Comparison of Data Mining Techniques for Money Laundering Detection System

Rafał Dreżewski, Grzegorz Dziuban, Łukasz Hernik, Michał Pączek
AGH University of Science and Technology, Department of Computer Science, Kraków, Poland
Email: drezew@agh.edu.pl, grzegorz.dziuban4@gmail.com, lukaszhernik@gmail.com, michal.paczek@hotmail.com

***Abstract*—The work of a police analyst, who inspects money laundering cases, is strongly based on ability to find patterns in a large amounts of financial data, consisting of entries describing bank accounts and money transfers. Out of the need for tools simplifying process of analyzing such data, came the Money Laundering Detection System (MLDS). The mechanisms of money laundering and the system itself are shortly presented in this paper. The main part focuses on implemented algorithms that help finding suspicious patterns in the money flow. Two independent sets of experiments were performed focusing on different algorithms implemented in MLDS, in which their performance was measured, compared and summarized.**

***Keywords*—decision support systems; data mining; pattern analysis**

I. Introduction

The term "money laundering" originated in the twenties of the previous century, and describes the process of turning profit from criminal activity into money that seems to originate from legal sources. Through the following years this process changed its shape, matured into a precise and complicated mechanism, consisting of many different techniques. Parallel to it, police criminal analysis was also perfected in a constant struggle to keep up with the creativity of criminals.

Nowadays, complexity of analyzing huge amounts of data, which are used as a basis of investigations, requires use of dedicated software. The whole process of money laundering can be described as a procedure in which the offenders make large number of small money transfers between hundreds of entities or individuals. Bank records of such activities are the main source of data for police analysts, but it has been proven almost impossible to keep track of investigation with use of simple spreadsheets. Such situation led to creating a new kind of software, which was based on algorithms that could operate on vast amount of data, from which only a small part might be of any importance.

Money Laundering Detection System (MLDS) was developed as a solution for problems mentioned above. Its main goal is to analyze large amounts of bank statements and find patterns that may resemble criminal activity. The whole process is interactive and iterative, giving the user control over shape of the search. MLDS was implemented as a module of CAST/LINK system developed at AGH University of Science and Technology in cooperation with Polish Police and is described in more details in [1]–[3].

There are three main steps that need to be done to obtain any search results—that is importing data, clustering, and frequent pattern mining. From these, the last one may be considered a bottleneck, being the most demanding in matter of computational power, and its implementation is crucial to the overall performance. In this paper we present four different algorithms that can be used to obtain valid results. In section III we present their characteristics and in section IV the comparison of their performance.

II. Automatic Analysis of Bank Transfers

Methods of artificial intelligence include wide area of techniques for data mining. Wherever there is already a database, there is a need to analyze these data in order to discover previously unknown knowledge. The idea of data mining involves using computer's computational power to find patterns in the data that are hidden to a man.

The basis for the software designed to analyze bank transfers is the platform named CAST/LINK, for which plug-ins with analytical algorithms were implemented [1], [3]. This tool will help criminal analyst, simplifying analysis of data describing banking operations performed. One of the extensions of the platform is able to import the data supplied in form of electronic bank statements, which is then checked for any suspicious activity (i.e. repetitive sequences of operations on the accounts), in an attempt to detect money laundering processes. Mechanisms implemented to perform these functions are based on the use of artificial intelligence methods for mining large data sets—"data mining" techniques.

The term data mining is often used as a synonym for the process of knowledge discovery in databases [4]. Methods of data mining can be divided, very broadly, into six main groups: discovering associations, clustering, exploring the sequence patterns, discovering classification, discovering similarities in the time sequences, detection of changes and variations.

Among the six methods for data mining the most interesting (from the analysis of bank transfers perspective) is exploring the sequence patterns that will allow to discover repeated operations that will be the subject of further analysis.

