A One-Pass Triclustering Approach: Is There any Room for Big Data?

Dmitry V. Gnatyshak[1], Dmitry I. Ignatov[1], Sergei O. Kuznetsov[1], and Lhouari Nourine[2]

[1] National Research University Higher School of Economics, Russian Federation
dmitry.gnatyshak@gmail.com
http://www.hse.ru
[2] Blaise Pascal University, LIMOS, CNRS, France
http://www.univ-bpclermont.fr/

Abstract. An efficient one-pass online algorithm for triclustering of binary data (triadic formal contexts) is proposed. This algorithm is a modified version of the basic algorithm for OAC-triclustering approach, but it has linear time and memory complexities with respect to the cardinality of the underlying ternary relation and can be easily parallelized in order to be applied for the analysis of big datasets. The results of computer experiments show the efficiency of the proposed algorithm.

Keywords: Formal Concept Analysis, triclustering, triadic data, data mining, big data

1 Introduction

Cluster analysis of multimodal data and specifically of dyadic and triadic relations is a natural extension of the idea of normal clustering. In dyadic case biclustering methods (the term bicluster was coined by B. Mirkin [17]) are used to simultaneously find subsets of the sets of objects and attributes that form homogeneous patterns of the input object-attribute data. One of the most popular applications of biclustering is gene expression analysis in Bionformatics [16,3]. Triclustering methods operate in triadic case in which for each object-attribute pair one assigns a set of some conditions [18,8,5]. Both biclustering and triclustering algorithms are widely used in such areas as the analysis of gene expression [21,15,13], recommender systems [19,10,9], social networks analysis [6], etc. The processing of numeric multimodal data is also possible by modifications of existing approaches for mining binary relations [12].

Though there are methods that can enumerate all triclusters satisfying certain constraints [1] (in most cases they ensure that triclusters are dense), their time complexity is rather high, as in the worst case the maximal number of triclusters usually is exponential (e.g. in case of formal triconcepts), showing that these methods are hardly scalable. To process big data algorithms need to have at most linear time complexity and be easily parallelizable. Also, in most cases, it is necessary that such algorithms output the results in one pass.

In order to create an algorithm satisfying these requirements we adapted a triclustering method based on prime operators (prime OAC-triclustering method) [5]. As the result we developed an online version of prime OAC-triclustering method, which is linear, one-pass and easily parallelizable.

The rest of the paper is organized as follows: in Section 2 we recall the method and the basic version of the algorithm of prime OAC-triclustering. In Section 3 we describe the online setting for the problem and the corresponding online version of the basic algorithm with some optimizations. Finally, in Section 4 we show the results of some experiments which demonstrate the efficiency of the online version of the algorithm.

2 Prime object-attribute-condition triclustering method

Prime object-attribute-condition triclustering method based on the framework of Formal Concept Analysis [20,4,2] is an extension for the triadic case of object-attribute biclustering method [7]. Triclusters generated by this method have the same structure as the corresponding biclusters, namely the cross-like structure of triples inside the iput data cuboid (i.e. formal tricontext).

Let $\mathbb{K} = (G, M, B, I)$ be a triadic context, where G, M, B are respectively the sets of objects, attributes, and conditions, and $I \subseteq G \times M \times B$ is a triadic incidence relation. Each prime OAC-tricluster is generated by applying the following prime operators to each pair of components of some triple:

$$\begin{gathered}(X,Y)' = \{b \in B \,|\, (g,m,b) \in I \text{ for all } g \in X, m \in Y\},\\ (X,Z)' = \{m \in M \,|\, (g,m,b) \in I \text{ for all } g \in X, b \in Z\},\\ (Y,Z)' = \{g \in G \,|\, (g,m,b) \in I \text{ for all } m \in Y, b \in Z\}\end{gathered} \tag{1}$$

Then the triple $T = ((m,b)', (g,b)', (g,m)')$ is called *prime OAC-tricluster* based on triple $(g,m,b) \in I$. The components of tricluster are called, respectively, *extent*, *intent*, and *modus*. The triple (g,m,b) is called a *generating triple* of the tricluster T. Figure 2 shows the structure of an OAC-tricluster (X,Y,Z) based on triple $(\widetilde{g}, \widetilde{m}, \widetilde{b})$, triples corresponding to the gray cells are contained in the context, other triples may be contained in the tricluster (cuboid) as well.

