Multimodal Clustering of Boolean Tensors on MapReduce: Experiments Revisited

Dmitry I. Ignatov, Dmitry Tochilkin, and Dmitry Egurnov

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

Abstract. This paper presents further development of distributed multimodal clustering. We introduce a new version of multimodal clustering algorithm for distributed processing in Apache Hadoop on computer clusters. Its implementation allows a user to conduct clustering on data with modality greater than two. We provide time and space complexity of the algorithm and justify its relevance. The algorithm is adapted for MapReduce distributed processing model. The program implemented by means of Apache Hadoop framework is able to perform parallel computing on thousands of nodes.

Keywords: Formal Concept Analysis, n-ary relations, Boolean tensors, data mining, big data, MapReduce

1 Introduction

Mining of multimodal patterns in n-ary relations or Boolean tensors is among popular topics in Data Mining and Machine Learning [4,1,28,10,21,27,9]. 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 [22]) 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 Bioinformatics [20,2]. Triclustering methods operate in triadic case where for each object-attribute pair one assigns a set of some conditions [23,12,5]. Both biclustering and triclustering algorithms are widely used in such areas as gene expression analysis [33,19,16], recommender systems [24,14,13], social networks analysis [7], natural language processing [29], etc. The processing of numeric multimodal data is also possible by modifications of existing approaches for mining binary relations [15].

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 is usually exponential (e.g. in case of formal triconcepts), showing that

these methods are hardly scalable. To process big data algorithms require at most linear time complexity (e.g., $O(|I|)$ in case of n-ary relation I) and be easily parallelisable. In addition, especially in case of data streams [26], the output patterns should be the results of one pass over data.

Earlier, in order to create an algorithm satisfying these requirements, we adapted a triclustering method based on prime operators (prime OAC-triclustering method) [5] and proposed its online version, which has linear time complexity; it is also one-pass and easily parallelisable [6]. However, its parallelisation is possible in different ways. For example, one can use a popular framework for commodity hardware, Map-Reduce (M/R) [25].In the past, there were several successful M/R implementations in the FCA community and other lattice-oriented domains. Thus, in [17], the authors adapted Close-by-One algorithm to M/R framework and showed its efficiency. At the same year, in [18], an efficient M/R algorithm for computation of closed cube lattices was proposed. The authors of [32] demonstrated that iterative algorithms like Ganter's NextClosure can benefit from the usage of iterative M/R schemes.

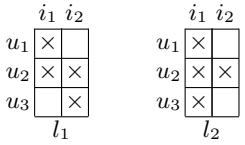
Our previous M/R implementation of triclustering method based on prime operators was proposed in [34] showing computational benefits on rather large datasets. M/R triclustering algorithm [34] is a successful distributed adaptation of the online version of prime OAC-triclustering [6]. This method uses MapReduce approach as means for task allocation on computational clusters, launching the online version of prime OAC-triclustering on each reducer of the first phase. However, due to simplicity of this adaptation, the algorithm does not use the advantages of MapReduce to the full extent. On the first stage all the input triples are split into the number of groups equals to the number of reducers by means of hash-function for entities of one of the types, object, attribute, or condition, which values are used as keys. It is clear that this way of data allocation cannot guarantee uniformness in terms of group sizes. The JobTracker used in Apache Hadoop is able to evenly allocate tasks by nodes [1]. To do so, the number of tasks should be larger than than the number of working nodes, which is not fulfilled in this implementation. For example, let us assume we have 10 reduce SlaveNodes; respectively, $r = 10$ and the hash-function is applied to the objects (the first element in each input triple). However, due to non-uniformity of hash-function values by modulo 10, it may happen that the set of objects will result in less than 10 different residuals during division by 10. In this case, the input triples will be distributed between parts of different sizes and processed by only a part of cluster nodes. Such cases are rather rare; it could be possible only for slicing by entities (objects, attributes, or conditions) with a small number of different elements. However, they may slow down the cluster work drastically.

The weakest link is the second stage of the algorithm. First of all, during the first stage it finds triclusters computed for each data slice separately. Hence, they are not the final triclusters; we need to merge the obtained results.

