Putting OAC-triclustering on MapReduce

Sergey Zudin, Dmitry V. Gnatyshak, and Dmitry I. Ignatov

National Research University Higher School of Economics, Russian Federation
dignatov@hse.ru
http://www.hse.ru

Abstract. In our previous work an efficient one-pass online algorithm for triclustering of binary data (triadic formal contexts) was proposed. This algorithm is a modified version of the basic algorithm for OAC-triclustering approach; it has linear time and memory complexities. In this paper we parallelise it via map-reduce framework in order to make it suitable for big datasets. The results of computer experiments show the efficiency of the proposed algorithm; for example, it outperforms the online counterpart on Bibsonomy dataset with $\approx 800,000$ triples.

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

1 Introduction

Mining of multimodal patterns is one of the hot topics in Data Mining and Machine Learning [1,2,3,4]. Thus, cluster analysis of multimodal data and specifically of dyadic and triadic relations is a natural extension of the idea of original clustering. In dyadic case biclustering methods (the term bicluster was coined in [5]) are used to simultaneously find subsets of the sets of objects and attributes that form homogeneous patterns of the input object-attribute data. In fact, one of the most popular applications of biclustering is gene expression analysis in Bionformatics [6,7]. Triclustering methods operate in triadic case where for each object-attribute pair one assigns a set of some conditions [8,9,10]. Both biclustering and triclustering algorithms are widely used in such areas as gene expression analysis [11,12,13], recommender systems [14,15,16], social networks analysis [17], etc. The processing of numeric multimodal data is also possible by modifications of existing approaches for mining dyadic binary relations [18].

Though there are methods that can enumerate all triclusters satisfying certain constraints [2] (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 (e.g., $O(|I|)$ in case of n-ary relation I) and be easily parallelisable. In addition, in most cases, it is necessary that such algorithms output the results in one pass.

Earlier, in order to create an algorithm satisfying these requirements, we adapted a triclustering method based on prime operators (prime OAC-triclustering

ISBN 978–2–9544948–0–7, ISSN 2311–701X, Blaise Pascal University, LIMOS laboratory, Clermont-Ferrand, 2015.

method) [10] and proposed its online version, which is linear, one-pass and easily parallelisable [19]. However, its parallelisation is possible in different ways. For example, one can use a popular framework for commodity hardware, MapReduce (M/R) [20]. By the way, there were several successful M/R implementations in the FCA community and other lattice-oriented domains. Thus, in [21], the authors adapted Close-by-One algorithm to M/R framework and showed its efficiency. At the same year, in [22], an efficient M/R algorithm for computation of closed cube lattices was proposed. The authors of [23] demonstrated that iterative algorithms like Ganter's NextClosure can benefit from the usage of iterative M/R schemes.

Note that experts aware potential users that M/R is like a big cannon that requires long preparations to shot but fires fast: "the entire distributed-file-system milieu makes sense only when files are very large and are rarely updated in place " [20]. In this work, in contrast to our previous study, we assume that there is a large bulk of data to process that are not coming online.

The rest of the paper is organized as follows: in Section 2, we recall the original method and the online version of the algorithm of prime OAC-triclustering. In Section 3, we describe the M/R setting of the problem and the corresponding M/R version of the original algorithm with important implementation aspects. Finally, in Section 4 we show the results of several experiments which demonstrate the efficiency of the M/R version of the algorithm. As an addendum, in the Appendix section, the reader may find our proposal for alternative models of M/R-based variants of prime OAC-triclustering.

2 Prime object-attribute-condition triclustering

Prime object-attribute-condition triclustering method (OAC-prime) based on Formal Concept Analysis [24,25] is an extension for the triadic case of object-attribute biclustering method [26]. Triclusters generated by this method have similar structure as the corresponding biclusters, namely the cross-like structure of triples inside the input 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:

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

where $X \subseteq G$, $Y \subseteq M$, and $Z \subseteq B$.

