The Solution of the Profiling Problem Based on Data Analysis

Vyacheslav Busarov
Saint Petersburg State University
St. Petersburg, Russia
VyacheslavBusarov@gmail.com

Natalia Grafeeva
Saint Petersburg State University
St. Petersburg, Russia
N.Grafeeva@spbu.ru

Abstract—This article describes the problem of profiling of objects with varied characteristics. This is relevant for today task, which requires working with large amounts of data and cannot be solved manually. In this paper we describe a solution based on search algorithms of frequent itemsets. The article also presents the results of a comparative analysis of the algorithms (which was the subject of a separate research). The main focus of the article is aimed at substantiation of the choice of an appropriate algorithm based on some characteristic of the source dataset. The study is based on work with actual data of the restaurant industry. Despite this, the results have a wide application, as the approach described in this article can easily be generalized to datasets in any other industry. This article is a logical continuation of the article published in [5].

I. Introduction

There are a number of objects and an ordered set of homogeneous common features characterizing these objects. The input data are sets of signs. Each set of signs is specific to a particular object and describes it. For each object we want to find a set of signs which are most fully and succinctly characterize the object. This set of signs will be called a profile. In the general case it will be useful for:

- acquiring a brief and succinct characteristic of an object;
- studying properties associated with the profiles of the object;
- acquiring new knowledge about the object;
- forming of assumptions about the future behavior/use of the object.

One particular case of this problem is the problem of profiling the network of restaurants. Objects in this case are restaurants, signs are dishes offered in the menus, sets of signs are paid bills at the restaurants. If we solve the problem of profiling, we'll get the standard set of the most popular dishes in each restaurant. This will help to optimize the purchase of products, to manage prices and assortment, to generate personalized recommendations, to plan advertising campaign, etc.

To solve the problem of profiling of the network of restaurants we decided to use the algorithm for finding frequent itemsets, which is a part of the solution of the famous problem of finding Association rules. The subject of Association rules was introduced in 1993 by R. Agrawal [1] for solving problems of market basket analysis. All the algorithms solving this problem operate in two stages: the search of frequent itemsets and the formation of Association rules based on the frequent itemsets. The second part has long been considered as the best and has a widely recognized decision. All the research up to 2015 [9] aim to improve the performance of the first part, because it is recognized as the most computationally difficult.

We will try to find the most computationally efficient algorithm among currently available algorithms.

II. Comparative analysis of algorithms

Finding frequent itemsets was first considered critical to mining association rules in the early 1990s. In the subsequent two decades, there have appeared numerous new methods of finding frequent itemsets, which underlines the importance of this problem. The number of algorithms has increased, thus making it more difficult to select a proper one for a particular task and/or a particular type of data. All algorithms need to work with large databases of source data. This circumstance complicates the analysis, therefore, an important aspect of these algorithms is the method of storing input data. The most relevant solutions move from characteristics to symbols (or sequences of characters of fixed length), and thus reduce the task of storing the original samples to storage of the vocabulary of transactions.

For the comparative study, we analyzed all known algorithms for finding frequent itemsets. But we did not consider algorithms with any specific assumptions.

Numerous scientific studies in the direction of fast search of frequent itemsets can be divided into two classes [5].

The first is "candidate-generation-and-test". A characteristic example of the first category of algorithms is Apriori (R. Agrawal, 1994[2]). This category also includes all the subsequent variations of this algorithm based on the anti-monotony principle (the support of a set of attributes does not exceed the support of any of its subsets [1]). Such algorithms generate a set of length $k+1$ based on the previous sets having length k. Even though the anti-monotony property principle allows us to disregard quite a few variants, such algorithms are not efficient computationally if initial data is extensive (the number of itemsets or the length of sets).

The second class is "pattern-growth method". A good example of the second category of algorithms is called FP-

Growth (J. Han2000 [10]).Algorithms of this type perform search in a recursive manner by breaking down a database into several parts and looking for local answers that are subsequently combined into an overall result. The algorithms of this type generate fewer candidates than the algorithms described above, which allows to save a considerable amount of memory. However, the productivity of such algorithms largely depends on the homogeneity of initial data.