Let us consider example in Table 1 with the ternary relation on users-items-labels. Let us assume that the first mapper splits data according to their la-

[1] https://wiki.apache.org/hadoop/JobTracker

Table 1. An example with triadic data



bels' component, $r = 2$, then triples containing label l_1 and those related to label l_2 are processed on different nodes. After the first stage completion on the first node, we have tricluster $(\{u_2\}, \{i_1, i_2\}, \{l_1\})$ among the others, while the second node results in tricluster $(\{u_2\}, \{i_1, i_2\}, \{l_2\})$. It is clear that both triclusters are not complete for the whole input dataset and should be merged into $(\{u_2\}, \{i_1, i_2\}, \{l_1, l_2\})$. The second stage of the algorithm is responsible for this type of merging. However, as one can see, this merging assumes that all intermediate data should be located on the same node. In big data setting, this allocation of all the intermediate results on a single node is a critical point for application performance.

To calculate tricluster components (or cumuli, see Section 3) and assemble the final triclusters from them, we need to have large data slices on the computational nodes. Moreover, during parallelisation of the algorithm those data slices can be required simultaneously on different nodes. Thus, to solve these problems one needs to fulfil data centralisation (all the required slices should be present at the same node simultaneously). However, it leads to accumulation of too large parts of the data as described. Another approach is to perform data replication. In this case the total amount of data processed on computational nodes is increased, while the data are evenly distributed in the system. The latter approach has been chosen for our updated study on multimodal clustering.

Note that experts aware potential M/R users: "the entire distributed-file-system milieu makes sense only when files are very large and are rarely updated in place" [25]. In this work, as in 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 organised as follows: in Section 2, we recall the original method and the online version of the algorithm of prime OAC-triclustering. Section 3 generalise prime OAC-triclustering for the case of multimodal data. In Section 4, 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 5 we show the results of several experiments which demonstrate the efficiency of the M/R version of the algorithm.

2 Prime object-attribute-condition triclustering

Prime object-attribute-condition triclustering method (OAC-prime) based on Formal Concept Analysis [31,3] is an extension for the triadic case of object-attribute biclustering method [11]. 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.

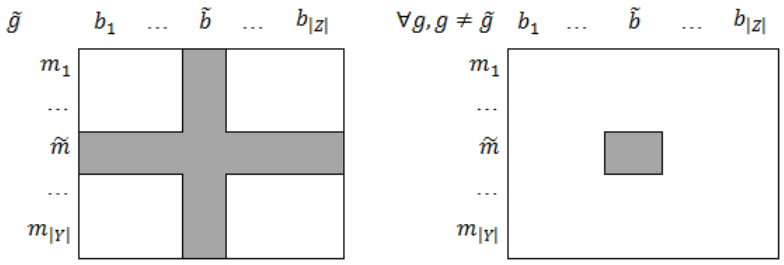


Fig. 1. Structure of prime OAC-triclusters: the dense cross-like central layer containing $\tilde{g}$ (left) and the layer for an object g (right) in $M \times B$ dimensions.

The basic algorithm for prime OAC-triclustering method is rather straightforward (see [5]). 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.

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 afterwards. This fact results in our previous MapReduce implementation [34].

3 Multimodal clustering

The direct extension of the prime object-attribute-condition triclustering is multimodal clustering for higher input relation arities. For the input polyadic context $\mathbb{K}_N = (A_1, A_2, \ldots, A_N, I \subseteq A_1 \times A_2 \times \cdots \times A_N)$ [30], we introduce the notion of *cumulus* for each input tuple $i = (e_1, e_2, \ldots, e_N) \in I$ and the corresponding entity e_k, where $k \in \{1, \ldots, N\}$ as follows:

$$cum(i,k) = \{e \mid (e_1, e_2, \ldots, e_{k-1}, e, e_{k+1}, \ldots, e_N) \in I\}.$$

The multimodal cluster generated by the tuple $i \in I$ is defined as follows:

$$((cum(i,1), \ldots, cum(i,N)).$$

Those cumuli operators are similar to primes for pairs (or tuples) of sets Eq. 1:

$$cum(i,k) = (\{e_1\}, \{e_2\}, \ldots, \{e_{k-1}\}, \{e_{k+1}\}, \ldots, \{e_N\})'.$$

However, here, they are applied to the tuples of input relation rather than to pairs (tuples) of sets.

