Lamari and Slaoui *J Big Data* (2017) 4:28
DOI 10.1186/s40537-017-0090-7

Journal of Big Data

METHODOLOGY Open Access

Clustering categorical data based on the relational analysis approach and MapReduce

Yasmine Lamari* and Said Chah Slaoui

*Correspondence:
lamari.yasmine@gmail.com
Department of Computer Science, Faculty of Science of Rabat, Mohammed V University, 1014RP, Rabat, Morocco

Abstract

The traditional methods of clustering are unable to cope with the exploding volume of data that the world is currently facing. As a solution to this problem, the research is intensified in the direction of parallel clustering methods. Although there is a variety of parallel programming models, the MapReduce paradigm is considered as the most prominent model for problems of large scale data processing of which the clustering. This paper introduces a new parallel design of a recently appeared heuristic for hard clustering using the MapReduce programming model. In this heuristic, clustering is performed by efficiently partitioning categorical large data sets according to the relational analysis approach. The proposed design, called PMR-Transitive, is a single-scan and parameter-free heuristic which determines the number of clusters automatically. The experimental results on real-life and synthetic data sets demonstrate that PMR-Transitive produces good quality results.

Keywords: Categorical data, Clustering, MapReduce, Relational analysis approach

Introduction

Over the past decade, the amount of information accumulated every second has become a treasure of inestimable value. Social media sites, sensors, transactions records and many other sources that come from everywhere are behind the Big Data phenomenon. Consequently, considerable efforts have been devoted to exploring such massive data in order to gain the maximum benefit from this treasure. To cope with the huge volume of data, various parallel programming frameworks have recently emerged.

Clearly, MapReduce is the most prominent model for problems of large-scale data processing. It was proposed by Dean and Ghemawat [1] at Google where it was successfully used for various purposes. The strengths of this model are summarized in the fact that it allows automatic parallelism and distribution. In addition to the fault-tolerant mechanism that helps in overcoming failures, it provides also tools to manage the status, monitoring, and load balancing. The locality optimization is ensured by storing data on local disks to avoid network bandwidth consumption. Thereby, MapReduce allows us to focus on the problem rather than on complex details of parallel programming.

Recently, the research studies in data mining are increasingly interested in the concept of parallel programming. Data mining covers a wide variety of data analysis procedures, including the classification, the regression, the clustering, and so on. In this paper, we

focus on the clustering procedure, which aims to partition data into groups of similar objects fulfilling the conditions of the maximizing the similarity between objects in the same group, and the minimization of the similarity between objects in different groups [2]. In order to solve this problem, we propose PMR-Transitive, which is a new parallel heuristic based on the MapReduce programming model of a recently appeared method, named *Transitive* heuristic [3]. In this heuristic, clusters are obtained by partitioning categorical large data sets according to the relational analysis approach [4]. The relational analysis approach provides a mathematical formalism where the problem of clustering takes the form of a linear program with n^2 integer attributes (with n the number of instances). Heuristics are the most convenient solution to produce satisfactory clustering results in the fastest time, particularly in the context of Big Data, where the number of instances is large and the response time is a critical factor. Since the original heuristic is sequential, it needs to be adjusted to the MapReduce model. This paper provides a detailed description of the new design based on the key methods of the MapReduce model, namely, Map and Reduce. And advantageously, most steps which produce high computational costs involved in *Transitive* heuristic can be processed in parallel.

The remainder of this paper is organized as follows: "Motivation and related work", presents briefly the MapReduce model and some related work. In "Relational analysis approach", the problem statement is formalized. The overview of the *Transitive* heuristic and its new version based on MapReduce are described in "Transitive heuristic" and "MapReduce-based implementation of transitive" respectively. Subsequently, the experimental results are presented in "Results and discussion". Finally, conclusions and propositions for future studies are drawn briefly in "Conclusions".

Motivation and related work

With the continuous increase of the data volume, the traditional methods of clustering have reached their limitations giving rise to the parallel clustering. In this section, we review the MapReduce frameworks and some related clustering algorithms.