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, *tricluster extent*, *tricluster intent*, and *tricluster modus*. The triple (g,m,b) is called a *generating triple* of the tricluster T. Figure 1 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.

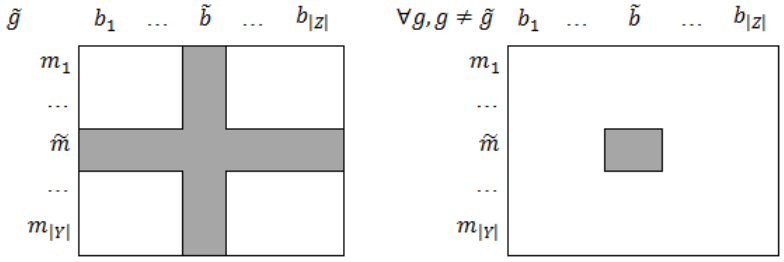

Fig. 1. Structure of prime OAC-triclusters: the dense cross-like central layer containing $\tilde{g}$ (left) and the layer for an object g (right) in $M \times B$ dimensions.

The basic algorithm for prime OAC-triclustering method is rather straightforward (see [10]). First of all, for each combination of elements from each two sets of $\mathbb{K}$ we apply the corresponding prime operator (we 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 contained in the tricluster set (by using hashing) and also check extra 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 without a minimal density threshold, the total time complexity is $O(|G||M||B| + |I|(|G| + |M| + |B|))$; in case of a non-zero minimal density threshold, it 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.

In online setting, for triples coming from triadic context $\mathbb{K} = (G, M, B, I)$, 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 as being different as they have different generating triples, even if their respective extents, intents, and modi are equal. Thus, any other triple can change only one of these two triclusters, making them different.

To efficiently access prime sets for their processing, the dictionaries containing the prime sets are implemented as hash-tables.

The algorithm is straightforward as well (Alg. 1). It takes some set of triples (J), the current tricluster set ($\mathcal{T}$), and the dictionaries containing prime sets ($Primes$) 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). At each iteration the algorithm modifies the corresponding prime sets (lines 2-4).

Finally, it adds a new tricluster to the tricluster set. 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) which allows to lower the memory and access costs.

Algorithm 1 Add function for the online algorithm for prime OAC-triclustering.

Input: J is a set of triples;
$\mathcal{T} = \{T = (*X, *Y, *Z)\}$ is a 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**

The algorithm is one-pass and its time and memory complexities are $O(|I|)$.

Duplicate elimination and selection patterns by user-specific constraints are done as post-processing to avoid patterns' loss. The time complexity of the basic post-processing is $O(|I|)$ and it does not require any additional memory.

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

3 Map-reduce OAC-triclustering

3.1 Map-reduce decomposition

We use a two-stage M/R approach. The first M/R allows us to efficiently calculate all the primes of the existed pairs. The second M/R permits to assemble the found primes into triclusters. During the first map phase, each triple from the input context is indexed by a key using hash function depending on one of the basic entity types, object, attribute, or condition (see Alg. 2). The number of map keys is equal to the number of reducers.

Then each first reducer receives the portion of data for a particular key (see Alg. 3). The internal reducer algorithm is almost a replication of Online OAC-prime. However, it does not assemble all found triclusters into a final collection; the reducer simply writes the file with the current triclusters for a given portion of data to a file or pass it to the second-stage mapper. Since in Hadoop MapReduce

Algorithm 2 Distributed OAC-triclustering: First Map

Input: S is a set of input triples as strings;
r is a number of reducers;
i is a grouping index (objects, attributes or conditions).
Output: $\tilde{J}$ is a list of $\langle key, triple \rangle$ pairs.

```
1: for all s ∈ S do
2:     t := transform(s)
3:     key := hash(t[i]) mod r
4:     J̃ := J̃ ∪ {⟨key, t⟩}
5: end for
```

we should work with text input files and our data are mainly in a tuple-based form, we use encode/decode function $encode()/transform()$ to switch between the internal tuple representation and the text-based one.