Deep comparative analysis of algorithms for finding frequent itemsets was made in [5]. In this work algorithms published in [1-4], [6-13], [15,16] were examined. In order to obtain an overall view of the compared algorithms, we introduced binary directed relationships between the algorithms to reflect improvement of productivity and decreasing of memory used (based only on the experiments described in the articles mentioned above). According to the authors of the articles, each pair of algorithms was compared under identical conditions pertaining to the hardware characteristics, data, the language of realization, etc. Based on the relationships introduced between the algorithms, we made a chart of execution time (Fig. 1) and a chart of memory requirements (Fig. 2).

It should be noted that there is a two-direction relationship between some of the algorithms (for example, between FPGrowth and LCMFreq v.2/v.3 in Fig. 1, EClat and dEClat in Fig. 2), which means that, depending on the type of initial data, such algorithms demonstrate approximately equal results and are considered on a par in this respect.

In the same paper [5] we have described experiments on a single platform with a common dataset Mushrooms with the participation of all the currently known algorithms. The experiment results (Table I) do not contradict the results given above, which again confirms their authenticity. All the further experiments were run on a computer with the following processor: Intel(R) Core(TM) i7-4700HQ CPU @ 2.40GHz, 12.0 Gb RAM. All the algorithms were realized on Java 8. The average transaction length is 23, the glossary of attributes consists of 119 elements, and the total number of transactions amounts to 8124.

TABLE I. EXECUTION TIME OF EXPERIMENTS WITH MUSHROOM DATASET (SEC)

Minimal support	70%	75%	80%	85%	90%
Relim	397,62	372,18	341,12	332,89	327,92
PrePost+	401,15	373,32	351,87	337,6	328,12
FIN	412,45	383,78	362,91	349,45	340,11
PrePost	415,23	385,11	365,12	348,56	345,61
PPV	417,56	389,98	373,32	358,76	349,08
H-mine	419,35	393,02	374,79	360,16	350,38
dEclat	426,12	403,43	379,81	373,14	359,26
FP-Growth	427,89	404,14	377,02	375,51	362,97
LCMFreq	429,96	409,07	382,59	380,31	364,82
Eclat	434,92	416,07	391,19	390,69	375,19
Apriori Hybrid	437,53	419,23	392,98	394,11	378,85
Apriori	438,24	422,91	395,01	394,09	379,02

The fact that the results were obtained exactly from a practical point of view is important since the performance of the algorithms depends on real data, not from theoretical estimates of complexity. The experiments show that Relim [4] and Prepost+ [9] are the most effective. However, they are representatives of different approaches: the “pattern-growth method" and the "candidate generate and test". We continued the experiments to choose the optimal algorithm for future research. We got similar results on the Chess dataset (Table II). The average transaction length is 18, the glossary of attributes consists of 94 elements, and the total number of transactions amounts to 14587.

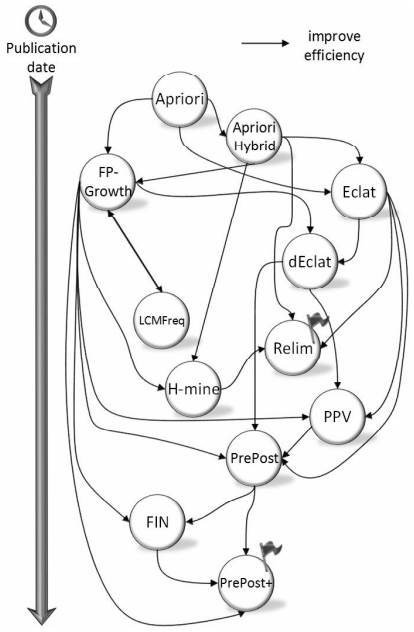

Fig. 1.The relationships between algorithms in terms of execution time. *Source:[5]*

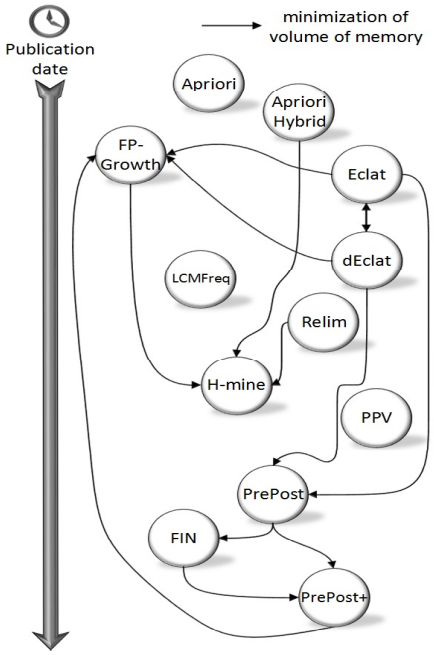

Fig. 2.The relationships between algorithms in terms of memory requirements. *Source:[5]*