III. Data Mining Techniques

Data mining techniques are algorithms which purpose is to find a pattern in a large amount of data, from which only a small amount (or even none in case in which no pattern is

978-1-4799-8386-5/15/$31.00

present), might be of any importance [4]. It is easy to imagine that when dealing with such a problem, deterministic approach would cause exponential rise of the numerical complexity. Out of the whole process of analyzing data, which can contain evidence of criminal activity, finding patterns is the most crucial step, so it is very natural, that we must be very cautious in choosing specific implementation of algorithms.

In this paper we present four different algorithms, which lay at the core of MLDS analytical functionality: *Apriori*, *PrefixSpan*, *FP-growth* and *Eclat*.

A. Apriori Algorithm

The Apriori algorithm is one of the first data mining procedures ever created [5]. To fully understand its functionality we must introduce a few terms. *Transaction* is an ordered itemset of elements meeting specified prerequisites. Money transfers can be divided into transactions on the basis of two criteria—operations performed in a certain period of time, or within a specific boundary of value. We can say that a transaction supports an itemset if its elements occur in that particular transaction. *Sequence* is an ordered list of transactions. *Support* is a number of transactions supporting a set. *Frequent itemset* is a set with support equal or greater to the assumed minimal support. Elements of an itemset must be unique. *Frequent sequence* is a sequence with support equal or greater to an assumed minimal support. Elements can recur, but the order must be preserved. *Subsequence* is a set of items which are part of a specific sequence. Elements in the set must occur in the same order as in the sequence, and can not contain any additional spaces.

The purpose of all pattern-seeking algorithms is to find as much frequent sequences as possible [5]. The basis for the Apriori procedure is a discovery that if a set is infrequent, any superset created from it also is infrequent. This leads us to an observation that if we tried to build every possible set starting from one element and adding one element per step, we would only need to remember in each iteration these sets (or in a more narrow definition—sequences) which are already frequent. This simplifies the whole process, reducing complexity, and giving opportunity to obtain results in a reasonable time.

We can divide the Apriori algorithm into two simple steps [5]:

1) Finding all frequent itemsets (sets which meet the required minimal support). If it is the first iteration of algorithm, we extract all one element itemsets and check their support.
2) On the basis of itemsets found so far we create candidate itemsets by adding one element. We then check their support, and if the set turns out to be frequent we append them to our solution set.

In a single iteration of the Apriori algorithm we acquire sequences that are one element longer, so to search the database for a pattern of length N we need only N iterations. The most important part of Apriori procedure is generating candidate itemsets. These are created on the basis of frequent sequences acquired so far, by combining two rules with the same prefix, which spares us time spent on searching for every possible combination of sequences with new elements.

Summarizing, the Apriori algorithm is a simple, yet effective, procedure, which we can be used for finding patterns in a large amount of data.

B. PrefixSpan Algorithm

Another approach to data mining problem applied in MLDS system is the PrefixSpan algorithm [6]. Its main idea is to divide a database which we are processing into a number of projections in order to reduce the problem.

PrefixSpan is a variation of previously invented FreeSpan algorithm, which was an attempt to overcome a few of disadvantages of Apriori-derived procedures: large number of candidate sequences, multiple database scans and deteriorating performance when mining longer sequences [6].

The idea that was introduced in FreeSpan was to recursively project sequence databases to smaller ones, based on fragments of sequences. Using this approach one could grow subsequences in logically separated areas that divided data and frequent sets. Unfortunately, substrings were generated using any combination in a sequence, which led to a large performance overhead.

PrefixSpan evolved the mechanism of FreeSpan by taking into account only prefixes of sequences, and projecting into new databases only postfix subsequences. This algorithm can be described in three steps [6]:

1) Finding all sequence patterns with length and size of one. We avoid sequences with length one, but of a larger size—these will be found in the next steps. In this part of procedure we also perform a initial transformation of database DS, to a form devoided of any infrequent elements.
2) Second step is based on an observation that any sequence must begin with one of the elements isolated in previous step. Keeping this in mind we can partition the database into N parts (projections), where N is the number of elements found in the first step.
3) The last part of the procedure analyzes subsets of sequences kept in created partitions. For any one-element sequence with length of one we create a projection of database with its postfixes. Everyone of these partitions is then recursively explored to generate sequence patterns. We try to find multi-element frequent sequences of length one or two, create new projections of database, and then explore them, until the algorithm terminates.

