An Empirical Evaluation of Sequential Pattern Mining Algorithms

Marjana Prifti Skenduli[1](✉), Corrado Loglisci[2], Michelangelo Ceci[2], Marenglen Biba[1], and Donato Malerba[2]

[1] University of New York Tirana, Tirana, Albania
{marjanaprifti,marenglenbiba}@unyt.edu.al
[2] Universita' degli Studi di Bari, Bari, Italy
{corrado.loglisci,michelangelo.ceci,donato.malerba}@uniba.it

Abstract. Sequence mining is one of the most investigated tasks in data mining and it has been studied under several perspectives. With the rise of Big Data technologies, the perspective of efficiency becomes prominent especially when mining massive sequences. In this paper, we perform a thorough experimental evaluation of several algorithms for sequential pattern mining and we provide an analysis of the results focusing on the different algorithmic choices and how these affect the performance of each algorithm. Experiments performed on real-world and synthetic datasets highlight relevant differences between existing algorithms and provide indications for Big Data scenarios.

1 Introduction

Sequences are elements arranged according to a total or partial ordering. They are very common in many real-world scenarios. For instance, click-streams and trajectories. Click-streams represent sequences produced by the web browsing activity of a user, the web pages visited by the user represent the elements of these sequence, while the order is established by the time-stamp when a web page is visited. Trajectories are sequences of geo-referenced positions produced by the movement, while the order is established by the motion of the moving objects [8]. The order is not necessarily related to time, but also to space. Likewise, in textual documents and biological studies, sequences are series of chars or series of nucleotides whose order is based on the position that they have with respect to the other elements. In many applications, the elements denote complex entities and often represent sets of single basic elements where no order relation holds. Market basket analysis is one of these applications, where a set of items corresponds to a purchase of a customer in a mall, while a sequence represents the series of purchases made in a week. Analyzing sequences can thus become profitable and advantageous for many real-life applications and one of the technologies adopted is represented by *Sequential pattern mining* (SPM), which comprises techniques for mining sequential data and discovering interesting sub-sequences in a set of sequences [4].

L. Barolli et al. (Eds.): EIDWT 2018, LNDECT 17, pp. 615–626, 2018.
https://doi.org/10.1007/978-3-319-75928-9_55

The blueprint for sequence mining algorithms proposed in the literature is enumerating all the interesting sub-sequences in the space of all the possible sub-sequences. Typically the notion of interestingness relies on the relative frequency, which provides statistical evidence to a sequential pattern and thus provides arguments about the regularity of a sub-sequence in a database [10]. Therefore, sub-sequences that have a frequency greater than a user-defined threshold are those selected as valid sequential patterns. To discover all the frequent sub-sequences, we should explore the whole search space by means of a generate-and-test strategy that builds all possible sub-sequences and tests the frequency against the minimum threshold.

This problem has been often tackled by approaches specifically designed for selected domains (e.g., [2,9,14,18]), while others have a more general characterization (e.g., [5,6,15]. However, regardless of the domain or specific approach, most of research studies mainly focus on three specific algorithmic features and on how these can be effectively and efficiently developed: *(i)* the method to explore the search space, *(ii)* the representation of the database of sequences, and *(iii)* the generation of the sequential patterns. Although, there are several theoretical studies and surveys [4,11,12], we ascertain the lack of empirical studies. An experimental viewpoint that highlights the characteristics of the three algorithmic features may be helpful when facing the sequence mining problem in the context of Big Data scenarios, where the necessity for methods able to analyze time-ordered and unbounded data produced at high rate becomes more and more pressing.

In this paper, we investigate how those three features have been developed in representative algorithms and propose a comparative evaluation on real and synthetic sequence datasets. Our contribution is not a theoretical discussion, but it should be intended as an empirical study that complements experiments presented in papers of specific, relevant sequential pattern mining algorithms.

This paper is organized as follows. In the next section, we introduce classical definitions and necessary notions to understand the problem of SPM and existing solutions to solve it. In Sect. 3, we discuss the most representative SPM algorithms by illustrating the core algorithmic decisions behind them. Then, in Sect. 4, we present the empirical evaluation upon real and synthetic datasets, further discussing on how the three features aforementioned work. Finally, the conclusions drawn in Sect. 5 mark the closure of this paper.

