ADVANCED PATTERN MINING: THE EVALUATION OF SEQUENTIAL PATTERN MINING ALGORITHMS

İbrahim Erdem KALKAN[1]

ABSTRACT

As a problem, in order to extract sequential patterns from a sequence database, there are plenty of algorithms introduced so far. They use different techniques in terms of scanning database, support counting, etc. It is possible to obtain different performance measurement with different sequential pattern mining algorithms and different size of data. In this paper it is tried to measure the efficiency of three algorithms: SPADE, PrefixSpan, and CM-SPADE. To do this, it is used three different real data set from UCI Machine Learning Repository: 1) MSNBC, 2) Online Retail, 3) DNA Sequence, with an open source data mining tool, SPMF, specialized in pattern mining. It is obvious that each algorithm have its own advantages and drawbacks. Whereas some features seem to be advantageous for a specific type of data set, some are disadvantageous. It is obtained that the runtime efficiency of one algorithm not only depends upon the data set characteristics, but also minimum support threshold values are also significant impact on process times.

Keywords: Frequent Pattern Mining, Sequential Pattern Mining, PrefixSpan, SPADE, CM-SPADE, Efficiency of Algorithms

[1] Cukurova University, Institute of Natural and Applied Sciences, Industrial Engineering, 2020911063

1. Introduction

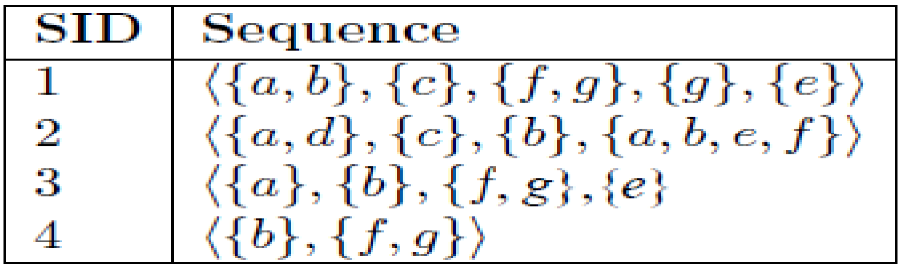
Since, Agrawal and Srikant (1994), introduced the “Apriori” algorithm, pattern mining has become an emerging area with many kind of application. There has been a lot of work in the field of data mining about pattern mining. The goal of pattern mining is to discover useful, novel, or unexpected patterns in databases (Fournier-Viger et al, 2017).

Even though pattern mining has become very popular, it does not take into account the sequential ordering of events. This may result in the failure to discover important patterns in the data, or finding patterns that may not be useful because they ignore the sequential relationship between events or elements. To address this issue, the task of sequential pattern mining was proposed (Fournier-Viger et al, 2017).

Sequential pattern mining was first introduced by Agrawal and Srikant (1995), and it was considered among advanced pattern mining techniques by Han et al. (2007), with multilevel, multidimensional frequent pattern mining, maximal and closed pattern mining, high-dimensional and colossal pattern mining.

Agrawal and Srikant (1995), have been defined sequential pattern mining based on customer transactions: Given a database D of customer transactions, the problem of mining sequential patterns is to find the maximal sequences among all sequences that have a certain user specified minimum support. Each such maximal sequence represents a sequential pattern.

Example. A sequence database contains some sequences like following example shown in Fig – 1 taken from Fournier-Viger et al, (2017).



SID	Sequence
1	$\langle\{a,b\},\{c\},\{f,g\},\{g\},\{e\}\rangle$
2	$\langle\{a,d\},\{c\},\{b\},\{a,b,e,f\}\rangle$
3	$\langle\{a\},\{b\},\{f,g\},\{e\}$
4	$\langle\{b\},\{f,g\}\rangle$

Fig – 1. A sequence database containing four sequences

A sequence database *S* is a list of sequences *S* = <*sid, s*> having sequence identifiers (SIDs) 1, 2 … p. These sequences could, for example, represent purchases made by four customers. If a sequence s_a is contained in a sequence s_b, s_a is said to be a subsequence of s_b. For example, the sequence <*{b}{f,g}*> is contained in sequence <*{a,b}{c}{f,g}{g}{e}*> while the sequence <*{b}{g}{f}*> is not.

The support of a sequence s_a in a sequence database *S* is defined as the number of sequences that contain s_a, and is denoted by $sup(s_a)$. For example, the support of the sequence <*{b}{f,g}*> in the database of Fig – 1 is 3 because this sequence appears in three sequences (Sequence 1,2 and 4).

Sequential pattern mining, which discovers frequent subsequences as patterns in a sequence database, is an important data mining problem with broad applications, including the analysis of customer purchase patterns or web access patterns, the analysis of sequencing or time-related processes such as scientific experiments, natural disasters, and disease treatments, the analysis of DNA sequences, etc.(Pei et al, 2004).

In the following chapter, it is introduced three algorithms of SPADE, PrefixSpan, and CM-SPADE respectively with a brief of information from literature. In Ch. 3, the test environment and measuring methods are briefly introduced. Ch. 4 contains experiment results and finally it is taken place final words in Ch. 5.

2. Finding Sequential Patterns

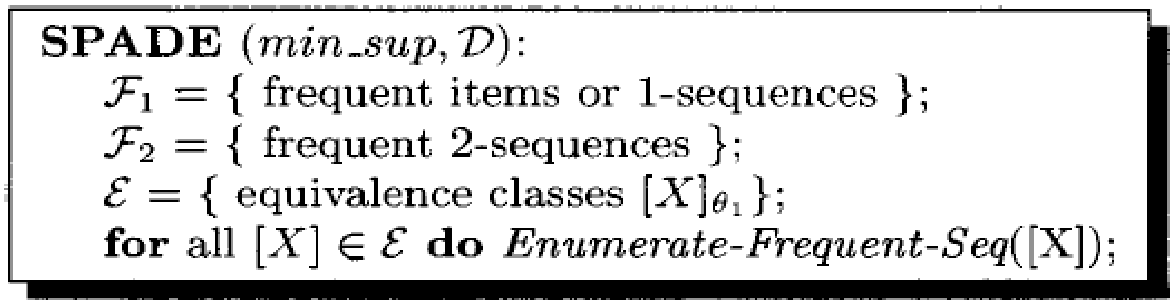
There are plenty of algorithms have been designed to discover sequential patterns in sequence databases. Some of the most popular are GSP, SPADE, PrefixSpan, SPAM, Lapin, CM-SPAM, and CM-SPADE. In general, sequential pattern mining algorithms differ in 1) whether they use a depth-first or breadth-first search, 2) the type of database representation that they use internally or externally, 3) how they generate or determine the next patterns to be explored in the search space, and 4) how they count the support of patterns to determine if they satisfy the minimum support constraint. (Fournier-Viger et al, (2017).

2.1. SPADE

An algorithm was presented by Zaki, (2001), to discover all frequent sequences set called SPADE (Sequential Pattern Discovery Using Equivalence Classes). Three key features were remarked in that paper.

1. It uses vertical id-list database format. Temporal join on id-lists is also the key concept.
2. It uses lattice-theoretic[2] approach. To minimizing search space databases or lattices are split into small pieces or sub-lattices.
3. It proposes two different search strategies: Depth-first and breadth-first search.

It is shown the high-level structure of the algorithm in Fig – 2. A vertical database format contains sequence id and event id pairs to identify item occurrence.