In a certain sense, cumuli accumulate all the elements of a fixed type that are related by I.

As its triadic version, multimodal clustering is not greater than the number of tuples in the input relation, whereas the complete set of polyadic concepts may be exponential w.r.t. the input size [30].

4 Map-reduce-based multimodal clustering

4.1 Map-reduce decomposition

We follow three stage approach here. On each stage, we sequentially run the map and reduce procedures: First map → First reduce → Second map → Second reduce → Third map → Third reduce. Each map/reduce procedure of a certain stage is executed in parallel on all the available nodes/clusters. The way in tasks are distributed among the nodes/clusters depends on the concrete MapReduce technology implementation (in our case, Apache Hadoop). Below, we describe data flow between computational nodes and their processing.

1) The first map (Algorithm 2) takes a set of input tuples. Each tuple $(e_1, e_2, \ldots, e_N)$ is transformed into N key-value pairs: $\langle (e_2, \ldots, e_N), e_1 \rangle$, $\langle e_1, e_3, \ldots, e_N), e_2 \rangle$, $\ldots$, $\langle (e_1, e_2, \ldots, e_{N-1}), e_N \rangle$. The resulting pairs are passed to the further step.

2) The first reduce (Algorithm 3) receives all the accumulated values of each key. Thus, for each $(e_1, \ldots, e_N) \in I$ and the context entity type

Algorithm 2 Distributed Multimodal clustering: First Map

Input: I is a set of tuples of length N each
Output: $\langle subrelation, entity \rangle$ pairs.
1: **for all** $(e_1, e_2, \ldots, e_N) \in I$ **do**
2: **for all** $k \in \{1, \ldots, N\}$ **do**
3: $subrelation := (e_1, \ldots, e_{k-1}, e_{k+1}, \ldots, e_N)$
4: **emit** $\langle subrelation, e_k \rangle$
5: **end for**
6: **end for**

$k \in \{1, 2, \ldots, N\}$, we compute the cumulus $(e_1, \ldots, e_{k-1}, e_{k+1}, \ldots, e_N)'$. The values are passed to the next MapReduce stage with the key $(e_1, \ldots, e_{k-1}, e_{k+1}, \ldots, e_N)$.

Algorithm 3 Distributed Multimodal clustering: First Reduce

Input: key-value pairs $\langle subrelation, entities\ \{e_k^1, \ldots, e_k^L\} \rangle$
Output: $\langle subrelation, cumulus \rangle$
1: cumulus:={}
2: **for all** $e_k \in \{e_k^1, \ldots, e_k^L\}$ **do**
3: $cumulus := cumulus \cup \{e_k\}$
4: **emit** $\langle subrelation, cumulus \rangle$
5: **end for**

3) Second map (Algorithm 4). All the received keys are transformed into the original relations and passed to the second reduce procedure with unchanged values.

Algorithm 4 Distributed Multimodal clustering: Second Map

Input: $\langle subrelation, cumulus \rangle$, where $subrelation = (e_1, \ldots, e_{k-1}, e_{k+1}, \ldots, e_N)$
Output: $\langle generating\_relation, cumulus \rangle$ pairs.
1: **for all** $e_k \in cumulus$ **do**
2: $generating\_relation := (e_1, \ldots, e_{k-1}, e_k, e_{k+1}, \ldots, e_N)$
3: **emit** $\langle generating\_relation, cumulus \rangle$
4: **end for**

4) Second reduce (Algorithm 5). All the cumuli obtained for each input tuple of the original relation I are reduced to a single set. On this stage, we obtain all the original tuples and generated multimodal clusters. These clusters are presented as tuples of cumuli for respective entity types. All the obtained pairs $\langle generating\_relation, multimodal\_cluster \rangle$ are passed to the next stage.