2 Background and Basics

SPM has originally been formalized in [15] and the subsequent research has inherited the same formulation, which revises one of the association rules mining problem.

Let $I : \{i_1, i_2, \ldots, i_m\}$ a set of elements with nominal values, termed *items*. An itemset X is a unordered set of items such that $X \subseteq I$, the cardinality of X, $|X|$ corresponds to the number of contained items. Without loss of generality, we assume that items of an itemset are sorted in lexicographic order. An itemset

X of cardinality $|X| = k$ is said to be of length k. For example, given the set of items $I : \{a, b, c, d, e, f\}$, the set $\{a, b, c\}$ has length 3 and consists of the items a, b and c.

A *sequence* is an ordered set of itemsets $s = \langle I_1, I_2, \ldots, I_n\rangle$, such that $I_h \subseteq I$ $(1 \leq h \leq n)$. A sequence s of cardinality $|s| = n$ is said to be of length n. An example of sequence is $s = \langle\{a, b\}, \{c\}, \{f, g\}, \{g\}, \{e\}\rangle$, where it is assumed an order relation which establishes the itemset $\{a, b\}$ to precede the itemset $\{c\}$. For instance, in the scenario of the click-stream, by assuming the order relation holding on the hours, the itemset $\{a, b\}$ indicates a set of two web-pages a and b visited in the same hour, while the itemset $\{c\}$ indicates a set consisting of the sole page c visited in a subsequent hour.

A *sequence database* is a list of sequences $SDB = \langle s_1, s_2, \ldots, s_p\rangle$, each assigned to an identifier. Sequential patterns are mined by searching the occurrences of sequences in a database SDB and computing their *support*. The *support* of a sequence s_a is the number of sequences of a database SDB that contain s_a. If this value exceeds a user-defined minimum threshold of support, denoted as *minSUP*, the sequence s_a is considered *frequent* and identified as valid sequential pattern. To check if a sequence $s_b : \langle B_1, B_2, \ldots, B_m\rangle$ contains a sequence $s_a : \langle A_1, A_2, \ldots, A_n\rangle$, we search for a series of integers $1 \leq i_1 < i_2 < \ldots < i_n \leq m$ such that $A_1 \subseteq B_{i1}$, $A_2 \subseteq B_{i2}, \ldots, A_n \subseteq B_{in}$.

The exploration of the search space, common to all the sequential pattern mining algorithms, consists on mining valid sequential patterns based on the shorter sequential patterns, starting with sequential patterns of length 1. This is typically done by means of two basic operations which extend the sequences by inserting either one itemset or one item. These two operations are termed *s-extension* and *i-extension*. A sequence $s_b : \langle I_1, I_2, \ldots, I_m, \{x\}\rangle$ is a s-extension of a sequence $s_a : \langle I_1, I_2, \ldots, I_m\rangle$ if s_a is a *prefix* of s_b and the item x appears in an itemset occurring after all the itemsets of s_a. A sequence $s_a : \langle A_1, A_2, \ldots, A_n\rangle$ is a prefix of a sequence $s_b : \langle B_1, B_2, \ldots, B_m\rangle$ if $n < m$, $A_1 = B_1, A_2 = B_2, \ldots, An - 1 = Bn - 1$ and A_n is equal to the first $|A_n|$ items of B_n according to the lexicographic order on the items. A sequence $s_c : \langle I_1, I_2, \ldots, I_m \cup \{x\}\rangle$ is a i-extension of a sequence $s_a : \langle I_1, I_2, \ldots, I_m\rangle$ if s_a is a prefix of s_c, the item x is appended to the last itemset of s_a and the item x is the last one in I_m according to the lexicographic order. Next we report two examples for illustration purposes. The sequences $\langle\{a\}, \{a\}\rangle$, $\langle\{a\}, \{b\}\rangle$ and $\langle\{a\}, \{c\}\rangle$ are s-extensions of the sequence $\langle\{a\}\rangle$. The sequences $\langle\{a, b\}\rangle$ and $\langle\{a, c\}\rangle$ are i-extensions of the sequence $\langle\{a\}\rangle$.

