

# A Novel Sequential Pattern Mining Algorithm for Large Scale Data Sequences

Ali Burak Can[1(✉)], Meryem Uzun-Per[1,2], and Mehmet S. Aktas[3]

[1] BiletBank Research and Development Center, Akdeniz PE-TUR A.S., Istanbul, Turkey
{aliburak.can,meryem.uzunper}@petour.com
[2] Computer Engineering Department, Istanbul Health and Technology University, Istanbul, Turkey
meryem.uzunper@istun.edu.tr
[3] Computer Engineering Department, Yildiz Technical University, Istanbul, Turkey
aktas@yildiz.edu.tr

**Abstract.** Sequential pattern mining algorithms are unsupervised machine learning algorithms that allow finding sequential patterns on data sequences that have been put together based on a particular order. These algorithms are mostly optimized for finding sequential data sequences containing more than one element. Hence, we argue that there is a need for algorithms that are particularly optimized for data sequences that contain only one element. Within the scope of this research, we study the design and development of a novel algorithm that is optimized for data sets containing data sequences with single elements and that can detect sequential patterns with high performance. The time and memory requirements of the proposed algorithm are examined experimentally. The results show that the proposed algorithm has low running times, while it has the same accuracy results as the algorithms in the similar category in the literature. The obtained results are promising.

**Keywords:** Sequential pattern mining · GSP · PrefixSpan · Large scale data sequences · MapReduce programming model

## 1 Introduction

Due to the increase in e-commerce transaction volume in recent years, the data that must be stored rises exponentially day by day. Therefore, it becomes inevitable to develop both data mining and big data processing methods. One of the main tasks of data mining is to discover useful patterns normally hard to distinguish in data by eye [7,12,24]. Pattern mining have been an active research topic in data mining for years. Apriori [1], ECLAT (Equivalence Class Clustering and bottom-up Lattice Traversal) [33], FP-Growth (Frequent Pattern Growth) [8] algorithms are some example algorithms for association rule mining, while GSP (Generalized Sequential Pattern) [19], SPADE (Sequential PAttern Discovery using Equivalence classes) [32], PrefixSpan (Prefix-projected Sequential

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022
O. Gervasi et al. (Eds.): ICCSA 2022 Workshops, LNCS 13377, pp. 698–708, 2022.
https://doi.org/10.1007/978-3-031-10536-4_46

pattern mining) [13] methods are used for sequential pattern mining applications as both of them are sub-branches of pattern mining. These two different algorithm families differ in their goals. Sequential pattern mining methods consider the order of the items, unlike the association rule mining methods.

As the nature of sequential pattern mining and association rule mining algorithms, they work on different types of datasets. Sequential pattern mining algorithms like PrefixSpan are designed for datasets containing sequential items. On the other hand, association rule mining methods like FP-Growth can be applied for transactional datasets. A transactional dataset consists of unique items corresponding to each Transaction-ID. A symbolic representation could be such as $TID$-1: A, B, C and $TID$-2: A, C, D. This example could be interpreted like having two basket records on the dataset of a grocery and product A, product B, and product C in one basket and product A, product C, and product D in another basket. The order of the items is immaterial in this type of dataset. On the contrary, in a dataset containing sequential items corresponding to each Sequence-ID, the items are arranged in order, called Sequential Dataset. It is seen by examining sequences here that there may be more than one element in an itemset. For example, if we consider this symbolic representation $SID$-1: A, (A, B, C), (A, C), D, (C, F) as the shopping data of a market, in the row corresponding to $SID$-1; a customer purchases product A on Monday, products A, B, and C on Tuesday, products A and C on Wednesday, product D on Thursday, and product C and F on Friday. Differently, some kinds of datasets like a genome sequence data portrayed in this symbolic representation $SID$-1: (A), (G), (T) have different characteristics. Even though these datasets are ordered from left to right as the sequential dataset, each itemset has only one element. In other words, each itemset is one-dimensional, or the length of the itemset is 1.