C. FP-growth Algorithm

FP-growth algorithm is one of the fastest and most popular methods of discovering frequent sets [4], [7]. It displays better performance than Apriori algorithm for small minimal support values and in the case of very dense data sets.

FP-growth algorithm is based on a structure called FP-tree, to which we must transform our database in order to execute the procedure. To achieve this we must carry out a few steps [7]:

1) Find all one-element frequent sets.
2) Compress all transactions by deleting all non-frequent elements.
3) Sort elements of transactions in a descending support order.
4) Transform to FP-tree.

FP-tree is a compact and coherent data structure that keeps information about frequent patterns. Every vertex of a FP-tree consists of three attributes: name, telling which element does it represent, number of transactions leading to this particular vertex and connections to next vertexes with the same names (or a null element).

When transformation of transaction database to FP-tree is completed, the FP-growth algorithm analyzes created graph in order to find frequent sets. In the first step we must find all paths for a one-element frequent set α, which supersets are represented by paths that contain at least one vertex α. That means that for each one-element frequent set α all prefix paths of the FP-tree, in which set α is the last vertex, must be found.

Main advantages of FP-growth algorithm are [4], [7]:

- Transparent structure of FP-tree, which reduces the necessity of subsequent database access.
- FP-tree will never become larger than original data set. Usually data undergoes lossless conversion in the process of searching for sequences.
- *Divide and Conquer*—leads to partitioning the whole process of searching into smaller ones, operating on reduced amount of data.
- No need to re-scan the data set during generation of candidates, which leads to a large enhancement in performance.
- Usually faster than Apriori algorithm.
- Good for use on transactional data sets, in which there are no relations between elements.

D. Eclat Algorithm

Eclat is one of the best association rule finding algorithms in the case of dense databases [8]. It fulfilled the need for a quick association rule finding algorithm, with a single database reading, which significantly reduces usage of resources.

Most of association rule finding algorithms, i.e. Apriori and Eclat, utilize the alphabetical ordering of data sets *P(A)* in order to avoid additional, unnecessary checks and consequently eliminate redundant calculations. Order of elements is defined by their prefixes, which are arranged in a lexicographic succession.

Eclat algorithm utilizes vertical orientation of data in a database, where every important information is stored in a tid-list (sorted list of transaction identifiers, in which a particular element exists) [8]. A vertical arrangement of data does not include any additional calculation costs related to finding potential candidates, as it is in the case of horizontal orientation of database.

In the case of a parallel version of Eclat algorithm, every processor calculates all frequent sets in a single equivalency class before going to another. Thanks to that, local database is scanned only once. Eclat does not require additional calculations during building and searching complex data structures, neither does it generate all subsets of every transaction. It must be noted, that tid-lists automatically drop unimportant transactions and with the growth of number of elements the size of the tid-list shrinks. It is also worth noting that conversion of a database from vertical to horizontal orientation proceeds in a very simple way and we can always recover our initial database.

Eclat algorithm utilizes new clustering techniques for approximating a set of potentially maximal frequent sets and finding frequent sets contained within every cluster. Its searching techniques are based on bottom-up strategy.

To calculate support of any set, common part of tid-lists from its two subsets must be calculated. Frequent sets are generated by discovering common parts of tid-lists of all pairs of different atoms and by checking their support. In a bottom-up approach we recursively repeat this procedure for subsequent levels of frequent sets found. This process is executed until all frequent sets are found.

It must be mentioned, that frequent sets are sorted in ascending order based on their support value, calculated from tid-list. It reduces number of generated candidate sets. In Eclat algorithm sorting is executed in each recursive call of procedure.

Generally speaking, Eclat is one of the best currently available algorithms for frequent sets mining. Its only downside is the need to remember (sometimes quite large) amount of intermediate tid-lists, but rarely we can experience effects of this flaw.