MapReduce [1] is considered one of the most prominent programming models for problems of large scale data processing of which the cluster analysis. It consists of two phases: Map and Reduce. The Map phase is responsible for filtering and sorting, while the Reduce phase is in charge of summarizing the outputs of the previous phase. The Map function receives records from the input files as key-value pairs and produces intermediate key-value pairs. When the Map phase is completely finished, the Reduce phase starts. Each Reducer works on the values of a specific intermediate key and produces one final value for the same key.

There are several implementations of the MapReduce programming model. Hadoop offers the most popular framework in Java. Authored by Apache Software Foundation, the project includes modules enabling a reliable and scalable distributed computing as an open source framework. For the qualities that it provides, namely, its organized architecture, scalability, cost effectiveness, flexibility and resilience to failure, Hadoop MapReduce framework is used for the implementation of the proposed method.

Since there are several MapReduce-based clustering algorithms, we mention in this section only a few relevant works such as the PKMeans [5] algorithm. PKMeans is a MapReduce-based implementation of the k-means algorithm. It is designed with a single

MapReduce job, in which the Map function is responsible for the assignment of each sample to the nearest center, and the Reduce function is responsible for updating the new centers. In addition to the combiner function which aggregates partially the values of the points assigned to the same cluster in order to Reduce the communication cost. The experiments, which have been performed in a cluster of four nodes, demonstrate that this approach can process large data sets.

In 2011, He et al. [6] proposed the MR-DBSCAN algorithm, which is a MapReduce-based implementation of the well-known DBSCAN algorithm. The proposed parallel method consists of four steps. In the first step, the size and the general spatial distribution of the total records are summarized, then, a list of dimensional index indicated an approximate grid partitioning is generated for the next step. The second step performs the main DBSCAN process for each subspace divided by the partition profile. The third step handles the cross border issues when merging the subspaces. At the end, a cluster ID mapping, from local clusters to global one, is built for the entire data set based on pairs lists collected from the previous step. Then, the local ID's are changed by the global ones for points from all partitions in order to produce a united output. The experiments, which have been performed in a cluster of 13 nodes, demonstrated that the MR-DBSCAN is efficient on large data sets since it was tested with data sets up to 50.4 GB.

In the same context, Kim et al. [7] suggested a new density-based clustering algorithm, called DBCURE, in addition to its parallel version, called DBCURE-MR, which is implemented using the MapReduce programming model. DBCURE acts as DBSCAN by reiterating two steps. The first step selects an unvisited point in the data set which is considered as a seed and then inserts it to the seed set. In the second step, all points that are density-reachable from the seed set are retrieved. This process produces clusters one at a time and stops when the seed set becomes empty, contrary to its parallel version, which finds several clusters at the same time by treating each core point in parallel through four steps. The first step is responsible for the estimation of the neighborhood covariance matrices and it is performed using two MapReduce algorithms. The second step performs the computation of ellipsoidal τ-neighbourhoods and it is performed using two other MapReduce algorithms. The third step discovers core clusters, which is done by a single MapReduce algorithm. Finally, the last step is responsible for the merge of core clusters and it is performed with a single MapReduce algorithm. The experiments, which were performed in a cluster of 20 nodes with data sets reaching 0.5 GB, demonstrated that the proposed approach scales up well with the MapReduce programming model.

Most the proposed MapReduce-based clustering algorithms focused on the k-means and the DBSCAN methods which deal only with numerical data (points). It is therefore not obvious to compare the results produced by the PMR-Transitive, which operates on categorical data (records), with such methods. So in terms of quality, we suggest comparing the clustering results obtained by the proposed method with two serial clustering algorithms well-known for clustering categorical data, which are the MMR [8] and some enhanced versions of k-modes [9]. The MMR (Min-Min-Roughness) algorithm is based on the rough set theory, which requires the number of clusters as an input and uses a new similarity method based on the roughness concept to produce stable results. This algorithm is distinguished by the ability to handle uncertainty in the clustering process. Bai and Liang proposed to use the between-cluster information to improve the

effectiveness of some existing versions of the k-modes algorithm. Clustering results of categorical data sets have demonstrated that the improvements brought to the k-modes algorithms are effective and scalable.