The basic algorithm for prime OAC-triclustering method is rather simple (Alg. 1). First of all, for each combination of elements from each two sets of $\mathbb{K}$ we compute the results of applying the corresponding prime operator (we will call the resulting sets *prime sets*). After that we enumerate all triples from I and on each step we must generate a tricluster based on the corresponding triple, check whether this tricluster is already presented in the tricluster set (by using hashing) and also check conditions.

The total time complexity of the algorithm depends on whether there is a non-zero minimal density threshold or not and on the complexity of the hashing algorithm used. In case we use some basic hashing algorithm processing the tricluster's extent, intent and modus and have a minimal density threshold equal to 0, the total time complexity of the main loop is $O(|I|(|G|+|M|+|B|))$, and of

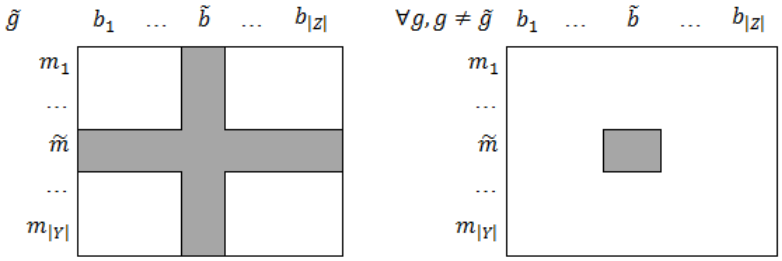

Fig. 1. Structure of prime OAC-triclusters

Algorithm 1 Algorithm for prime OAC-triclustering.

Input: $\mathbb{K} = (G, M, B, I)$ — tricontext;
ρ_{min} — minimal density threshold
Output: $\mathcal{T} = \{T = (X, Y, Z)\}$

$\mathcal{T} := \emptyset$
for all (g, m): $g \in G$,$m \in M$ **do**
  $PrimesOA[g, m] = (g, m)'$
end for
for all (g, b): $g \in G$,$b \in B$ **do**
  $PrimesOC[g, b] = (g, b)'$
end for
for all (m, b): $m \in M$,$b \in B$ **do**
  $PrimesAC[m, b] = (m, b)'$
end for
for all $(g, m, b) \in I$ **do**
  $T = (PrimesAC[m, b], PrimesOC[g, b], PrimesOA[g, m])$
  $Tkey = hash(T)$
  if $Tkey \notin \mathcal{T}.keys \wedge \rho(T) \geq \rho_{min}$ **then**
    $\mathcal{T}[Tkey] := T$
  end if
end for

the whole algorithm is $O(|G||M||B|+|I|(|G|+|M|+|B|))$. If we have a non-zero minimal density threshold, the time complexity of the main loop, as well as the time complexity of the algorithm, is $O(|I||G||M||B|)$.

The memory complexity is $O(|I|(|G|+|M|+|B|))$, as we need to keep the dictionaries with the prime sets in memory.

3 Online version of the OAC-triclustering algorithm

At first, let us describe the online problem of finding the set of prime OAC-triclusters. Let $\mathbb{K} = (G, M, B, I)$ be a triadic context. The user has no a priori knowledge of the elements and even cardinalities of G, M, B, and I. At each iteration we receive some set of triples from I: $J \subseteq I$. After that we must process J and get the current version of the set of all triclusters. It is important in this setting to consider every pair of triclusters different if they have different generating triples, event if their extents, intents, and modi are equal, because any other triple can change only one of them, thus making them different. The picture 2 shows the example of such situation (dark gray cells are the generating triples, light gray — prime sets).