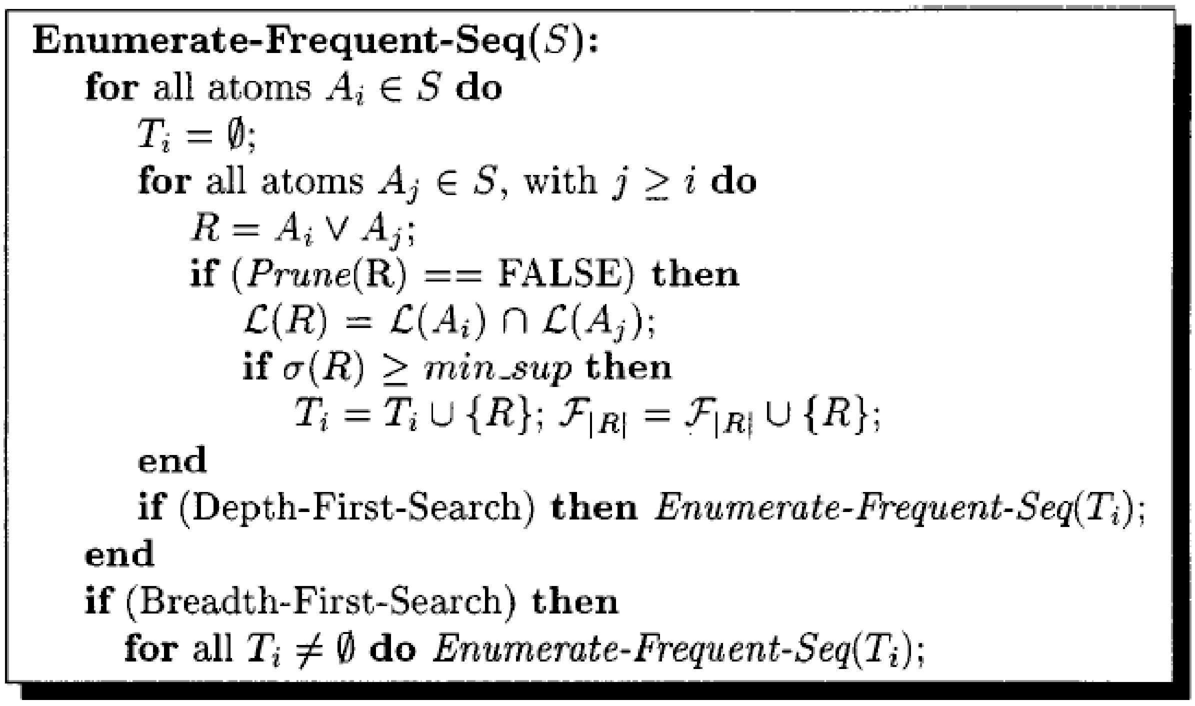
```
SPADE ($min\_sup, \mathcal{D}$):
    $\mathcal{F}_1$ = { frequent items or 1-sequences };
    $\mathcal{F}_2$ = { frequent 2-sequences };
    $\mathcal{E}$ = { equivalence classes $[X]_{\theta_1}$ };
    for all $[X] \in \mathcal{E}$ do Enumerate-Frequent-Seq([X]);
```

Fig – 2. High level structure of SPADE (Zaki, 2001)

First, to calculate frequent *1-sequences (F1)* one scan of database is needed. It can be read from the disc-based vertical database the sequence ids of each item. Second, to compute frequent *2-sequences (F2)* it proposes 2 alternate steps instead of naïve methods in order to minimize search space. 1) Using a user specified lower bound or, 2) performing a vertical-to-horizontal transformation on-the-fly.

[2] For details about using lattices must see the Zaki, 2001. Davey and Priestley, 1990 is also pointed for a good introduction.

It can be seen pseudocode of third step enumerating frequent sequences of a class in Fig – 3 where S represents a sub-lattice, and all atoms represents all items in S.



```
Enumerate-Frequent-Seq($S$):
    for all atoms $A_i \in S$ do
        $T_i = \emptyset$;
        for all atoms $A_j \in S$, with $j \geq i$ do
            $R = A_i \vee A_j$;
            if ($Prune$(R) == FALSE) then
                $\mathcal{L}(R) = \mathcal{L}(A_i) \cap \mathcal{L}(A_j)$;
                if $\sigma(R) \geq min\_sup$ then
                    $T_i = T_i \cup \{R\}$; $\mathcal{F}_{|R|} = \mathcal{F}_{|R|} \cup \{R\}$;
        end
        if (Depth-First-Search) then Enumerate-Frequent-Seq($T_i$);
    end
    if (Breadth-First-Search) then
        for all $T_i \neq \emptyset$ do Enumerate-Frequent-Seq($T_i$);
```

Fig – 3. Pseudocode for enumerating frequent sequences of a class (Zaki, 2001)

Frequent sequences are generated by joining the id-lists of all pairs of atoms (including a self-join) and checking the cardinality of the resulting id-list against *min_sup.* Before joining the id-lists a pruning step can be inserted to ensure that all subsequences of the resulting sequence are frequent. The sequences found to be frequent at the current level form the atoms of classes for the next level. This recursive process is repeated until all frequent sequences have been enumerated.

Although, SPADE supports a pruning step, it has been observed that pruning was not efficient in some experiments. On the other hand, pruning can eliminate a lot of unnecessary candidates in large sequences. Thus, the algorithm can be used with a disabled pruning step.

2.2. PrefixSpan

PrefixSpan (Prefix-projected sequential pattern mining) is an efficient sequential pattern mining algorithm based on the pattern-growth principle was introduced by Pei et al. (2004). According to paper it follows that philosophy:

“Sequence databases are recursively projected into a set of smaller projected databases based on the current sequential pattern(s), and sequential patterns are grown in each projected databases by exploring only locally frequent fragments.”

Prefix, suffix, and projected-database are key concepts of PrefixSpan. According to paper, the definition of the prefix:

Suppose all the items within an element are listed alphabetically. Given a sequence a = <e_1 e_2 e_3 ...e_n> (where each e_i corresponds to a frequent element in a sequence database S), a sequence b = <e'_1 e'_2 e'_3 ...e'_m> (m<n) is called a prefix of a if and only if 1) (e'_i =

e_i) for ($i \leq m$-1);2) e'_m is subset of e_m; 3) all the frequent items in e_m are alphabetically after those in e'_m.

And the suffix:

Given a sequence a = <e_1 e_2 e_3 ...e_n> (where each e_i corresponds to a frequent element in a sequence database S), let b = <e_1 e_2 e_3 ... e_{m-1} e'_m> (m<n) be the prefix of a. the sequence c = <e''_m e_{m+1} ... e_n> is called the suffix of a with regards to prefix b, denoted as c = a/b where $e''_m = (e_m - e'_m)$.

For example, for the sequence *<a(abc)(ac)d(cf)>; <a>, <aa>, <a(ab)>, <a(abc)>* are prefixes but neither *<ab>* nor *<a(bc)>* is considered as a prefix. *<(abc)(ac)d(cf)>* is suffix with regards to prefix *<a>, <(_bc)(ac)d(cf)>* is suffix with regards to prefix *<aa>, and <(_c)(ac)d(cf)>* is suffix with regards to prefix *<a(ab)>*.

Projected-database is another important definition:

Let "a" be sequential pattern in a sequence database S, the a-projected database denoted as S|a,is the collection of suffixes of sequences in S with regards to prefix a.

Therefore, based on the above definitions, the algorithm of PrefixSpan is presented as follows:

Method: Call *PrefixSpan (< >, 0, S)*

Subroutine: *PrefixSpan (a, l, S|a)*

The parameters are 1) "*a*" is a sequential pattern; 2) *l* is the length of *a*; and 3) *S|a* is the *a*-projected database if *a* ≠ < >, otherwise, it is the sequence database *S.*

Method:

1. Scan *S|a* once, find each frequent item, *b*, such that

   (a) *b* can be assembled to the last element of "*a*" to form a sequential pattern; or

   (b) *<b>* can be appended to "*a*" to form a sequential pattern.

2. For each frequent item *b*, append it to *a* to form a sequential pattern "*a'*", and output "*a'*"

3. For each "*a'*", construct *a'*-projected database *S|a'*, and call *PrefixSpan(a', l+1, S|a')*

2.3. CM – SPADE

Despite, using vertical database algorithms are more efficient generating candidate sequences, there is still a high costs to generate-and-test candidates approach. Fournier-Viger et al, (2014) targeted to reduce this cost by developing a pruning technique based on *Co-occurrence Map* structure (CMAP), integrating three algorithms that use vertical database format: SPADE, SPAM, and ClaSP. They introduced three new algorithms CM-SPADE, CM-SPAM, and CM-ClaSP corresponding each of the state-of-the-art algorithms. According to their experimental results CM-SPADE is the best algorithm with regards of performance.

According to Fournier-Viger et al. (2014), to define the co-occurrence map it is required two new definitions of *i-extension* and *s-extension*.

An item k is said to succeed by i-extension to an item j in a sequence <I_1 I_2 I_3 ...I_n> if and only if j, k ∈ Ix for an integer x such that $1 \leq x \leq n$. An item k is said to succeed by s-extension to an item j in a sequence <I_1 I_2 I_3 ...I_n> if and only if j ∈ I_v and k ∈ I_w for some integers v and w such that $1 \leq v < w \leq n$. A Co-occurrence MAP (CMAP) is a structure mapping each item k ∈ I to a set of items succeeding it. We define two CMAPs named CMAPi and CMAPs. CMAPi maps each item k to the set $cm_i(k)$ of all items j ∈ I succeeding k by i-extension in no less than min-sup sequences of SDB. CMAPs maps each item k to the set $cm_s(k)$ of all items j ∈ I succeeding k by s-extension in no less than min-sup sequences of SDB.

Example. The CMAP structures built for the sequence database of Fig – 1 are shown in Fig – 4 being CMAPi on the left part and CMAPs on the right part. Both tables have been created considering a *min-sup* of two sequences. For instance, for the item *f*, it can be seen that it is associated with an item, $cm_i(f) = \{g\}$, in CMAPi, whereas it is associated with two items, $cm_s(f) = \{e, g\}$, in CMAPs. This indicates that both items e and g succeed to f by s-extension and only item g does the same for i-extension, being all of them in no less than *min-sup* sequences.