Within the scope of this research, an algorithm is proposed which is designed to detect sequential patterns on big data processing and analysis platforms on sequential datasets (e.g., genome sequence data) containing ordered sequences and containing only one element in each sub-sequence. The algorithm is named $Proposed\text{-}SPMA$ (Sequential Pattern Mining Algorithm) in this article. In order to demonstrate the success of the proposed algorithm, a prototype implementation has been done.

The algorithm we propose within this study has been compared with the sequential pattern mining methods (PrefixSpan and GSP) in the literature. Apache Spark [11] open source library, which is one of the widely used platforms, is used for the big data processing and analysis platform. While the ready-made function in the Apache Spark MLlib library was used for the PrefixSpan algorithm, the GSP algorithm was developed by using the Apache Spark library.

The rest of this article is organized as follows: Sect. 2 describes the research questions we examined thoroughly in this study in detail. In Sect. 3, sequential pattern mining, and big data processing are examined and also sequential pattern mining studies in big data are summarized. Section 4 gives the pseudocode of the proposed algorithm, its explanation, and its textual explanation through an

example. Experimental results and their evaluation are done in Sect. 5. In the last section, a summary of the whole study is given.

## 2 Research Questions

In this study, it is examined how a high-performance sequential pattern mining algorithm can be created that will allow finding sequential patterns on one-dimensional sequences that are large enough to be stored in a single computer's data storage media (memory and physical disk). We summarize below the research questions that we have been working on within the scope of this research.

1) How a sequential pattern mining algorithm that will work with high performance on datasets with big data characteristics should be to enable finding sequential patterns on single-element itemsets that have come together in a certain order?
2) How can the running time and memory requirements of a sequential pattern mining algorithm be determined experimentally? How is the performance of the proposed sequential pattern mining algorithm in this research, when compared to other algorithms in the same category, in terms of running time and memory requirements?

## 3 Literature Review

Sequential pattern mining is a subtopic of pattern mining, and it is used to find common patterns in sequential data [7,12,24]. GSP [19] and PrefixSpan [13] algorithms are among the frequently used algorithms for sequential pattern mining, which has many application areas such as bioinformatics [25], market basket analysis [19], and text analysis [14]. Within the scope of this research, a comparative analysis is conducted for the proposed sequential pattern mining algorithm with the GSP and PrefixSpan algorithms.

By inspired by apriori method, GSP algorithm was invented. As Apriori, *support* and $confidence$ metrics are also calculated in this algorithm [19]. Support is the probability of a pattern in all of the transactions. Confidence is the probability that a pattern coexists with another pattern. In the GSP algorithm, the dataset is first scanned for candidates for sequential patterns. Then, the candidates that do not meet the predefined minimum support (*minsup*) are removed. Possible candidates of sequential patterns are generated with the remaining candidates. The dataset is scanned again, and the frequencies of the newly generated pattern candidates are calculated. Again, candidates that are below the *minsup* are deleted. Then, potential candidates are generated with patterns that exceed the value of *minsup*, and these processes continue until no new pattern candidates can be created. Similar to the Apriori algorithm, the GSP algorithm has overheads like creating a numerous sequential pattern candidates, dataset scanning several times, and having trouble to find long patterns.

PrefixSpan algorithm does not necessitate the generation of pattern candidates. [13]. In the PrefixSpan algorithm, basically, the projections of the dataset are generated, and sequential patterns are created. Instead of candidates, PrefixSpan uses frequent prefixes in the projections of the dataset. This method diminishes the processing time and boosts the efficiency of the algorithm.

Big data is often characterized by variety, volume, and velocity, and it may be comprised of structured, semi-structured, or unstructured data. One of the most significant aspects of big data processing is the ability to handle data fast [5,16]. Data storage, data processing, and workflow management are the three main categories of the open source big data processing ecosystem. To process big data, there are several system such as Apache Hadoop, Spark, Flink, and Storm. They use HDFS (The Hadoop Distributed File System) for storing technology and YARN (Yet Another Resource Negotiator) for scheduling tasks and managing resources. Compared to Apache Hadoop, Apache Spark [31], an open source platform for distributed big data processing, is faster in terms of iterative transactions because it prevents writing to disk as its capability. In-memory computing and RDDs (Resilient Distributed Datasets) are main attributes of Apache Spark. Hence, it runs 100 times faster in memory and ten times faster on disk than Apache Hadoop. For machine learning applications, Hadoop uses Mahout [2], Spark uses MLlib [11]. Because machine learning algorithms often involve iterative operations, Spark MLlib is faster than Mahout to run these algorithms. The algorithm we propose in this study is designed to work on big data processing and analysis platforms that implement the map-reduce programming model. Apache Spark has been used as the big data processing and analysis platform since it uses the iterative map-reduce programming model, provides high-performance data parallelism, and is easy to use. The Apache Spark MLlib library has been extended, and the proposed algorithm has been implemented on this library.