The PMR-Transitive, which is proposed in this paper, is a new parallel design of the *Transitive* heuristic implemented using the Hadoop MapReduce framework. As mentioned in the "Introduction" section, the *Transitive* heuristic is a fast heuristic which finds clusters by partitioning categorical large data sets according to the relational analysis approach. The proposed method presents some relevant points; it processes categorical large data sets rapidly, without any prior settings or sampling method, and guarantees a good quality solution in a reasonable time. Indeed, contrary to other algorithms, the number of clusters is automatically detectable by the *Transitive* heuristic and its new parallel design.

Relational analysis approach

The relational analysis approach is a mathematical data analysis model used in different fields including clustering. It was conceived by J F Marcotorchino and P Michaud in the late 1970s at the IBM European Center of Applied Mathematics [4]. The relational analysis is defined as an optimization problem under linear constraints of the Condorcet's criterion. The detailed mathematical representation of the relational analysis approach can be found in [10]. In this subsection, we review some basic notions of this model, since it is the basis of the proposed parallel method and its original version.

Let $E = \{1, 2, ..., n\}$ be the data set of n instances described by the set $V = \{v_1, v_2, ..., v_k\}$ of k categorical attributes. We denote by $v_l(i)$ the value domain of the categorical attribute v_l for the instance i.

To each attribute v_l is assigned a matrix C_l of general term $c_l(i,j)$ with $(i,j) \in E^2$:

$$C_l = \big(c_l(i,j)\big)_{1\le i\le n, 1\le j\le n}. \tag{1}$$

where $c_l(i,j) = \begin{cases} 1 & if\ v_l(i) = v_l(j) \\ 0 & otherwise \end{cases}$

Then, the following matrix, which is called the collective table or the table of Condorcet, is deduced:

$$C = \big(c(i,j)\big)_{1\le i\le n, 1\le j\le n}. \tag{2}$$

where $c(i,j) = \sum_{l=1}^{k} c_l(i,j)$

In other words, $c(i, j)$ is the number of attributes for which the instances i and j share the same value domain.

In the relational analysis approach, a partition of the data set E is represented by the matrix Y:

$$Y = \big(y(i,j)\big)_{1\le i\le n, 1\le j\le n}. \tag{3}$$

where $y(i,j) = \begin{cases} 1 & if\ i\mathcal{R}j \\ 0 & otherwise \end{cases}$ $\mathcal{R}$ is an equivalence relation.

Theoretically, the problem proposed by the relational analysis approach, which needs to be solved, is formulated as the following: find the best partition , which represents the clustering result, by optimizing the Condorcet's criterion [11] accordance with the constraints defining an equivalence relation.

$$C(Y) = \sum_i \sum_j c(i,j)\, y(i,j) + \overline{c(i,j)}\,\overline{y(i,j)}. \tag{4}$$

The model of linear programming, which results from the relational analysis approach, is given below:

$$\begin{cases} Max\, C(Y) \\ Y: partition\ of\ the\ set\ E \\ \forall\, i \in E,\ y(i,i) = 1 (Reflexivity) \\ \forall\, (i,j) \in E^2,\ y(i,j) - y(j,i) = 0 (Symmetry) \\ \forall\, (i,j,k) \in E^3,\ y(i,j) + y(j,k) - y(i,k) \leq 1 (Transitivity) \\ \forall\, (i,j) \in E^2,\ y(i,j) \in \{0,1\} (Binary) \end{cases} \tag{5}$$

The original *Transitive* heuristic and its new parallel design, which is the subject of this paper, are based on the relational analysis model. This choice is mainly made for the qualities that this model presents when applied to clustering. Indeed, when applying this formalism, a good quality clustering result can be achieved without defining the maximum number of clusters. In addition to the good quality partitions which are guaranteed by the concept of the paired comparisons concept.

Transitive heuristic

The purpose of the *Transitive* heuristic is to transform a structure of covering, which is not transitive, into a transitive solution. In this section, we present the overview of *Transitive* heuristic and we formalize the definitions and concepts related to this method.

Preliminary

Let $M = \{m_1, m_2, ..., m_q\}$ be the set formed by all value domain of the k categorical attributes. Then we denote by $M(i)$ the set of values terms for the instance i.