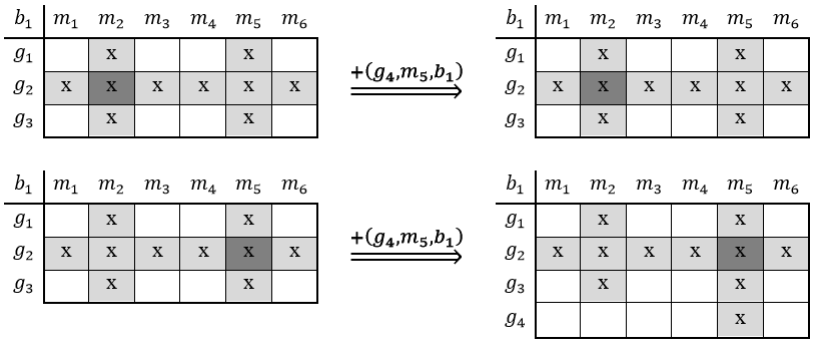

Fig. 2. Example of modification of triclusters by adding a triple

Also the algorithm requires that the dictionaries containing the prime sets are implemented as hash-tables. Because of this data structure the algorithm can efficiently access prime sets for their processing.

The algorithm itself is also quite simple (Alg. 2). It takes some set of triples (J) and current versions of the tricluster set ($\mathcal{T}$) and the dictionaries containing prime sets ($PrimesOA$, $PrimesOC$, $PrimesAC$) as input and outputs the modified versions of the tricluster set and dictionaries. The algorithm processes each triple (g, m, b) of J sequentially (line 1). On each iteration the algorithm modifies the corresponding prime sets:

– adds b to $(g, m)'$ (line 2)
– adds m to $(g, b)'$ (line 3)
– adds g to $(m, b)'$ (line 4)

Finally, it adds a new tricluster to the tricluster set. It is important to note that this tricluster contains pointers to the corresponding prime sets (in the corresponding dictionaries) instead of the copies of the prime sets (line 5).

In effect this algorithm is the same as the basic one but with some optimizations. First of all, instead of computing prime sets at the beginning, we modify them on spot, as adding an additional triple to the relation modifies only three prime sets by one element. Secondly, we remove the main loop by using pointers for the triclusters' extents, intents, and modi, as we can generate triclusters at the same step as we modify the prime sets. And the third important optimization is the use of only one pass through the triples of the ternary relation I, instead of enumeration of different pairwise combinations of objects, attributes, and conditions.

Algorithm 2 Add function for the online algorithm for prime OAC-triclustering.

Input: J — set of triples;
$\mathcal{T} = \{T = (*X, *Y, *Z)\}$ — current set of triclusters;
$PrimesOA$, $PrimesOC$, $PrimesAC$;
Output: $\mathcal{T} = \{T = (*X, *Y, *Z)\}$;
$PrimesOA$, $PrimesOC$, $PrimesAC$;
1: **for all** $(g, m, b) \in J$ **do**
2: $PrimesOA[g, m] := PrimesOA[g, m] \cup b$
3: $PrimesOC[g, b] := PrimesOC[g, b] \cup m$
4: $PrimesAC[m, b] := PrimesAC[m, b] \cup g$
5: $\mathcal{T} := \mathcal{T} \cup (\&PrimesAC[m, b], \&PrimesOC[g, b], \&PrimesOA[g, m])$
6: **end for**

Let us estimate the complexities of this algorithm. Each step requires the constant time: we need to modify three sets and add one tricluster to the set of triclusters. The total number of steps is equal to $|I|$. Thus the time complexity is linear $O(|I|)$. Beside that the algorithms is one-pass.

The memory complexity is the same: for each of $|I|$ steps the size of each dictionary containing prime sets is increased either by one element (if the required prime set is already present), or by one key-value pair (if not). Still, each of these dictionary requires $O(|I|)$ memory. Thus, the memory complexity is also linear $O(|I|)$.