Algorithm 3 Distributed OAC-triclustering: First Reduce

Input: J is a list of triples (for a certain key);
$\mathcal{T} = \{T = (X, Y, Z)\}$ is a current set of triclusters;
$PrimesOA$, $PrimesOC$, $PrimesAC$.
Output: file of strings – encoded $\langle triple, tricluster \rangle$ pairs.

```
1: Primes ← initialise a new multimap
2: for all (g, m, b) ∈ J do
3:     Primes[g, m] := Primes[g, m] ∪ {b}
4:     Primes[g, b] := Primes[g, b] ∪ {m}
5:     Primes[m, b] := Primes[m, b] ∪ {g}
6: end for
7: for all (g, m, b) ∈ J do
8:     T := (set(Primes[m, b]), set(Primes[g, b]), set(Primes[g, m]))
9:     s := {encode(⟨(g, m, b), T⟩)}
10:    store s
11: end for
```

The second mapper takes the found intermediate triclusters (with their keys) as strings from the files produced by the first-stage reducers (see Alg. 4). It fills $Primes$ multimap in one pass through all $\langle triple, tricluster \rangle$ pairs. In the next loop for each key (g, m, b) the corresponding tricluster is formed and $\langle tricluster, tricluster \rangle$ pairs are passed to the second-stage reducer (the key $tricluster$ can be efficiently implemented by a proper hashing). In its turn, the second stage reducer eliminates duplicates and outputs the resulting file (Alg. 4). The $set()$ function helps to avoid duplicates among the values of $Primes[,]$, which is closer to our implementation. However, one can easily omit $set()$ in line 8, provided that $Primes$ is properly implemented.

The time complexity of the M/R solution is composed from two terms for each stage: $O(|I|/r)$ and $O(|I|)$. However, there are communication costs that

Algorithm 4 Distributed OAC-triclustering: Second Map

Input: S is a list of strings.
Output: $\tilde{\mathcal{T}}$ is an list of $\langle tricluster, tricluster\rangle$ pairs.

```
Primes ← initialise a new multimap
for all s ∈ S do
    ⟨(g, m, b), T⟩ := decode(s)
    update Primes multimap appropriately
    I := I ∪ {(g, m, b)}
end for
for all (g, m, b) ∈ I do
    T := (set(Primes[m, b]), set(Primes[g, b]), set(Primes[g, m]))
    T̃ := T̃ ∪ {⟨T, T⟩}
end for
```

Algorithm 5 Distributed OAC-triclustering: Second Reduce

Input: $\hat{\mathcal{T}}$ is a list of $\langle tricluster, list\ of\ triclusters\rangle$ pairs.
Output: File with a final set of triclusters $\{T = (X, Y, Z)\}$.

```
for all ⟨T, [T, ..., T]⟩ ∈ T̂ do
    store T
end for
```

should be inevitably paid and can be theoretically estimated as follows [20]: the replication rate for the first M/R stage $r_1 = 1$ (each triple is passed as one key-value pair), the reducer size $q_1 = |I|/r$; the replication rate for the second M/R stage is $r_2 = 1$ (it assign one key-value pair for each tricluster), but the reducer size varies from $q_2^{min} = 1$ (no duplicate triclusters) and $q_2^{max} = |I|$ (one final tricluster when all the initial triples belong to one absolutely dense cuboid).

3.2 Implementation aspects and used technologies

The application [1] has been implemented in Java within JRE 8 and as distributed computation framework we use Apache Hadoop [2].

We have used many other technologies: Apache Maven (framework for automatic project assembling), Apache Commons (for work with extended Java collections), Google Guava (utilities and data structures), Jackson JSON (open-source library for transformation of object-oriented representation of an object like tricluster to string), TypeTools (for real-time type resolution of inbound and outbound key-value pairs), etc.