We see that there are no fundamental differences in the relative efficiency of algorithms. The rating of the algorithms in terms of execution time has not changed. What is common in data structures of the selected datasets? It turns out that they are similar in the terms of indicator, which is called “average cover of a glossary". This concept for a dataset is defined as follows:

$$Average\ cover\ of\ a\ glossary = 1 / n * \sum(|\tau_i| / |D|) * 100$$

where D - the glossary of attributes, τ_i– transactions, n - the number of transactions.

TABLE II. EXECUTION TIME OF EXPERIMENTS WITH CHESS DATASET (SEC)

Minimal support	70%	75%	80%	85%	90%
Relim	788,24	735,89	682,24	675,43	647,88
PrePost+	807,21	740,64	703,74	678,12	648,24
FIN	825,82	753,93	723,85	698,99	672,26
PrePost	838,84	778,92	742,24	709,12	672,22
PPV	833,61	773,23	736,64	705,52	690,16
H-mine	840,74	787,78	749,61	720,47	700,76
dEclat	853,25	804,1	759,62	716,29	710,52
FP-Growth	854,22	816,82	762,04	751,02	716,14
LCMFreq	858,41	818,14	765,21	760,31	724,92
Eclat	880,64	832,14	782,39	780,46	740,47
Apriori Hybrid	874,42	838,46	785,66	788,98	751,85
Apriori	883,42	845,82	790,02	789,09	754,02

Datasets Mushrooms and Chess have *Average cover of a glossary* equal to 23.7% and 22.1%, respectively. However other datasets can have very different values of average cover indicator. How will the algorithms work with such datasets? Since public datasets do not have sufficient flexibility to more thoroughly examine this question, we wrote a utility to generate random datasets according to the specified parameters: the number of transactions, the capacity of a glossary and the average cover of a glossary.

We have generated a glossary consisted of 70 items, datasets consisted of 90 000 transactions, and varied the value of the average cover of the glossary. We have conducted experiments with a minimum support (MinSupport) equal to 80 and got the results shown in Table III.

TABLE III. EXECUTION TIME OF EXPERIMENTS WITH DIFFERENT VALUES OF AVERAGE COVER (SEC). MINSUPPORT = 80%

Avg cover	2%	5%	10%	15%	25%	30%
Relim	2356,47	2294,43	2263,08	2192,67	2208,48	2237,16
PrePost+	2104,21	2168,28	2237,52	2255,73	2270,33	2356,02
FIN	2197,34	2231,83	2280,14	2270,59	2299,36	2360,98
PrePost	2188,63	2215,87	2261,59	2249,41	2312,42	2367,82
PPV	2191,33	2206,87	2265,59	2255,41	2324,42	2373,82
H-mine	2446,26	2374,65	2306,13	2237,79	2340,69	2398,08
dEclat	2211,33	2226,87	2269,59	2275,41	2345,42	2393,82
FP-Growth	2516,67	2445,36	2405,94	2310,72	2347,95	2405,07
LCMFreq	2551,95	2471,91	2423,79	2350,38	2363,73	2431,08
Eclat	2381,24	2386,92	2419,91	2440,59	2478,63	2510,37
Apriori Hybrid	2390,42	2414,31	2452,13	2462,72	2491,82	2525,09
Apriori	2461,42	2493,31	2551,13	2574,72	2590,82	2610,85

It is interesting to note that the results of the experiments in Table III clearly demonstrate the difference in behavior of "candidate-generation-and-test" and "pattern-growth method" algorithms. If we place the results on different charts this difference becomes more noticeable (Fig. 3 and Fig. 4). Certain patterns of behavior common to all algorithms of each type become visible on these charts. "Candidate-generation-and-test" algorithms are better for small values of the average cover of a glossary (2-15%). At the same time, they are noticeably less efficient for large values of this index (15-30%). This fact may be explained: for small values of the average cover of a glossary heuristics effectively cut off unsuitable candidates, that improves the speed of the algorithm work. The growth of the average cover of a glossary makes heuristics work more and more rarely. It finally decreases the efficiency of the whole algorithm. However in this interval "pattern-growth method" algorithms effectively use their main advantage – the search of local results with their further extension. There are fewer local results, and they expand for a smaller number of steps because of smaller differences in transactions. The charts shows that the performance rankings of the algorithms when the value of the average cover of a glossary is equal 25% corresponds to the rankings of the experiments with the bases Mushrooms and Chess (whose indexes are equal to 23.7% and 22.1% respectively). This reinforces the validity of our experiments.