$CMAP_i$		$CMAP_s$	
item	is succeeded by (i-extension)	item	is succeeded by (s-extension)
a	$\{b\}$	a	$\{b, c, e, f\}$
b	$\emptyset$	b	$\{e, f, g\}$
c	$\emptyset$	c	$\{e, f\}$
e	$\emptyset$	e	$\emptyset$
f	$\{g\}$	f	$\{e, g\}$
g	$\emptyset$	g	$\emptyset$

Fig – 4. CMAPi and CMAPs structure for the database of Fig – 1. (Fournier-Viger et al, 2014)

Before integration to SPADE the following properties have been introduced:

Let be a frequent sequential pattern A and an item k. If there exists an item j in the last item-set of A such that k belongs to $cm_i(j)$, then the i-extension of A with k is infrequent. Let be a frequent sequential pattern A and an item k. If there exists an item j ∈ A such that the item k belongs to $cm_s(j)$, then the s-extension of A with k is infrequent. To generalize, let be a frequent sequential pattern A and an item k. If there exists an item j ∈ A (equivalently j in the last item-set of A) such that there is an item k ∈ $cm_s(j)$ (equivalently in $cm_i(j)$), then all super-sequences B having A as prefix and where k succeeds j by s-extension (equivalently i-extension to the last item-set) in A in B are infrequent.

It can be seen in the Fig – 3 the pseudo-code of SPADE algorithm. To integrate properties above, in the *Enumerate-Frequent-Seq* procedure, consider a pattern *r* obtained by merging two patterns $Ai = P \cup x$ and $Aj = P \cup y$, being *P* a common prefix for *Ai* and *Aj*. Let *y* be the item that is appended to *Ai* to generate *r*. If *r* is an i-extension, it can be used the CMAPi structure, otherwise, if *r* is an s-extension, it can be used CMAPs. If the last item *a* of *r* does not have an item $x \in cm_i(a)$ (equivalently in $cm_s(a)$), then the pattern *r* is infrequent and *r* can be immediately discarded, avoiding the join operation to calculate the support of *r*.

3. Method

In this paper it is tried to measure the performance and compare the efficiency of three algorithms: SPADE, PrefixSpan, and CM-SPADE. To do this, it is used a free open source data mining tool, SPMF, and three different real data from UCI Machine Learning Repository[3]: 1) MSNBC, 2) Online Retail, and 3) DNA Sequence.

A methodology of data mining or so called "knowledge discovery process" is introduced by Han et al. (2007), as an iterative sequence of steps. Based on their methodology, it is shown in the fig-5 the steps which is to be followed in this work. Although it is seen the implementation alternative as generic, an implemented tool is used in this work instead re-implementation of an algorithm by coding.

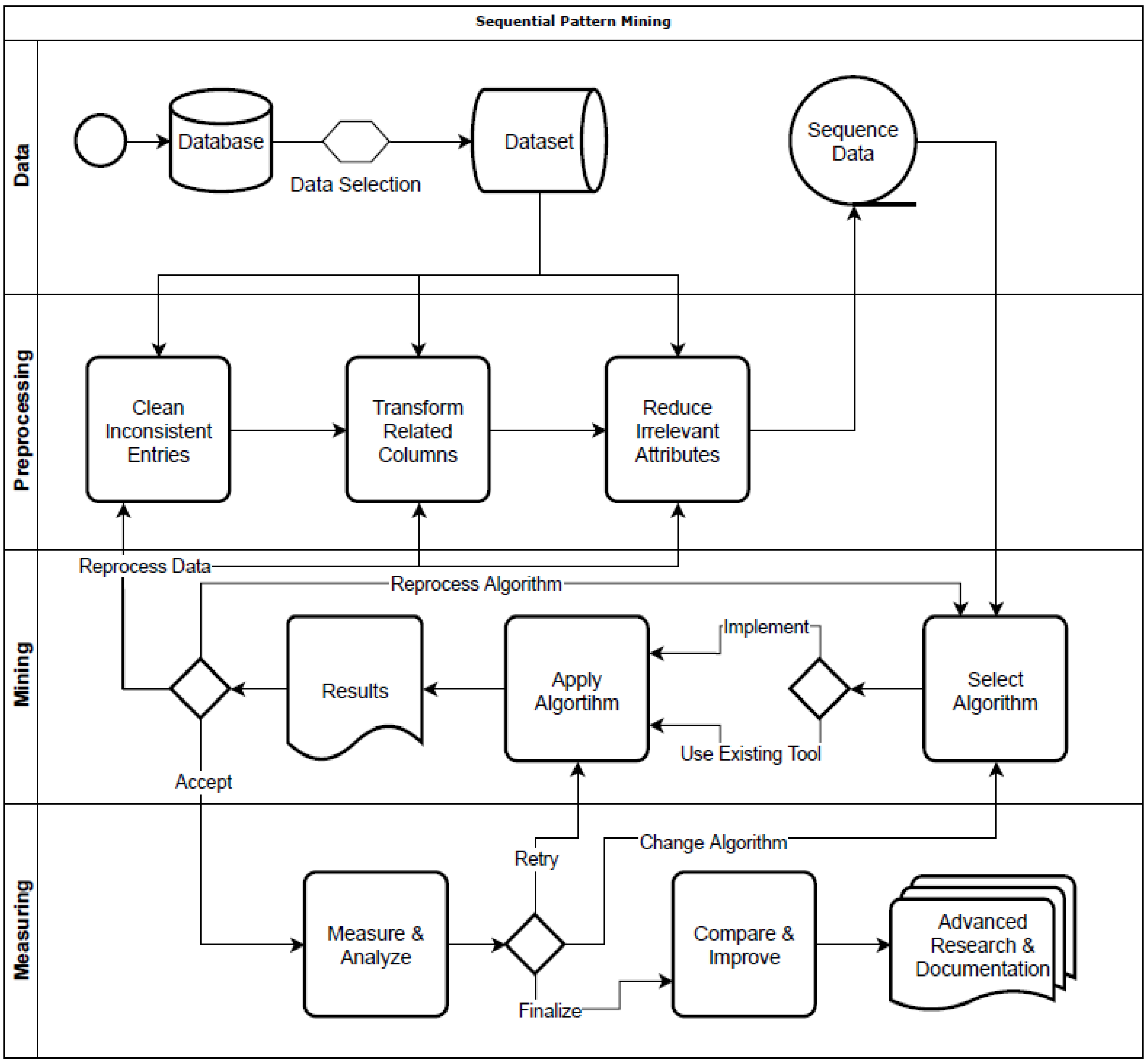


Fig – 5. The workflow knowledge discovery process

[3] [http://archive.ics.uci.edu/ml/index.php]

3.1. Datasets

The first dataset have been used is msnbc.com (MSNBC) anonymous web data (Heckerman, n.d.). This data describes the page visits of users who visited msnbc.com on September 28, 1999. Each sequence in the dataset corresponds to page views of a user during that twenty-four hour period. Each event in the sequence corresponds to a user's request for a page. Records are at the level of page category. The categories are *"frontpage", "news", "tech", "local", "opinion", "on-air", "misc", "weather", "health", "living", "business", "sports", "summary", "bbs" (bulletin board service), "travel", "msn-news", and "msn-sports".*

MSNBC data file includes web click data sequences in terms of integers corresponding page category belongs to 989.818 visitor. For example a record from data is like “3 2 2 4 2 2 2 3 3” represents that a visitor first clicks 3 (tech), and then clicks 2 (news), and then clicks again 2 (news) and so on...

The second dataset is called “Online Retail” (Chen et al., 2012). It is a transnational data set which contains all the transactions occurring between 01.12.2010 and 09.12.2011 for a UK-based and registered non-store online retail. The company mainly sells unique all-occasion gifts. Many customers of the company are wholesalers.

Online Retail includes entries of 541.909 transactions which belong to different customers in terms of “CustomerID”, but some transactions don’t have any customer ID. Therefore, in the data preprocessing section some transactions which don’t contain any customer info had to be eliminated.

The third dataset is E. coli promoter gene sequences (DNA Sequence) with associated imperfect domain theory (Reynolds and Harley, 1987; Towell et al, 1990). According to UCI Repository the data set is described as: *“a dataset has been developed to help evaluate a "hybrid" learning algorithm (KBANN) that uses examples to inductively refine preexisting knowledge.”*

DNA Sequence includes 106 instances with 59 attributes. The attributes of 3-59 contain 57 fields filled by one of {a, g, t, c} corresponding a nucleotide. In this paper the last 57 attributes have been used by creating a sequence. Therefore, the dataset contains 106 sequences and each sequence is 57 item long.

3.2. Tool

As a data mining tool, SPMF: An Open-Source Data Mining Library (Fournier-Viger et al., 2016) have been used for testing algorithms. SPMF is a free open source data mining library containing the implementations of many data mining algorithms, written in Java, specialized in pattern mining**.** It is distributed under the GPL v3 license. It is available to download from “philippe-fournier-viger.com”.