5) Third map (Algorithm 6). The task of the third mapreduce stage is duplicate elimination and filtration by density threshold. It is beneficial to im-

Algorithm 5 Distributed Multimodal clustering: Second Reduce

Input: $\langle generating\_relation, cumuli\ \{A_1, A_2, \cdots, A_N\}\rangle$
Output: $\langle generating\_relation, multimodal\_cluster\rangle$ pairs
1: $multimodal\_cluster := (A_1, A_2, \cdots, A_N)$
2: **emit** $\langle generating\_relation, multimodal\_cluster\rangle$

plement within the reduce step, but to do so each obtained key-value pair $\langle generating\_relation, multimodal\_cluster\rangle$ should be passed further as follows $\langle multimodal\_cluster, generating\_relation\rangle$.

Algorithm 6 Distributed Multimodal clustering: Third Map

Input: $\langle generating\_relation, multimodal\_cluster\rangle$
1: **emit** $\langle multimodal\_cluster, generating\_relation\rangle$

6) Third reduce (Algorithm 7). For each input multimodal cluster and its generating tuples it is possible to directly compute density. All the unique clusters will be stored.

Algorithm 7 Distributed Multimodal clustering: Third Reduce

Input: $\langle multimodal\_cluster, generating\_relations\ \{r_1, r_2 \ldots, r_M\}\rangle$
1: **if** $\dfrac{|\{r_1, r_2 \ldots, r_M\}|}{vol(multimodal\_cluster)} \geq \theta$ **then**
2: **store** $\langle multimodal\_cluster\rangle$
3: **end if**

The time and memory complexities are provided assuming that the worst case scenario corresponds to the absolutely dense cuboid, i.e. polyadic context $\mathbb{K}_N = (A_1, A_2, \ldots, A_N, A_1 \times A_2 \times \cdots \times A_N)$. Thus, after careful analysis, the worst-case time complexity of the proposed three stage algorithm is $O(|I| \sum_{j=1}^{N} |A_j|)$. Not surprisingly it has the same worst-case memory complexity, since the stored and passed maximal number of multimodal clusters is $|I|$, and the size of each of them is not greater $\sum_{j=1}^{N} |A_j|$. However, from implementation point of view, since HDFS has default replication factor 3, those data elements are copied thrice to fulfil fault-tolerance.

4.2 Implementation aspects and used technologies

The application[2] has been implemented in Java and as distributed computation framework we use Apache Hadoop[3].

[2] `https://github.com/kostarion/multimodal-clustering-hadoop`
[3] `http://hadoop.apache.org/`

We have used many other technologies: Apache Maven (framework for automatic project assembling), Apache Commons (for work with extended Java collections), 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.

To provide the reader with basic information on the most important classes for M/R implementation, let us shortly describe them below.

Entity. This is a basic abstraction for an element of a certain type. Each entity is defined by its type index from 0 to n-1, where n is the arity of the input formal context. An entity value is a string that need to be kept during the program execution. This class inherit Writable interface for storing its objects in temporary and permanent Hadoop files. This is a mandatory requirement for all classes that pass or take their objects as keys and values of map and reduce methods.

Tuple. This class implements a representation of relation. Each object of the class Tuple contains the list of objects of Entity class and arity of its data given by its numeric value. This class implements interface WritableComparable<Tuple> to make it possible to use an object class Tuple as a key. The interface is similar to Writable, however one need to define comparison function to use in the key sorting phase.

Cumulus. This is an abstraction of cumulus, introduced earlier. It contains the list of string values and the index of respective entity. It also implements the following interfaces: WritableComparable<Cumulus> for using cumulus as a key and Iterable<String> for iteration by its values.

FormalConcept. This is an abstraction of both formal concepts and multimodal clusters, it contains the list of cumuli and implements interface Writable.

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

FirstMapper, SecondMapper, ThirdMapper. These are the classes that extend class Mapper<> of the Hadoop MapReduce library by respective mapping function from Subsection 4.1.

FirstReducer, SecondReducer, ThirdReducer. These classes extend class Reducer<> of the Hadoop MapReduce library for fulfilling Algorithms 2,4,6.

TextCumulusInputFormat, TextCumulusOutputFormat. These classes implement reading and writing for objects of Cumulus class; they also inherit RecordReader and RecordWriter interfaces, respectively. They are required to exchange results between different MapReduce phases within one MapReduce program.