Previous studies have shown that the algorithms of the category "candidate-generation-and-test" are more efficient for small values of the average cover of a glossary, and "pattern-growth method" algorithms for large values. The absolute leaders in each of the categories are PrePost+ and Relim, respectively. We continued to look for the relationship between the operation time of the algorithms and metrics based on the lengths of the transaction, and considered the mathematical expectation of the length of the transaction. This concept for a dataset is defined as follows:

$$Mathematical\ expectation = \sum p_l * l$$

where l in $(1,N)$,
N - the maximum length of a transaction,
p_l - the relative probability of length l $(p_l = q_l / n)$,
n - the number of transactions,
q_l -the number of transactions of length l .

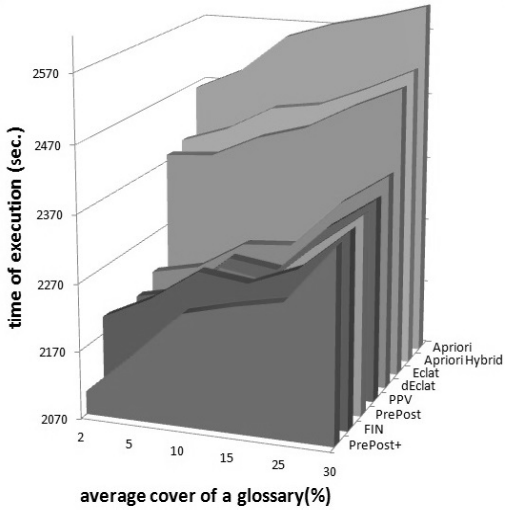

Fig. 3. The chart of execution time of "candidate-generation- and-test" algorithms (sec)

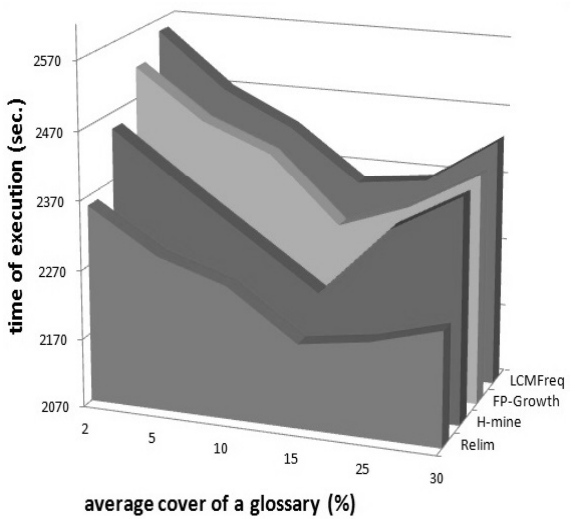

Fig. 4. The chart of execution time of "pattern-growth method" algorithms (sec)

Relim and PrePost+ algorithms are not only the most effective but also reflect the behavior of all algorithms for the respective classes. That is why in the future we have only considered them. We have upgraded the existing tool for generating test data set for the study of dependencies with the new metric and added the required mathematical expectation of the length of the transaction as a new input parameter. We left the experimental value of the minimum support equal to 80%, generated a database with 50000 transactions and varied the size of a glossary: 20, 50, 80 and 100 items.

TABLE IV. EXECUTION TIME OF PREPOST+ AND RELIM ALGORITHMS IN THE EXPERIMENTS WITH 20, 50, 80, 100 ITEMS

glossary size	expectation	0,2	0,3	0,4	0,5	0,6	0,7
20	PrePost+	382,72	389,48	405,23	413,07	420,15	429,69
20	Relim	430,61	417,8	411,45	398,84	404,19	411,26
50	PrePost+	771,34	780,91	792,32	798,64	810,49	817,82
50	Relim	830,14	813,79	807,02	790,41	795,65	801,74
80	PrePost+	1069,9	1088,8	1092,24	1108,75	1122,36	1136,67
80	Relim	1130,4	1110,7	1102,37	1084,52	1095,93	1101,66
100	PrePost+	1172,4	1194	1207,58	1224,97	1246,52	1271,18
100	Relim	1274,7	1245	1230,45	1204,24	1211,43	1220,51