The tool uses two single parameter as input: sequence file and a minimum support threshold. The support of a sequential pattern is the number of sequences where the pattern occurs divided by the total number of sequences in the database. The minimum support threshold (min-sup) is a user-specified ratio like 0.4 or 0.1.

The input file of the tool must be a text file where each line represents a sequence from a sequence database. Each item from a sequence is a positive integer and items from the same item set within a sequence are separated by single space. Note that it is assumed that items within a same item set are sorted according to a total order and that no item can appear twice in the same item set. The value "-1" indicates the end of an item set. The value "-2" indicates the end of a sequence (it appears at the end of each line). For example, the input file for using PrefixSpan algorithm contains the following four lines (four sequences):

```
1 -1 1 2 3 -1 1 3 -1 4 -1 3 6 -1 -2
1 4 -1 3 -1 2 3 -1 1 5 -1 -2
5 6 -1 1 2 -1 4 6 -1 3 -1 2 -1 -2
5 -1 7 -1 1 6 -1 3 -1 2 -1 3 -1 -2
```

The first line represents a sequence where the item set {1} is followed by the item set {1, 2, 3}, followed by the item set {1, 3}, followed by the item set {4}, followed by the item set {3, 6}. The next lines follow the same format.

The output file format of the tool is also a text file. Each line in the output is a frequent sequential pattern with a support value. The sequencing of each item and item set is as it is in the input file.

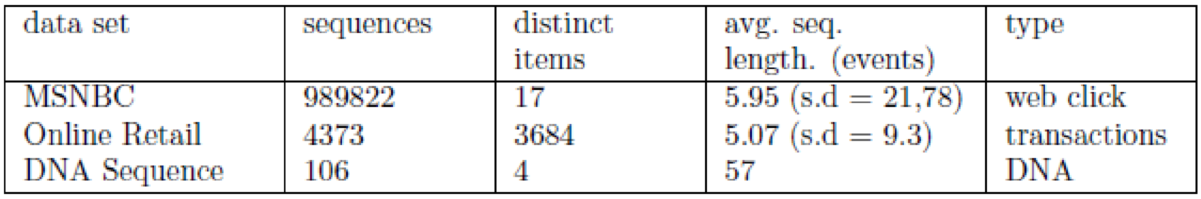
3.3. Preprocessing

Since SPMF uses specific type of data and document as input, all data sets are had to be preprocessed. Some data sets also have inconsistent entries, and some null cells. Due to the data format of MSNBC is nearly ready for process, it is not applied any preprocessing step but column transformation. After that step MSNBC contains 989,822 sequences and the longest sequence contains 6,553 events.

The original data set of Online Retail contains 541,910 rows. However, since it is deleted all 135,080 rows that have no *CustomerID*, it shrinks to 406,830 rows. At first step of data transformation, all *StockCode* values are transformed into unique integers. Second, the unique values of *InvoiceNo* attribute are handled as events and the unique values of *StockCode* (integers) attribute are handled as items. Third, by iterating through data set the new data frame is produced with column *CustomerID*, and *Sequences* made of *InvoiceNo* as events, and *StockCode* as items. The preprocessed Online Retail contains 4,373 sequences which belongs to each customer. The longest sequence contains 248 events.

The DNA Sequence contains 106 rows made of 57 items. All rows of the data set has the same length. It is applied only that data structure transformation and deletion of irrelevant columns. After all preprocessing steps are committed all data sets characteristics can be summarized in Table – 1. It can be seen that, whereas MSNBC has the highest number of sequences, DNA Sequence includes the longest sequences.

Table – 1. Data set characteristics



data set	sequences	distinct items	avg. seq. length. (events)	type
MSNBC	989822	17	5.95 (s.d = 21,78)	web click
Online Retail	4373	3684	5.07 (s.d = 9.3)	transactions
DNA Sequence	106	4	57	DNA

3.4. Execution of Experiments

All experiments have been performed in the 2.4 GHz laptop running a Windows10 operating system with Intel(R) Core(TM) i7-5500U CPU and 5.3 GB free memory. Each experiment consists of two independent trials, the results are read and averages are calculated from SPMF output files.

4. Experimental Results and Discussion

4.1. Frequent Sequence Distribution

Figs – 6, 7, 8, show the distribution of frequent sequences of the data sets for different support threshold values. The minimum support threshold range have been chosen by trial-and-error manner in order to extract more meaningful information with respect to data set characteristics. For example, with the data set DNA Sequence, it is obtained over 1 million frequent sequences when the minimum support threshold value is chosen below 90 %. Moreover, execution times of algorithms can grow easily into the non-reasonable ranges for the data sets such as DNA Sequences.

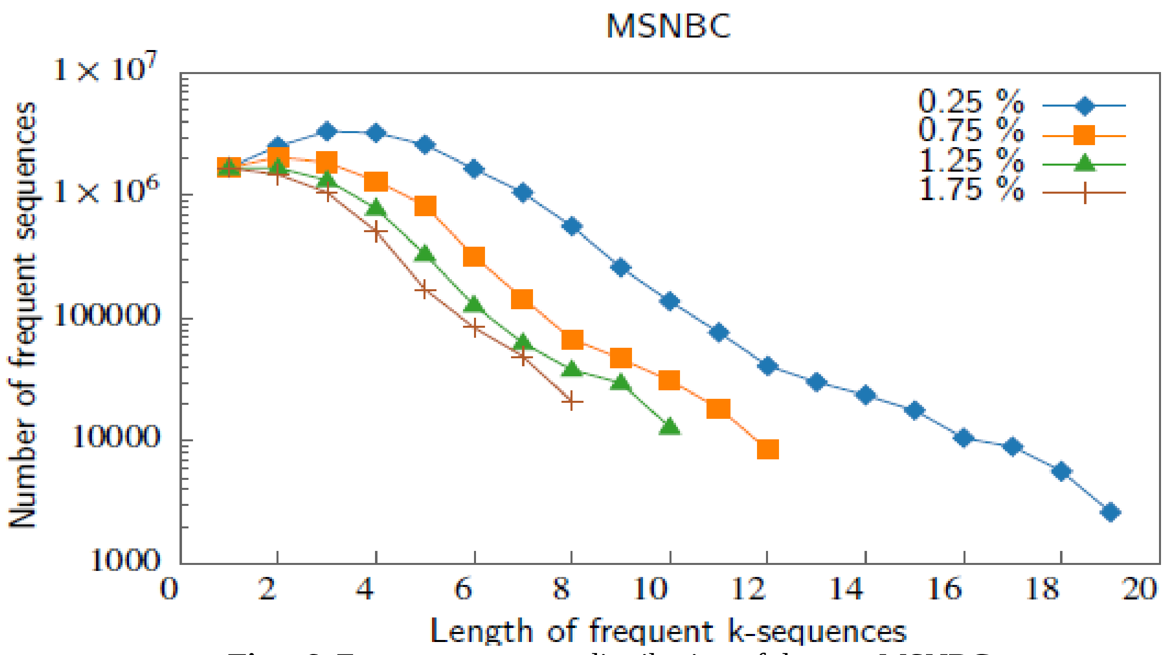


Fig – 6. Frequent sequence distribution of data set MSNBC

It can be seen that algorithms generate higher number of frequent sequences for the decreasing threshold values. For chosen threshold values, the data set of MSNBC has the longest frequent sequences. On the other hand it is obtained huge number of frequent sequences, and higher number of longer frequent sequences with the data set of DNA Sequence, since it has longer sequences to mine comparing the others.