Another important step used as an addition to this algorithm is post-processing. In addition to the user-specific post-processing there are some common useful steps. First of all, in the fixed moment of time we may want to remove additional triclusters with the same extent, intent, and modus from the output. Also some simple conditions like minimal support condition can be processed during

this step without increasing the original complexity. It should be done only during the post-processing step, as the addition of a triple in the main algorithm can drastically change the set of triclusters, and, respectively, the values used to check the conditions. Finally, if we need to check more difficult conditions like minimal density condition the time complexity of the post-processing will be higher than the time complexity of the original algorithm, but it can be also efficiently implemented.

To remove the same triclusters we need to use an efficient hashing procedure that can be improved by implementing it in the main algorithm. For this for all prime sets we need to keep their hash-values with them in the memory. And finally, when using hash-functions other than LSH function (Locality-Sensitive Hashing) [14] we can calculate hash-values of prime sets as some function of their elements (for example, exclusive disjunction or sum). Then when we modify prime sets we just need to get the result of this function and the new element. In this case, the hash-value of the tricluster can be calculated as the same function of the hash-values of its extent, intent, and modus.

Then it would be enough to implement the tricluster set as a hash-set in order to efficiently remove the additional entries of the same tricluster.

Pseudo-code for the basic post-processing (Alg. 3).

Algorithm 3 Post-processing for the online algorithm for prime OAC-triclustering.

Input: $\mathcal{T} = \{T = (*X, *Y, *Z)\}$ — full set of triclusters;
Output: $\overline{\mathcal{T}} = \{T = (*X, *Y, *Z)\}$ — processed hash-set of triclusters;

```
for all T ∈ 𝒯 do
    Calculate hash(T)
    if hash(T) ∉ 𝒯̄ then
        𝒯̄ := 𝒯̄ ∪ T
    end if
end for
```

If the names of the objects, attributes, and conditions are small enough (so that we can consider the time complexity of computing their hash values as $O(1)$), the time complexity of the post-processing is $O(|I|)$ if we do not need to calculate densities, and $O(|I||G||M||B|)$ otherwise. Also, the basic version of the post-processing does not require any additional memory, so its memory complexity is $O(1)$.

Finally, the algorithm can be easily paralleled by splitting the subset of triples J into several subsets, processing each of them independently, and merging the resulting sets afterwards.

4 Experiments

Two series of experiments were conducted in order to verify the time complexities and efficiency of the online algorithm: first one was conducted on the first set of synthetic contexts and on real world datasets, the second one — on the second set of synthetic contexts with large number of triples in each. In each experiment for the first set both versions of the OAC-triclustering algorithm were used to extract triclusters from a given context. Only the online version of the algorithm was applied to the second set of contexts as the computation time of the basic version of the algorithm was too high. To evaluate the time more precisely, for each context there were 5 runs of the algorithms with the average result recorded.

4.1 Datasets

Synthetic datasets. As it was mentioned, two sets of synthetic contexts were generated.

First five contexts have the same size, but different average densities. The sets of objects, attributes, and conditions of these contexts consist of 50 elements each (thus, the maximal number of triples for them is equal to 125,000). To form the relation I a pseudo-random number generator was used. It added each triple to the context with the given probability that was different for each context. These probabilities were: 0.02, 0.04, 0.06, 0.08, and 0.1.

The second set of uniform synthetic contexts consists of 10 contexts with the same probability for each triple to be included (0.001), but with different sizes of the sets of objects, attributes, and conditions. These sizes were 100, 200, 300, ..., 1000.

IMDB. This dataset consists of Top-250 list of the Internet Movie Database (250 best movies based on user reviews). For the analysis the following triadic context was extracted: the set of objects consists of movie names, the set of attributes — of tags, the set of conditions — of genres, and a triple of the ternary relation means that the given movie has the given genre and is assigned the given tag.