The experiment results (Table IV, Fig. 5) confirm the concept of the common patterns of behavior for the algorithms that belong to the same class. At small values of the mathematical expectations of the lengths of the transactions, "candidate-generation-and-test" algorithms are faster than the “pattern-growth method" algorithms, however, for large values of mathematical expectations, the situation changes exactly the opposite. It is also worth noting that the running time of an algorithm PrePost+ with the increase of the mathematical expectation is growing disproportionately faster than the operation time of the Relim algorithm. This observation is also true for all members of each class relative to the average cover of a glossary (Fig. 3, Fig. 4).

The second observation that can be made on the results of these experiments: the size of the dictionary influences the angle of the graph execution time of the algorithm PrePost+. That is, the execution time depending on the mathematical expectation grows faster with increasing size of the dictionary. While Relim behaves much more stable, almost not changing the angle of the graph in different experiments.

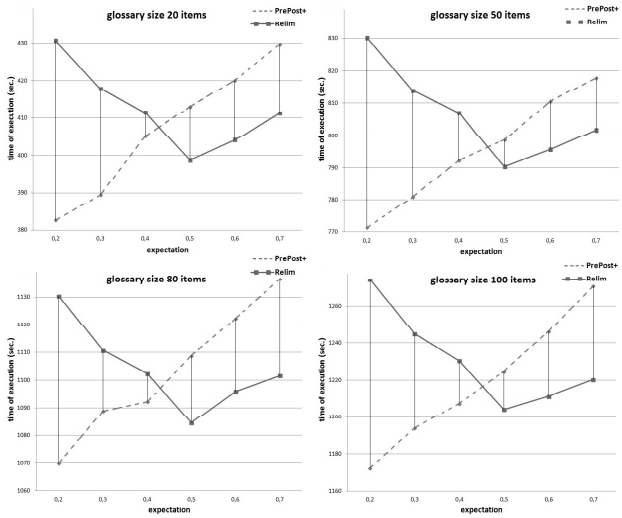

Fig. 5. The charts of execution time of PrePost+ and Relim algorithms at experiments with 20, 50, 80, 100 items

The experiments show that for small values of mathematical expectation of transaction length "candidate-generation-and-test" algorithms more efficient, however, with the increase of mathematical expectation, their efficiency is dramatically reduced. In this case are leading representatives of the "pattern-growth method", working with greater speed. Their rate of increase in execution time is much more stable. Brief descriptions of the algorithms Relim and PrePost+ are given below.

- ***Relim.*** This algorithm was proposed in 2005 in the study by Christian Borgelt [4]. The acronym "Relim" illustrates the underlying principle of the algorithm: "REcursiveELIMination scheme". Relim tries to find all frequent itemsets with a given prefix by lengthening it recursively and renewing support at the same time. The approach utilized here is called a "pattern-growth method".

- ***PrePost+.*** Proposed by Z. H. Deng [9] in 2015, it is the latest algorithm for identifying frequent itemsets. PrePost+ uses three data structures at a time: N-list, PPC-tree and set-enumeration tree, which explains why it requires more memory than FP-Growth does (as you can see in Fig. 2). Although it is an "Apriori-like" algorithm, it has empirically proved superior to many other algorithms in terms of execution time (Fig. 1).

III. THE SOLUTION OF THE PROBLEM OF PROFILING

We will describe the solution of the profiling of a chain of restaurants where each restaurant profile is a set of the most popular dishes. We solve the task for each object separately, despite the fact that they have common glossary of attributes

and the input set of transactions. The solution is divided into 5 stages:

1) Filtering of "empty" data. In order not to work with information, which made no sense, we need to get rid of attributes not used to describe objects. This obviously is done in linear time O(N) using O(M) additional memory, where N is the initial number of transactions, M is the number of attributes.

2) *Eliminating redundancy.* Each transaction from the original dataset actually contains information about a single client order, which is redundant information to build a profile of the restaurant. We don't care whether the data contained meals in a single check, it is important for us that they were ordered in the same place at the same day. So, on the basis of the initial transactions we can build generalized transactions with the help of aggregating orders inside of a restaurant for each date. In this case, associative rules for the generalized transaction will look like this: "If *a* is ordered in the restaurant, on the same day will be the ordered *b*." This optimization significantly reduces the number of input data. In addition, at this stage quantitative indicators are canceled. Therefore, the aggregated data will have the form, shown on Fig. 6. This step is accomplished in O(N) using O(1) additional memory.