3 Approaches for Sequential Pattern Mining

Before illustrating the comparative analysis, we present and discuss the most representative solutions for implementing the three features that characterize the algorithms of SPM. The features are *(i)* method to explore the search space, *(ii)* representation of the database of sequences, and *(iii)* generation of the sequential patterns. Although, these are not independent aspects from each other when designing an algorithm, thereby we try to inspect them separately highlighting their properties. The detailed description of specific algorithms is not subject of the current paper, so the interested reader may refer to the relative papers.

Exploration of the search space. The space of all the possible sequences, where searching the sequential pattern, can be modeled as a lattice, which is a partially-ordered set, whose elements are the patterns defined on the set of the items (I) present in the database. The space is organized by levels, the patterns of a level are prefixes of the patterns of the next level and have the same length. The patterns of a level are thus obtained as s-extensions or i-extensions of the patterns of the previous level, so their length is increased by one. The exploration of the lattice can be done in two different ways, *breadth-first* search and *depth-first* search. The breadth-first search follows a level-wise strategy, in that it visits a level of the lattice at time and processes all the patterns of a level before considering the next level. Once a level has been visited, it will not be explored again. Procedurally, the search first considers all the patterns of length 1, then it visits the next level to process all the patterns of length 2 and, subsequently, the level with the patterns of length 3. It follows this strategy until it reaches the longest patterns. The most representative algorithm implementing the breadth-first search is GSP [15].

The depth-first search explores the lattice in depth and processes the patterns of a level without necessarily completing that level. When it reaches the leaves, that is, when there are no patterns, it backtracks to the first level (patterns of length 1) and re-starts visiting the remaining patterns of the levels which it had previously visited. Procedurally, the search first considers all the patterns of length 1, then it takes one and processes one pattern of length 2 originated by the pattern of length 1. Subsequently, the pattern of length 2 is used to process one pattern of length 3 of the next level. This procedure is recursively performed until no pattern can be visited. The most representative algorithms implementing the depth-first search are PrefixSpan [13], SPADE [17] and SPAM [1].

The exploration of the lattice of the sequential patterns is costly, especially when the number of items is very large, considering that we should generate a set of patterns with magnitude order equal to 2^i from a database with i elements. To solve this problem and make the exploration efficient, the algorithms (regardless of the space search technique) have implemented pruning techniques aiming at removing sub-spaces that could contain uninteresting patterns. The most used technique relies on the *anti-monotonicity* property of the support, according to which we can avoid to generate the sequence s_a if there exists a sequence s_b, which is contained in s_a, whose support does not exceed the minimum threshold $minSUP$. The sub-space containing sequences longer than s_a can therefore be pruned, since those sequences will not exceed the threshold (intuitively, if a sequence s_a is not frequent, then all the sequences which contain s_a will be not frequent).

Representation of the database. The representation format of the input sequences becomes relevant when counting the number of the occurrences of a pattern in the database. There are three main solutions, (i) horizontal databases, (ii) vertical databases, (iii) projected databases. In the horizontal format, we transform the original transaction database SDB in a list of sequences ordered by identifier. For each sequence, the itemsets are sorted by relation order (e.g., time). This way, in order to count the occurrences of a pattern, we should match the itemsets of the pattern against those of a sequence of the database SDB. In case of GSP implementation, this solution requires to access the database a number of times equal to the number of input sequences, which may significantly raise the time consumption, especially in massive databases.

In the vertical format, we transform the original transaction database in a set of lists (*IDLists*), each associated to one item. A list indicates the itemsets of the input sequences where the corresponding item occurs. These structures are built by accessing only once the database, exactly at the beginning of the process when mining the frequent items. It is not necessary repeating the access operation because the number of the occurrences of the patterns of length greater than 1 is determined by joining the $IDlists$ of the frequent items. This solution is particularly effective in the algorithms that perform depth-first search, for instance SPAM, but it losses efficiency when the IDlists are very large, as typically encountered in dense databases and databases with very long sequences. A popular optimization approach is to encode IDLists as bit vectors [1].