ChainingJob module. During the development we found that in Hadoop one MapReduce process can contain only one Mapper and one Reducer. Thus, in order to develop an application with three "map" phases and one "reduce", one needs to create three processes. One process creation (even without various adjustments) takes 8-10 lines of code. After our vain search of an appropriate

[1] `https://github.com/zydins/DistributedTriclustering`
[2] `http://hadoop.apache.org/`

library, we developed "chaining-job" module [3]. Its main class contains the following fields: "jobs" (list of all scheduled processes), "name" (common name for all processes), and "tempDir" (folder name for intermediate results). First, the algorithm set input path for the first chaining process and path to the result of the last job; the rest jobs are connected by input and output "key-value" pairs and directory for intermediate files storage. Then this algorithm runs processes according to the schedule and waits their completion. In other words, it connects the input and output of chaining processes that run sequentially.

Let us shortly describe the most important classes our M/R implementation.

Entity. It is a basic class for object oriented representation of input strings and maintains three entity types: EXTENT, INTENT, and MODUS. For example: {"Leon", EXTENT }.

Tuple. An object of this class stores references to objects of class Entity and represents two basic entities: triple and tricluster. Mapper and Reducer classes operate with objects of this type.

FormalContext. This class is an object oriented representation of the underlying binary relation; it keeps the reference to an object of EntityStorage class (see below). It also contains methods "add" (add triple) and "getTriclusters" (get the output set of unique triclusters).

EntityStorage. This class manages the work with extents, intents and modi of triclustes. It also contains three dictionaries with composite keys. For example, for $(g1, m1, c1)$ object $c1$ will be added by key $(g1, m1)$ to the first dictionary; analogously for keys $(g1, c1)$ and $(m1, c1)$.

The process-like M/R classes are summarised below.

TupleReadMapper. Its main goal is reading a triple from the input file and transform the triple to an object of class Tuple.

TupleContextReducer. It receives input tuples and fills the underlying tricontext by them. It also sets the number of first reducers. This number depends on the available nodes in a distributed system and the structure of input data. The more unique entities are in triples, the more that value should be.

PrepareMapper. The "map" method receives files from the previous stage. They contain intermediate triclusters from each object of class TupleContextReducer. It fills the dictionary with primes. Further, each tricluster triple is transformed to Tuple structure and is passed to the second reduce phase.

CollectReduce. This class gathers all intermediate triclusters and obtains the final tricluster set. This process runs in several threads for speed up. The number of threads is a user-specified parameter.

Executor. It is a starting class of the application, which receives the input parameters, activates "chaining-job" utility for making a chain of jobs, and starts the execution.

[3] `https://github.com/zydins/chaining-job`

4 Experiments

Two series of experiments have been conducted in order to test the application on the synthetic contexts and real world datasets with moderate and large number of triples in each. In each experiment both versions of the OAC-triclustering algorithm have been used to extract triclusters from a given context. Only online and M/R versions of OAC-triclustering algorithm have managed to result patterns for large contexts since the computation time of the compared algorithms was too high (>3000 s). To evaluate the runtime more carefully, for each context the average result of 5 runs of the algorithms has been recorded.

4.1 Datasets

Synthetic datasets. As it was mentioned, synthetic contexts were randomly generated: 1) 20,000 triples (25 unique entities of each type); 2) 100,000 triples (50 unique entities of each type); 3) 1,000,000 triples (all possible combinations of 100 unique entities of each type). However, it is easy to see that some datasets are not correct formal contexts from algebraic viewpoint. Thus, the first dataset inevitably contains duplicates since $25 \times 25 \times 25$ gives only 15,625 unique triples. The second one contains less triples than $50^3 = 125,000$, the number of all possible combinations. The third one is just an absolutely dense cuboid $100 \times 100 \times 100$ (it contains only one formal concept (OAC-tricluster), the whole context).