Bibsonomy. Finally, a sample of the data of bibsonomy.org was used. This website allows users to share bookmarks and lists of literature and tag them. For the research the following triadic context was extracted: the set of objects consists of users, the set of attributes (tags), the set of conditions (bookmarks), and a triple of the ternary relation means that the given user has assigned the given tag to the given bookmark.

The table 1 contains the summary of the contexts.

4.2 Results

The experiments were conducted on the computer running under Windows 8, using Intel Core i7-3517U 2.40 GHz processor, having 8 GB RAM. The algorithms

Table 1. Contexts for the experiments

Context	$\|G\|$	$\|M\|$	$\|B\|$	# triples	Density
Synthetic[1], 0.02	50	50	50	2530	0.02024
Synthetic[1], 0.04	50	50	50	5001	0.04001
Synthetic[1], 0.06	50	50	50	7454	0.05963
Synthetic[1], 0.08	50	50	50	10046	0.08037
Synthetic[1], 0.1	50	50	50	12462	0.09970
Synthetic[2], 100	100	100	100	996	0.001
Synthetic[2], 200	200	200	200	7995	0.001
Synthetic[2], 300	300	300	300	27161	0.001
Synthetic[2], 400	400	400	400	63921	0.001
Synthetic[2], 500	500	500	500	125104	0.001
Synthetic[2], 600	600	600	600	216021	0.001
Synthetic[2], 700	700	700	700	343157	0.001
Synthetic[2], 800	800	800	800	512097	0.001
Synthetic[2], 900	900	900	900	729395	0.001
Synthetic[2], 1000	1000	1000	1000	1000589	0.001
IMDB	250	795	22	3818	0.00087
BibSonomy	51	924	2844	3000	0.000022

were implemented in C# under .NET Framework 4.5. Jenkins' hash-function [11] was used to generate hash-values.

Figure 3 shows the time performance of both versions of the algorithms for different values of minimal density threshold. Figure 4 shows the computation time for the online version of the algorithm on the second set of synthetic contexts. "Basic" graph refers to the average time required by the basic algorithm, "Online, algorithm" — to average time required by the main algorithm part of the online algorithm (addition of new triples), "Online, total" — to the average time required by both the main algorithm and post-processing. Table 4.2 contains summary of the results for the case of zero minimal threshold.

As it can be clearly seen from all the graphs, online version of the algorithm significantly outperforms the basic version. However, post-processing in case of non-zero minimal density threshold can minimize the difference, especially in cases with small sets of objects, attributes, and conditions and large ternary relation.

In the case of several contexts of the fixed size, but increasing density, total computation time converges to the same value for the both algorithms, with the time for the online one being slightly smaller. For the non-zero minimal density threshold this convergence takes place for almost any average density value. In this case there is a rather large number of triclusters of big size, with many intersections, thus it takes much time to calculate all the triclusters' densities. This situation is close to the worst case, where time complexity is $O(|G||M||B|)$ for the main algorithm (because $|I|$ converges to $|G||M||B|$) and $O(|I||G||M||B|)$ for the post-processing. Also, in the case where the context's