E. Utilization of Algorithms in MLDS

Money Laundering Detection System uses the above described algorithms to find recurring pattern sequences. Data mining algorithm returns as a result a tree, which presents sequences that fulfill conditions imposed by parameters supplied by user. List of operations in this structure is a representation of connections between two bank accounts, which are elements of a sequence. As we can see in Fig. 1 existence of transfer between accounts ending with 0057 and 6902 is directly related to operation between accounts 9604 and 5188. Graphical presentation of results should allow police analyst to discover any suspicious operations. This should also help tracking numerous transactions which seem to be not connected with each other but in reality may be linked with a money laundering process.

An important factor of searching for frequent sequences is the fact, that order of operations in a single transaction does not matter. Presented order is provisional and is only supposed to point out to a narrower area of analysis.

Criminal analyst builds his project in many stages, unfortunately, he often does not have access to all the necessary data but has to repeatedly ask banks or other officials for them. On the basis of provided data he can then adapt analysis scope, and formulate conclusions.

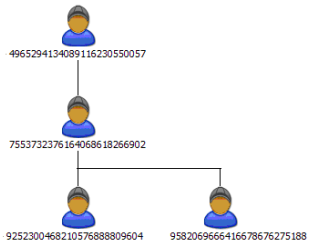


Fig. 1. Graph of frequent sequences

IV. Experimental Comparison of Algorithms

A. First Study: Apriori and PrefixSpan

In the first study, the comparison of the characteristics of algorithms was based on additional software module, which allows to run algorithms in a sequence with parameters chosen for the analysis. These values of parameters are set on the basis of supplied value of incremental factor and are always contained within provided boundaries.

Comparison was based on the following parameters: desired minimal support for frequent sequences, size of the input file with the data for analysis, diversity of connections (number of existing accounts) and type of the algorithm that performs the analysis.

Experiments were also conducted on the basis of changing diversity of test data. The third parameter—the diversification of ties—had an effect on the input files (that had always the same number of transactions), but also changed the internal links between the operations and thus changed the number of existing accounts.

During the next step, algorithm comparing mechanism presents in tabular form the following data: run-time, memory usage, volume of input set, logs or algorithms' run parameters.

Experimental comparison of time performance of algorithms depending on the specified minimum support was made. In the case of lesser minimal support the candidate set is greater. Results presented in Fig. 2 shows that in the case when minimal support is small and thus number of candidates is larger in consecutive iterations, the Apriori algorithm is considerably slower than PrefixSpan. The difference is almost sevenfold with support set to 0.02.

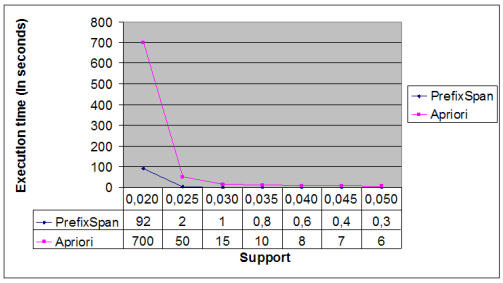


Fig. 2. Relationship between run-time and minimal support value

The presented results show that PrefixSpan is characterized by a much higher efficiency than the basic Apriori algorithm in the case of discovering sequence patterns. There are two main reasons:

1) In contrast to methods for pattern discovery based on the idea of Apriori algorithm, PrefixSpan builds frequent sequences incrementally on the basis of previously discovered ones.
2) In the step in which we search for frequent sequences, PrefixSpan analyzes projections of database, which are considerably smaller than the original one. Moreover, in subsequent iterations of the algorithm, the size of these databases significantly diminishes.

It is easy to note, however, that the main cost of PrefixSpan algorithm lies in the construction of projection databases. If we could reduce the size and the number of projection data bases generated by the algorithm, it would significantly improve the efficiency of the algorithm.

One way to reduce the number and size of projection databases generated by the algorithm is to use two levels PrefixSpan projection scheme (called bi-level projection).