These tests look more like crush test, but they have sense since in M/R setting the triples can be (partially) repeated, e.g., because of M/R task failures on some nodes (i.e. restarting processing of some key-value pairs). Even though the third dataset does not result in $3^{min(|G|,|M|,|B|)}$ formal triconcepts, the worst case for formal triconcepts generation in terms of the number of patterns, this is an example of the worst case scenario for the second reducer since its size is maximal ($q_2^{max} = |I|$). By the way, our algorithm should correctly assemble the only one tricluster (G, M, B) and it actually does.

IMDB. This dataset consists of Top-250 list of the Internet Movie Database (250 best movies based on user reviews). The following triadic context is composed: the set of objects consists of movie names, the set of attributes (tags), the set of conditions (genres), and each 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 from ECML PKDD discovery challenge 2008 has been used. This website allows users to share bookmarks and lists of literature and tag them. For the tests the following triadic context has been prepared: 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.

Table 1. Contexts for the experiments

Context	$\|G\|$	$\|M\|$	$\|B\|$	# triples	Density
20k	25	25	25	20,000	1
100k	50	50	50	100,000	0.8
1m	100	100	100	1,000,000	1
IMDB	250	795	22	3,818	0.00087
BibSonomy	2,337	67,464	28,920	816,197	$1.8 \cdot 10^{-7}$

4.2 Results

The experiments has been conducted on the computer running under OS X 10, using 1,8 GHz Intel Core i5, having 4 Gb 1600 MHz DDR3 and having 8 Gb free space on its hard drive (a typical commodity hardware). Two M/R modes have been tested: sequential mode of tasks completion and emulation of distributed one with 16 first reducers and 32 threads for the second stage.

Table 2. Results of comparison (time is given in seconds)

Algorithm/Context	IMDB (≈3k triples)	20k triples	100k triples	1m triples	Bibsonomy (≈800k triples)
Tribox	324	800	1,265	>3,000	>3,000
TRIAS	189	362	862	>3,000	>3,000
OAC Box	374	756	1,265	>3,000	>3,000
OAC Prime	7	8	734	>3,000	>3,000
Online OAC prime	3	3	3	5	>3,000
M/R OAC prime seq.	12	30	81	166	1,534
M/R OAC prime distr.	1	15	20	25	520

In Table 2 we summarise the results of performed tests. It is clear that on average our application has fewer execution time than its competitors, except of online version of OAC-triclustering. If we compare the implemented program with its original online version, the results are worse for not that big but dense datasets (closer to the worst case scenario $q_2 = |I|$. It is the consequence of the fact that the application architecture aimed at processing of large amounts of data; in particular, it is implemented in two stages with time consuming communication. Launching and stopping Apache Hadoop, data writing and passing between Map and Reduce steps in both stages requires substantial time, that is why for not that big datasets, when execution time is comparable with time for infrastructure management, time performance is not perfect. However, with data size increase the relative performance is growing. Thus, the last test for BibSonomy data has been successfully passed, but the competitors were not able to finish it within 50 min, but our M/R program did it even in sequential mode within 25 min.

5 Conclusion

In this paper we have presented a map-reduce version of OAC-triclustering algorithm. We have shown that the algorithm is efficient from both theoretical and practical points of view. It remains of linear time complexity and is performed in two stages (with each stage being M/R distributed); this allows us to use it for big data problems. However, we believe that it is possible to propose another variants of map-reduce based algorithm where the reducer exploits composite keys directly (see Appendix section). So, such algorithms and their comparison with the current M/R version on real and artificial data is still be in our plans. However, in despite the step towards Big Data technologies, a proper comparison of the proposed OAC triclustering and noise tolerant patterns in n-ary relations by DataPeeler and its descendants [2] is not yet conducted.

Acknowledgments. The study was implemented in the framework of the Basic Research Program at the National Research University Higher School of Economics in 2014-2015, in the Laboratory of Intelligent Systems and Structural Analysis. The last two authors were partially supported by Russian Foundation for Basic Research, grant no. 13-07-00504. The authors would like to thank Yuri Kudriavtsev from PM-Square and Dominik Slezak from Infobright and Warsaw University for their encouragement given to our studies of M/R technologies.