Profile definition

A profile is a vector constructed from the complete disjunctive coding of an instance i.

$$Pr(i,m) = \begin{cases} 1 & if\ m \in M(i) \\ 0 & otherwise \end{cases} \tag{6}$$

The profile of the instance i is defined by:

$$P(i) = (Pr(i,m))_{1 \leq m \leq q}. \tag{7}$$

Through to the concept of the profile, one can easily find the term $c(i, j)$ previously described in the relational analysis approach:

$$c(i,j) = P(i) \cdot P(j). \tag{8}$$

By adopting this technique, we avoid the computation of the C_l matrices, and consequently the C matrix described in the previous section. Nonetheless, $c(i, j)$ quantities

are widely involved in the *Transitive* heuristic to perform the coefficient computation and in particular for calculating the contribution function. This calls into question the usefulness of the transformation of categorical data into binary data using the complete disjunctive coding. In order to simplify calculating $c(i, j)$ without needing any transformation of the original data and Reduce significantly the computational time, we propose the following way:

$$c(i,j) = |M(i) \cap M(j)|. \tag{9}$$

Thus, $c(i, j)$ is the cardinal of the set formed by the intersection of the two sets of values of the terms for instances i and j.

Cluster definition

Using a representative $r \in E$, which is a random instance selected from the data set, we define its corresponding cluster Cl_r by:

$$Cl_r = \{i \in E \ / \ coef(i,r) > 0\}. \tag{10}$$

with $coef(i,r) = c(i,r) - \overline{c(i,r)}$.

The representative element is the generator of the cluster; it helps to speed up the heuristic in the phase of separation of non-disjoint clusters. Indeed, it allows calculating the similarity of a shared instance with the representative of each cluster, instead of calculating the similarity with all the instances contained in the cluster.

Contribution function

The contribution of an instance i to a cluster Cl_r is defined as follows:

$$cont(i, Cl_r) = \sum_{j \in Cl_r} coef(i,j). \tag{11}$$

Computing the contribution using the above formula consumes a lot of time since the disjunctive table is calculated as often as there are instances in the cluster. Reducing the computational time of the contribution function can be done by restricting comparison of the instance solely to the cluster representative r:

$$cont(i, Cl_r) = coef(i,r). \tag{12}$$

Overview of transitive heuristic

The process of the *Transitive* heuristic is shown in Fig. 1. This heuristic consists of four main steps: initialization, construction, intersection, and evaluation.

Firstly, in order to build the first cluster, a random instance, called representative, is selected randomly and used to cluster instances which resemble it using the coefficient function. All identifiers of clustered instances are saved in order to avoid selecting a new representative which is already clustered for the next iterations.

The repetition of iterations can generate fuzzy clustering. So in order to have distinctive clusters, the intersection of clusters is calculated. Then, for each shared instance a decision of the suitable cluster is made. This decision consists of computing the contribution of the shared instance based on the representatives of clusters. Then, the highest