Projected databases are subsets of the database SDB and they provide the means to reduce the search space. They are built simultaneously with the mining process and contain the only input sequences in which a pattern, which has been previously mined, occurs. More precisely, once mined the patterns of length k, for each pattern s_a we scan the database to create a (reduced) database with the only sequences in which s_a is present, while counting the occurrences. Recursively, new databases are created with the sequences in which the patterns s_b, built from the pattern s_a ($s_a \subseteq s_b$), are present. This representation allows us to work on the sequences really appearing in the database, but it has the disadvantage of repeatedly scanning the databases previously created.

Generation of frequent sequential patterns. The generation of sequential patterns is a procedure that consists of the operative steps to build the lattice of the patterns, which we explore through space search methods. The existing works differ on the use of the *generate-and-test* strategy, which is defined in terms of two main steps, (i) generation of candidate patterns and (ii) selection of the only candidates that appear in the input database and that meet the threshold $minSUP$. A candidate pattern of length k is generated by joining two frequent patterns of length $k-1$ that have $k-2$ itemsets in common. The other two items are used to reach the length k. More precisely, we first find all the frequent patterns of length 1, then we generate those of length 2 by using those of length 1. This step goes on until no longer patterns can be generated. Therefore, the patterns of the level k of the lattice cannot be built if we have not completed the level $k-1$. This explains why the generate-and-test strategy is often coupled with breadth-first search, as in the GSP algorithm [15]. The generate-and-test strategy has two main limitations. First, it may generate candidates which do not appear in the database. In fact, they are derived from the patterns present in the lattice and not from the sequences contained in the database. This may clearly affect the efficiency of the mining process. Second, it is necessary to store all the patterns of a level prior to building the candidates of the next level. In turn, this may require huge memory.

Alternative solutions have been designed in order to (i) avoid generating candidates that do not appear in the database and (ii) work on a smaller search space. There is a category of algorithms that resorts to the depth-first search in order to generate a candidate from one frequent pattern, which is taken from those previously mined [1,17]. More precisely, once mined the frequent items (length 1), the candidates of length 2 are built by appending an itemset (by means of the operations s-extension and i-extension) to one frequent item, then recursively one more itemset is added to the frequent pattern previously mined until no further itemsets can be appended. The procedure re-starts with another frequent item, with which patterns of increasing length can be built by appending one itemset at time. This kind of algorithms holds the advantage of generating patterns of length k by keeping only one pattern of length $k-1$, contrarily to the generate-and-test strategy. Extensions to this solution have been addressed to make the candidate generation efficient. In [3], the authors upgrade the algorithms SPADE [17] and SPAM [1] in order to avoid infrequent candidates. More precisely, they propose a preliminary step in which sequential patterns of length 2 are discovered. These are used later to eliminate the candidates in which the items of length 2 patterns are not present.

Another category of algorithms combines depth-first search and projected databases. They extend the frequent patterns, mined in previously created data bases, by using the items present in the newly created databases as suffixes or prefixes for longer patterns [7,13]. This way, they avoid building uninteresting candidates and early prune (sub)spaces of the lattice, achieving a two-fold result (i) utilizing much less memory and (ii) keeping the mining process focused only on those subsets of the database which can give frequent patterns.

4 Empirical Evaluation

To empirically evaluate the differences between the three main features described above, in this section we present experiments conducted on some algorithms that adopt different solutions to implement the three features. To this end, we considered GSP, PrefixSpan, SPAM, Spade and CM-Spade. More detailed description of these algorithms can be found in [11,12]. In order to perform a fair analysis, we used the SPMF framework [16], which collects many sequential pattern mining algorithms implemented in Java using the same design pattern. The experiments aim at evaluating the time consumption (in milliseconds) and memory consumption (in MB).

We used real-word datasets and synthetically generated sequence datasets. As to the real-world datasets, the experiments were performed by manually tuning the minimum threshold $minSUP$. Three categories of real-world sequences

Table 1. Characteristics of the real-world datasets used for the experiments.