References

1. Georgii, E., Tsuda, K., Schölkopf, B.: Multi-way set enumeration in weight tensors. Machine Learning **82**(2) (2011) 123–155
2. 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) (2013) 574–619
3. Spyropoulou, E., De Bie, T., Boley, M.: Interesting pattern mining in multi-relational data. Data Mining and Knowledge Discovery **28**(3) (2014) 808–849
4. Ignatov, D.I., Gnatyshak, D.V., Kuznetsov, S.O., Mirkin, B.: Triadic formal concept analysis and triclustering: searching for optimal patterns. Machine Learning (2015) 1–32
5. Mirkin, B.: Mathematical Classification and Clustering. Kluwer, Dordrecht (1996)
6. Madeira, S.C., Oliveira, A.L.: Biclustering algorithms for biological data analysis: A survey. IEEE/ACM Trans. Comput. Biology Bioinform. **1**(1) (2004) 24–45
7. Eren, K., Deveci, M., Kucuktunc, O., Catalyurek, Umit V.: A comparative analysis of biclustering algorithms for gene expression data. Briefings in Bioinform. (2012)
8. 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. Volume 6743 of Lecture Notes in Computer Science., Springer (2011) 248–256
9. Ignatov, D.I., Kuznetsov, S.O., Poelmans, J., Zhukov, L.E.: Can triconcepts become triclusters? International Journal of General Systems **42**(6) (2013) 572–593
10. Gnatyshak, D.V., Ignatov, D.I., Kuznetsov, S.O.: From triadic FCA to triclustering: Experimental comparison of some triclustering algorithms. In: CLA. (2013) 249–260

11. Zhao, L., Zaki, M.J.: Tricluster: An effective algorithm for mining coherent clusters in 3d microarray data. In: SIGMOD 2005 Conference. (2005) 694–705
12. Li, A., Tuck, D.: An effective tri-clustering algorithm combining expression data with gene regulation information. Gene regul. and syst. biol. **3** (2009) 49–64
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) (2011) 1989–2001
14. 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) (2010) 407–412
15. 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), ACM (2013) 1215–1224
16. Ignatov, D.I., Nenova, E., Konstantinova, N., Konstantinov, A.V.: Boolean Matrix Factorisation for Collaborative Filtering: An FCA-Based Approach. In: AIMSA 2014, Varna, Bulgaria, Proceedings. Volume LNCS 8722. (2014) 47–58
17. Gnatyshak, D.V., Ignatov, D.I., Semenov, A.V., Poelmans, J.: Gaining insight in social networks with biclustering and triclustering. In: BIR. Volume 128 of Lecture Notes in Business Information Processing., Springer (2012) 162–171
18. Kaytoue, M., Kuznetsov, S.O., Macko, J., Napoli, A.: Biclustering meets triadic concept analysis. Ann. Math. Artif. Intell. **70**(1-2) (2014) 55–79
19. Gnatyshak, D.V., Ignatov, D.I., Kuznetsov, S.O., Nourine, L.: A one-pass triclustering approach: Is there any room for big data? In: CLA 2014. (2014)
20. Rajaraman, A., Leskovec, J., Ullman, J.D.: MapReduce and the New Software Stack. In: Mining of Massive Datasets. Cambridge University Press, England, Cambridge (2013) 19–70
21. Krajca, P., Vychodil, V.: Distributed algorithm for computing formal concepts using map-reduce framework. In: N. Adams et al. (Eds.): IDA 2009. Volume LNCS 5772. (2009) 333–344
22. Kuznecov, S., Kudryavcev, Y.: Applying map-reduce paradigm for parallel closed cube computation. In: 1st Int. Conf. on Advances in Databases, Knowledge, and Data Applications, DBKDS 2009. (2009) 62–67
23. Xu, B., de Frein, R., Robson, E., Foghlu, M.O.: Distributed formal concept analysis algorithms based on an iterative mapreduce framework. In Domenach, F., Ignatov, D., Poelmans, J., eds.: ICFCA 2012. Volume LNAI 7278. (2012) 292–308
24. Wille, R.: Restructuring lattice theory: An approach based on hierarchies of concepts. In Rival, I., ed.: Ordered Sets. Volume 83 of NATO Advanced Study Institutes Series. Springer Netherlands (1982) 445–470
25. Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. 1st edn. Springer-Verlag New York, Inc., Secaucus, NJ, USA (1999)
26. Ignatov, D.I., Kuznetsov, S.O., Poelmans, J.: Concept-based biclustering for internet advertisement. In: ICDM Workshops, IEEE Computer Society (2012) 123–130