On Spark and Hadoop, Sequential Pattern Mining methods are implemented. They are also parallelized to avoid the overhead of loading and serialization [10,15,26,29,30]. Yu et al. have proposed a MapReduce-based distributed GSP (DGSP) algorithm on Hadoop [30]. The DGSP method partitions the dataset and assigns tasks to worker nodes, thus optimizing the balance of workload. Li et al. proposed the Ha-GSP algorithm in Hadoop which uses MapReduce's Divide and Conquer functions [10]. Yu et al. also showed an implementation of the GSP algorithm on Apache Spark. [29]. Several database partitioning methods have been proposed for the unbalanced loading problem. To reduce the input-output problem, stored records on HDFS are loaded to Spark RDDs, the database is read from the RDDs, and the intermediate results are stored on the RDDs.

Parallel PrefixSpan algorithms using Hadoop-based MapReduce operations are proposed by Wei et al. [26] and Sabrina et al. [15]. I/O operations on HDFS are necessary for Hadoop, it causes a high I/O issue. Yu et al. [29] proposed a PrefixSpan algorithm on Spark to reduce load by making it scalable, which partitions the projected database, loads them to RDDs to reduce I/O operations, and uses them from RDDs.

**Algorithm 1.** Proposed-SPMA

**Inputs:** A sequence database (*database*) which is formed like id-transactions, Minimum Support Value (*minsup*)
**Output:** *result* (Sequential Patterns and their counts)
1: Count $C_1$'s for each sequence in *database*
2: $F_1 \leftarrow C_1$'s such that its frequency exceeds the *minsup*
3: $database_k \leftarrow$ Database contains only $F_1$
4: $result \leftarrow F_1$
5: $k \leftarrow 2$
6: **while** $F_k \neq \emptyset$ **do**
7: Generate candidate sets $C_k$ for each sequence
8: Count $C_k$'s for each sequence
9: $F_k \leftarrow C_k$'s such that its frequency exceeds the *minsup*
10: $result \leftarrow result + F_k$
11: $database_k \leftarrow$ Database contains only $F_k$
12: $k \leftarrow k + 1$
13: **end while**

Another parallel algorithm based on PrefixSpan on Spark has been developed by Kim et al. [9]. PrefixSpan-based methods run faster than GSP-based methods as GSP-based scans database more than one and PrefixSpan-based algorithms scan only the databse which is projected last step. Unlike the studies above, this research proposes an algorithm that can work on big data processing and analysis platforms, optimized to detect sequential patterns on large-scale datasets consisting of single-element, sequential datasets.

We observe a number of studies that focus on big data processing in different studies [3,4,6,17,20–23,27,28]. Different from previous work, in this study, we mainly focus on big data processing techniques for sequential data mining algorithms.

## 4 Methodology

This study introduces a sequential pattern mining algorithm that can mine sequential patterns on datasets consisting of single-element sequential and large-scale transactions. The pseudocode of the proposed algorithm is given in Algorithm 1.

The algorithm we propose has been designed and developed inspired by the GSP algorithm, which has been presented in Sect. 3. The algorithm first scans the dataset and identifies candidate patterns to find the sequential patterns. Candidate patterns with a frequency less than the minimum support (*minsup*) value are eliminated. The algorithm, unlike the GSP algorithm, aims to reduce the search space in each iteration. In order to reduce the search space, the values below the *minsup* value are deleted from the database, and new candidate patterns are produced based on this cleaned database. While generating the candidate patterns, the *ID*s are also taken into account. In this way, it enables

the search space to be narrowed by clearing the database from elements below the *minsup* value. In the narrowed search space, possible patterns to be examined in the next iteration are created based on the candidates above the *minsup* value.