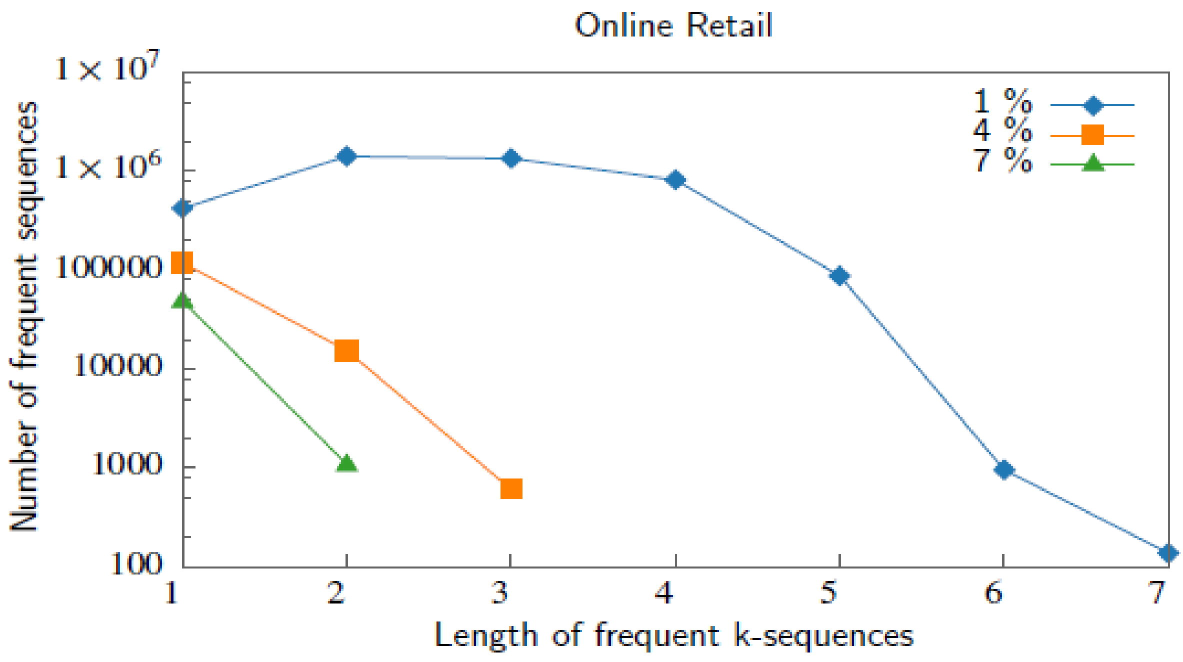


Fig – 7. Frequent sequence distribution of data set Online Retail

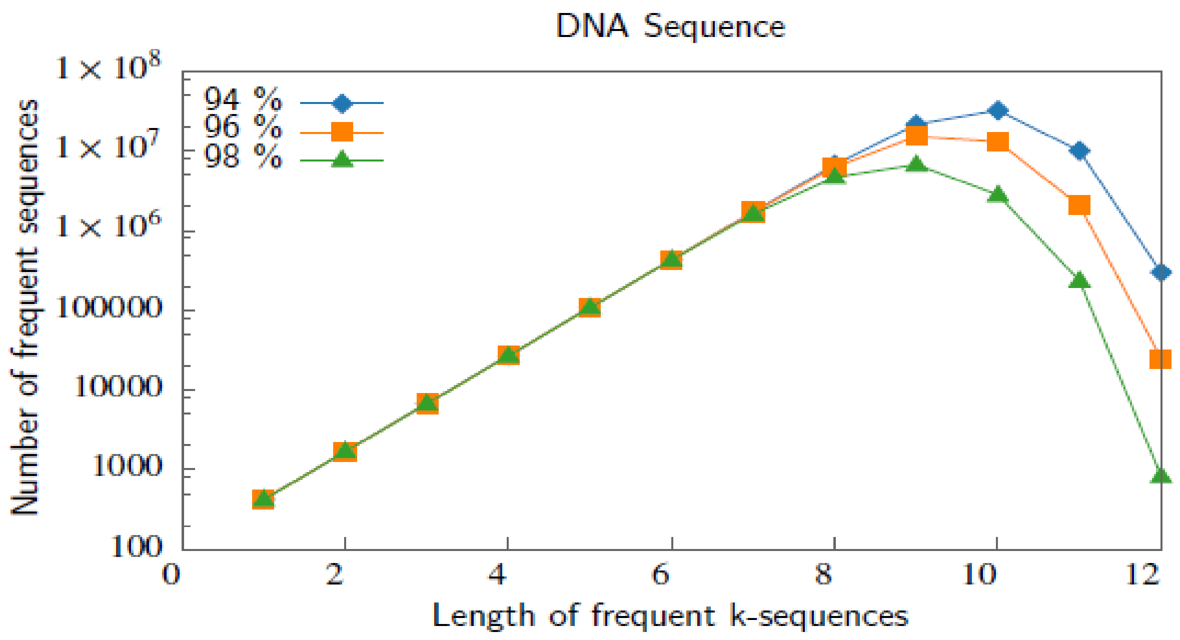


Fig – 8. Frequent sequence distribution of data set Online Retail

4.2. Process Time

The first test is performed in terms of execution time of the three algorithms on the each data set. It is represented in the Fig – 9 that the runtime measurement values of the three algorithms as milliseconds, and the increasing values of *minsup* as the ratio of the number of the frequent sequences. With the MSNBC data set, it is obtained that the process times of each algorithm could be sorted as PrefixSpan < CM-SPADE < SPADE.

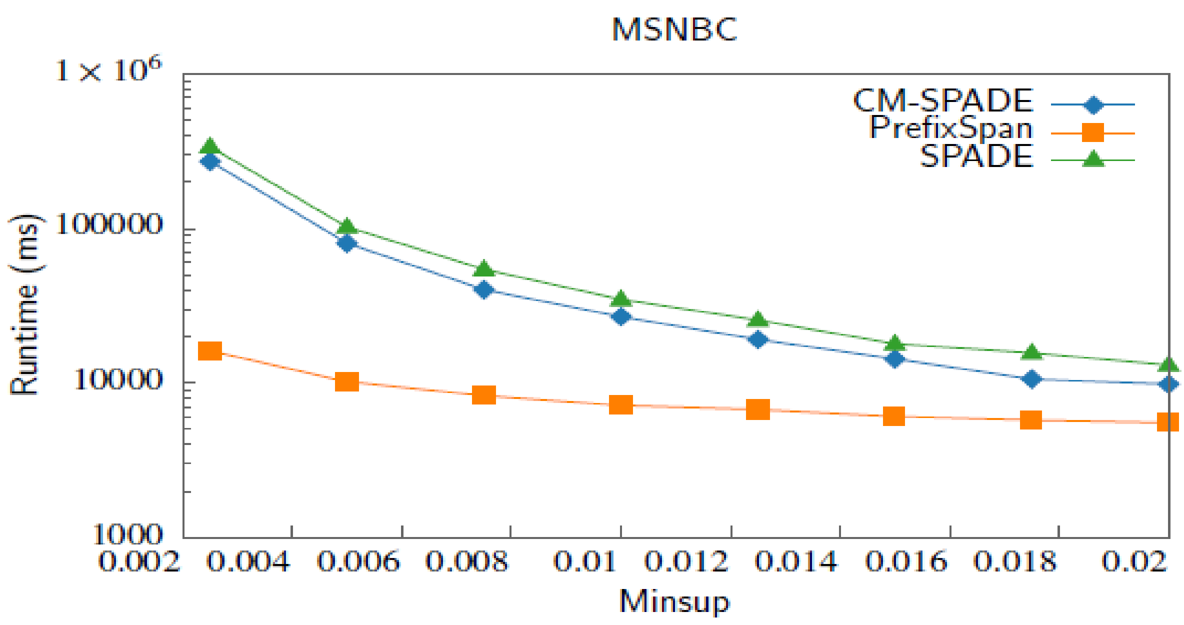


Fig – 9. Execution times on the data set MSNBC

Whereas with the small *minsup* values PrefixSpan has had the best performance radically, with higher *minsup* values other algorithms have caught its performance. Furthermore, with all *minsup* values CM-SPADE is shown slightly better performance than SPADE. With 2 % *minsup* value, while PrefixSpan completes the execution in 2,767 milliseconds, the others complete the execution in 4,906 and 6,533 milliseconds respectively. When *minsup* value has the last experiment value of 0.25 %, the algorithms have 8,070, 135,870, and 168,566 milliseconds process runtimes respectively.

Fig – 10 shows the results of the experiments on the data set Online Retail. 1 – 8 % *minsup* values are applied to this experiment in order to attain reasonable results. With the results of the experiments with that data set, it is obvious that the SPADE algorithm has the best performance with all *minsup* values. As it is shown CM-SPADE is the second best algorithm with that data set.

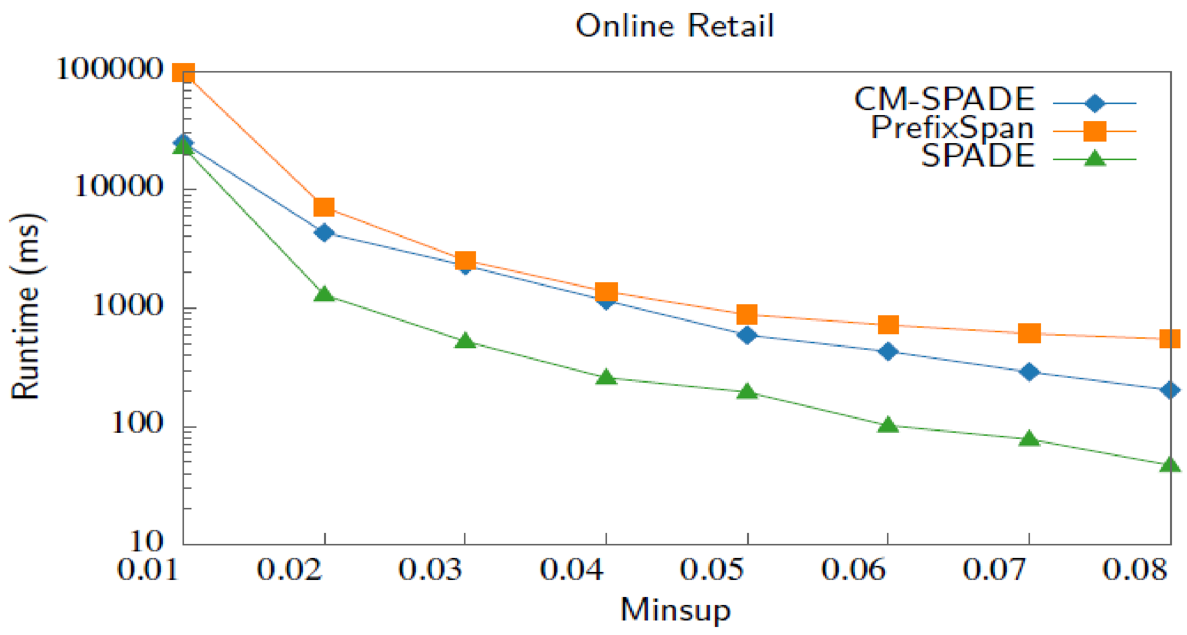


Fig – 10. Execution times on the data set Online Retail

Whereas with the lowest *minsup* value of 1 %, algorithms execute 22,881, 25,053 and 97,848 milliseconds respectively, with the highest *minsup* value of 8 % they have respectively 46.5, 203, and 547 milliseconds runtimes.

