Journal of Theoretical and Applied Information Technology
15th July 2018. Vol.96. No 13
© 2005 – ongoing JATIT & LLS

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A STUDY OF SEQUENTIAL PATTERN MINING ALGORITHMS FOR USE IN DETECTION OF USER ACTIVITY PATTERNS

MAXIM DUNAEV[1], KONSTANTIN ZAYTSEV[1], MIKHAIL TITOV[2]

[1]National Research Nuclear University MEPhI, 115409, Russia, Moscow, Kashirskoe Avenue 31
[2]National Research Center "Kurchatov Institute", 123182, Russia, Moscow, Academician Kurchatov sq., 1

ABSTRACT

In the last decade, the significant growth of the volume of analysis data has set the high level of importance of data mining field. This field contains a vast amount of different methods and techniques for knowledge extraction. One of the highly-demanded areas of this field is sequential pattern mining (SPM), which includes many methods for detection of frequent sequential patterns in different types of input ordered data sets. The goal of this work is to compare the efficiency of several types of SPM algorithms, and to identify the most applicable algorithm to deal with data from physical experiments used in scientific analysis tasks (e.g., analysis data from the ATLAS experiment at the Large Hadron Collider, CERN, Switzerland), and to extract association rules from experimental data samples. This paper presents the analysis of 3 types of SPM algorithms - horizontal and vertical, as well as pattern-growth, with the emphasis on algorithms' performance. There were prepared corresponding test data sets which are specific and typical for analysis tasks in the ATLAS experiment.
Keywords: *Sequences, Sequential Pattern Mining, Frequent Pattern Mining, Items Mining, Searching Patterns, Association Rules, Vertical Format, Horizontal Format, Pattern-Growth*

1. INTRODUCTION

In the last 5-7 years, due to the explosive growth of the volume of stored information, more and more attention has been paid to intellectual analysis of stored data (i.e., data mining) in various fields of human activities [1]. The data obtained as a result of analysis may be presented as Sequential Patterns, as properties of data (e.g. decision trees), as clustering (e.g. as Chernoff faces), etc. This article is devoted to representation of knowledge as Sequential Patterns. Searching for Sequential Patterns is one of the fundamental goals of Data Mining [2]. This area of activities starts with the article of Agrawal and Shrikant [3]. Sequential Pattern Mining is widely used today in various fields, for example, in analyzing web clicks of users [4], in bioinformatics [5], in market [6] and text [7] analysis, mining [8] and many other areas of human activity.

The Large Hadron Collider is the most powerful tool in high-energy physics today, as well as a source of large amounts of data, with which scientists collaborate around the world. One of the general-purpose experiments the ATLAS experiment [9] uses the PanDA (Production and Distributed Analysis System) workload management system [10] to process experimental data at distributed computing centers around the world (up to 2 million computing jobs daily). The placement of new and presented data, as well as corresponding copies (i.e., replicas), is regulated by an intelligent system for managing distributed data and various internal policies of the experiment.

Thus, if it is impossible to route the processing task to the corresponding data center where the input data sets are located, this data is moved to the selected center (which can lead to an increase in the job execution timeout). As part of the research work carried out at the University of Texas at Arlington [11], a study was conducted to investigate user activity in the PanDA system and to identify the systematic use of certain groups of data sets that showed that there are established sequences of data sets used for different classes and groups of computing jobs.

Therefore, this information allows us to predict in advance the next set of data in which the user might be interested based on the identified "pattern" of the data used by other users and, if necessary, prepare this data for movement (which in this case will shorten the possible waiting time). To detect such sequences, SPM algorithms are used, depending on the methods of presenting the original data and differing in the required computing resources. The purpose of this article is to compare Sequential Pattern Mining algorithms, and to identify the most effective of them (for the possibility of using when analyzing user activity in the PanDA system of the ATLAS experiment).

To ensure reliability, records for data processing are stored in logs having different semantic structure. Therefore, to check SPM algorithms under the same conditions, it is necessary to either to correct the code of the ready algorithms or to use / create a parser, which would lead the data to the format that is understood by this or that algorithm.

Another obstacle is the presence of a sufficiently representative set of SPM algorithms, the appearance of new, more efficient algorithms, and also works comparing individual, often two, algorithms of a certain type. For example, there are comparisons between SPAM and PrefixSpan [12], GSP and PrefixSpan, [13], TreeProjection and FP-growth [14], and many others [15, 16].

In this article, the focus is made on comparison of the best in their types of algorithms SPADE, PrefixSpan, FP-growth with the basic underlying algorithm of Apriori in solving the task of effective Sequential Pattern Mining when processing the sets of events of different semantic structure obtained (using test data of the PanDA system).

2. MATERIALS AND METHODS

Let us describe the formal statement of Sequential Pattern Mining problem, and then a pseudo-code of SPM algorithms, which we will compare to each other when working with sets of events of various semantic structures.

2.1 Formulation of the problem

There is a training sample represented as matrix "objects, symptoms". Let X be the object space;

$F = \{f_1, \ldots, f_n\}$ – is the space of binary attributes (items), where $f_j: X \rightarrow \{0,1\}$;

$X^l = \{x_1, \ldots, x_l\} \subset X$ is the training set.

Each subset (set of attributes) $\varphi \subseteq F$ corresponds to a conjunction:

$$\varphi(x) = \bigwedge_{f \in \varphi} f(x), x \in X, \quad (1)$$

If $\varphi(x) = 1$, then "attributes from φ are also found in set x".

Let us introduce a concept like frequency of occurrence or support for φ in sample X^l

$$\nu(\varphi) = \frac{1}{l} \sum_{i=1}^{l} \varphi(x_i), \quad (2)$$

If $\nu(\varphi) \geq \delta$, then "set φ is frequent", where parameter δ is the minimum support (Minsupp).