In Algorithm 1, a step-by-step explanation of the pseudocode of the proposed algorithm is given:

- The inputs are *database*, a sequential dataset consisting of single-element, and *minsup*, a minimum support value.
- The output is *result*, frequent sequential patterns with their corresponding frequencies.

- In line 1, 1-length subsequences in the dataset are counted.
- In line 2, sequences below the *minsup* value are eliminated, and in line 3, these sequences are also deleted from the dataset. Subsequences above the *minsup* value become the initial values of the output *result*, on line 4.
- In line 6, the loop begins and continues until no new candidate can be created.
- The candidate generation part of the algorithm for each sequence is done in the 7th line. While generating the candidate, all other sequences in the dataset are scanned for each sequence. For each sequence, the first element of the first sequence is deleted, and the last element of the sequence is deleted. If the remaining ones are equal, a new candidate is generated. This process may be explained with the following example:
  - For 2-sequence examples, AB and BC, the first element of the first sequence A is deleted, thus remaining is B. The last element of the second sequence C is deleted, therefore remaining is B, the ABC data series candidate can be generated because $B = B$. But, for AB and CB, since $C \neq B$ is, a new candidate cannot be generated from these two.
- Line 8 shows counting generated candidates, and those below the *minsup* value are eliminated in line 9, and these items are also deleted from the dataset as shown on line 11.
- Candidates that meet the *minsup* value requirement in each loop become the sequential patterns that are looked for, and the output is appended to the *result* value in line 10.
- The value $k$, which specifies the candidate length, is incremented with each iteration.

A textual explanation of the pseudocode given in Algorithm 1 is as follows. The *minsup* value is determined as 2.

The operation of the algorithm is simulated on a dataset consisting of 4 transactions. In each transaction, there are single-element, sequenced data. Their transaction IDs and sequential data in each transaction are as follows: ID-0: B, C, D, F; ID-1:B, D, E, C, F; ID-2: A; ID-3: B, C, D.

The algorithm first calculates the frequency values of single-elements in the dataset. It then eliminates elements with a support value less than 2. It then deletes the eliminated single-elements from the dataset. As a result of this process, in the example given, the next state of the dataset became: ID-0: B, C, D, F; ID-1:B, D, C, F; ID-3: B, C, D.

After this stage, the algorithm performs the following operations on the dataset in a loop.

In the first iteration; As seen in the example, 2-element candidate sequential data arrays were created based on elements. The frequency values of the 2-element candidates on the dataset were calculated. 2-element candidates which are below the support value were eliminated. Using the remaining 2-element frequently recurring sequential data, an updated dataset of 2-element datasets to be used in the next iteration was generated as ID-0: BC, BD, BF, CD, CF, DF; ID-1:BD, BC, BF, DF, CF; ID-3: BC, BD, CD.

In the second iteration, the operations performed in the previous iteration are applied to the dataset consisting of 2-element sequential data. The updated dataset resulting from this iteration is as follows: ID-0: BCD, BCF, BDF; ID-1:BDF, BCF; ID-3: BCD.

In the example, in the third iteration the algorithm exits the loop, since data sequences with a length of 4 elements cannot be created. The output of the algorithm, which shows sequential patterns in the given dataset, are B, C, D, F, BC, BD, BF, CD, CF, DF, BCD, BCF, and BDF.

The time and memory complexity of the proposed algorithm was investigated by experimental tests.

## 5 Experiments

### 5.1 Test Design

An Apache Spark cluster was set up to test the proposed algorithm. This setup consisted of 4 worker nodes and 1 master node. Each worker node consisted of 1 CPU core and 2 GB of RAM, while the master node consisted of 1 CPU core and 3 GB of RAM. This Apache Spark environment was built on a virtual machine. Processor speeds within the virtual machine were dynamically adjusted by the hypervisor. The mainframe includes 16 GB of RAM and an Intel Core i7-10510 processor. For comparison, PrefixSpan, GSP, and the proposed algorithm were implemented on Spark. PrefixSpan is readily available in the PySpark library.

