• Fuzzifying quantitative data
• Deriving fuzzy customer-separated database
Constrained Sequence Mining Algorithm
• Mining Sequential Patterns by Sliding process algorithm in each fuzzy customer-separated database
• Updating the sequence tree
• Extracting patterns from the sequence tree

Figure 2: Flowchart of the CFSSM method

The second phase fuzzifies the quantitative data in order to represent them as fuzzy categorical data. This representation avoids data loss caused by a binary representation of numerical data. And finally, fuzzy sequential patterns are mined by CFSSM algorithm in the third phase. These phases are described in detail in Subsections IV-A to IV-C. The solution quality of the proposed method is discussed in Subsection IV-D.

```
CFSSM_Method
Inputs: Data_Stream, Mining_Parameters
Output: Set_of_Sequential_Patterns

 1: While (~EndOfStream && ~UserDemandFlag)
 2:    Data Stream Buffering
 3:    Update Customer Fuzzy Database
 4:    For each Customer
 5:        Sliding_Process_Algorithm
 6:        Update_Sequence_Tree_Algorithm
 7:    End For
 8: End While
 9: Scan the Sequence Tree in depth
10: Show Sequential Patterns where  count > min_count
```

Figure 3: CFSSM pseudo code

A. Data Stream Buffering

In data streams, the entire data of each customer is not available, and they are generated continuously. Each transaction in a quantitative data stream has a customer-ID, a transaction type, a time-stamp, and a quantity value, as shown in Table 1.

Table 1: Quantitative data stream sample

Customer-ID	*Transaction*	*Time-Stamp*	*Quantity*

In CFSSM, to face the limitations of data streams, as mentioned in section 2, a new time-based moving sliding window is proposed to buffer the most recent transactions of which their time-stamp value satisfies user-specified sliding window and gap constraints. Thus, the last part of the stream (current database) which has been buffered is updated when a new transaction from the stream is received. It should be noted that the number of transactions in the current database is related to the frequency by which the data are received, and it might be variable in time.

For example, suppose the constraint values in Figure 1, are set to p=2, q=6, and the stream be as given in Table 2. Also let the most recent transaction, called now, be the one whose time-stamp is 15. By receiving this data, the moving sliding window and gap are moved in such a way that "q", as shown in Figure 1, is located at now, Time Stamp of the last Transaction in Stream.

At this time, Figure 4 shows the correspondence between the time-stamps and the parameters of the sliding window and the gap constraints. Thus, some transactions are removed from the current database whose time-stamps are less than 9. These transactions are shown in grey in Table 2, and the buffered ones are shown in black in Table 2.

Table 2: Quantitative data stream sample

Customer-ID	*Transaction*	*Time-Stamp*	*Quantity*
1	A	1	5
2	D	4	8
3	B	8	2
1	**A**	**11**	**3**
1	**B**	**14**	**1**
2	**A**	**14**	**5**
1	**C**	**15**	**1**
2	**C**	**15**	**3**

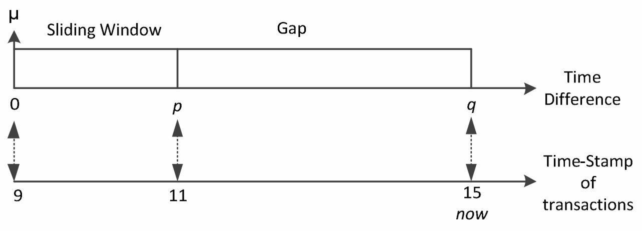


Figure 4: Correspondence between the time-stamps and the parameters of the sliding window and the gap constraints

B. Fuzzifying Quantitative Data and Deriving Fuzzy Customer-Separated Database

In order to avoid data loss caused by the binary representation of numerical data, the quantitative data buffered in current database is fuzzified, hereinafter called fuzzy database, and is expressed by the linguistic terms as shown in Figure 5.

It's possible to use one of the following fuzzy cardinality count types to fuzzify the quantities [18]:

- The fuzzy membership value $\mu(x)$,
- The fuzzy membership value if it's bigger than a specified threshold (*thr*),
- The crisp count if the fuzzy membership value is bigger than a specified threshold (*thr*).

This paper uses the first and the third cardinality count types in the examples given in the next subsections.