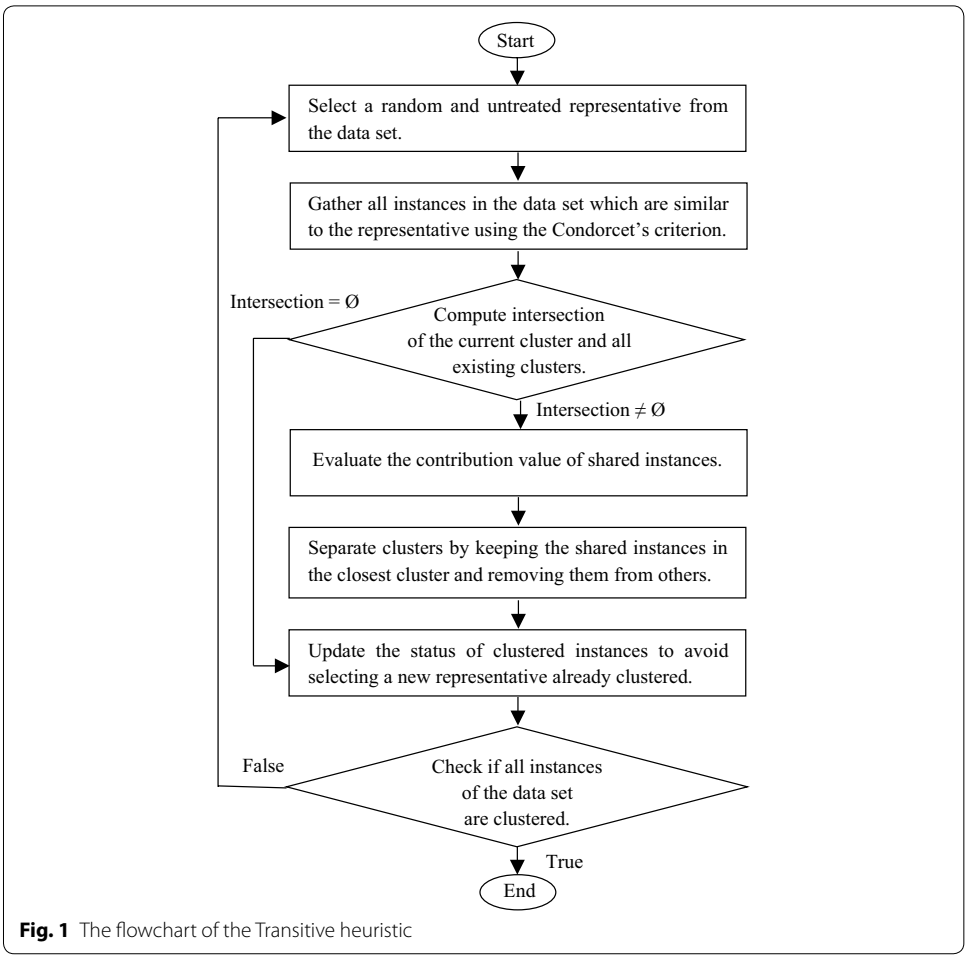

Fig. 1 The flowchart of the Transitive heuristic

value means that the corresponding representative is the closest to the shared instance in where it will be kept and removed from the others.

By virtue of its suitable features, *Transitive* heuristic provides good quality results in reasonable computing time and that without using the traditional sampling methods or even the setting of input parameters, such as the number of clusters, thresholds, and other parameters. However, in its serial version, it cannot take advantage of the distributed systems to process big data. To make the *Transitive* heuristic run in a parallel environment, some adjustments are necessary that we will discuss in the next section.

MapReduce-based implementation of transitive

The new design of *Transitive* heuristic based on the MapReduce framework is illustrated in Fig. 2. Multiple mappers run in parallel and produce partial clustering. Then, a single reducer runs and transforms the initially obtained partitions into a final result of hard clustering.

Map-function

The Map function performs an initial clustering of the input data block. It gathers the input instances (pairs) in clusters using the Condorcet's criterion as a similarity measure

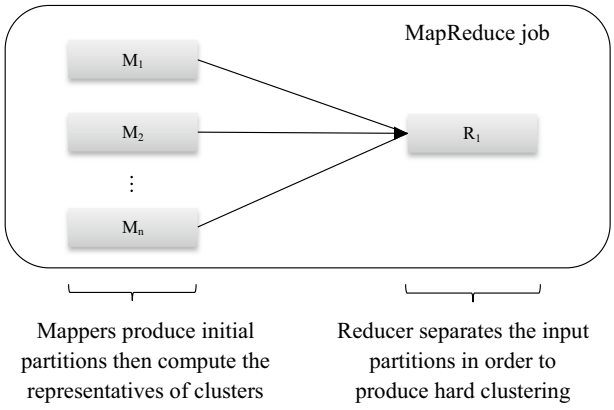

Fig. 2 The design of PMR-Transitive using a single MapReduce job

and it allows the instances to belong to more than one cluster. Then, it recalculates the representatives of clusters in the initial solution (partition). The representative of a cluster is calculated on the basis of the frequency of occurrence of features in the cluster. When the mapper is complete, it returns as an intermediate result a fuzzy clustering of the received part of the input file that will be refined later in the Reduce phase.