Dataset	No. of sequences	No. of distinct items	Avg no. of itemsets per sequence	Density
Sign	730	267	52	0,0037
Bible	36369	13905	21,64	7,19E−05
Kosarak	69999	21144	7,98	4,72E−05
Msnbc	989818	17	4,75	0,058
Fifa	20450	2990	36,24	0,00033
Pumsb	49046	2088	50,48	0,00048

were considered: textual data, click-streams and census data. In the textual data, sequences correspond to sentences, while items correspond to words. The structure of the discourse defines the order between the words. Two datasets of textual data were used. *Sign*, which contains transcriptions of sign language utterances. *Bible*, which contains the transcription of Bible. In the context of click-stream data, sequences correspond to sessions of browsing on the Web, while the items correspond to Web pages. The order is defined in terms of the time-stamp when clicking on the page link. Three datasets were used. *Kosarak*, which contains sequences of click-stream data from an hungarian news portal. *Msnbc*, which contains click-stream data collected from logs of www.msnbc.com and news-related portions of www.msn.com for the entire day of September 28, 1999. *Fifa*, which contains the data of the requests made to the 1998 World Cup Web site between April 30, 1998 and July 26, 1998. In the census data, sequences correspond to population and demographic statistics collected across time. We used the dataset *Pumsb* that contains federal census data collected for the IPUMS project from Los Angeles – Long Beach area for the years 1970, 1980, and 1990. The real-world datasets are available at the link http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php. The experiments were performed on a machine equipped with Windows 10 operating system, i3 3.3 GHz processor and 8 GB main memory. Table 1 reports a summary of the characteristics of these datasets. The density variable is obtained as the average number of items present in the itemsets divided by the number of distinct items.

As to the synthetically generated sequences, we used the datasets available at the link http://www.di.uniba.it/~ceci/micFiles/systems/CloFAST/ (also used in [5]). The experiments were performed by manually regulating one characteristic of the data at a time while leaving the others fixed. In particular, three characteristics were considered: average number of itemsets per sequence, density and number of sequences (denoted as C, T, D afterwards). The experiments were executed on a machine equipped with CentOS Linux operating system, Intel Xeon 2.4 GHz processor and 64 GB main memory.

In Fig. 1, we report the time spent by the algorithms on the real-world datasets. As expected, the larger the value of *minSUP* the larger the time consumption (the scales are logarithmic). In particular, the running time of GSP crosses at least three magnitude orders for all the datasets, while the algorithms SPAM, PrefixSpan and CM-Spade cross two magnitude orders on Sign, Bible, Kosarak and three orders on Fifa and Pumsb. The performance related to running time on Fifa and Pumsb can be attributed to a particular conjunction of the properties of these datasets, they have simultaneously many items (no. of distinct items), many itemsets per sequence (no. of distinct items) and large density, compared to the other datasets. When these characteristics are not present simultaneously, the running time is smaller, for instance the plots of SPAM and PrefixSpan algorithms on Kosarak and Sign are basically linear. Indeed, Kosarak has small sets of itemsets per sequence, while Sign has small sets of distinct items. A specific consideration can be drawn for SPAM on Kosarak and Bible. Although their respective running time has not an exponential tendency, the magnitude orders are greater compared to the other algorithms, including GSP. This can be explained with the data representation implemented in SPAM, where a bit vector is associated with each item. Considering that Kosarak and Bible have the larger sets of distinct items, the running time of SPAM can be affected by the cost of the operations for building the bit vectors.

The memory consumption (Fig. 2) follows the tendency of the results on the running time, although it increases quite linearly. The algorithms that take more memory space are those compliant with the generate-and-test procedure, such as, GSP and SPAM, because they keep the frequent patterns previously mined. This is more evident in GSP, where the breadth-first search forces us to keep the patterns of the levels of the lattice already visited. In particular, GSP needs more memory (and more running time) on Sign, Fifa and Msnbc for small values of $minSUP$ because the lattice grows up significantly. On the contrary, the choice of PrefixSpan to keep smaller search spaces is successful in terms of running time and memory consumption. In the case of SPAM, it consumes memory on Kosarak and Bible because it needs a huge number of bit vectors. The output of CM-Spade deserves a specific consideration. In Bible and Kosarak, we have no result because CM-Spade stops due to insufficient memory. This is quite expected because the two datasets have the maximum number of distinct items (compared to the other datasets) and CM-Spade uses all the memory because it has to perform a preparatory operation on the itemsets of length 2 (Sect. 3). On the contrary, it performs faster on the datasets with lower number of items (Sign, Msnbc and Pumsb), although it uses more memory compared to other choices, for instance, PrefixSpan.