The association rule $\varphi \rightarrow y$ is a pair of disjoint sets $\subseteq F$, such that:

a) Sets φ and y are often found together, i.e. $\nu(\varphi \cup y) \geq \delta$;

b) If φ is found, y is also often found,

$$\nu(y|\varphi) \equiv \frac{\nu(\varphi \cup y)}{\nu(\varphi)} \geq \aleph, \quad (3)$$

where $\nu(y|\varphi)$ is the confidence of the rule;

$\aleph$ is the minimum confidence (MinConf).

Example: if dataset i - "φ" is processed, dataset j - "y" will also be processed with probability $\nu(y|\varphi) = 60\%$; both datasets will be processed with probability $\nu(\varphi \cup y) = 2\%$.

From (1), we have the antimonotone property: for any ψ, $\varphi \subseteq F$ $\varphi \subseteq \psi$ it follows that $\nu(\varphi) \geq \nu(\psi)$. Consequences:

a) if ψ is frequent, all its subset $\varphi \subset \psi$ are frequent,

b) if φ is not frequent, then all sets $\psi \supset \varphi$ are also not frequent,

c) $\nu(\varphi \cup \psi) \leq \nu(\varphi)$ for all φ, ψ.

The task of finding association rules consists of two stages. First, searching for frequently met (further referred to as "frequent") sets, followed by searching for association rules for these sets. The search for frequent sets is based on viewing the entire transaction database, while searching for association rules, it is sufficient to use a simple procedure in RAM (in memory).

We need the pseudo-code of analyzed algorithms to understand in which steps some of them are superior to others, and why.

2.2 Algorithm Apriori

The idea of the algorithm involves the following four steps.

Step 1. Allocating one-element (single) frequent subsequences.

Step 2. From the list of single subsequences, we generate lists of two-element subsequences by joining sequences to one-element sequences while they are listed one by one.

Step 3. From the list of two-element subsequences we generate a list of three-element subsequences by the analogy with step 2.

Step 4. The subsequences satisfying the Minsupp value are chosen.

The pseudo-code of the algorithm can be seen below.

Input: X^l – is the training sample;
minimum support for δ.

Output: $R = \{(\varphi, y)\}$ is a list of association rules;

```
1: the set of all frequent source characteristics:
        $G_1 := \{f \in F \mid \nu(f) \geq \delta\}$;
2: for all j= 2,...,n
3: the set of all frequent sets of power j:
$G_j := \{\varphi \cup \{f\} \mid \varphi \in G_{j-1}, f \in G_1, \nu(\varphi \cup \{f\}) \geq \delta\}$;
4: if $G_j = \emptyset$ then
5: exiting the loop on j;
6: $R := \emptyset$;
7: for all $\psi \in G_j$, j= 2,...,n
8: AssocRules($R, \psi, \emptyset$).
```

2.3 Algorithm SPADE

The idea of the algorithm consists in building a vertical database (table), where the following will be specified for each one-element sequence:

- sequence identifier (SequenceID or SID);

- element identifier (ElementID or EID) is the sequence number of a subsequence in a large sequence, which can be separated by timestamps (timei) or some separators, for example, brackets;

- a set of elements (items).

Next two-element tables are obtained from one-element tables with a "join" operation.

The pseudo-code of the algorithm can be seen below.

```
Input: C – is the Atomset;
        minimum support for $\delta$;
Output: F – is a list with frequent sets;
SPADE (A, $\delta$, F)
1: for all $A_i \in C$
2: $T_i \leftarrow \{\}$
3: for all $A_j \in C$, j≥I and all combinations of B from $A_i, A_j$
4: L(B) = Temporary TID-list of join of $L(A_i)$ from $L(A_j)$
5: if Supp(B) ≥ $\delta$ then
6: $T_i \leftarrow T_i \cup \{B\}$
7: $F = F \cup B$
8: Spade($T_i$, $\delta$, F )
```

2.4. Algorithm PrefixSpan

The idea of PrefixSpan algorithm is to first find all frequent items in the original database and add them to the current template, thereby obtaining new frequent sequences, and then to search for frequent sequences of greater length based on projected databases.

In order to find all patterns of sequential events in database D, PrefixSpan(<>, D) is to be invoked. Creating projected databases may greatly affect performance when working with large amounts of data, therefore, instead of physical creation of projections, the so-called pseudo-projection is used. In case of recursive invocation of the PrefixSpan method, instead of the created projection, it is passed pointers to the minimum position possible occurrences of the elements into client sequences after the current template. A set consisting of the client ID, the transaction ID in the client sequence and the position in the transaction is considered to be a pointer. Due to the pseudo-projection, the speed of the algorithm operation is significantly increased. In addition, running the algorithm requires much less memory.

The pseudo-code of the algorithm can be seen below.

```
PrefixSpan(s, D|s):
Input: s – is the pattern of consecutive events;
    $D|_s$ – is s-projection of the original database D if s is not an empty sequence <>, otherwise $D|_s$ = D.
Output: s – is the pattern of sequential events.
1. Find all frequent items b from $D|_s$, such that:
    A) b may be attached to the last substantive set of s, forming a frequent sequence;
    B) subject set (b) may be added to s, forming a frequent sequence.
2. For each frequent item b:
    A) Add b to s, forming new pattern s';
    B) Add s' to the result;
3. For each new template s':
    A) Build s'-projection $D|_{s'}$;
    B) Invoke PrefixSpan(s', $D|_{s'}$);
```

2.5 Algorithm FP-growth

The basis of this algorithm is a new data structure in the form of a prefix tree. First, a prefix tree (phase 1) is built, then frequent sets are searched for in it (phase 2).