In order to mine sequential patterns from customer behaviors, the current fuzzy database should be divided into fuzzy customer-separated databases. It should be noted that some transactions may satisfy two fuzzy membership functions and will be expressed by two linguistic terms, like the first two rows in Table 3, which shows the fuzzy customer-separated databases of the example given in Table 2, for w=2, x=6, y=9, z=13 in Figure 5, using μ(x) for fuzzy cardinality count. The pseudo-code of this phase, called fuzzifying quantitative data and deriving fuzzy customer-separated database, is shown in Figure 6.

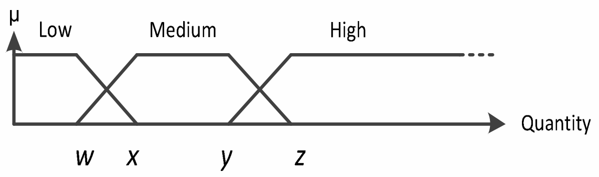


Figure 5: Fuzzy membership functions for quantitative items

Update_Fuzzy_Customer_separated_Database_Algorithm
Inputs: Buffered_Data_Stream, is_Qunatitative, Cardinality_Type
Output: Fuzzy_Customer_Separated_Databases

```
1:  For each transaction in Buffered Data Stream
2:    If new_customer_ID
3:       create new_customer_database
4:       Fuzzify the quantities
5:       Generate new fuzzy transactions
6:    Else
7:       Fuzzify the quantities
8:       Generate new fuzzy transactions
9:  End For
```

Figure 6: Updating fuzzy customer-separated database pseudo code

Table 3: Quantitative data stream sample

Customer-ID	*Transaction*	*Time-Stamp*	$\mu_{quantity}$
1	A-low	11	0.75
	A-med	11	0.25
	B-low	14	1
	C-low	15	1
2	A-low	14	0.25
	A-med	14	0.75
	C-low	15	0.75
	C-med	15	0.25

C. Constrained Sequence Mining Algorithm

In this phase, sequential patterns are mined by the Sliding Process algorithm from fuzzy customer-separated databases. The pseudo-code of the Sliding Process algorithm is shown in Figure 7.

In the Sliding Process algorithm, the sequence to be mined is initialized as a null sequence. For each fuzzy customer-separated database, the start of the sliding window constraint, shown in Figure 1, slides to the Time-stamp of the first transaction and all the transactions whose timestamps are in this sliding window are assumed to be the first itemset (lines 1-7 in Figure 7). After finding the first itemset of a sequence, the start of the moving sliding window slides to the time-stamp of the next transaction whose time-stamp wasn't in the moving sliding window constraint in the previous iteration, and all the transactions whose timestamps are in this sliding window are assumed to be the next itemset (lines 8-12 in Figure 7), this loop (lines 8-12 in Figure 7) is repeated until all transactions satisfy one of the moving sliding windows, i.e., the end of the first pass.
At the end of the first pass, one sequence is mined from the fuzzy customer-separated database. In order to mine more sequences from the fuzzy customer-separated database, pass 2 starts in which the start of the moving sliding window slides to the timestamp of the next transaction that has not yet been the start of any previous moving sliding windows, and another new sequence is mined. This procedure repeats until all sequences are mined from the fuzzy customer-separated database.

The Sliding Process algorithm passes for the customer 1's database in Table 3 are shown in Table 4. The starting transaction in each pass is shown in bold; and the transactions whose time-stamp don't satisfy the moving sliding window, are shown in gray. The count of each sequence can be computed by the user-specified fuzzy T-norm between fuzzy membership functions of itemsets ($\mu_{quantity}$) in each sequence.

Sliding_Process_Algorithm
Inputs: Customer_Fuzzy_Database, Mining_Parameters
Output: set_of(Sequence, Count)

```
1:  For each transaction t whose time-stamp had not been at the
        start of any previous moving sliding windows
2:    sequence = < >
3:    itemset = { }
4:    Slide the start of the moving sliding window to transaction
        t's time-stamp
5:    itemset = set of transactions whose timestamps are in the
        sliding window
6:    itemset_count = T-norm (transactions $\mu_{quantity}$)
7:    add itemset to sequence
8:    For each transaction t2 whose timestamp wasn't in the
          previous sliding window
9:      Slide the start of the moving sliding window to
          transaction t2's time-stamp
10:     itemset = set of transactions whose timestamps are in
                 the sliding window
11:     itemset_count = T-norm (transactions $\mu_{quantity}$)
12:     add itemset to sequence
13:   End For
14:   sequence_count = T-norm(itemsets of sequence)
15:   Add (sequence, sequence_count) to the output set
16: End For
```