Next experiment was aimed at finding out how diversification of the number of elements in the transaction impacts the efficiency and speed of algorithms. The goal was to show the effects of variation in the number of elements of transactions. Differentiation of the transaction was carried out on the basis of the number of underlying accounts, which continue to be generated by a set of fixed-length transaction. Diversification has an impact on the length of the candidate sets, and thus affects the size of candidate sets and the use of memory, and finally run time.

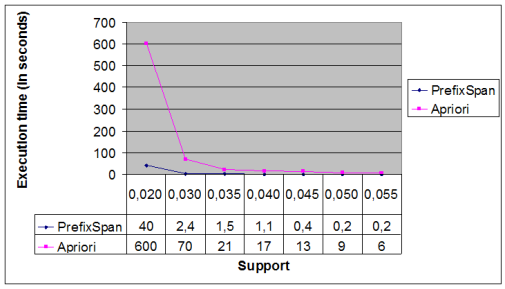


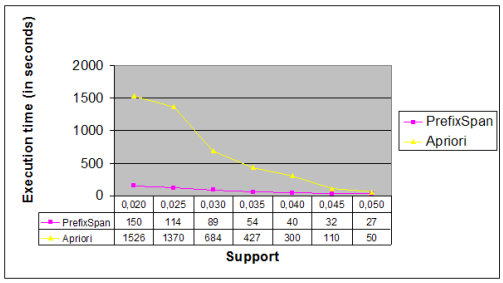
Fig. 3. Relationship between run time and elements diversification (maximum 100 elements in the transaction)

In Fig. 4 we can see clearly that diversity has an impact on performance of algorithms. Execution time was greatly extended when diversity reached 500, due to the long amount of candidate sets, which directly affect the performance of the algorithms.

After an increase of the number of elements that could be appended to a transaction, run times of algorithms significantly increased. However, we note that the PrefixSpan algorithm, due to the cutting of branches of early-generated sets of candidates behaved much more efficiently than Apriori algorithm.

B. Second Study: Apriori, FP-growth and Eclat

For the second study a number of tests utilizing proprietary testing software were executed. In each case a random trans-



	0,020	0,025	0,030	0,035	0,040	0,045	0,050
PrefixSpan	150	114	89	54	40	32	27
Apriori	1526	1370	684	427	300	110	50

Fig. 4. Relationship between run time and support value (diversity set to 500)

action database was generated. Tests were focused on number of elements in a single transaction, number of transactions and minimal support. Each transaction element was a number randomly generated in a range from 0 to a set number. In Tab. I values of parameters are shown.

TABLE I
VALUES OF PARAMETERS USED IN THE SECOND SET OF EXPERIMENTS

Exp. no.	Parameters				Frequent sets						
	Min. support	No. of trans.	No. of elements in trans.	Range of values	L1	L2	L3	L4	L5	L6	L7
1	0.002 (2/1000)	1000	1–3	0–10	10	45	68	-	-	-	-
2	0.002 (2/1000)	1000	1–4	0–10	10	45	116	27	-	-	-
3	0.002 (2/1000)	1000	1–5	0–10	10	45	120	156	11	-	-
4	0.002 (2/1000)	1000	1–8	0–10	10	45	120	210	251	107	8
5	0.0007 (2/3000)	3000	1–3	0–10	10	45	119	-	-	-	-
6	0.0007 (2/3000)	3000	1–4	0–10	10	45	120	121	-	-	-

TABLE II
RESULTS OBTAINED IN THE SECOND SET OF EXPERIMENTS

Exp. no.	Apriori	FP-Growth	Eclat
1	13 ms	21 ms	3 ms
2	16 ms	24 ms	4 ms
3	21 ms	30 ms	4 ms
4	42 ms	58 ms	9 ms
5	26 ms	257 ms	6 ms
6	34 ms	63 ms	9 ms

Results presented in Tab. II highlight the character of Eclat algorithm that behaves very good when applied to multi-element transactions, which means that it is immune to number of elements in transactions.