The pseudo-code of the algorithm is shown below.

```
Input: $X^l$ – is the training sample;
Output: FP-tree T, $<f_v, c_v, S_v>_{v \in T}$;
1: normalize symptoms $f \in F$ : $\nu(f) \geq \delta$ in descending order $\nu(f)$;
Stage 1: building an FP-tree T on sample $X^l$
2: for all $x_i \in X^l$
3: v:=v0;
4: for all $f \in F$ such that $f(x_i) \neq 0$
5: if there is no child vertex $u \in S_v : f_u = f$ then
6: create new vertex u; $S_v := S_v \cup \{u\}$; $f_u := f$; $c_u := 0$; $S_u := \emptyset$;
7: $c_u := c_u + 1/\ell$; v: =u;
8: Stage 2: recursive search for frequent sets by the FP-tree FP T-find(T, $\emptyset$,$\emptyset$);
Input: FP-tree T, set $\varphi \subset F$, list of rules R;
Output: add all frequent sets that contain $\varphi$ to R;
1: PROCEDURE FP-find(T,$\varphi$,R);
2: for all $f \in F$ : $V(T,f) \neq \emptyset$ by levels from bottom to top
3: if $C(T,f) \geq \delta$ then
```

4: add frequent set $\varphi \cup \{f\}$ to list R: $R := R \cup \{\varphi \wedge f\}$;
5: build a conditional FP-tree $T' := T|f$, namely
$T' :=$ FP-tree by sub-sample $\{x_i \in X^l : f(x_i) = 1\}$;
6: from T', find all frequent sets, including φ and f:
FP-find $(T', \varphi \cup \{f\}, R)$;

Conditional FP-tree $T' := T|f$ may be built quickly, by using only the FP-tree T and without looking into the
sample.

Input: FP-tree T, attribute $f \in F$;
Output: conditional FP-tree $T' = T|f$;
1: leave in the tree only the vertices on the paths from vertices v of attribute f from bottom up to root v_0:

$$T' := \bigcup_{v \in V(T,f)} [v, v_0];$$

2: raise the value of counters c_v from vertices $v \in V(T',f)$ bottom-up according to rule

$$c_u := \sum_{v \in X_u} c_v \text{ for all } u \in T';$$

3: remove from T' all vertices of attribute f; their subtrees are not required and are even not created, since at the moment
FP-find invocation, all sets that contain elements below f have already been viewed.

2.6 Associative Rules

On the example of continuing the Apriori algorithm, we will show how association rules are built on found sets. This procedure is versatile and may be applied for all considered algorithms.

Input: R – is the list of association rules;
(φ, y) – is the associative rule.
Output: R – is the list of association rules;
(φ, y) –is the associative rule.

```
1: PROCEDURE AssocRules (R, $\varphi$, y);
2: for all $f \in \varphi$
3: $\varphi' := \varphi \setminus \{f\}$; $y' := y \cup \{f\}$;
4: if $\nu(y'|\varphi') \geq \aleph$ then
5: add associative rule $(\varphi', y')$ to list R;
6: if $|\varphi'| > 1$ then
7: AssocRules $(\varphi', y')$;
```

The provided pseudo-code of the algorithms allows revealing the following ideas, which are easily visible in texts:

1) the Apriori algorithm uses almost full enumeration of variants, which will presumably lead to bad performance during the test.
2) the use of a vertical data format by algorithm SPADE will supposedly lead to good performance in processing frequent sequences with many datasets;
3) the PrefixSpan algorithm uses pseudo-projection, which implies good performance when working with non-homogeneous data.
4) the FP-growth algorithm builds prefix trees, which will supposedly result in good enough performance when working with homogeneous data.

3. RESULTS

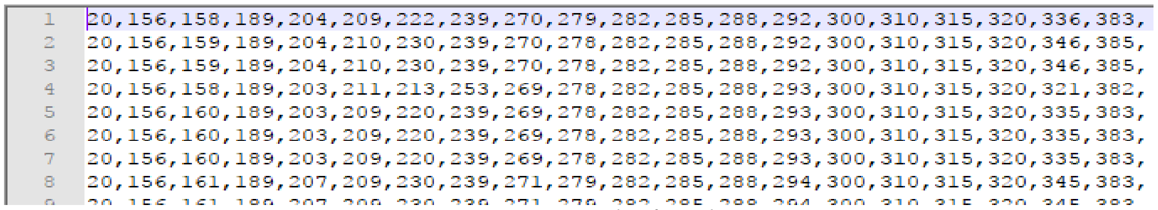
3.1 Test Data And Information About The Test Computer

Testing was performed with the use of a MSI GT70 notebook PC with Intel Core i7 processor (clock speed 2.4 GHz) and 16Gb RAM. To avoid introducing errors into the operation of the algorithms from the features of a certain development environment, it was decided to use Windows prompt.

The algorithms were tested on sequences of user activities (i.e., user Id with a list of ordered used items Ids) that were collected from user computing tasks/jobs (figure 1) controlled by the PanDA workload management system ("Production and Distributed Analysis") [23].

PanDA database contains records about computing jobs that are run to perform user and group analysis (to process ATLAS data). These records brings the connection between a particular user (job's owner) and dataset (job's input data) that will be processed, thus jobs describe the relation between the user and required items with a certain set of attributes. The most significant job attributes in context of our research are the following: prodUserId (the owner of the job), prodDBlock (input data or data pattern that contain job input files),creationTime (job creation time), jobStatus (job processing state, e.g., successfully finished), prodSourceLabel (type of the job, e.g., user analysis, production, local system, test, etc.), transferType (method and type of file transfer), processingType (type of the client that processes the job), computingSite (computing center where job is processed), destinationDBlock (name of the destination dataset, that is used for the outputs of an associated set of jobs).