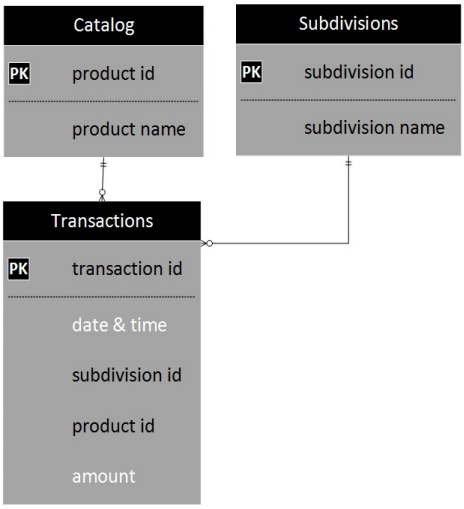

Fig. 6. Data structure. White attributes was deleted after upgrade

3) Formatting. Algorithms for finding frequent itemsets require input in the form of transactions, i.e. a set of elements, corresponding to the real transaction. Formatting the source data is performed in O(N) time and requires O(1) memory.

4) The choice of a suitable algorithm. We found that the average cover of a glossary y for the original dataset is equal to 29.5%. Therefore, we will use the most computationally efficient algorithm - Relim[4] (if the index was less than 15%, we would have used PrePost+[9]).

5) *The allocation of profiles*. Input parameter Minimal Support directly affects the number of items in frequent itemsets. Adjustment of this parameter is carried out experimentally. The transition from frequent itemsets to the profiles provided by the intersection of the results obtained for the selected values of minimal support.

The undoubted advantage of the above solution is the analysis of features in combination, rather than a simple sort by quantity of item. There is also a possibility of seasonal analysis, i.e. analysis of data corresponding periodic time intervals (for example: winter, spring, fall or summer). Optimization and clipping that occur in stages 1 - 2 significantly speed up the algorithm without compromising the end result, and the choice in step IV the most suitable algorithm allows to obtain the most computationally efficient solution.

IV. Experiments

The restaurant chain has provided a dataset, which consists of three tables (Fig. 6). Initially the table "Transaction" consisted of approximately 4900000 lines describing the client checks on all restaurants in the city. It is obvious that profiling wouldn't make sense, if the network consisted of one restaurant because one of the motivations of this task - the search for distinguishing features of objects. In this case, there were 23 restaurants. Each restaurant was analyzed individually, but because of the common glossary of items (dishes) we got matching answers.

It should be noted that the operation of a generalization conducted on the second step, significantly improved the operating time of the entire algorithm as a whole, as it reduced the initial number of transactions on average 2 orders of magnitude. For example, in one of the restaurants, the number of transactions was equal to 299095 before generalization and 1459 after generalization. There is some knowledge, which was extracted from source data:

- In the restaurants located on the territory of the airport, the largest demand is for different kinds of sandwiches. It can be assumed that customers take sandwiches on board the aircraft;
- In many restaurants, customers often order different kinds of coffee and desserts, while in "White gardens", "White nights" and "Moscow city", according to a set of dishes, people prefer to dine (in the profiles, almost no coffee and desserts, mostly hearty meals);
- In restaurants "Garage" and "Coffee shop on Fontanka" customers often have breakfast (porridge, scrambled eggs, pancakes and etc.).

In all experiments, the minimal support is large enough. This is due to the peculiarities of the basic problem: too large profiles obtained with a larger number of frequent itemsets, not so aptly characterize the object, as more compact. What facts can be also extracted from the resulting profiles? We can make the most popular combination of drinks and desserts in individual restaurants, prepare promotions and integrated offerings or make something similar. Do not think that the examples of the profiles are evident for the employees of the restaurants. This is absolutely wrong, as each of the 23 restaurants initially has approximately 300000 transactions.

Large and compact profiles were found during the analysis process. We can get even more non-obvious knowledge, if we will take their difference for each feature and thus will throw away all trivial combinations.

Thus, the experiments conducted in section II, support the hypothesis that there is a correlation between *Average cover of a glossary*, *Mathematical expectation* and execution time of algorithms for finding frequent itemsets. In the same section were formulated the selection criteria for the most efficient algorithm depending on the properties of the original data. The experiments conducted in section IV demonstrated the practical results of the profiling algorithm on real data and illustrated important optimization stages.