Figure 7: Sliding process pseudo code

Table 4: Sliding process on customer 1's database

Pass	*Transaction*	*Time-Stamp*	$\mu_{quantity}$
1	**A-low**	**11**	**0.75**
	A-med	11	0.25
	B-low	14	1
	C-low	15	1
	A-low	11	0.75
	A-med	11	0.25
	B-low	**14**	**1**
	C-low	15	1
2	A-low	11	0.75
	A-med	11	0.25
	B-low	14	1
	C-low	**15**	**1**

After the Sliding Process algorithm, Update Sequence Tree Algorithm is called to insert the sequences into the sequence tree that contains the sequences and the count of each item (Figure 8).

```
Update_Sequence_Tree_Algorithm
Input: Set_of(sequences, count)

 1:  For each Sequence
 2:  currrntNode = Root
 3:      For each itemset of the sequence
 4:          If itemset is in currrntNode's children
 5:              update the count of the child
 6:              currrntNode = child
 7:          Else
 8:              Generate new node with itemset properties
 9:              currrntNode = new node
10:      End For
11:  End For
```
Figure 8: Update sequence tree pseudo code

According to the definition of stream mining, sequential patterns should be available whenever the user demands. Thus, when that happens, the algorithm scans the Sequence Tree in depth to output the sequential patterns.

For example, if the stream was only the bold part of the stream given in Table 2, and user demands the patterns with minimum as T-norm and min_count=0, after processing the current database of customer 1, the tree would be like Figure 9. By scanning the Sequence Tree in depth, the Sequential Patterns shown in Table 5 would be generated; and after Sliding Process on the current database of customer 2, the tree would be updated to Figure 10.

Table 5: Sequential patterns

Sequence	*Count*
< A-low, B-low >	0.75
< A-low, C-low >	0.75
< A-low(B-low, C-low) >	0.75
< C-low >	1.25

In the proposed method, two sequence counting types can be used:

- Customer-Transaction-based Sequence counting
- Transaction-based Sequence counting

The above example shows a customer-Transaction-based sequence counting method, wherein the stream is divided into customer's databases and the sequence found in each customer's database would be updated in the sequence tree. However, in the transaction-based sequence counting, the whole stream would be assumed to be generated by one customer, thus the stream wouldn't be divided into customer databases.
Also, the user can specify whether the algorithm mine quantitative sequences or not. In the non-quantitative case, the module Fuzzifying quantitative data of the proposed method (Figure 2) is discarded, and the module Deriving fuzzy customer-separated database is discarded in the case of transaction-based sequence counting.

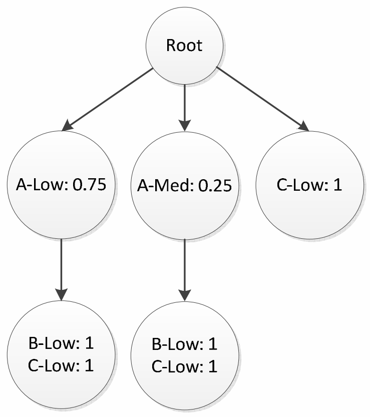


Figure 9: The sequence tree after sliding process on the current database of customer 1

D. *Solution Quality*

As mentioned in subsection IV-C, the proposed method in

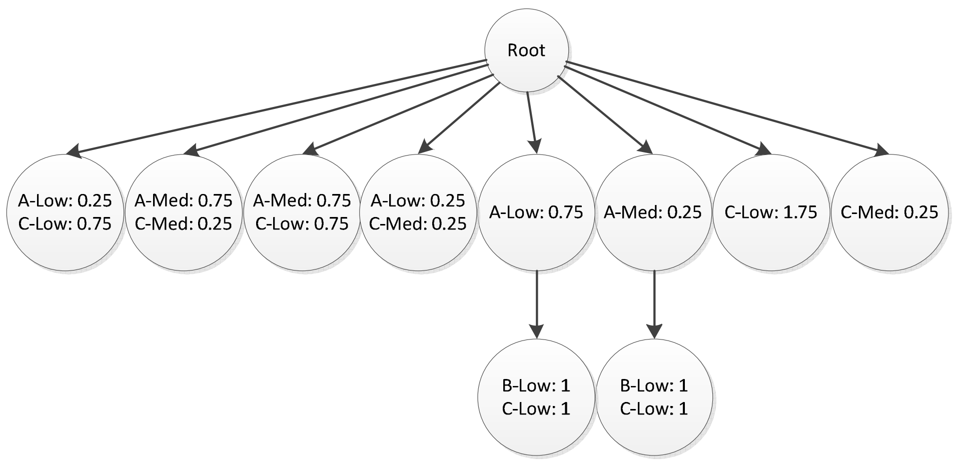


Figure 10: The sequence tree after sliding process on the current database of customer 2