Dataset sequences have different indicators:
- the average number of transactions (T) for heterogeneous databases or the exact number of transactions for homogeneous databases (C) in client sequences,
- the average number of items in transactions (I) and
- number of client transactions (D).



```
20,156,158,189,204,209,222,239,270,279,282,285,288,292,300,310,315,320,336,383,
20,156,159,189,204,210,230,239,270,278,282,285,288,292,300,310,315,320,346,385,
20,156,159,189,204,210,230,239,270,278,282,285,288,292,300,310,315,320,346,385,
20,156,158,189,203,211,213,253,269,278,282,285,288,293,300,310,315,320,321,382,
20,156,160,189,203,209,220,239,269,278,282,285,288,293,300,310,315,320,335,383,
20,156,160,189,203,209,220,239,269,278,282,285,288,293,300,310,315,320,335,383,
20,156,160,189,203,209,220,239,269,278,282,285,288,293,300,310,315,320,335,383,
20,156,161,189,207,209,230,239,271,279,282,285,288,294,300,310,315,320,345,383,
```

Figure 1. An example of test data

Figure 1 shows that the first PanDA user has used a sequence of datasets (i.e., items) with numbers: 20,156,158,189,204 ..., etc. Minimizing waiting time for loading the next dataset, it is required to build such a template/pattern so that when the user uses datasets with the numbers 20,156,158,189, we preload the dataset with the number 204, if we guessed, and the user used it for the analysis, then we load the dataset with the number 209 etc.

To test the algorithms of searching for effective sequences, six types of journals (see Table 1).

Table 1. Types of source data for testing SPM algorithms

The type of source data	T	C	I	D
Dataset1	-	10	10	1,000
Dataset2	-	20	20	10,000
Dataset3	-	73	73	10,000
Dataset4	25	-	10	10,000
Dataset5	10	-	4	100,000
Dataset6	40	-	10	100,000

3.2 Results of the comparative analysis

As a result of the testing the algorithms on provided datasets, the following results have been obtained. For Dataset1, the best time for all support indicators was shown by the FP-growth algorithm (Figure 2).

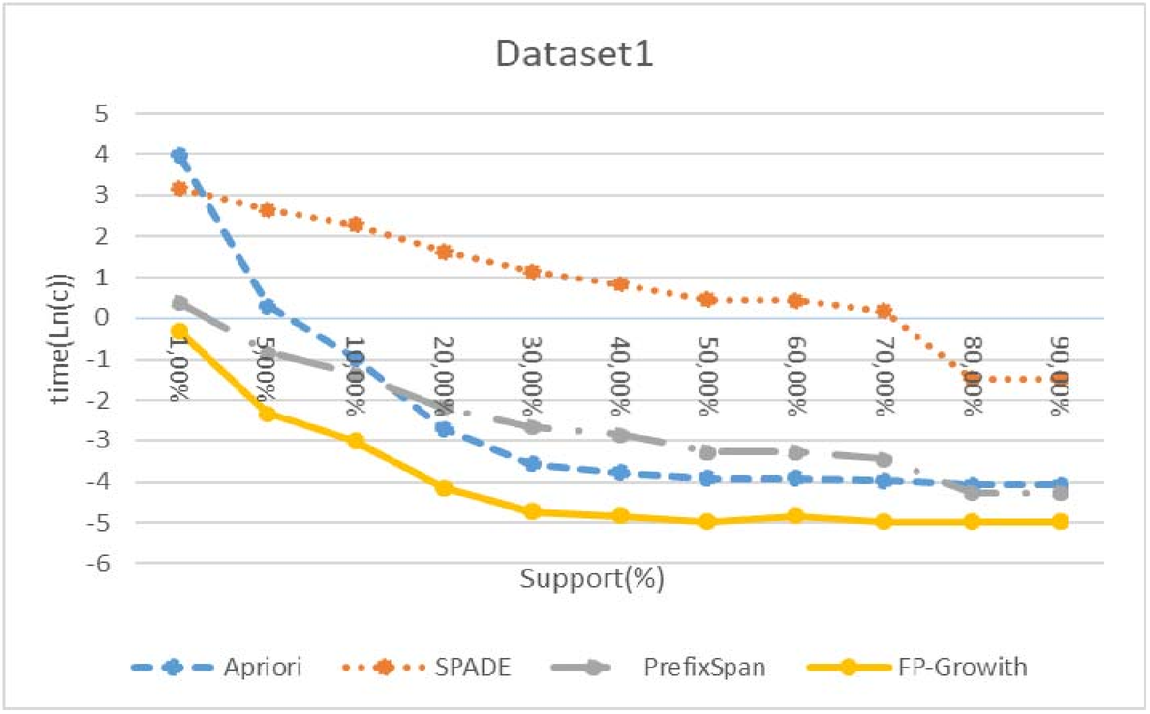


Figure 2. Schedule of algorithms for Dataset1

For Dataset2 and Dataset3, the best time for all support indicators was also shown by the FP-growth algorithm (Figures 3 and 4). This algorithm was able to calculate on the test computer patterns even for 60% support. It can also be assumed that if a supercomputer is used, this algorithm would be able to find the sequence for 1% of support, too. For Dataset4, the best for minimum support was the PrefixSpan algorithm, and for large values - the FP-Growth algorithm (Figure 5).

For Dataset 5, the best time for one percent support was shown by the PrefixSpan algorithm, and for 10% – by the FP-Growth algorithm (Figure 6).