JobConfigurator. This is the class for setting configuration of a single MapReduce stage. It defines the classes of input/output/intermediate keys and values of the mapper and reducer as well as formats of an input and output data.

App. This class is responsible for launching the application and chaining of M/R stages.

5 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.

5.1 Datasets

Synthetic datasets. The following synthetic dataset were generated.
The dense context $\mathbb{K}_1 = (G, M, B, I)$, where $G = M = B = \{1, \ldots, 60\}$ and $I = G \times M \times B \setminus \{(g, m, b) \in I \mid g = m = b\}$. In total, 60^3 - 60 =215,940 triples.
The context of three non-overlapped cuboids $\mathbb{K}_2 = (G_1 \sqcup G_2 \sqcup G_3, M_1 \sqcup M_2 \sqcup M_3, B_1 \sqcup B_2 \sqcup B_3, I)$, where $I = (G_1 \times M_1 \times B_1) \cup (G_2 \times M_2 \times B_2) \cup (G_3 \times M_3 \times B_3)$. In total, $3 \cdot 50^3 = 375,000$ triples.
The context $\mathbb{K}_3 = (A_1, A_2, A_3, A_4, A \times B \times C \times D)$ is a dense fourth dimensional cuboid with $|A_1| = |A_2| = |A_3| = |A_4| = 30$ containing $30^4 = 810,000$ triples.

These tests have sense since in M/R setting due to the tuples 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 $3^{min(|A_1|,|A_2|,|A_3|,|A_4|)}$ 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 reducers since the input has its maximal size w.r.t. to the size of A_i-s and the number of duplicates. In fact, our algorithm should correctly assemble the only one tricluster (A_1, A_2, A_3, A_4) 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. In total, there are 3,818 triples.
Movielens. The dataset contains 1,000,000 tuples that relate 6,040 users, 3,952 movies, ratings, and timestamps, where ratings are made on a 5-star scale [8].
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.

Table 2 contains the summary of IMDB and Bibsonomy contexts.

Table 2. Tricontexts based of real data systems for the experiments

Context	$\lvert G\rvert$	$\lvert M\rvert$	$\lvert B\rvert$	# triples	Density
IMDB	250	795	22	3,818	0.00087
BibSonomy	2,337	67,464	28,920	816,197	$1.8 \cdot 10^{-7}$

5.2 Results

The experiments has been conducted on the computer Intel®Core(TM) i5-2450M CPU @ 2.50GHz, 4Gb RAM (a typical commodity hardware) in the emulation mode, when Hadoop cluster contains only one node and operates locally and sequentially. By time execution results one can estimate the performance in a real distributed environment assuming that each node workload is (roughly) the same.

Table 3. Three-stage MapReduce multimodal clustering

Results, ms	IMDB	MovieLens100k	$\mathbb{K}_1$	$\mathbb{K}_2$	$\mathbb{K}_3$
Online OAC prime clustering	368	16,298	96,990	185,072	643,978
MapReduce multimodal clustering	7,124	14,582	37,572	61,367	102,699

Table 4. Three-stage MapReduce multimodal clustering

Dataset	Online OAC Prime	M/R total	MapReduce stages			# clusters
			1st	2nd	3rd	
MovieLens100k	89,931	16,348	8,724	5,292	2,332	89,932
MovieLens250k	225,242	42,708	10,075	20,338	12,295	225,251
MovieLens500k	461,198	94,701	15,016	46,300	33,384	461,238
MovieLens1M	958,345	217,694	28,027	114,221	74,446	942,757
Bibsonomy (≈800k triples)	> 6 hours	3,651,072 (≈1 hour)	19,117	1,972,135	1,659,820	486,221

In Tables 3 and 4 we summarise the results of performed tests. It is clear that on average our application has fewer execution time than its competitor, the online version of OAC-triclustering. If we compare the implemented program with its original online version, the results are worse for not that big and sparse dataset as IMDB. 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 three 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