this paper uses a new time-based moving sliding window and time gap which are defined as user constraints. These enable the proposed method to mine the following patterns:

- Complex sequential patterns which cannot be mined by batch-based algorithms,
- The exact set of constrained sequential patterns over a data stream, which cannot be mined by existent algorithms which work on binary representation of data.

Also, the outputs of the proposed method, which mines the exact set of constrained sequential patterns over a data streams, and the Constrained PrefixSPAN on the same static database, would be the same, in terms of equality constraints.

Table 6: non-quantitative data stream

Transaction	*Time-Stamp*
A	1
D	4
B	5
A	11
B	14
A	14
C	15
C	15

Here is a simple transaction-based and non-quantitative example to show the differences between the proposed method and batch-based algorithms. For the stream given in Table 2, suppose the constraint parameters in Figure 1 be p=2, q=5, and the crisp count, as mentioned in section IV-B, be used. For batch_size=4 in batch-based algorithms, two different batches will exist in Table 2 as shown in Table 6. As said before, the relations between transactions in these two batches are discarded in the batch-based algorithms, e.g. the relation between the fourth and fifth transaction is discarded, and the sequence <AB> won't be mined for min_count=2.

In this paper, the new time-based moving sliding window and time gap are proposed. For the given example, these constraints are shown in Figure 11, for different time-stamps in the stream. Thus, the proposed method can mine the sequence <AB> with a count equal to two in this stream, while in batch-based algorithms, the count of the sequence <AB> will be one.

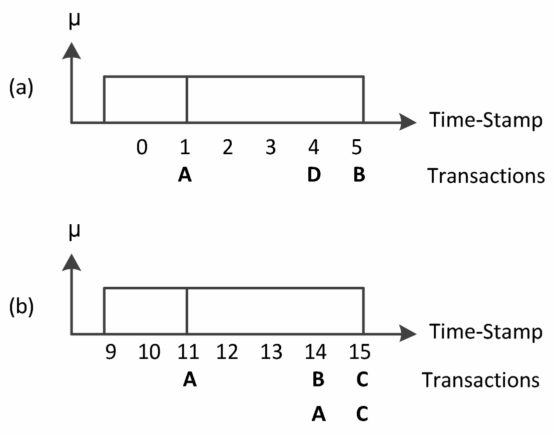


Figure 11: Stream buffering in different time-stamps with corresponding transactions (a) now = 5, (b) now = 15

The main advantages of the proposed algorithm are:

- Mines the exact set of fuzzy sequential patterns over quantitative or non-quantitative data stream,
- The ability to mine non-constrained or constrained sequential patterns over data streams,
- The ability to mine both transaction-sensitive streams and customer-transaction-sensitive data streams.

V. Experimental Results

In this section, the performance of the proposed method, CFSSM, is investigated by some experimental studies. All experiments were conducted on a 2.53GHz Intel Core 2 Duo PC with 4GB main Memory, running Microsoft Windows 7 operating system. All algorithms were implemented in C#. The results are shown for two data streams, T10I4D100K and T40I10D100K, which are generated from the synthetic data generator described by Agrawal et al in [19]. The parameters of these datasets are T: the average size of transactions, I: the average size of the maximal potentially frequent itemsets, and D: the number of transactions (in 1000). To use these datasets in this algorithm, a customer-ID, in range of 1-100 is randomly assigned to each transaction (itemset), and also a quantity value, in range of 1-10 is randomly assigned to each item. It is well-worth mentioning that there is no numerical sequential data stream available in standard datasets. Several experiments were conducted for different characteristics of transactions and types of data. The results are labeled as follows: Q, nQ, CT and T which indicate to Quantitative data, non-Quantitative data, Customer-Transaction-based Sequence Counting, and Transaction-based sequence counting, respectively. Figure 12 and Figure 13 show the results on T10I4D100K and Figure 14 and Figure 15 show the results on T40I10D100K. In the following figures, the mining type "Q, CT" is the main routine of the CFSSM algorithm, which is described by example in previous sections.