During the tests, the execution times and memory usage of these three algorithms were recorded. In order to obtain more consistent results, each test was performed 10 times, and their standard deviations and averages were calculated.

- Execution time: It measures the time between the start and end times of the algorithm for each run.
- Memory usage: Instant memory usage per second is recorded and averaged during operations on a single node.

### 5.2 Dataset

In the experiments, ChainStore dataset is used. it is a open-source dataset which includes customer purchses from a huge grocery in the United States of America, was used as the dataset. This dataset was obtained from the digital database SPMF [18]. The ChainStore dataset contains 1,112,949 transaction rows, 46,086 unique products, and an average of 7.23 products per transaction.

### 5.3 Results

Figure 1 shows the execution times for the ChainStore dataset of the three algorithms we ran on Apache Spark. These execution times are also classified according to the total number of worker nodes in the running environment. As seen from the obtained results, the proposed algorithm performs better than the other algorithms in terms of speed.

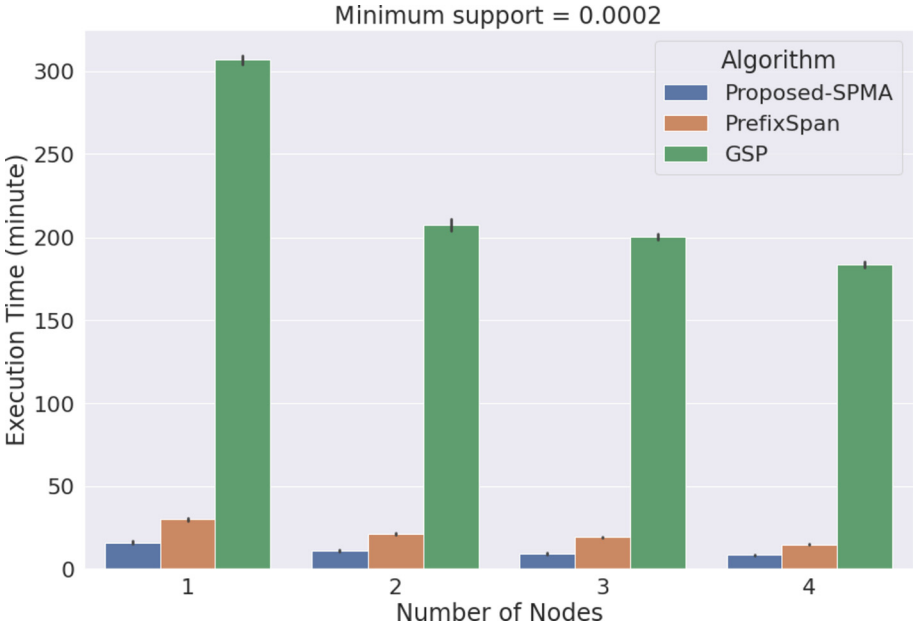


**Fig. 1.** Execution times of algorithms on different number of worker nodes.

Tables 1 and 2 show the minimum, maximum, and average memory usages of the three algorithms for the master and a worker node. The results show that the proposed algorithm requires less memory than GSP but more memory than PrefixSpan.

**Table 1.** Memory usage of a worker node

| Algorithm | Min. | Max. | Mean |
|---|---|---|---|
| Proposed-SPMA | 18% | 80% | 60% |
| GSP | 20% | 85% | 74% |
| PrefixSpan | 20% | 82% | 62% |

In this study, a series of tests were conducted for the proposed algorithm in the Apache Spark environment. In the tests carried out to reveal the running time and memory requirements of the proposed algorithm, the proposed algorithm was compared with GSP and PrefixSpan on a different number of worker nodes in terms of speed and memory usage.