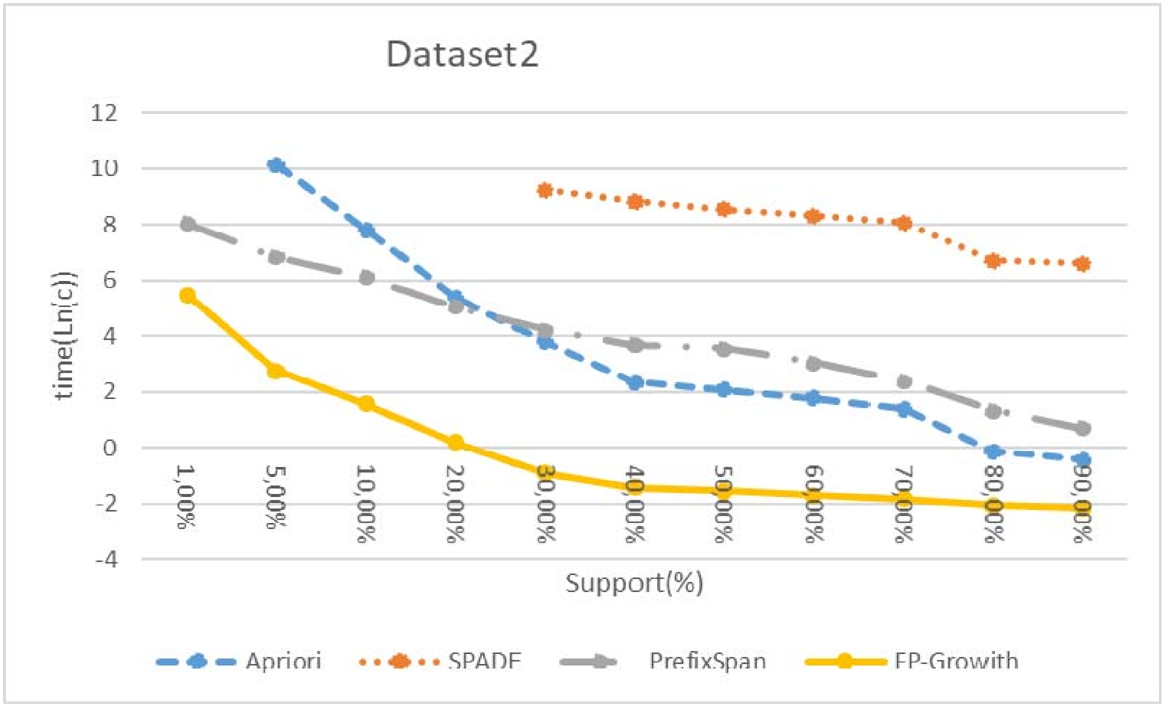


Figure 3. Schedule of algorithms for Dataset2

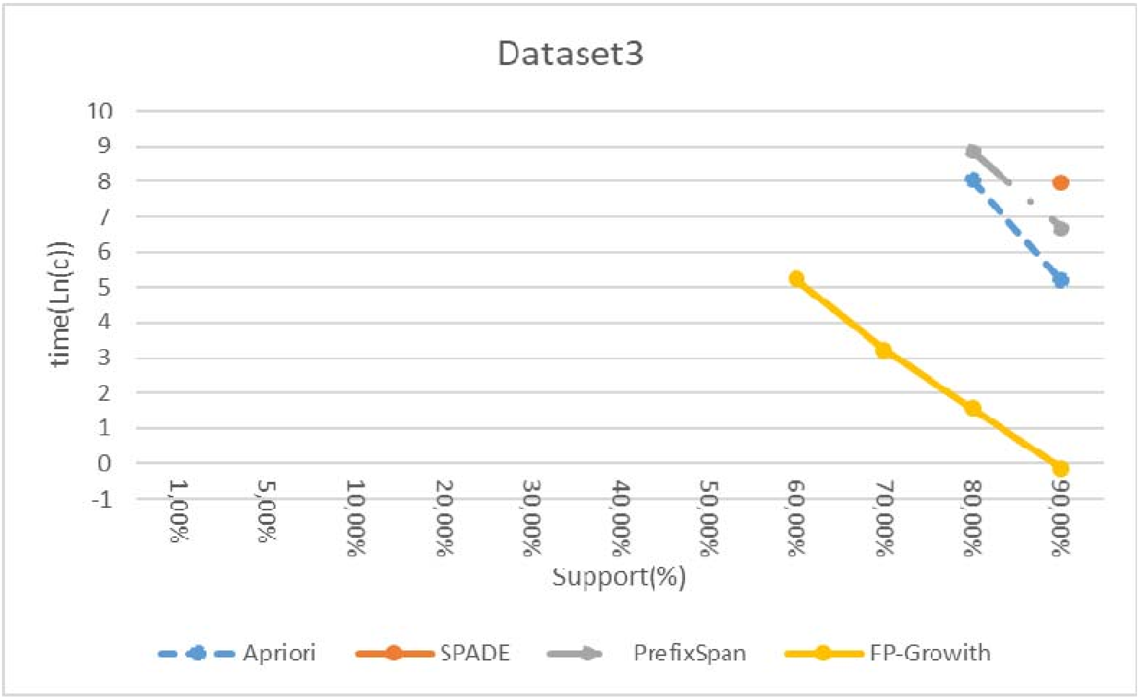


Figure 4. Schedule of algorithms for Dataset3

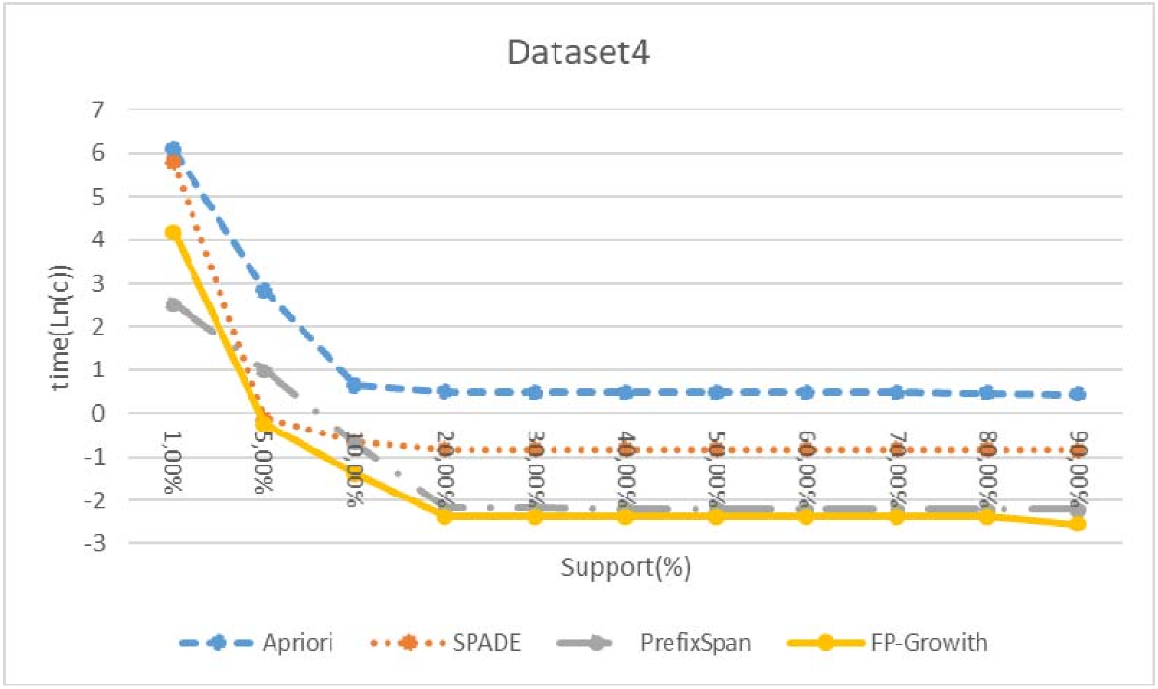


Figure 5. Schedule Of Algorithms For Dataset4

For Dataset 6, the best for support in 1%, 5% and 10% was shown by the PrefixSpan algorithm, and for support in 20% - by the FP-Growth algorithm (Figure 7).

By the results of comparative analysis of SPM algorithms all ordered transactions about activities of PanDA users may be divided into two types:

a) highly homogeneous data. To process data of this type, the FP-growth algorithm proved very efficient, since for many equal (or very similar) sequences, a tree is built, where the number of vertices is the same; this greatly reduces the dimensionality of the problem and the search time;