Figure 12 and Figure 14 show the results under different gap sizes, where the sliding window size is 10, for different mining parameters. As the gap grows the running time and memory usage will increase because more transactions are stored in the memory to scan. The running time and memory usage are smaller in the case of mining non-Quantitative sequential patterns rather than mining Quantitative ones, because of less computational burden and also smaller data structures created during the process. The memory usage is smaller in the case of mining Customer-Transaction-based sequential patterns rather than Transaction-based sequence counting ones, because of more sliding processes in a bigger customer database with more transactions.

Figure 13 and Figure 15 show the results under different sliding window sizes, where the gap size is 150, in the case of Q, CT. The running time increases when the sliding window size is growing up, as more transactions are stored in memory, thus there will be more iterations in the Sliding Process algorithm, in Figure 7. This means the more available memory, the longer and the more complex sequential patterns CFSSM can mine.

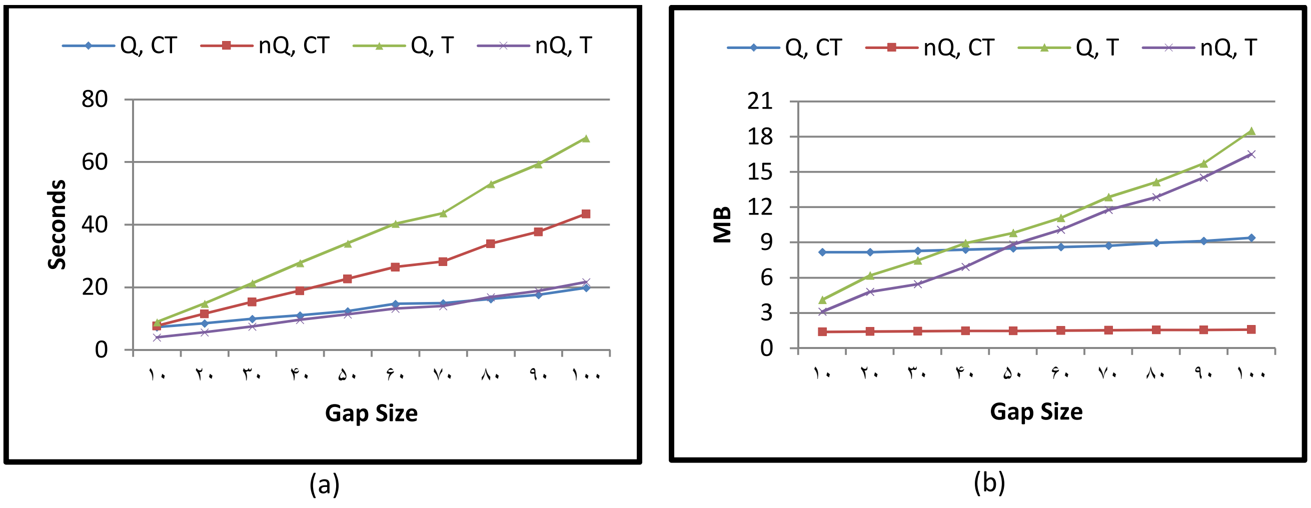


Figure 12: Results on T10I4D100K Dataset:
(a) runtime for different gap sizes for different mining parameters, where sliding window size = 10,
(b) memory usage for different gap sizes for different mining parameters, where sliding window size = 10