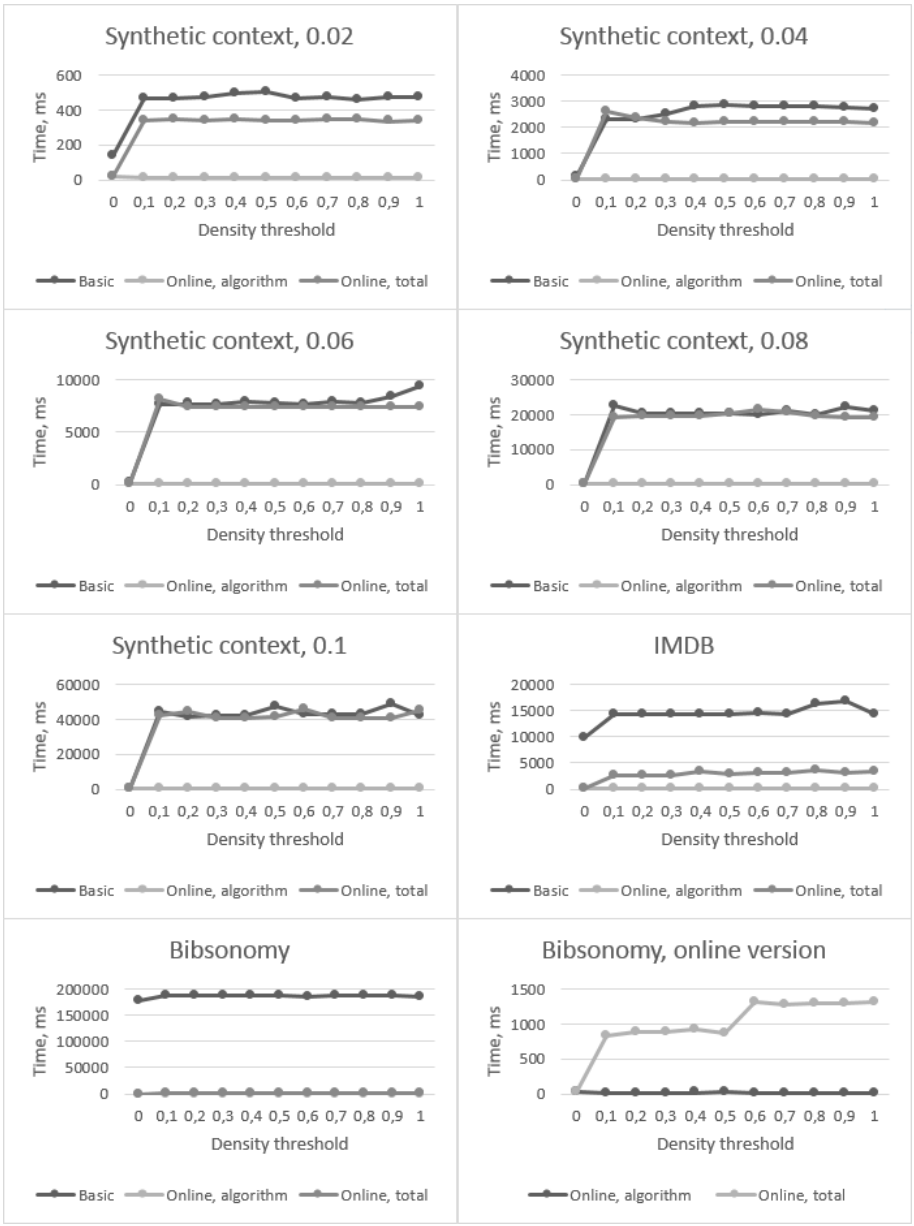

Fig. 3. Results of the experiments for both versions of OAC-triclustering algorithm