The experiments on the synthetic datasets have been set on the properties of the input data sequences, which we can control. Figures 3a and d illustrate the running time and memory usage obtained while increasing values of C ($minSUP = 0.4$). We note that when C (average number of itemsets per sequence) increases, the time consumption increases too and for all the algorithms the efficiency drops of four orders of magnitude. The explanation is that the value of C affects the length of the patterns and consequently the size of the lattice. Thus, when C increases, we need to perform more s-extension operations, which lead the width and depth of the lattice to grow. This is particularly

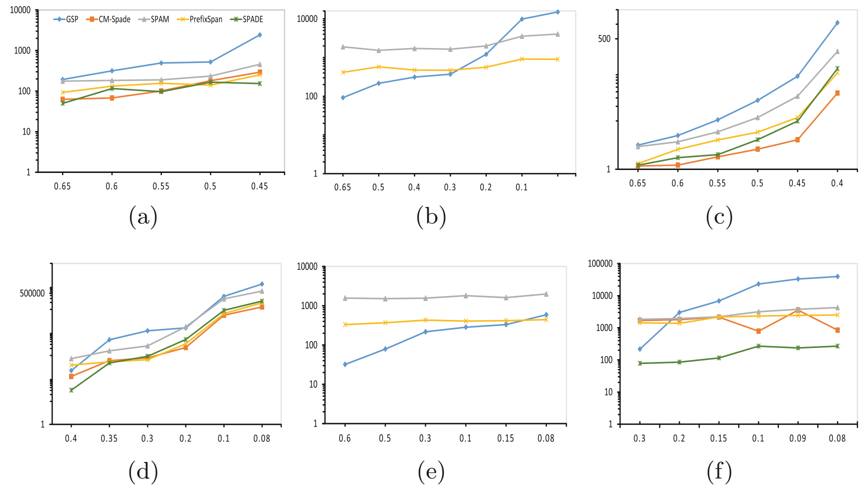

Fig. 1. Running time (in milliseconds) obtained on the real-world datasets (a) Sign, (b) Bible, (c) Pumsb, (d) Fifa, (e) Kosarak, (f) Msnbc.

evident on the algorithms that use the level-wise search, and candidate generation techniques, for instance, GSP, because they have to evaluate all the candidates of a level (which will be larger because there are more itemsets) before proceeding to the next level. A different behavior can be observed on the memory consumption, which grows linearly with C. This reveals a choice common to many algorithms, that is, limiting the use of the main memory at the cost of the running time. The exception is represented by CM-Spade, which asks for less time, but uses more memory for keeping the itemsets of length 2, especially for large values of C.

The performances obtained on T (density) follow those of C. With respect to running time, all the algorithms cross four orders of magnitude when T increases (Figs. 3b and e). The reason is that when the sequences are densest, the number of items present in the itemsets increases and the algorithms need more i-extension operations. This makes the lattice "more complex" and the pattern generation procedures costly. As to the memory, it is worth noting that the consumption of Prefix-Span is greater than GSP, especially for large densities. This is due to the fact that the size of the projected databases is not smaller than the input database for densest sequences.

The results obtained by varying D (Figs. 3c and f) allow us to evaluate the scalability properties of the algorithms, whose running time follows an exponential tendency. The algorithm GSP has at least one more magnitude order, compared to the others, and this is due to the algorithmic choice on the access and representation of the input sequences. In GSP, the computation of the occurrences is performed by scanning the whole database, hence the data representation relies on traditional transaction-based format, which requires costly I/O operations. Contrarily, PrefixSpan does not scan the whole database, but only partitions of the input sequences and this guarantees better efficiency. This gain