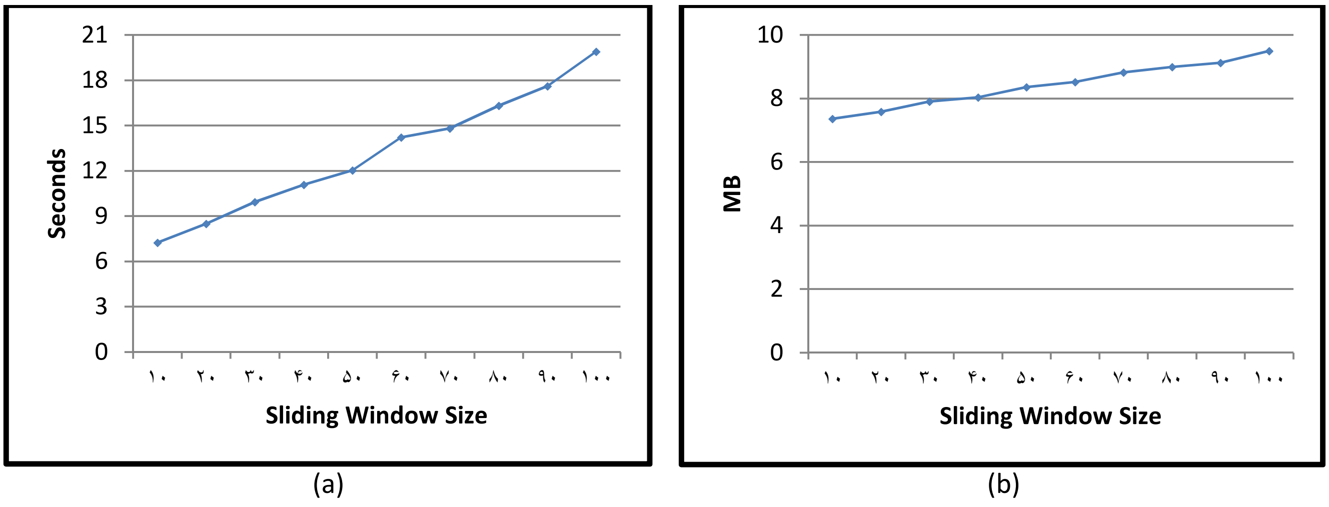


Figure 13: Results on T10I4D100K Dataset in case of Q, CT:
(a) runtime for different sliding window sizes, where gap size = 150,
(b) memory usage for different sliding window sizes, where gap size = 150

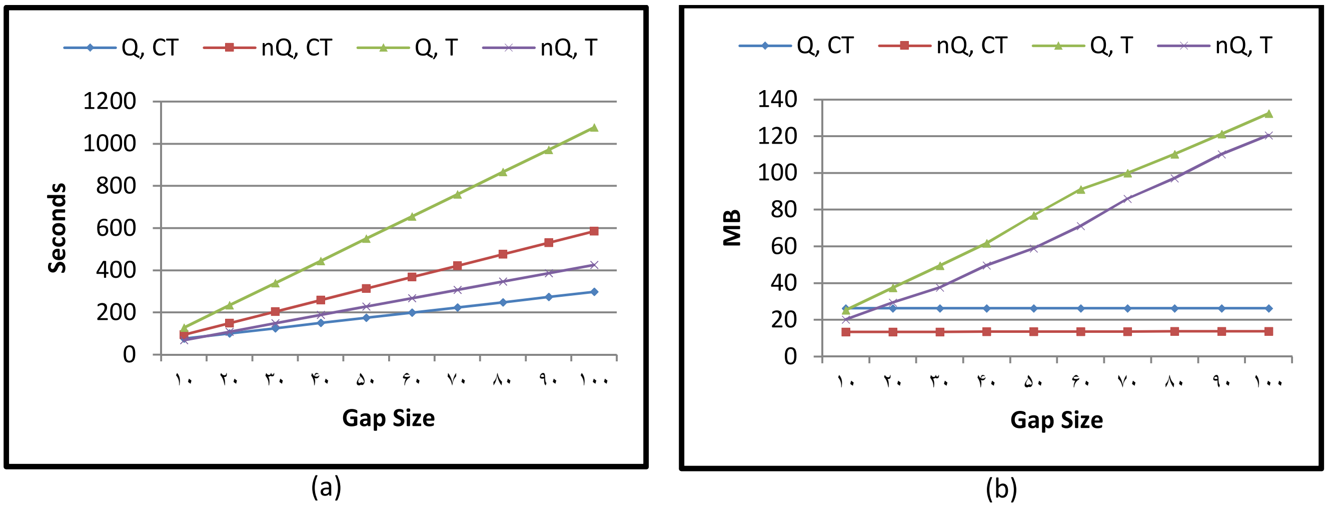
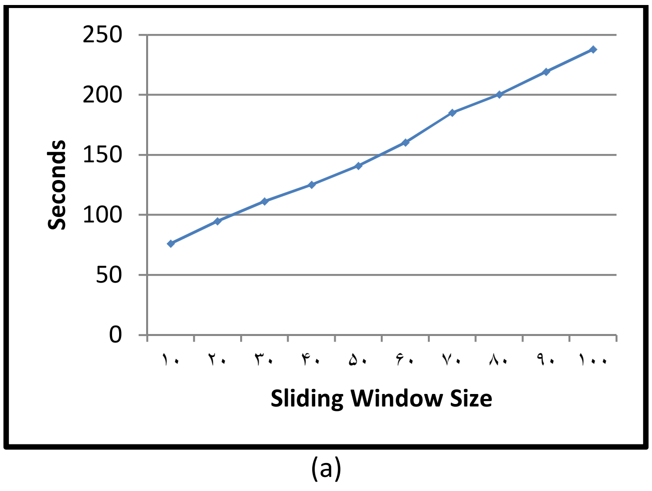