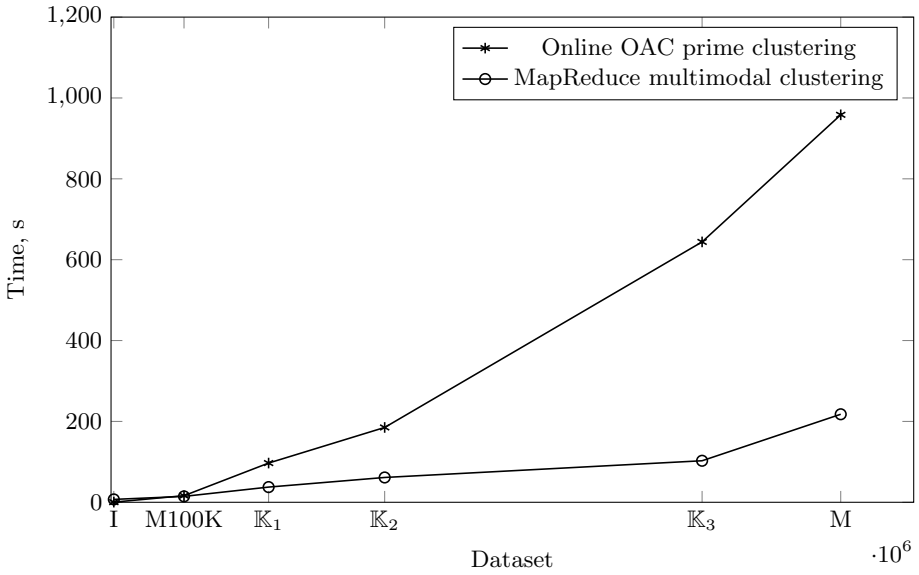


Fig. 2. Performance curves for six datasets: I stands for IMDB dataset with 3,818 triples, M100K – MovieLens dataset with 100K tuples, M – MovieLens dataset with 1M tuples

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 up to five-six times (see Fig. 2). Thus, the last test for BibSonomy data has been successfully passed, but the competitor was not able to finish it within one hour. As for the M/R stages, the most time-consuming phases are 2nd and 3rd stages.

6 Conclusion

In this paper we have presented a map-reduce version of multimodal clustering algorithm, which extends triclustering approaches and copes with bottlenecks of the earlier M/R triclustering version [34]. We have shown that the proposed algorithm is efficient from both theoretical and practical points of view. This is a variant of map-reduce based algorithm where the reducer exploits composite keys directly (see also Appendix section [34]). However, in despite the step towards Big Data technologies, a proper comparison of the proposed multimodal clustering and noise tolerant patterns in n-ary relations by DataPeeler and its descendants [1] is not yet conducted (including MapReduce setting).

Acknowledgements. The study was implemented in the framework of the Basic Research Program at the National Research University Higher School of Economics, in the Laboratory of Intelligent Systems and Structural Analysis

(Sections 2 and 5), and funded by the Russian Academic Excellence Project '5-100'. The first author was also supported by Russian Scientific Foundation (Section 1, 3, and 4) under grant 17-11-01294. The authors would like to thank anonymous reviewers as well as 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. 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. Eren, K., Deveci, M., Kucuktunc, O., Catalyurek, Umit V.: A comparative analysis of biclustering algorithms for gene expression data. Briefings in Bioinform. (2012)
3. Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edn. (1999)
4. Georgii, E., Tsuda, K., Schölkopf, B.: Multi-way set enumeration in weight tensors. Machine Learning **82**(2), 123–155 (2011)
5. Gnatyshak, D.V., Ignatov, D.I., Kuznetsov, S.O.: From triadic FCA to triclustering: Experimental comparison of some triclustering algorithms. In: CLA. pp. 249–260 (2013)
6. 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)
7. 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)
8. Harper, F.M., Konstan, J.A.: The movielens datasets: History and context. TiiS **5**(4), 19:1–19:19 (2016). https://doi.org/10.1145/2827872, `https://doi.org/10.1145/2827872`
9. Henriques, R., Madeira, S.C.: Triclustering algorithms for three-dimensional data analysis: A comprehensive survey. ACM Comput. Surv. **51**(5), 95:1–95:43 (Sep 2018). https://doi.org/10.1145/3195833, `http://doi.acm.org/10.1145/3195833`
10. Ignatov, D.I., Gnatyshak, D.V., Kuznetsov, S.O., Mirkin, B.: Triadic formal concept analysis and triclustering: searching for optimal patterns. Machine Learning pp. 1–32 (2015)
11. Ignatov, D.I., Kuznetsov, S.O., Poelmans, J.: Concept-based biclustering for internet advertisement. In: ICDM Workshops. pp. 123–130. IEEE Computer Society (2012)
12. 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)
13. 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. vol. LNCS 8722, pp. 47–58 (2014)
14. 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)
15. Kaytoue, M., Kuznetsov, S.O., Macko, J., Napoli, A.: Biclustering meets triadic concept analysis. Ann. Math. Artif. Intell. **70**(1-2), 55–79 (2014)