For the rest of tests the following naming convention was established: *id*—id number, *sup*—minimal support, *tNum*—number of transactions in the database, *tEl*—maximum number of elements in the transaction, *tRg*—range of generated element values, *fqNum*—number of returned frequent sets, *Ap[ms]*—execution time of Apriori algorithm in milliseconds, *FP[ms]*—execution time of FP-growth algorithm in milliseconds, *Ec[ms]*—execution time of Eclat algorithm in milliseconds.

id	sup	tNum	tEl	tRg	fqNum	Ap[ms]	FP[ms]	Ec[ms]
1	0,0500 (500/10000)	10000	10	1000	0	859	1656	35
2	0,0200 (200/10000)	10000	10	1000	0	163	1576	24
3	0,0100 (100/10000)	10000	10	1000	0	154	1626	28
4	0,0050 (50/10000)	10000	10	1000	764	10099	3326	2265
5	0,0020 (20/10000)	10000	10	1000	1000	15917	4359	2948
6	0,010 (10/10000)	10000	10	1000	1000	15823	3688	2963
7	0,0005 (5/10000)	10000	10	1000	1013	16237	4539	3397

Fig. 5. Results for tNum=10000, tEl=10, tRg=1000

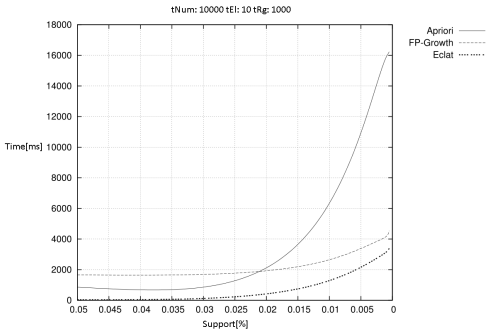


Fig. 6. Results for tNum=10000, tEl=10, tRg=1000

Results presented in Fig. 5–10 show that Eclat algorithm is the fastest, but does not perform that well in the case of sets with a large diversity of elements. This technique has the largest advantage in the case of dense data sets.

Performed tests show clearly, that Apriori algorithm is outperformed by FP-growth and Eclat. Apriori does not perform well in the case of sets with large diversity and large number of transaction elements. Eclat algorithm increased its distance from FP-growth during third test (Fig. 9 and Fig. 10) as compared to the performance displayed during second test (Fig. 7 and Fig. 8), showing that it performs better in the case of large sets of data with minimal diversity. As mentioned before, Eclat algorithm is not influenced by the number of

id	sup	tNum	tEl	tRg	fqNum	Ap[ms]	FP[ms]	Ec[ms]
1	0,0500 (500/10000)	5000	25	1000	0	798	567	32
2	0,0200 (200/10000)	5000	25	1000	0	177	486	34
3	0,0100 (100/10000)	5000	25	1000	0	152	519	26
4	0,0050 (50/10000)	5000	25	1000	975	19582	1695	3570
5	0,0020 (20/10000)	5000	25	1000	1000	18842	2131	3760
6	0,010 (10/10000)	5000	25	1000	1000	18082	1321	4063
7	0,0005 (5/10000)	5000	25	1000	3043	18093	1337	4026