Figure 14: Results on T40I10D100K Dataset:
(a) runtime for different gap sizes for different mining parameters, where sliding window size = 10,
(b) memory usage for different gap sizes for different mining parameters, where sliding window size = 10



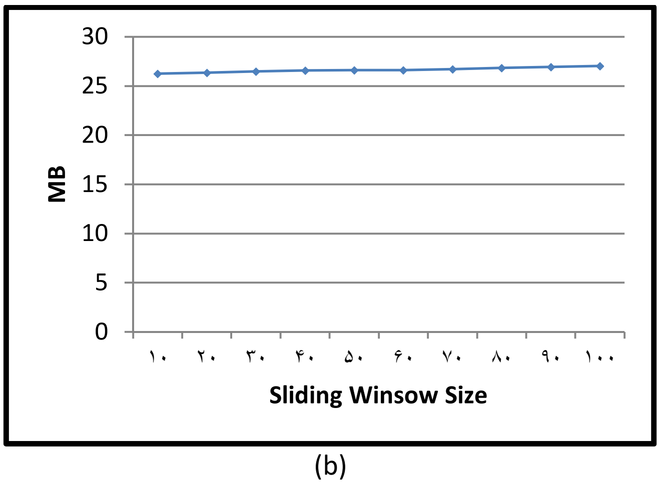


Figure 15: Results on T40I10D100K Dataset in case of Q, CT:
(a) runtime for different sliding window sizes, where gap size = 150,
(b) memory usage for different sliding window sizes, where gap size = 150

VI. CONCLUSION

In this paper, a new method, CFSSM, including new time-based moving sliding window and gap constraints, was proposed to mine quantitative streams which can mine the exact set of constrained fuzzy sequential patterns. It also can mine non-Quantitative or Transaction-based sequential patterns over a data stream. The proposed method uses the fuzzy set concept to mine fuzzy sequential patterns from numerical data stream. Experimental studies showed that the running time and the memory usage of the proposed algorithm in the case of quantitative and customer-transaction-based sequence counting are nearly proportional to the size of the fuzzy sliding window and gap constraints.

ACKNOWLEDGEMENT

This research was partially supported by *Research Institute for Information and Communication Technology* of Iran.