16. 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). https://doi.org/10.1016/j.ins.2010.07.007, http://dx.doi.org/10.1016/j.ins.2010.07.007
17. Krajca, P., Vychodil, V.: Distributed algorithm for computing formal concepts using map-reduce framework. In: N. Adams et al. (Eds.): IDA 2009. vol. LNCS 5772, pp. 333–344 (2009)
18. 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. pp. 62–67 (2009). https://doi.org/10.1109/DBKDA.2009.32, http://dx.doi.org/10.1109/DBKDA.2009.32
19. Li, A., Tuck, D.: An effective tri-clustering algorithm combining expression data with gene regulation information. Gene regul. and syst. biol. **3**, 49–64 (2009), http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2758278/
20. 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)
21. Metzler, S., Miettinen, P.: Clustering boolean tensors. Data Min. Knowl. Discov. **29**(5), 1343–1373 (2015). https://doi.org/10.1007/s10618-015-0420-3, https://doi.org/10.1007/s10618-015-0420-3
22. Mirkin, B.: Mathematical Classification and Clustering. Kluwer, Dordrecht (1996)
23. 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)
24. 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)
25. Rajaraman, A., Leskovec, J., Ullman, J.D.: Mining of Massive Datasets, chap. MapReduce and the New Software Stack, pp. 19–70. Cambridge University Press, England, Cambridge (2013)
26. Schweikardt, N.: One-pass algorithm. In: Encyclopedia of Database Systems, Second Edition (2018). https://doi.org/10.1007/978-1-4614-8265-9_253, https://doi.org/10.1007/978-1-4614-8265-9_253
27. Shin, K., Hooi, B., Faloutsos, C.: Fast, accurate, and flexible algorithms for dense subtensor mining. TKDD **12**(3), 28:1–28:30 (2018). https://doi.org/10.1145/3154414, https://doi.org/10.1145/3154414
28. Spyropoulou, E., De Bie, T., Boley, M.: Interesting pattern mining in multi-relational data. Data Mining and Knowledge Discovery **28**(3), 808–849 (2014)
29. Ustalov, D., Panchenko, A., Kutuzov, A., Biemann, C., Ponzetto, S.P.: Unsupervised semantic frame induction using triclustering. In: Gurevych, I., Miyao, Y. (eds.) Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers. pp. 55–62. Association for Computational Linguistics (2018), https://aclanthology.info/papers/P18-2010/p18-2010
30. Voutsadakis, G.: Polyadic concept analysis. Order **19**(3), 295–304 (2002)
31. 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)
32. 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. vol. LNAI 7278, pp. 292–308 (2012)

33. Zhao, L., Zaki, M.J.: Tricluster: An effective algorithm for mining coherent clusters in 3d microarray data. In: SIGMOD 2005 Conference. pp. 694–705 (2005)
34. Zudin, S., Gnatyshak, D.V., Ignatov, D.I.: Putting oac-triclustering on mapreduce. In: Yahia, S.B., Konecny, J. (eds.) Proceedings of the Twelfth International Conference on Concept Lattices and Their Applications, Clermont-Ferrand, France, October 13-16, 2015. CEUR Workshop Proceedings, vol. 1466, pp. 47–58. CEUR-WS.org (2015), http://ceur-ws.org/Vol-1466/paper04.pdf