The last experiment is performed on the data set DNA Sequence. In that data set, since the huge number of frequent sequences have been obtained, it have been applied higher support threshold values (Between 90 – 97 %).

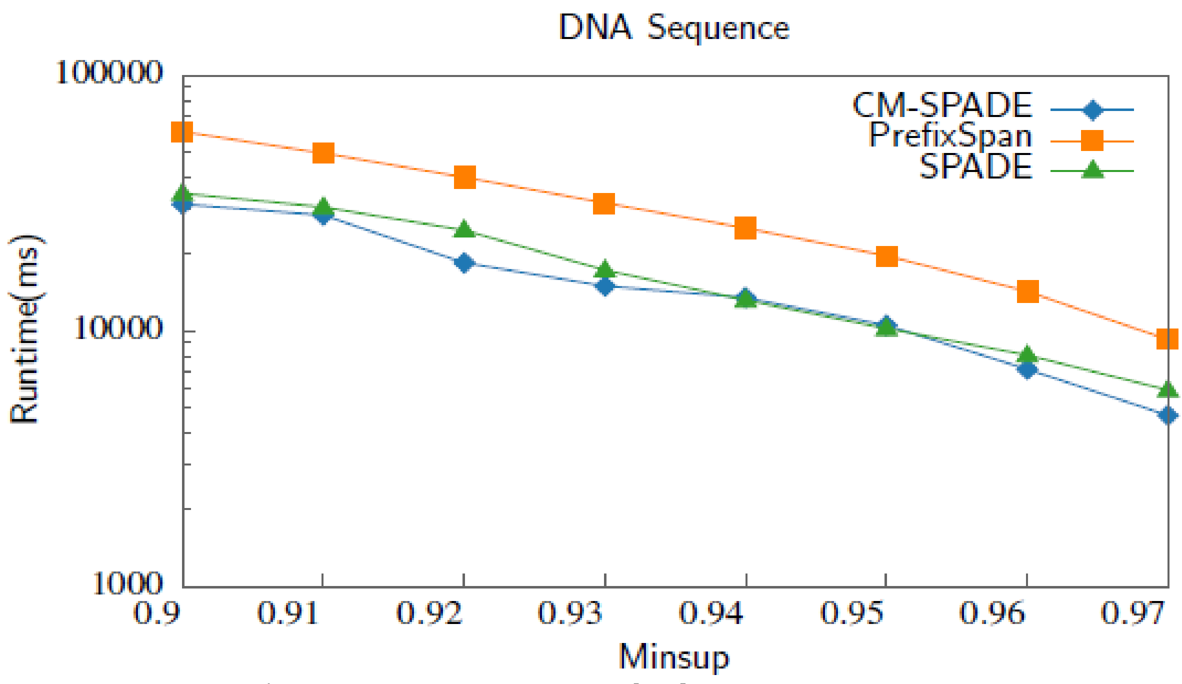


Fig – 11. Execution times on the data set DNA Sequence

As it is seen in the Fig – 11, CM-SPADE have become the slightly best algorithm comparing with SPADE with the runtime value range of 4,688 to 31,428 milliseconds corresponding to each support threshold value. PrefixSpan has resulted the worst performance with that data set with the runtime value range of 9,265 to 60,410 corresponding to each support threshold value.

4.3. Memory Usage

The second test on the data sets has been about the memory resource consumption. The measurements have been summarized in the Figs – 12, 13, 14.

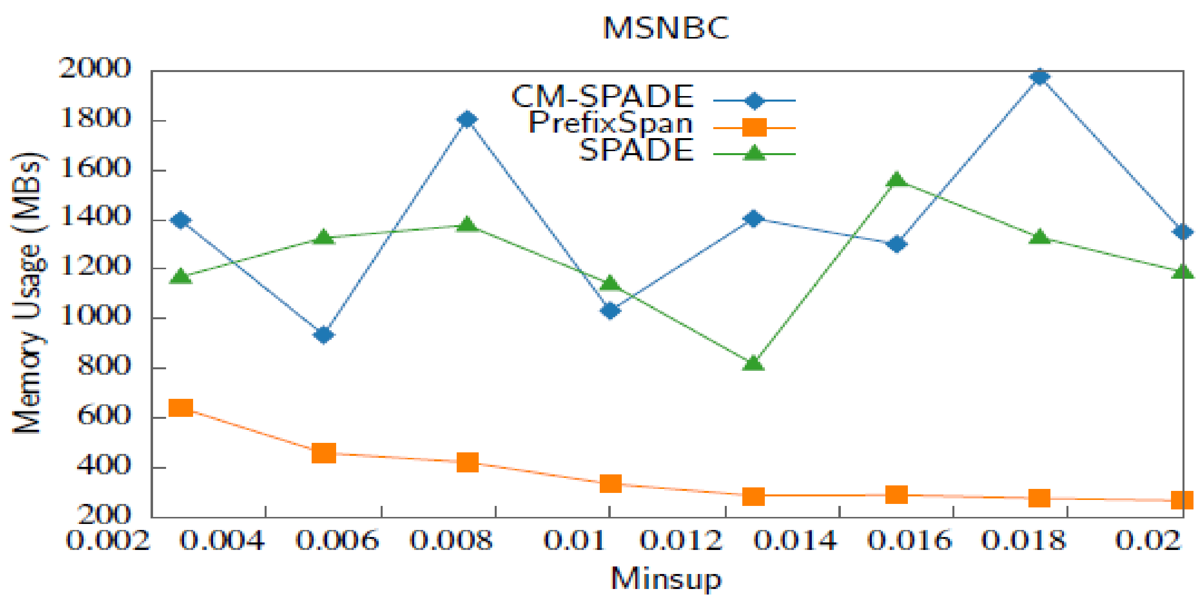


Fig – 12. Average memory usage with the data set MSNBC

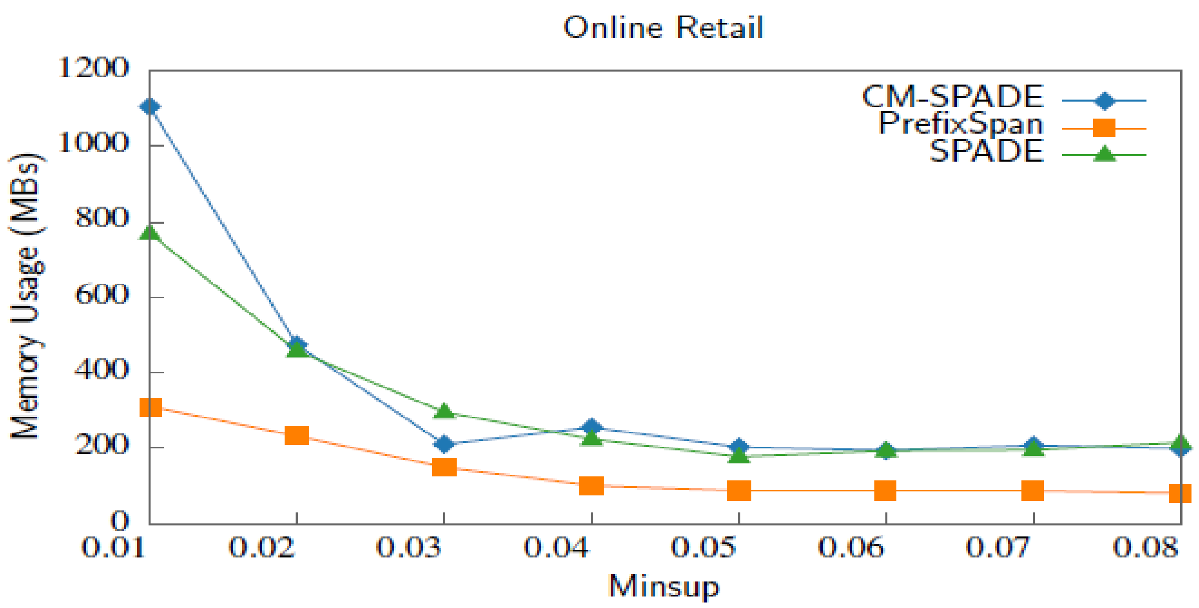


Fig – 13. Average memory usage with the data set Online Retail

The graphs represent the averages of the maximum memory usage values as megabytes extracted by SPMF with the increasing values of the support threshold ratios.