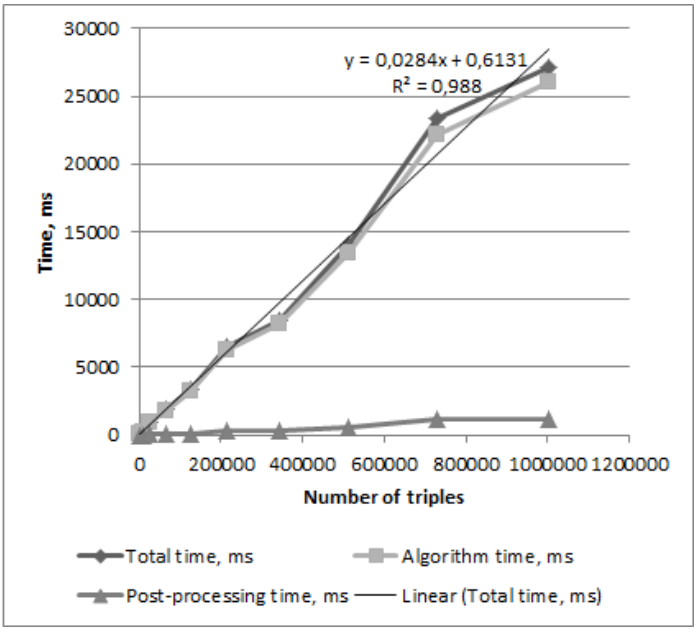

Fig. 4. Computation time for the online algorithm for various numbers of triples

density getting closer to 1, total time for both algorithms should be almost the same even in the case of zero minimal density threshold, as in the worst case for dense contexts $|I|$ is equal to $|G||M||B|$ (though it is an extremely rare case for real datasets).

The results for the second set of synthetic contexts confirm that the algorithm is indeed linear with respect to the number of triples. It also shows that the significant number of triples does not affect the performance as long as the context fits in the memory.

As for the other datasets with large sets of objects, attributes, and conditions and small ternary relation, the online algorithm significantly outperforms the basic one. The basic version spends much time on enumeration the large number of combinations of the elements of different sets of the context, while the online one just passes through the existing triples. Time to compute densities is quite small for these datasets since due to their sparseness they contain small number of rather small triclusters.

Finally, as it can be seen, for non-dense contexts the average density of triclusters is rather high even in the case of zero minimal density threshold. Because of that, it can be advised in most of the cases to use the online version of the algorithm without any hard conditions, like minimal density condition, as the results will still be good, but the performance will be significantly improved.

Table 2. Tricluster sets summary

Context	Number of triclusters	Average density
Synthetic[1], 0.02	2456	0.700
Synthetic[1], 0.04	4999	0.426
Synthetic[1], 0.06	7453	0.286
Synthetic[1], 0.08	10046	0.218
Synthetic[1], 0.1	12462	0.193
Synthetic[2], 100	897	0.993
Synthetic[2], 200	6972	0.972
Synthetic[2], 300	23645	0.941
Synthetic[2], 400	56584	0.909
Synthetic[2], 500	113041	0.871
Synthetic[2], 600	199210	0.834
Synthetic[2], 700	322447	0.796
Synthetic[2], 800	487982	0.759
Synthetic[2], 900	703374	0.722
Synthetic[2], 1000	973797	0.686
IMDB	1276	0.539
BibSonomy	1290	0.946

5 Conclusion

In this paper we have presented an online version of OAC-triclustering algorithm. We have shown that the algorithm is efficient from both theoretical and practical points of view. Its linear time complexity and performance in one pass (with an additional pass for the required post-processing) allows us to use it for big data problems. Moreover, the online algorithm as well as the basic one can be easily parallelized to attain even larger efficiency.

Acknowledgements. The study was implemented in the framework of the Basic Research Program at the National Research University Higher School of Economics in 2013-2014, in the Laboratory of Intelligent Systems and Structural Analysis (Russian Federation), and in the LIMOS (Laboratoire d'Informatique, de Modelisation et d'Optimisation des Systemes) (France). The first three authors were partially supported by Russian Foundation for Basic Research, grant no. 13-07-00504.

References

1. Cerf, L., Besson, J., Nguyen, K.N., Boulicaut, J.F.: Closed and noise-tolerant patterns in n-ary relations. Data Min. Knowl. Discov. 26(3), 574–619 (2013)
2. Davey, B.A., Priestley, H.A.: Introduction to Lattices and Order. Cambridge University Press, 2 edn. (2002)
3. Eren, K., Deveci, M., Kucuktunc, O., Catalyurek, Umit V.: A comparative analysis of biclustering algorithms for gene expression data. Briefings in Bioinform. (2012)

4. Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edn. (1999)
5. Gnatyshak, D.V., Ignatov, D.I., Kuznetsov, S.O.: From triadic FCA to triclustering: Experimental comparison of some triclustering algorithms. In: Ojeda-Aciego, M., Outrata, J. (eds.) CLA. CEUR Workshop Proceedings, vol. 1062, pp. 249–260. CEUR-WS.org (2013)
6. Gnatyshak, D.V., Ignatov, D.I., Semenov, A.V., Poelmans, J.: Gaining insight in social networks with biclustering and triclustering. In: BIR. Lecture Notes in Business Information Processing, vol. 128, pp. 162–171. Springer (2012)
7. Ignatov, D.I., Kuznetsov, S.O., Poelmans, J.: Concept-based biclustering for internet advertisement. In: ICDM Workshops. pp. 123–130. IEEE Computer Society (2012)
8. Ignatov, D.I., Kuznetsov, S.O., Poelmans, J., Zhukov, L.E.: Can triconcepts become triclusters? International Journal of General Systems 42(6), 572–593 (2013)
9. Ignatov, D.I., Nenova, E., Konstantinova, N., Konstantinov, A.V.: Boolean Matrix Factorisation for Collaborative Filtering: An FCA-Based Approach. In: Agre, G., et al. (eds.) AIMSA 2014, Varna, Bulgaria, September 11-13, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8722, pp. 47–58. Springer (2014)
10. Jelassi, M.N., Yahia, S.B., Nguifo, E.M.: A personalized recommender system based on users' information in folksonomies. In: Carr, L., et al. (eds.) WWW (Companion Volume). pp. 1215–1224. ACM (2013)
11. Jenkins, B.: A hash function for hash table lookup (2006), `http://www.burtleburtle.net/bob/hash/doobs.html`
12. Kaytoue, M., Kuznetsov, S.O., Macko, J., Napoli, A.: Biclustering meets triadic concept analysis. Ann. Math. Artif. Intell. 70(1-2), 55–79 (2014)
13. Kaytoue, M., Kuznetsov, S.O., Napoli, A., Duplessis, S.: Mining gene expression data with pattern structures in formal concept analysis. Inf. Sci. 181(10), 1989–2001 (2011), `http://dx.doi.org/10.1016/j.ins.2010.07.007`
14. Leskovec, J., Rajaraman, A., Ullman, J.: Mining of Massive Datasets, chap. Finding Similar Items, pp. 71–128. Cambridge University Press, England, Cambridge (2010)
15. Li, A., Tuck, D.: An effective tri-clustering algorithm combining expression data with gene regulation information. Gene regulation and systems biology 3, 49–64 (2009), `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2758278/`
16. Madeira, S.C., Oliveira, A.L.: Biclustering algorithms for biological data analysis: A survey. IEEE/ACM Trans. Comput. Biology Bioinform. 1(1), 24–45 (2004)
17. Mirkin, B.: Mathematical Classification and Clustering. Kluwer, Dordrecht (1996)
18. Mirkin, B.G., Kramarenko, A.V.: Approximate bicluster and tricluster boxes in the analysis of binary data. In: Kuznetsov, S.O., et al. (eds.) RSFDGrC 2011. Lecture Notes in Computer Science, vol. 6743, pp. 248–256. Springer (2011)
19. Nanopoulos, A., Rafailidis, D., Symeonidis, P., Manolopoulos, Y.: Musicbox: Personalized music recommendation based on cubic analysis of social tags. IEEE Transactions on Audio, Speech & Language Processing 18(2), 407–412 (2010)
20. Wille, R.: Restructuring lattice theory: An approach based on hierarchies of concepts. In: Rival, I. (ed.) Ordered Sets, NATO Advanced Study Institutes Series, vol. 83, pp. 445–470. Springer Netherlands (1982)
21. Zhao, L., Zaki, M.J.: Tricluster: An effective algorithm for mining coherent clusters in 3d microarray data. In: Özcan, F. (ed.) SIGMOD Conference. pp. 694–705. ACM (2005)