Appendix. Alternative variants of two-stage MapReduce

First Map: Finding primes. During this phase every input triple (g, m, b) is encoded by three key-value pairs $\langle (g, m), b\rangle$, $\langle (g, b), m\rangle$, and $\langle (m, b), g\rangle$. These pairs are passed to the first reducer. The replication rate is $r_1 = 3$.

First Reduce: Finding primes. This reducer fills three corresponding dictionaries for primes of keys. So, for example, the first dictionary, $PrimeOA$ contains key-value pairs $\langle (g,m), \{b_1, b_2, \ldots, b_n\}\rangle$. The reducer size is $q_1 = max(|G|, |M|, |B|)$

The process can be stopped after the first reduce phase and all the triclusters found as $(Prime[g,m], Prime[g,b], Prime[m,b])$ each by enumeration of $(g,m,b) \in I$. However, to do it faster and keep the result for further computation, it is possible to use M/R as well.

Second Map: Tricluster generation. The second map does tricluster combining job, i.e. for each triple (g,m,b) it composes the new key-value pair, $\langle (g,m,b), \emptyset\rangle$. And for each pair of either type, $\langle (g,m), Prime[g,m]\rangle$, $\langle (g,b), Prime[g,b]\rangle$, and $\langle (m,b), Prime[m,b]\rangle$ it generates key-values pairs $\langle (g,m,\tilde{b}), Prime[g,m]\rangle$, $\langle (g,\tilde{m},b), PrimeOC[g,b]\rangle$, and $\langle (\tilde{g},m,b), Prime[m,b]\rangle$, where $\tilde{g} \in G$, $\tilde{m} \in M$, and $\tilde{b} \in B$. $r_2 = (|I|+3|G||M||B|)/(|I|+|G||M|+|G||B|+|M||B|) \leq (\rho+3)/(\rho+3/max(|G|,|M|,|B|))$, where ρ is the input tricontext density.

Second Reduce: Tricluster generation. The second reducer just assembles only one value for each key (g,m,b), the generating triple, its tricluster, $(Prime[g,m], Prime[g,b], Prime[m,b])$. If there is no key-value pair $\langle (g,m,b), \emptyset\rangle$ for a particular triple (g,m,b), it does not output any key-value pair for the key. The reducer size q_2 is either 3 (no output) or 4 (tricluster assembled).

Second Map: Tricluster generation with duplicate generating triples. Second map does tricluster combining job, i.e. for each triple (g,m,b) it composes a new key-value pair: $\langle (Prime[g,m], Prime[g,b], Prime[m,b]), (g,m,b)\rangle$.

Second Map: Tricluster generation with duplicate generating triples. The second reducer just groups values for each key: $\langle (X,Y,Z), \{(g_1,m_1,b_1), \ldots, (g_n,m_n,b_n)\}\rangle$.

These two variations of the second stage have their merits: the first one is beneficial for further computations with a new portion of triples and the last one is more compact and informative. Of course, each variant of the second stage has its own runtime complexity which depends not only on the model representation but is also sensitive to datastructures implementation and M/R communication costs and settings.