With the data sets of MSNBC and DNA Sequence, SPADE and CM-SPADE have performed instable memory usage values. In contrast PrefixSpan have consumed relatively stable amount of memory.

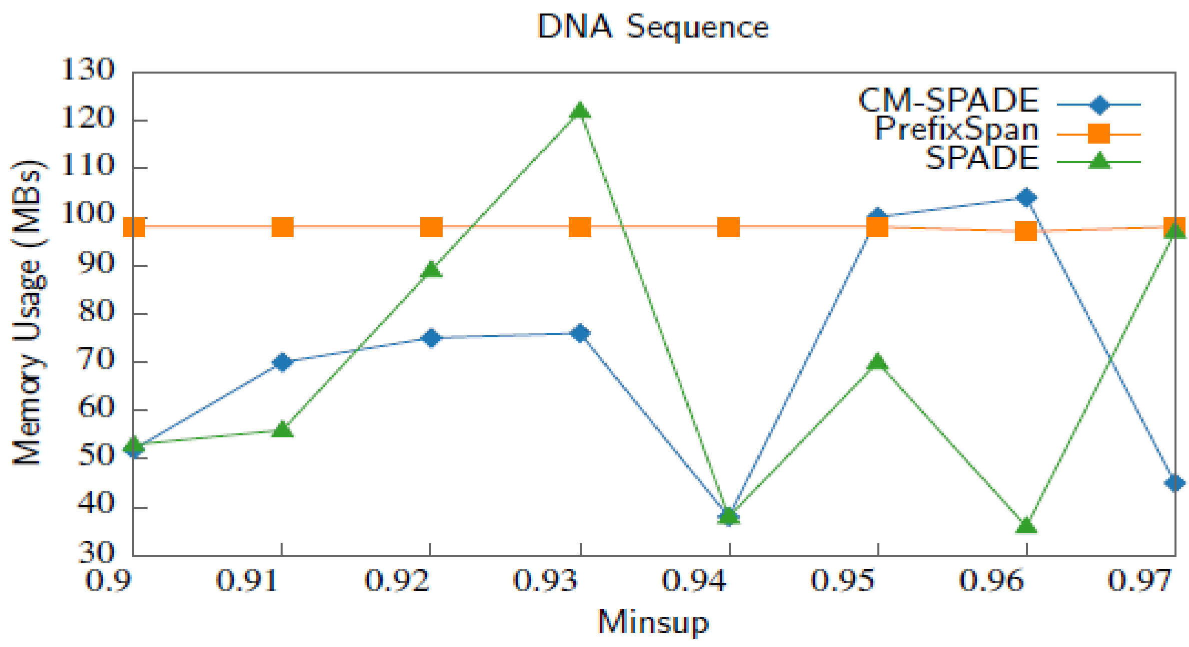


Fig – 14. Average memory usage with the data set DNA Sequence

In terms of average memory consumption, PrefixSpan is the slightly best algorithm when comparing the others since it has shown the best performance with the first two data sets. Furthermore, it is quite hard to compare clearly the CM-SPADE and SPADE performances with those inconsistent values.

4.4. Further Analysis

As a result of runtime experiment each algorithm have become the best once. Therefore, it has been obtained that there is no algorithm among those three algorithms which clearly overcome the others regarding with process times on data sets used. However, it can be seen that there is one phenomenon that the PrefixSpan works fast comparing to others with relatively low threshold values.

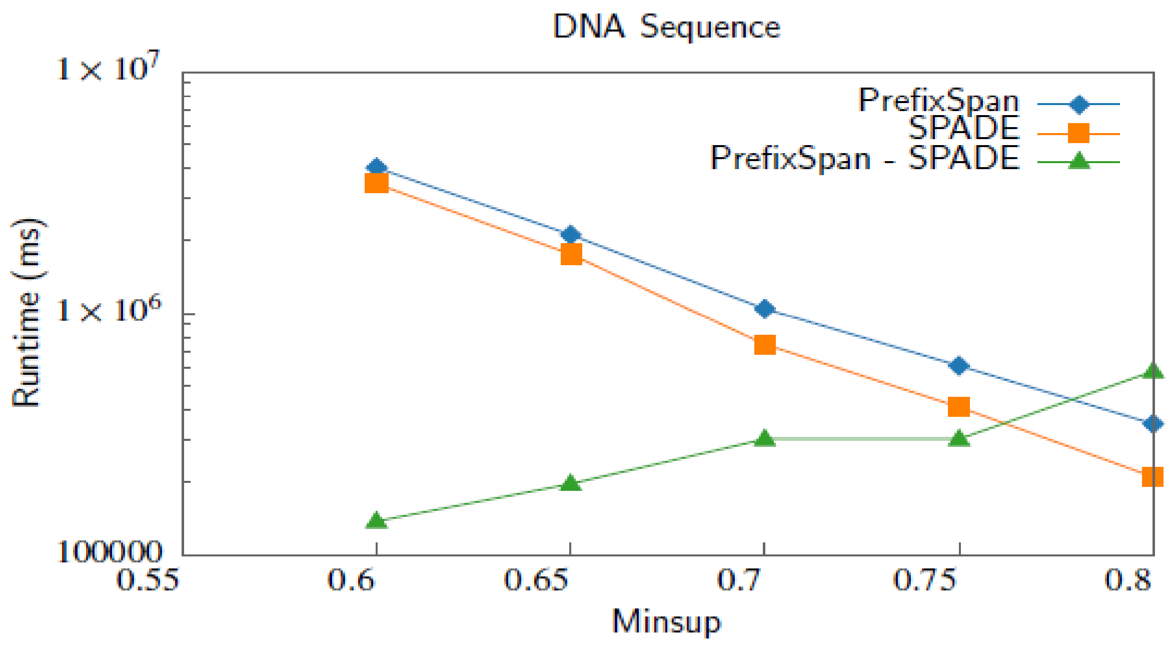


Fig – 15. Execution times on the data set DNA Sequence

It can be obtained easily in figs – 15, 16. Due to the CM-SPADE is a derivative of SPADE it has been accepted more useful to compare only PrefixSpan and SPADE. In terms of data sets DNA Sequence and Online Retail as minimum support threshold value decreases, the runtime of PrefixSpan is getting nearer to others. With the denser data set of DNA Sequence for below 8 % *minsup* value the runtimes are getting bigger exponentially. However, the execution time of PrefixSpan getting nearer to SPADE's. Finally, it seems to be reasonable that there is a critical minimum support threshold to obtain lower process times with PrefixSpan comparing to others.