**Table 2.** Memory usage of the master node

| Algorithm | Min. | Max. | Mean |
|---|---|---|---|
| Proposed-SPMA | 21% | 91% | 80% |
| GSP | 20% | 91% | 85% |
| PrefixSpan | 39% | 52% | 49% |

As a result of the experiments, the proposed algorithm performs almost 10 times better in terms of speed performance compared to GSP and also better in terms of memory usage. The reason for this result is that the proposed algorithm narrows the search space at each step. As the search space narrows, candidate generation and finding frequent candidates, which are the most costly parts, take less time. Likewise, as the search space narrows, the average memory usage decreases compared to the GSP.

When the proposed algorithm is compared to PrefixSpan, which is one of the fastest known algorithms of sequential pattern mining, although it is disadvantageous in terms of memory usage, it works almost twice as fast in terms of speed. When the obtained results are evaluated, it is seen that the proposed algorithm runs faster than the PrefixSpan algorithm. Since the dataset used in the tests is a dataset in which the search space can be narrowed easily, this property of the dataset may decrease the running time for our algorithm.

## 6 Conclusion

In this research, a new sequential pattern mining algorithm is proposed that will work on large-scale datasets containing sequential single-elements. The pseudocode of the proposed algorithm is given, and the working logic of the algorithm is shown in the example.

A prototype application has been developed to demonstrate the success of the proposed algorithm. Using this prototype implementation, the performance of the algorithm in terms of accuracy, running time, run-time memory requirement metrics are compared with the algorithms in similar categories.

The results of the experimental studies show that the proposed algorithm performs very high performance in terms of running time requirements for large-scale datasets.

Our future work will include examining the performance of our proposed sequential pattern mining algorithm on different large datasets. In addition, our future research will include a comparative evaluation of the proposed algorithms with other sequential pattern mining algorithms in the literature.

**Acknowledgements.** This study was supported by BiletBank R&D Center. We would like to thank BiletBank for providing us with the necessary hardware and access to their datasets.

## References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proceedings of the 20th International Conference on Very Large Data Bases, VLDB 1994, pp. 487–499. Morgan Kaufmann Publishers Inc., San Francisco (1994). https://doi.org/10.5555/645920.672836
2. Anil, R., et al.: Apache mahout: machine learning on distributed dataflow systems. J. Mach. Learn. Res. **21**, 1–6 (2020)
3. Bahadır, D., et al.: A big data processing framework for self-healing internet of things applications. In: 12th International Conference on Semantics, Knowledge and Grids (SKG) (2016)
4. Burak, C., et al.: Data feature selection methods on distributed big data processing platforms. In: 3rd International Conference On Computer Science And Engineering (2018)
5. Casado, R., Younas, M.: Emerging trends and technologies in big data processing. Concurr. Comput. Pract. Exp. (CCPE) J. **27**(8), 2078–2091 (2015)
6. Duygu, S., et al.: Implementation of association rule mining algorithms on distributed data processing platforms. In: 4th International Conference on Computer Science and Engineering (UBMK) (2019)
7. Fournier-Viger, P., Lin, J.C.W., Kiran, R.U., Koh, Y.S., Thomas, R.: A survey of sequential pattern mining. Data Sci. Pattern Recogn. **1**(1), 54–77 (2017)
8. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. SIGMOD Rec. **29**(2), 1–12 (2000). https://doi.org/10.1145/335191.335372
9. Kim, B., Yi, G.: Location-based parallel sequential pattern mining algorithm. IEEE Access **7**, 128651–128658 (2019)
10. Li, H., Zhou, X., Pan, C.: Study on GSP algorithm based on hadoop. In: 2015 IEEE 5th International Conference on Electronics Information and Emergency Communication, pp. 321–324 (2015)
11. Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., et al.: Mllib: machine learning in apache spark. J. Mach. Learn. Res. **17**(1), 1235–1241 (2016)
12. Mooney, C.H., Roddick, J.F.: Sequential pattern mining-approaches and algorithms. ACM Comput. Surv. (CSUR) **45**(2), 1–39 (2013)
13. Pei, J., et al.: Mining sequential patterns by pattern-growth: the prefixspan approach. IEEE Trans. Knowl. Data Eng. **16**(11), 1424–1440 (2004). https://doi.org/10.1109/TKDE.2004.77
14. Pokou, Y.J.M., Fournier-Viger, P., Moghrabi, C.: Authorship attribution using small sets of frequent part-of-speech skip-grams. In: The Twenty-Ninth International Flairs Conference (2016)
15. Sabrina, P.N., Saptawati, G.P.: Multiple mapreduce and derivative projected database: new approach for supporting prefixspan scalability. In: 2015 International Conference on Data and Software Engineering (ICoDSE), pp. 148–153. IEEE (2015)
16. Sagiroglu, S., Sinanc, D.: Big data: a review. In: 2013 International Conference on Collaboration Technologies and Systems (CTS), pp. 42–47 (2013)