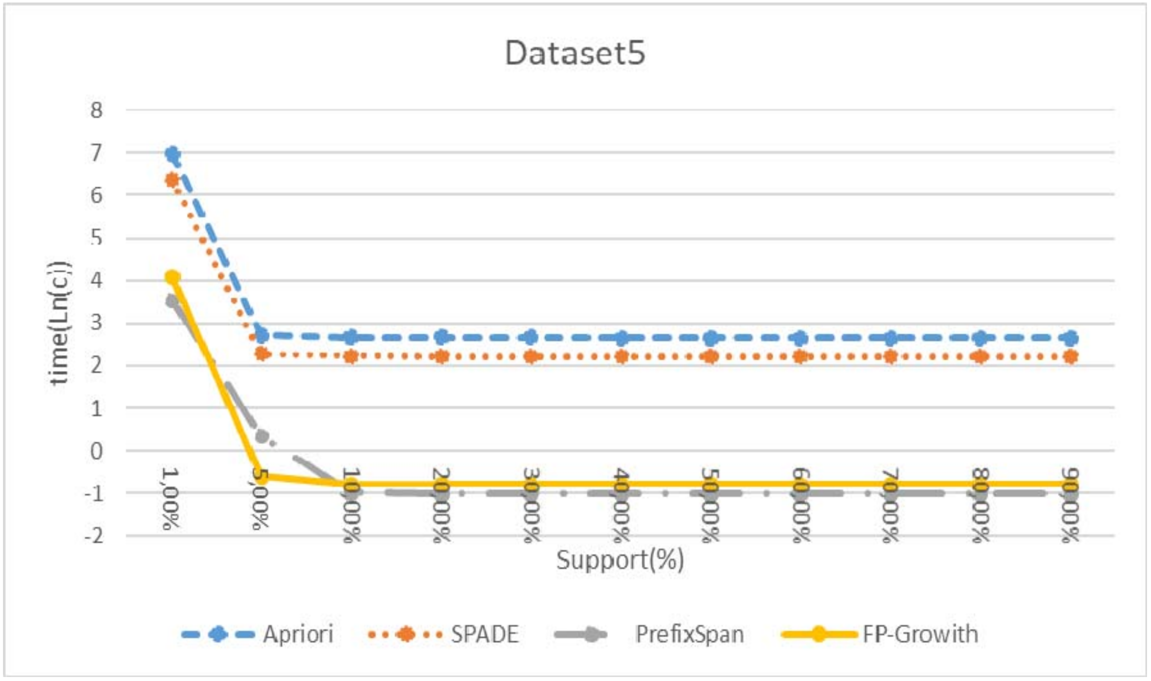


Figure 6. Schedule Of Algorithms For Dataset5

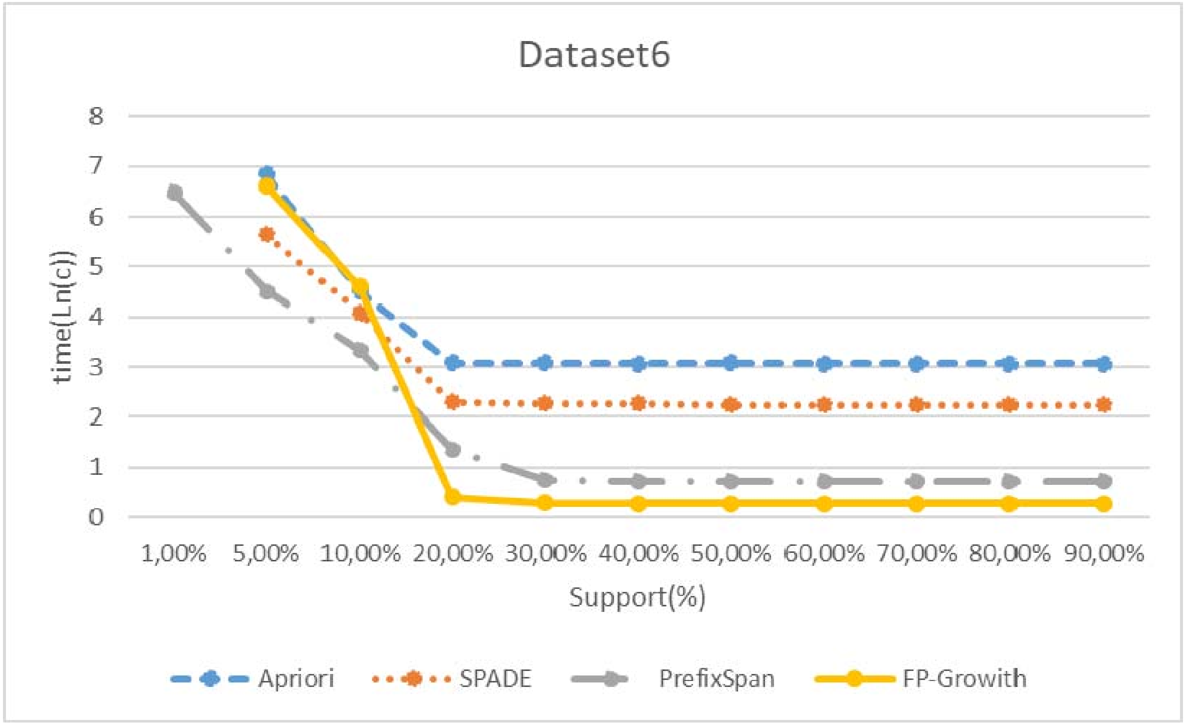


Figure 7. Schedule Of Algorithms For Dataset6

b) rare (heterogeneous) data. The PrefixSpan algorithm proved efficient to process data of this type. The FP-growth algorithm for small supports will have too large forests of trees, which will have a negative impact on the speed of search.

In addition, when building association rules for the minimum significance MinConf=70% for all samples it turned out that the time of the procedure of finding association rules was short, and even for large samples was less than a minute. An example of rules found for Dataset2 with MinSupp = 20% is shown in Figure 8.

```
Rule: ('282', '20', '156', '300', '310', '315', '320', '383', '239', '189', '285') ==> ('209',) , 1.000
Rule: ('282', '156', '300', '310', '315', '209', '320', '383', '239', '189', '285') ==> ('20',) , 1.000
```

Figure 8. An Example Of Built Association Rules

Figure 8 shows the activity patterns of the users of data sets represented in the form of standard rules: "If (a loaded sequence of datasets), then (download the specified dataset) with probability (...)". Here the order of the elements of the sequence is important.
For example, "If (282, 20, 156, 300, 310, 315, 320, 383, 239, 189, 285), then (209) with the probability of (1.0)".

4. DISCUSSION

The number of publications about SPM algorithms of various types is growing, as there is the growing number of attempts to use them in various fields [4-8]. There are also articles that describe algorithms of various types, for example, those described in [13, 17-19] and their use.

There is an opportunity to compare performance of the SPM algorithms in identical computation conditions on specially prepared test examples of various semantic structures. It would seem that this opportunity should arouse interest in determining the areas of efficient use of various algorithms, however, this far, in publications we see only attempts of comparative analysis of two algorithms on coherent repeatable examples, or comparisons of the algorithms, where the presented experimental results are difficult to repeat, or even explain. For example, in paper [13], the algorithms seem to be compared correctly, but it is not explained how and why the GSP algorithm should be compared to other algorithms if they initially do not match the types of the tasks being resolved. On the other hand, in the same article [13] the PrefixSpan algorithm, compared to SPADE, showed the best time results, which is also seen in our studies; however, in our experiments, the compared times are much more different.