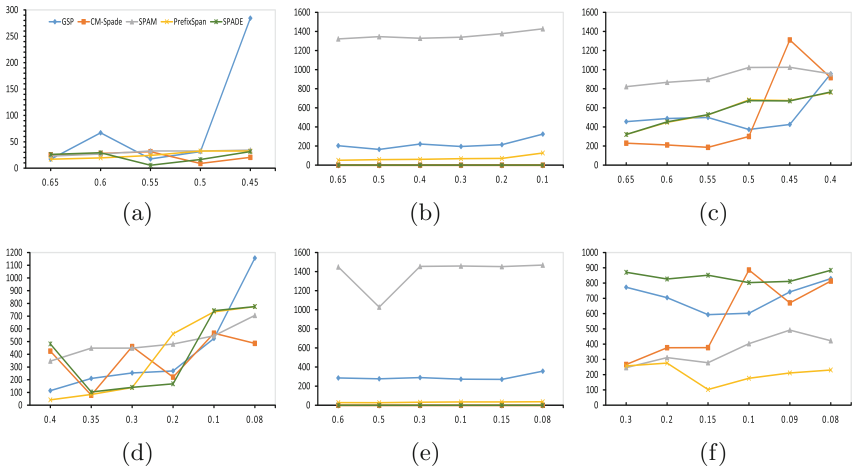

Fig. 2. Memory consumption (in MB) obtained on the real-world datasets (a) Sign, (b) Bible, (c) Pumsb, (d) Fifa, (e) Kosarak, (f) Msnbc.

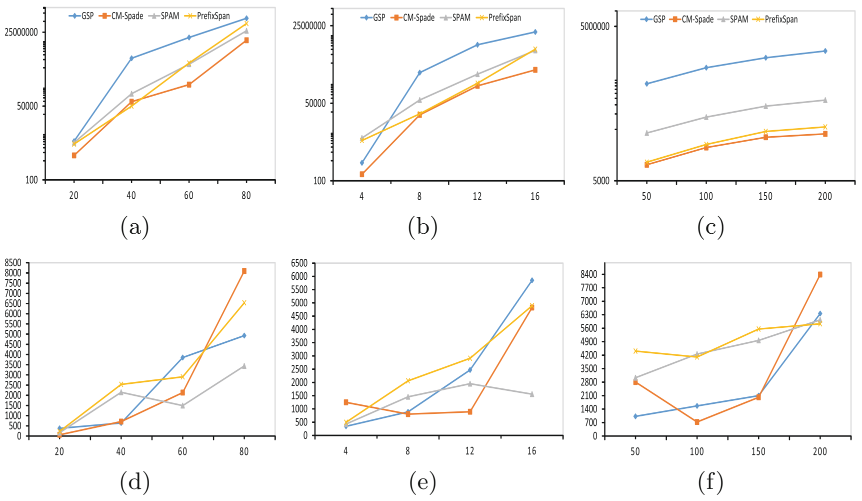

Fig. 3. Running time and memory consumption (in MB) obtained on the synthetic datasets (a), (d) Number of itemsets per sequences, (b), (e) Density and (c), (f) Number of sequences.

in running time implies a greater usage of the memory, as seen for the results of C. In the case of PrefixSpan, we need to create a large number of projected databases containing probably more sequences, while, in the case of SPAM and CM-Spade, we need very long IDlists. Finally, GSP asks for less because it uses memory essentially to store the lattice, whose size is related to the size of the input database.

5 Conclusions

In this work, we have conducted an experimental study to compare most representative algorithms designed for sequence mining. The main contribution lies in the empirical evaluation of the choices done in those algorithms for the *(i)* exploration of the lattice of the patterns, *(ii)* representation of the database of sequences, and *(iii)* generation of sequential patterns. This work can be intended as a preliminary investigation for the future design of a sequence mining approach on Big Data, considering solutions for data distribution and/or parallelization on different machines, including streaming approaches. Indeed, our evaluation, performed on real-world and synthetic datasets, has been set to draw indications on the performance in terms of time and memory consumption. We observed that the choices done to make the mining process efficient, for instance, in PrefixSpan and CM-Spade, imply greater memory consumption. Contrarily, solutions which are time consuming, for instance GSP, may perform relatively well in terms of main memory requirements. As future work, we plan to (i) investigate advanced data structures to adapt time-saving algorithms (for instance, PrefixSpan and CM-Spade) and (ii) consider distributed solutions for the generation of patterns in order to adapt memory-saving algorithms (for instance, GSP).