17. Secil, Y., et al.: On the performance analysis of map-reduce programming model on in-memory nosql storage platforms: a case study. In: International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT) (2018)
18. Spmf an open-source data mining library. http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php, Accessed 15 Sept 2021
19. Srikant, R., Agrawal, R.: Mining sequential patterns: generalizations and performance improvements. In: Apers, P., Bouzeghoub, M., Gardarin, G. (eds.) EDBT 1996. LNCS, vol. 1057, pp. 1–17. Springer, Heidelberg (1996). https://doi.org/10.1007/BFb0014140
20. Tas, Y., et al.: An approach to standalone provenance systems for big social provenance data. In: 12th International Conference on Semantics, Knowledge and Grids (SKG) (2016)
21. Tufek, A., et al.: On the provenance extraction techniques from large scale log files. In: Concurrency And Computation-Practice & Experience (Early Access) (2021) https://doi.org/10.1002/cpe.6559
22. Uzun-Per, M., Gürel, A.V., Can, A.B., Aktas, M.S.: An approach to recommendation systems using scalable association mining algorithms on big data processing platforms: A case study in airline industry. In: 2021 International Conference on INnovations in Intelligent SysTems and Applications (INISTA), pp. 1–6. IEEE (2021)
23. Uzun-Per, M., Can, A.B., Gürel, A.V., Aktas, M.S.: Big data testing framework for recommendation systems in e-science and e-commerce domains. In: 2021 IEEE International Conference on Big Data (Big Data), pp. 2353–2361. IEEE (2021)
24. Uzun-Per, M., Gurel, A.V., Can, A.B., Aktas, M.S.: Scalable recommendation systems based on finding similar items and sequences. Concurr. Comput. Pract. Exp., e6841 (2022)
25. Wang, J., Han, J., Li, C.: Frequent closed sequence mining without candidate maintenance. IEEE Trans. Knowl. Data Eng. **19**(8), 1042–1056 (2007)
26. Wei, Y.Q., Liu, D., Duan, L.S.: Distributed prefixspan algorithm based on mapreduce. In: 2012 International Symposium on Information Technologies in Medicine and Education, vol. 2, pp. 901–904 (2012)
27. Yasin, U., et al.: Technical analysis on financial time series data based on map-reduce programming model: a case study. In: International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT) (2018)
28. Yasin, U., et al.: On the large-scale graph data processing for user interface testing in big data science projects. In: 8th IEEE International Conference on Big Data (Big Data) (2020)
29. Yu, X., Li, Q., Liu, J.: Scalable and parallel sequential pattern mining using spark. World Wide Web **22**(1), 295–324 (2018). https://doi.org/10.1007/s11280-018-0566-1
30. Yu, X., Liu, J., Liu, X., Ma, C., Li, B.: A mapreduce reinforced distributed sequential pattern mining algorithm. In: International Conference on Algorithms and Architectures for Parallel Processing, pp. 183–197 (2015)
31. Zaharia, M., et al.: Apache spark: a unified engine for big data processing. Commun. ACM **59**(11), 56–65 (2016). https://doi.org/10.1145/2934664
32. Zaki, M.J.: Spade: an efficient algorithm for mining frequent sequences. Mach. Learn. **42**, 31–60 (2004)
33. Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W.: New algorithms for fast discovery of association rules. In: The Third International Conference on Knowledge Discovery and Data Mining (KDD-97), pp. 283–286. AAAI Press, Newport Beach (1997)