In paper [14] the FP-growth algorithm was also better than the Apriori algorithm, but it is not clear why there are so small values of algorithms' running time with such a large sample. Even from the theoretical point of view, such values are impossible. In paper [20], the SPADE algorithm proved to be much better than the PrefixSpan algorithm on many samples. It is very strange, as the SPADE algorithm is much more complex than PrefixSpan, which is evident during analysis of the pseudo-code of these algorithms. And such gain may only be in inhomogeneous samples with such support values, under which the PrefixSpan algorithm requires building many pseudo-projections.

The approach to the comparative analysis of SPM algorithms proposed in this article is based on comparing them by repeated test cases of various semantic structures, and is a comprehensive study of the most efficient algorithms in their classes. The results of work may be used for solving the problems of effective processing of datasets at LHC experiments..

It should be noted that there is no universal SPM algorithm that would be equally well suited for processing data with various semantic structure. Therefore, the search will continue.

5. CONCLUSION

This work is devoted to identifying areas of efficient use of SPM algorithms of three, the most progressive types - horizontal and vertical format and "pattern-growth". For this purpose, at the stage of preliminary filtering, the best in their class algorithms were selected, like SPADE, PrefixSpan, FP-growth.

For comparing the algorithms, their pseudo-code was given, and special test samples of the original data sets characteristic for the PanDa system of the ATLAS experiment at the Large Hadron Collider, were prepared and differed in their semantic structure.

Comparison of the algorithms for homogeneous initial data sets with few (10-20) transactions (courses) for all values of support identified the FP-growth algorithm as the leading one. When the number of transactions increases, this algorithm remains the best. The difficulty of using this algorithm arises when value MinSupp is <60%, which is explained by an increased size of built prefix trees to such a magnitude, where RAM is filled quickly (in our experiments, it was 190 seconds). Therefore, obtaining more accurate results requires using a more powerful computer.

Comparison of algorithms for heterogeneous source data sets for almost all additional testing conditions identified the PrefixSpan algorithm as the best one.

In conclusion, based on the constructed sequence patterns, associative rules for a priori loading of data sets into temporary memory of supercomputers or computational clusters.

Using the results of this work by the PanDA workload management system in the ATLAS experiment will allow to control the loading of the studied data files into the computing environment, without waiting for an explicit request from the user task.