REFERENCES

1. R. Agrawal, R. Srikant, "Mining Sequential Patterns," Data Engineering, Proceedings of the Eleventh International Conference on, pp. 3-14. March 1995.
2. J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, M. C. Hsu, "Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach," IEEE Transactions on Knowledge and Data Engineering, Vol. 16, pp. 1424-1440, May 2004.
3. R. Agrawal, R. Srikant, "Mining sequential patterns: generalizations and performance improvements" in 5th International Conference on Extending Database Technology, Springer, March 1996, pp. 3-17.
4. L. F. Mendes, B. Ding, J. Han, "Stream Sequential Pattern Mining with Precise Error Bounds," Data Mining, ICDM '08, Eighth IEEE International Conference on, pp. 941-946, 2008.
5. C. I. Ezeife, M. Mostafa, "A PLWAP-Based Algorithm for Mining Frequent Sequential Stream Patterns," Technology and Intelligent Computing (ITIC), Vol. 2, pp. 89-116, 2007.
6. Q. Huang, W. Ouyang, "Sequential Patterns Mining Scaling with Data Stream Based on LSP-tree" Sixth International Conference on Fuzzy Systems and Knowledge Discovery, Vol. 5, August 2009, pp. 614-618.
7. T. P. Hong, C. S. Kuo, S. C. Chi, "Mining fuzzy sequential patterns from quantitative data", in Systems, Man, and Cybernetics. IEEE SMC '99 Conference Proceedings, IEEE International Conference on, Vol. 3, 1999, pp. 962-966.
8. T. P. Hong, K. Y. Lin, S. L. Wang, "Mining Fuzzy Sequential Patterns from Multiple-Item Transactions" IFSA World Congress and 20th NAFIPS International Conference, Vol. 3, 2001, pp. 1317-1321.
9. J. Cheng, Y. Ke, W. Ng, "A Survey on Algorithms for Mining Frequent Itemsets over Data Streams", Springer Knowledge and Information Systems, Vol. 16, pp. 1-27, 2008.
10. J. Han, M. Kamber, Data Mining: Concepts and Tecniques, 2nd edition, Morgan Cufmann - Diane Cerra, 2006.
11. C. C. Aggrawal, Data Stream: Models and Algorithms, Purdue University, West Lafayette, Springer, 2007.
12. A. Bifet, Adaptive Stream Mining: Pattern Learning And Mining From Evolving Data Streams, Amsterdam, Netherlands: IOS Press BV, 2010.
13. J. Pei, J. Han, W. Wang, "Mining sequential patterns with constraints in large databases" ACM 11th International Conference on Information and Knowledge Management, 2002, pp. 18-25.
14. S. Bringay, A. Laurent, B. Orsetti, P. Salle, M. Teisseire, "Handling Fuzzy Gaps in Sequential Patterns: Application to Health" Fuzzy Systems, FUZZ-IEEE. IEEE International Conference on, 2009, pp. 1338-1345.
15. C. I. Chang, H.E. Chueh, N. P. Lin, "Sequential Patterns Mining with Fuzzy Time-Intervals" IEEE Sixth International Conference on Fuzzy Systems and Knowledge Discovery, Vol. 3, 2009, pp. 165-169.
16. C. Xu, Y. Chen, R. Bie, "Sequential Pattern Mining in Data Streams Using the Weighted Sliding Window Model", 15th International Conference on Parallel and Distributed System, 2009, pp. 886-890.
17. F. Zabihi, M. M. Pedram, M. Ramezan, A. Memariani, "Fuzzy Sequential Pattern Mining with Sliding Window Constraint" 2nd International Conference on Education Technology and Computer (ICETC), Vol. 5, 2010, pp. 396-400.
18. C. Fiot, A. Laurent, M. Teisseire, "From Crispness to Fuzziness: Three Algorithms for Soft Sequential Pattern Mining," IEEE Transactions on Fuzzy Systems, Vol. 15, pp. 1263-1277, December 2007.
19. R. Agrawal, R. Srikant, "Fast algorithms for mining association rules," 20th Intl. Conf. on Very Large Databases (VLDB'94), pp. 487-499. 1994.

AUTHORS' INFORMATION

Omid Shakeri received his Master Degree in Artificial Intelligence from Kharazmi University, Tehran, Iran in 2011. Since then he has been working as a web full stack developer at "IFITPRO" company, Tehran, Iran.

Manoochehr Kelarestaghi received his Ph.D. degree in System Control Engineering at Université de technologie de Compiègne (UTC), Compiègne, France in 1999, M.S. and B.S. degrees in Electronics Engineering from Tehran University and Sharif University, Tehran, Iran, in 1992 and 1988, respectively. He is currently an Assistant Professor in Electrical and Computer Engineering Department, Kharazmi University, Tehran, Iran. His current research interests are NLP, Pattern Recognition, Signal Processing and Optimization.

Farshad Eshghi has done a Post-Doc and received his Ph.D. in Electrical Engineering-Telecommunications from University of British Columbia, Vancouver, BC, Canada, and Concordia University, Montreal, QC, Canada in 2004 and 2006 respectively. From 2008 to 2011, he has served as a lecturer in the Dept. of Computer Science, Faculty of Mathematics and Computer Sciences, Amir kabir University of Technology. Since 2011, he has been with the Dept. of Electrical and Computer Engineering, Faculty of Engineering, Kharazmi University, Tehran, Iran as an Assistant Professor. His main research interests include different topics in Ad Hoc WLANs, WSNs, and Intelligent Management Systems with applications in BMS, Transportation, and Health.

Ahmad Ganjtabesh is a graduate student studying towards his Master degree in Computer Science at Kharazmi University, Tehran, Iran. His research interests include Recommender Systems, Machine Learning, and Fuzzy Modeling.