Fig. 7. Results for tNum=5000, tEl=25, tRg=1000

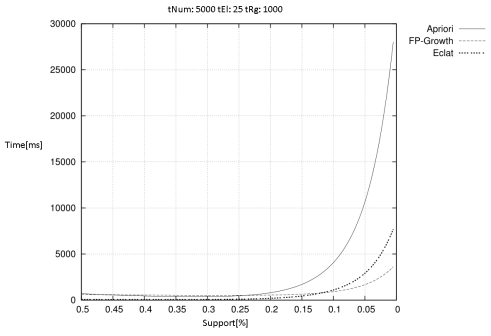


Fig. 8. Results for tNum=5000, tEl=25, tRg=1000

id	sup	tNum	tEl	tRg	fgNum	Ap[ms]	FP[ms]	Ec[ms]
1	0,0500 (500/10000)	5000	39	942	0	632	587	53
2	0,0200 (200/10000)	5000	39	942	0	223	555	55
3	0,0100 (100/10000)	5000	39	942	673	12864	2707	3057
4	0,0050 (50/10000)	5000	39	942	942	21621	3346	5558
5	0,0020 (20/10000)	5000	39	942	942	22739	3003	5644
6	0,010 (10/10000)	5000	39	942	1241	21954	2716	5246
7	0,0005 (5/10000)	5000	39	942	62790	862707	19920	15118

Fig. 9. Results for tNum=5000, tEl=39, tRg=942

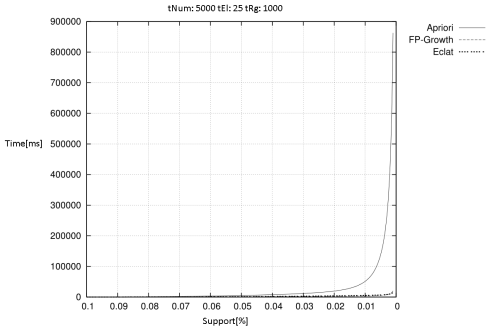


Fig. 10. Results for tNum=5000, tEl=39, tRg=942

elements in transactions.

id	sup	tNum	tEl	tRg	fgNum	Ap[ms]	FP[ms]	Ec[ms]
1	0,1000 (5/50)	50	3	3	6	1	2	0
2	0,2000 (10/50)	50	3	3	5	1	1	1
3	0,4000 (20/50)	50	3	3	3	1	2	0
4	0,8000 (40/50)	50	3	3	0	2	0	1
5	0,1000 (20/200)	200	3	3	6	2	3	0
6	0,0667 (20/300)	300	3	3	6	3	3	0
7	0,1000 (30/300)	300	3	3	6	3	2	1
8	0,0667 (20/300)	300	3	3	7	2	3	1
9	0,0500 (20/400)	400	3	3	7	2	4	0
10	0,0020 (2/1000)	1000	3	3	7	4	9	1
11	0,0010 (2/2000)	2000	3	3	7	8	19	2
12	0,0007 (2/3000)	3000	3	3	7	12	36	3
13	0,005 (2/4000)	4000	3	3	7	16	193	3
14	0,0004 (2/5000)	5000	3	3	7	20	263	5
15	0,0003 (2/6000)	6000	3	3	7	24	299	5
16	0,0003 (2/7000)	7000	3	3	7	30	566	8
17	0,0002 (2/8000)	8000	3	3	7	35	885	7
18	0,0002 (2/9000)	9000	3	3	7	37	782	8
19	0,0020 (2/1000)	1000	5	10	326	24	32	4
20	0,0010 (2/2000)	2000	5	10	391	33	53	8
21	0,0007 (2/3000)	3000	5	10	424	47	83	11
22	0,005 (2/4000)	4000	5	10	451	221	106	15
23	0,0004 (2/5000)	5000	5	10	461	71	314	17
24	0,0003 (2/6000)	6000	5	10	484	82	555	23
25	0,0003 (2/7000)	7000	5	10	498	96	827	26
26	0,0002 (2/8000)	8000	5	10	524	114	775	29
27	0,0002 (2/9000)	9000	5	10	559	131	1487	32
28	0,0200 (20/1000)	1000	3	10	51	8	11	2
29	0,0100 (20/2000)	2000	3	10	55	14	24	4
30	0,0067 (20/3000)	3000	3	10	55	22	44	5
31	0,0050 (20/4000)	4000	3	10	55	28	421	7
32	0,0040 (20/5000)	5000	3	10	56	37	360	10
33	0,0033 (20/6000)	6000	3	10	58	45	437	11
34	0,0029 (20/7000)	7000	3	10	72	53	756	13
35	0,0025 (20/8000)	8000	3	10	77	59	698	15
36	0,0022 (20/9000)	9000	3	10	101	68	1255	17
37	0,0200 (20/1000)	1000	5	10	62	11	17	3
38	0,0100 (20/2000)	2000	5	10	161	24	41	7
39	0,0067 (20/3000)	3000	5	10	175	37	72	10
40	0,0050 (20/4000)	4000	5	10	175	90	463	14
41	0,0040 (20/5000)	5000	5	10	180	62	401	17
42	0,0033 (20/6000)	6000	5	10	196	77	506	20
43	0,0029 (20/7000)	7000	5	10	218	78	584	25
44	0,0025 (20/8000)	8000	5	10	264	104	908	28
45	0,0022 (20/9000)	9000	5	10	311	119	933	32