Another important problem of sequential template modeling today is the verification of algorithms for completeness and correctness. This problem is important because incomplete or incorrect algorithms, for all their efficiency, will not work with a full set of correct templates, and accordingly will not be able to guarantee the quality of the solutions obtained.

For the problem of sequential modeling of regularities, a complete algorithm is one that can find all the necessary sequential patterns, and the correct algorithm is an algorithm that correctly calculates the number of occurrences.

For example, the article [21] covers the problem of the completeness of one of the most popular algorithms - the CloSpan algorithm, and in [22] it is proved that it is not complete.

This algorithm for reducing the search space applies some theoretical calculations based on heuristics, which, naturally, are not universal and suitable for all types of analyzed sequences. CloSpan works well for databases that contain sequences of sets of items that contain one item in each. But in cases where the database contains sequences of sets containing more than one element in a set of elements, the algorithm may skip some consecutive patterns, which indicates that it is incomplete.

ACKNOWLEDGMENTS

This work was supported by the Competitiveness Program of National Research Nuclear University MEPhI (Moscow Engineering Physics Institute), contract with the Ministry of Education and Science of the Russian Federation No. 02.A03.21.0005, 27.08.2013.

REFERENCES

[1] Frasconi, P., Landwehr, N., Manco, G. and Vreeken, J (2016) Machine Learning and Knowledge Discovery in Databases European Conference, Riva del Garda, Italy, September 19-23, 2016, Proceedings, Part III.

[2] Aggarwal C. C. (2016), Recommender Systems: the textbook, Heidelberg: Springer.

[3] Agrawal R. and Srikant R. (1994) Fast algorithms for mining association rules, *The International Conference on Very Large Databases*, pp. 487–499.

[4] Singh, H., Kaur, M. and Kaur P. (2017) Web page recommendation system based on partially ordered sequential rules. *Journal of Intelligent & Fuzzy Systems,* 32, No. 4, pp. 3009-3015.

[5] Krishnan R, Nair A.S. ans Dhar P.K. (2017). Computational study of 'HUB' microRNA in human cardiac diseases. Bioinformation, vol. 13(1), pp. 17-20.

[6] Maji, G., Sen, S. and Sarkar A. (2017) Share Market Sectoral Indices Movement Forecast with Lagged Correlation and Association Rule Mining. In IFIP International Conference on Computer Information Systems and Industrial Management, pp. 327-340. Springer: Cham

[7] Maylawati, D. S., and C PutriSaptawati V (2017) Set of Frequent Word Item sets as Feature Representation for Text with Indonesian Slang. In *Journal of Physics: Conference Series*, vol. 801, no. 1, p. 012066.

[8] Bai, P. and Ravi G.K. (2016) Efficient Incremental Itemset Tree for approximate Frequent Itemset mining on Data Stream. In *Applied and Theoretical Computing and Communication Technology* (iCATccT), 2016 2nd International Conference on, pp. 239-242. IEEE

[9] ATLAS. Access to Collaboration Site and Physics Results [Electronic resource] (2017) https://atlas.cern (Retrieved: 06.02.2018).

[10] D.D. Drizhuk, A.A., Poyda, D.A., Oleynik (2016) Systems of pilot assignments for the integration of the Kurchatov supercomputer into a heterogeneous computing environment. http://conf58.mipt.ru/static/reports_pdf/1118.pdf (Retrieved: 06.02.2018).

[11] Titov M. (2016) Personalization and Data Relation Exploration using Predictive Analytics for the Production and Distributed Analysis System (PanDA) Monday, August 15. https://uta-ir.tdl.org/uta-ir/handle/10106/26138?show=full (Retrieved: 06.02.2018).

[12] Rashmi V. Mane (2013) A Comparative Study of Spam and PrefixSpan Sequential Pattern Mining Algorithm for Protein Sequences. *Communications in Computer and Information Science*, ICAC3 2013, CCIS 361, pp. 147–155.

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

[14] Han, J., Pei, J., Yin, Y. and Mao R. (2004) Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach, Data Mining and Knowledge Discovery, no. 8, pp. 53–87

[15] Laoviboon, S., and Komate A. (2017) Mining high-utility itemsets with irregular occurrence. In Knowledge and Smart Technology (KST), 2017 *9th International Conference* on, pp. 89-94. IEEE

[16] Subramanian, K. and Surya, S., (2017). Mining Huge Data with Closed Sequential Pattern Model. International Journal, vol. 5(2).

[17] Yuan X. (2017). An improved Apriori algorithm for mining association rules. *AIP Conference Proceedings* 1820, 080005

[18] Zaki M. J. (2001). SPADE: An Efficient Algorithm for Mining. Frequent Sequences. *Machine Learning*, no. 42, pp. 31–60.

[19] Belwate N. (2017). Comparison of Data Mining Algorithms for Effective Performance, *InternationalJournal of Innovative Research in Computer and Communication Engineering*, Vol. 5, Issue 5.

[20] Philippe Fournier-Viger, Antonio Gomariz, Manuel Campos, and Rincy Thomas (2014) Fast Vertical Mining of Sequential Patterns Using Co-occurrence Information. V.S. Tseng et al. (Eds.): PAKDD 2014, Part I, LNAI 8443, pp. 40–52

[21] Philippe Fournier-Viger, On the Completeness of the CloSpan and IncSpan algorithms [Electronic resource] (2018) http://data-mining.philippe-fournier-viger.com/completeness-clospan-incspan-algorithms/ (Retrieved: 06.02.2018).

[22] Le, B., Duong, H., Truong, T., & Fournier-Viger, P. (2017). FCloSM, FGenSM: two efficient algorithms for mining frequent closed and generator sequences using the local pruning strategy. Knowledge and Information Systems, 53(1), p.71-107.

[23] The PanDA Production and Distributed Analysis System [Electronic resource] https://twiki.cern.ch/twiki/bin/view/PanDA/PanDA (Retrieved: 06.02.2018).