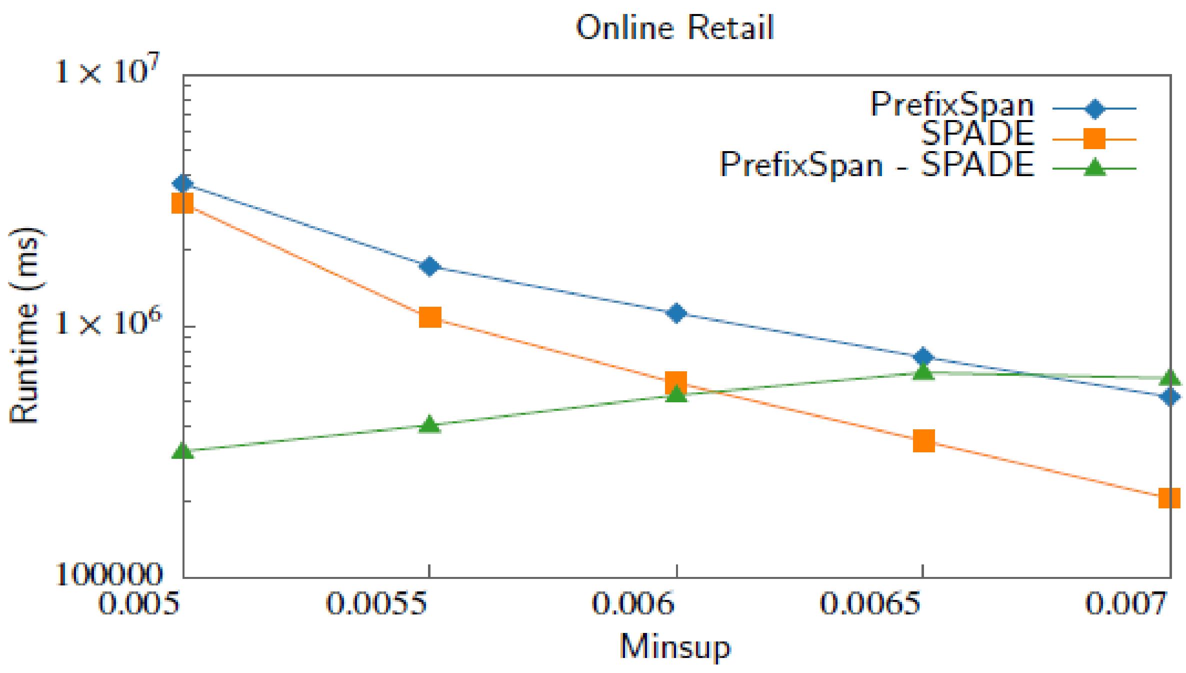


Fig – 16. Execution times on the data set Online Retail

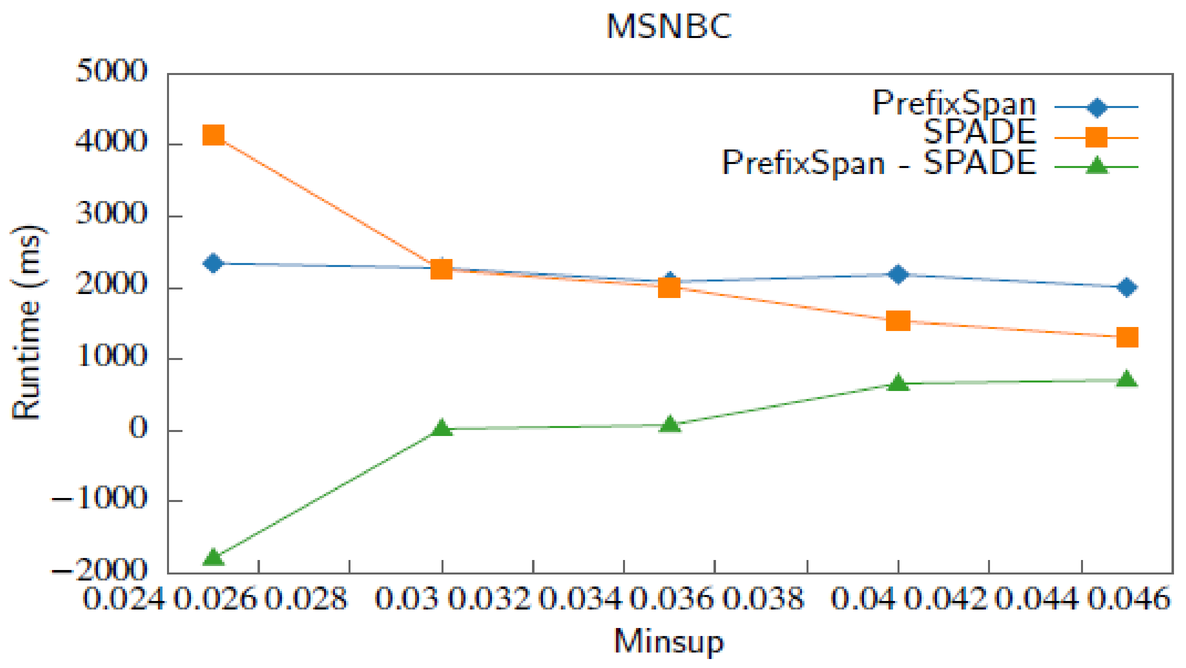


Fig – 17. Execution times on the data set MSNBC

On the other hand, it seems to be non-reasonable to mine data set with that level of minimum support thresholds. For example, it is generated 26,596,570 frequent sequences with the data set DNA Sequence for the minimum support 75 %, and 185,499,740 frequent sequences for 60%.

To support this idea, if it is taken one step back with the data set MSNBC, it can be seen that in the fig – 17, the process time of SPADE is lower than PrefixSpan. If minimum support threshold increases upper than 3 %, SPADE seems to be faster than PrefixSpan. However, it is needed deeper analysis with more data sets in order to determine if PrefixSpan is the fastest algorithm for minimum support threshold values below the critical value.

5. Conclusion

To evaluate the efficiency of three algorithms, three data sets having different characteristics are used. In terms of processing time, experiments yield ambiguous results since each algorithm become winner once. On the other hand, in respect to memory usage efficiency, PrefixSpan become slightly winner and with one data set it cannot be determined clearly which algorithm is winner.

Table – 2. Summary of experimental results

		SPADE	PrefixSpan	CM-SPADE
Runtime	MSNBC		✓	
	Online Retail	✓		
	DNA Sequence			✓
Memory Usage	MSNBC		✓	
	Online Retail		✓	
	DNA Sequence			

As it is seen in the Table – 2, algorithms' process time depend upon characteristics of data sets such as number of sequences, number of events, and number of distinct items. On the other hand, minimum support threshold values are also significant impact on process times.

It is obvious that each algorithm have its own advantages and drawbacks. Whereas some features seem to be advantageous for a specific type of data set, some are disadvantageous. It is known that, for apriori like algorithms' candidate sequence generation-and-test process give rise to non-trivial costs, but divide-and-conquer manner algorithms such as FreeSpan (Han et al., 2000), and PrefixSpan can overcome this drawback. On the other hand, PrefixSpan pays great cost to database projection. Instead of searching very large database spaces the lattice based algorithms such as SPADE, and SPAM (Ayres et al, 2002), can decompose the search space.

One can say that the CM-SPADE results seem slightly better than SPADE with two of data sets. Therefore, it can be accepted that CM-SPADE improves the SPADE one step further, and it overcome some limitations of SPADE. The summary of three algorithms tested in this paper is represented in Table – 3.

Table – 3. Summary of algorithms' characteristics

	SPADE	PrefixSpan	CM-SPADE
Main Philosophy	Reduce search space	Divide-and-conquer	Reduce search space
Advantage	Easy scanning database	Pattern growth method	Easy scanning & effective candidate pruning
Drawback	Candidate generation-and-test process	Projected-database generation	Candidate generation-and-test process
Data Type	Vertical	Horizontal	Vertical
Search Technique	Breadth-First & Depth-First	Depth-First	Breadth-First & Depth-First

As a further frequent sequence mining research topic, the idea of critical minimum support threshold can be considered. It is obtained that there can be a critical point in each data set where one algorithm can overtake the other one and beat it regarding process time. In this paper, it is observed that PrefixSpan may be the fastest algorithm for deeper minimum support threshold values.

References

Agrawal R., Srikant, R., (1994), Fast algorithms for mining association rules. In Proc. 1994 Int. Conf. Very Large Data Bases (VLDB'94), pages 487–499

Agrawal, R. Srikant, R. (1995) Mining sequential patterns, The International Conference on Data Engineering, pp. 3 – 14.

Ayres, J., Flannick, J., Gehrke, J., Yiu, T., (2002), Sequential pattern mining using a bitmap representation. In: Proc. 8th ACM SIGKDD Intern. Conf. Knowledge Discovery and Data Mining, pp. 429–435.

Chen, D., Sain, S. L., Guo, K., (2012), Data mining for the online retail industry: A case study of RFM model-based customer segmentation using data mining, Journal of Database Marketing and Customer Strategy Management, 19(3), pp. 197 – 208. DOI: 10.1057/dbm.2012.17

Fournier-Viger, P., Gomariz, A. Campos, M., Thomas, R., (2014), Fast Vertical Mining of Sequential Patterns Using Co-occurrence Information, The Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 40 – 52.

Fournier-Viger, P., Lin, C. W., Gomariz, A., Soltani, A., Deng, Z., Lam, H. T. (2016), The SPMF open-source data mining library version 2, The European Conference on Principles of Data Mining and Knowledge Discovery, pp. 36-40.

Fournier-Viger, P., Lin, J. C. W., Kiran, U. R., Koh, S. Y., Thomas, R. (2017), A survey of sequential pattern mining, Data Science and Pattern Recognition, 1(1), pp. 54-76.

Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., Hsu, M. (2000). FreeSpan: Frequent pattern-projected sequential pattern mining, Proceeding of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. (6) – pp. 355-359. DOI:10.1145/347090.347167

Han. J., Kamber, M., Pei, J., (2007), Data Mining Concepts and Techniques (3rd Edition), London: Elsevier

Harley, C. Reynolds, R., (1987), Analysis of E. Coli Promoter Sequences, Nucleic Acids Research, (15) pp. 2343 – 2361.

Heckerman, D., (n.d.), UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., Hsu, M. C., (2004), Mining sequential patterns by pattern-growth: The prefixspan approach, IEEE Transactions on knowledge and data engineering, 16(11), pp. 1424 – 1440.

Towell, G., Shavlik, J. and Noordewier, M., (1990), Refinement of Approximate Domain Theories by Knowledge-Based Artificial Neural Networks, In Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90).

Zaki, M. J., (2001), SPADE: An efficient algorithm for mining frequent sequences, Machine Learning 42(1), pp. 31-60. DOI:10.1023/A:1007652502315