Fig. 11. Collected results from tests with varying parameters

Fig. 11 shows tests with restricted value ranges for number of elements in transaction and number of transactions in data set. As we can see, Eclat algorithm performs really well in the case of such data sets, proving to be several times faster than the other methods. Nevertheless it is important to remember about the characteristics of the data set when we choose search algorithm. Analyzing the above results, we can say that Eclat algorithm is, on the average, several times better in terms of execution time, than Apriori and FP-growth methods implemented in the Money Laundering Detection System.

V. Summary and Conclusions

In this paper we presented four algorithms that are the core of Money Laundering Detection System. Their overall characteristics were presented from a perspective of incorporating them into a system for police analysts. We can clearly see why data mining problems are non-trivial and why searching for a better solution is always necessary. Construction of Apriori, PrefixSpan, FP-growth and Eclat algorithms, as well as their performance were compared, in favor of the last one. The MLDS consists not only of these four algorithms, but a lot of other algorithms are still tested and added to the system (for example social network analysis algorithms [3]).

In the current state the system allows the user to conduct all actions required for performing the analysis. This includes importing transactions and accounts from data supplied by banks (which may exist in many different formats), clustering them, analyzing in multiple iterations and presenting results in a graphical form. The MLDS is capable of being an invaluable aid to money laundering investigators and is being used in practice by Polish Police.

Acknowledgments

This research was partially supported by a grant “A system for collecting and generating information for criminal analysis and activities coordination of the Border Guard” (No. DOB-BIO6/08/129/2014) from the Polish National Centre for Research and Development and by Polish Ministry of Science and Higher Education under AGH University of Science and Technology Grant (statutory project).

References

[1] R. Dreżewski, J. Sepielak, and W. Filipkowski, “System supporting money laundering detection,” *Digital Investigation*, vol. 9, no. 1, pp. 8–21, 2012.

[2] J. Dajda, R. Dębski, A. Byrski, and M. Kisiel-Dorohinicki, “Component-based architecture for systems, services and data integration in support for criminal analysis,” *Journal of Telecommunications and Information Technology*, no. 1, pp. 67–73, 2012.

[3] R. Dreżewski, J. Sepielak, and W. Filipkowski, “The application of social network analysis algorithms in a system supporting money laundering detection,” *Information Sciences*, vol. 295, pp. 18–32, 2015.

[4] J. Han and M. Kamber, *Data Mining: Concept and Techniques*. Morgan Kaufmann, 2006.

[5] R. Agrawal and R. Srikant, “Fast algorithms for mining association rules,” in *VLDB ’94: Proceedings of the 20th International Conference on Very Large Data Bases*. Morgan Kaufmann Publishers Inc., 1994, pp. 487–499.

[6] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and H. M., “Prefixspan: Mining sequential patterns by prefix-projected growth,” *Proceedings of the 17th International Conference on Data Engineering*, pp. 215–224, 2001.

[7] J. Han, J. Pei, and Y. Yin, “Mining frequent patterns without candidate generation,” in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, W. Chen, J. F. Naughton, and P. A. Bernstein, Eds. ACM, 2000, pp. 1–12.

[8] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, “New algorithms for fast discovery of association rules,” in *3rd International Conference on Knowledge Discovery and Data Mining(KDD)*, 1997.