References

1. Ayres, J., Flannick, J., Gehrke, J., Yiu, T.: Sequential pattern mining using a bitmap representation. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2002, New York, NY, USA, pp. 429–435. ACM (2002)
2. Cheng, Y., Lin, Y., Chiang, K., Tseng, V.S.: Mining sequential risk patterns from large-scale clinical databases for early assessment of chronic diseases: a case study on chronic obstructive pulmonary disease. IEEE J. Biomed. Health Inform. **21**(2), 303–311 (2017)
3. Fournier-Viger, P., Gomariz, A., Campos, M., Thomas, R.: Fast vertical mining of sequential patterns using co-occurrence information. In: Advances in Knowledge Discovery and Data Mining - 18th Pacific-Asia Conference, PAKDD 2014, Proceedings, Part I, Tainan, Taiwan, 13–16 May 2014, pp. 40–52 (2014)
4. Fournier-Viger, P., Lin, J.C.-W., Kiran, R.U., Koh, Y.S.: A survey of sequential pattern mining. Data Sci. Pattern Recognit. **1**(1), 54–77 (2017)
5. Fumarola, F., Lanotte, P.F., Ceci, M., Malerba, D.: Clofast: closed sequential pattern mining using sparse and vertical id-lists. Knowl. Inf. Syst. **48**(2), 429–463 (2016)
6. Ge, J., Xia, Y., Wang, J., Nadungodage, C.H., Prabhakar, S.: Sequential pattern mining in databases with temporal uncertainty. Knowl. Inf. Syst. **51**(3), 821–850 (2017)
7. Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., Hsu, M.: Freespan: frequent pattern-projected sequential pattern mining. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Boston, MA, USA, 20–23 August 2000, pp. 355–359 (2000)
8. Loglisci, C.: Using interactions and dynamics for mining groups of moving objects from trajectory data. Int. J. Geograph. Inf. Sci. 1–33 (2017)

9. Loglisci, C., Ceci, M., Impedovo, A., Malerba, D.: Mining spatio-temporal patterns of periodic changes in climate data. In: New Frontiers in Mining Complex Patterns - 5th International Workshop, NFMCP 2016, Held in Conjunction with ECML-PKDD 2016, Riva del Garda, Italy, 19 September 2016, Revised Selected Papers, pp. 198–212 (2016)
10. Loglisci, C., Ceci, M., Malerba, D.: Relational mining for discovering changes in evolving networks. Neurocomputing **150**, 265–288 (2015)
11. Mabroukeh, N.R., Ezeife, C.I.: A taxonomy of sequential pattern mining algorithms. ACM Comput. Surv. **43**(1), 3:1–3:41 (2010)
12. Mooney, C., Roddick, J.F.: Sequential pattern mining - approaches and algorithms. ACM Comput. Surv. **45**(2), 19:1–19:39 (2013)
13. Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., Hsu, M.: Mining sequential patterns by pattern-growth: the prefixspan approach. IEEE Trans. Knowl. Data Eng. **16**(11), 1424–1440 (2004)
14. Schweizer, D., Zehnder, M., Wache, H., Witschel, H.F., Zanatta, D., Rodriguez, M.: Using consumer behavior data to reduce energy consumption in smart homes: applying machine learning to save energy without lowering comfort of inhabitants. In: 14th IEEE International Conference on Machine Learning and Applications, ICMLA 2015, Miami, FL, USA, 9–11 December 2015, pp. 1123–1129 (2015)
15. Srikant, R., Agrawal, R.: Mining sequential patterns: generalizations and performance improvements. In: Apers, P.M.G., Bouzeghoub, M., Gardarin, G. (eds.) Advances in Database Technology - EDBT 1996, 5th International Conference on Extending Database Technology, Proceedings, Avignon, France, 25–29 March 1996, vol. 1057. Lecture Notes in Computer Science, pp. 3–17. Springer (1996)
16. Viger, P.F., Gomariz, A., Gueniche, T., Soltani, A., Wu, C., Tseng, V.S.: SPMF: a java open-source pattern mining library. J. Mach. Learn. Res. **15**(1), 3389–3393 (2014)
17. Zaki, M.J.: SPADE: an efficient algorithm for mining frequent sequences. Mach. Learn. **42**(1/2), 31–60 (2001)
18. Ziebarth, S., Chounta, I., Hoppe, H.U.: Resource access patterns in exam preparation activities. In: Design for Teaching and Learning in a Networked World - 10th European Conference on Technology Enhanced Learning, EC-TEL 2015, Proceedings, Toledo, Spain, 15–18 September 2015, pp. 497–502 (2015)