V. Conclusion

So, in this paper we gave an algorithm solving the problem of profiling based on data analysis. The effectiveness of the proposed solutions is based on:

- preliminary eliminating redundancy to reduce the data volume;
- a reasonable selection of the most computationally efficient algorithm of finding frequent itemsets depending on the characteristics of the original dataset;
- an efficient algorithm for generating profile on the basis of the found frequent itemsets.

The paper presents a comparative analysis of all currently available algorithms for finding frequent itemsets. The paper presented a comparative analysis of all currently available algorithms for finding frequent itemsets, highlighted patterns of behavior algorithms of different classes and the results of experiments demonstrating the dependence of the behavior of the algorithms and the indicator "average cover of a glossary". As a result we received a criterion for the selection of the optimal algorithm for solving the problem of profiling depending on the source data. The practical result was the creation of profiles through the processing of relevant data in real chain of restaurants. The results may have wider application, as the solution can easily be generalized to similar data of any other industry and can be integrated into analytical software packages. In the future we plan to apply a more complex approach to the processing of quantitative indicators.

References

[1] R.Agrawal , T.Imielinski, A.Swami, Mining associations between sets of items in large databases, *in Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1993, pp.207-216.

[2] R.Agrawal, R.Srikant, Fast algorithms for mining association rules. *in Proc. 20th Int. Conf. on Very Large Databases*,1994, pp,487-499.

[3] R.Agrawal, H.Mannila, R.Srikant, H.Toivonen, Fast discovery of association rules. *Advances in knowledge discovery and data mining.* AAAI MIT Press, 1996,12(1), pp.307-328.

[4] C.Borgelt, Keeping things simple: finding frequent item sets by recursive elimination, *in Proc. Open Source Data Mining Workshop*, 2005, pp.66-70.

[5] V.Busarov, N.Grafeeva, and E.Mikhailova, A Comparative Analysis of Algorithms for Mining Frequent Itemsets, *in Proc. 12th International Baltic Conference, DB&IS*, 2016, pp. 136–150.

[6] Z.H.Deng, Z.Wang : A New Fast Vertical Method for Mining Frequent Patterns, *International Journal of Computational Intelligence Systems*, 2010, 3(6), pp.733-744 .

[7] Z.H.Deng, Z.Wang, J.Jiang: A new algorithm for fast mining frequent itemsets using N-Lists. *Science China Information Sciences*, 2012, 55 (9), pp.2008-2030.

[8] Z.H.Deng, S.L.Lv: Fast mining frequent itemsets using Nodesets. *Expert Systems with Applications*, 2014, 41(10), pp.4505–4512 .

[9] Z.H.Deng, S.L.Lv: PrePost+ : An efficient N-lists-based algorithm for mining frequent itemsets via Children–Parent Equivalence pruning. *Expert Systems with Applications,* 2015, 42(10), pp. 5424 – 5432 .

[10] J.Han, H.Pei, Y.Yin: Mining frequent patterns without candidate generation, *in Proc. ACM SIGMOD Int. Conf. on Management of data*, 2000, pp. 1-12.

[11] J.Pei, J.Han, H.Lu, S.Nishio, S.Tang , D Yang, H-Mine: Fast and space-preserving frequent pattern mining in large databases, *IIE Transactions*, 2007, vol. 39(6), pp. 593–605.

[12] T.Uno, M.Kiyomi, H. Arimura, LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets, *in Proc. Workshop on Frequent Itemset Mining Implementations*, 2004.

[13] T.Uno, M. Kiyomi, H.Arimura, LCM ver.3: Collaboration of array, bitmap and prefix tree for frequent itemset mining, *in Proc. Open Source Data Mining Workshop*, 2005, 77-86.

[14] S.Vinay, S.Nisarg, P.Samay, D.Nirali, C.Nimisha, P.Umang, P.Ankit, C.Vishvash, and P.Ashis, A Study of Various Projected Data based Pattern Mining Algorithms, *Research Journal of Material Sciences*, 2013, vol. 1(2), pp.1-5.

[15] M.J.Zaki, Scalable Algorithms for association mining, *IEEE transaction on knowledge and data engineering*, 2000, vol.12, no. 3, pp.372-390.

[16] M.J.Zaki, K.Gouda, Fast vertical mining using diffsets, *in Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2003, pp.326–335.