The Map function outputs a collection of cluster structures. Each cluster structure contains the representative instance and only the identifiers of the instances that are members of the cluster with their similarity scores. The similarity scores are useful in the Reduce phase because they avoid the recalculations of similarities in the evaluation of the contribution of shared instances. This technique allows dispensing with the data of clusters' members in the Reduce phase, thus, we decrease the amount of data sent from the Map phase to the Reduce phase. The pseudo code of the function of the Map phase is given below.

Algorithm 1 Map (key, value)

Input: $< k, v >$ pair, k: the identifier of the instance, v: the corresponding data record.
Output: $< k_i, v_i >$ pair, k_i: the data record of the cluster's representative, v_i: the members of the cluster (only their identifiers and similarity scores).

```
Initialize clusters: a collection to save the discovered clusters;
for each cluster in clusters do
    sim ← Compute_Condorcet_Score(cluster.representative,v);
    if sim > 0 then
        Add k and sim to cluster.members;
        Update cluster.representative;
    end if
end for
if < k, v > does not resemble any existing cluster then
    Add a new cluster to clusters with v as representative, k as member, 1 as similarity score;
end if
When the mapper is complete, output < k_i, v_i > pair for each cluster in clusters;
```

Reduce-function

The clusters produced during the Map phase of each host may share some instances. However, the aim of the proposed method is to produce a hard clustering. The Reduce phase is responsible for the separation of clusters. This is achieved by computing the intersection of clusters in order to determine the shared members. Then, for each shared

member we compare its similarity to the clusters in which it belongs. This information lies in the similarity scores of this member. In fact, the higher score indicates that the corresponding cluster is the most suitable for the shared member, and in which it will be kept in this cluster and removed from all others. This process stops when all clusters are disjointed.

In this proposed parallel design of *Transitive* heuristic, we considered a single reducer, and therefore it can be thought that this is expensive in term of run time. But the Reduce phase involves merely some comparisons between the members of clusters and there is no need to recalculate the similarities between members and representatives. And so, we obtain clusters which are consistent and accurate as a final steady result. The pseudo code of the function of the Reduce phase is given below.

Algorithm 2 Reduce (key, value)

Input: $< k, v >$ pair, k: the data record of the representative, v: the members of the cluster.
Output: $< k_i, v_i >$ pair, k_i: the identifier of the cluster, v_i: the corresponding members.

```
Initialize clusters: a collection to save the final clusters;
index ← 1;
if clusters = ∅ then
    Add a new cluster to clusters with index as identifier, v as members;
    index ← 1;
else
    for each cluster in clusters do
        intersect ← v ∩ cluster.members;
        if intersect ≠ ∅ then
            for each member in intersect do
                Keep member in the cluster corresponding to the highest similarity score;
            end for
        end if
    end for
    if v = ∅ then
        Add a new cluster to clusters with index as identifier, v as members;
        index ← index + 1;
    end if
end if
When the reducer is complete, output < k_i, v_i > pair for each cluster in clusters;
```

Discussion

Some noteworthy adjustments were brought to the original *Transitive* heuristic to match the MapReduce programming model:

- Unlike *Transitive* heuristic, the selection of representatives of clusters in PMR-Transitive is not carried out in a random way. In fact, the first key-value pair introduced to the Map function is considered to be an initial representative. The next inputs that follow are either added to existing clusters, or considered as initial representatives that will be updated after the construction step, and so on.
- In the *Transitive* heuristic, the initialization step takes place before the construction step. Indeed, first the representative is selected, and then its cluster is built. The PMR-Transitive shifts those steps because of the forsaken random selection and which is not applied to the MapReduce framework.
- In the *Transitive* heuristic, the representatives are instances that are selected randomly from the data set, while in the PMR-Transitive heuristic they are fictive and computed based on the profile of the cluster's members.

- Another important point, it concerns the fact that *Transitive* heuristic is an iterative method, while the proposed parallel design is a single-pass heuristic.

Results and discussion

In this section, we evaluate the quality of clustering results and the performance of PMR-Transitive regarding some commonly used categorical real-life and synthetic data sets. The clustering result is assessed by the purity (also called accuracy) measure, the normalized mutual information (NMI), and the adjusted rand index (ARI) [12].

The performance experiments were run on a single node, which has a quad-core processor of 3.60 GHz and 8 GB of memory and using Hadoop version 2.2.0 and Java 1.7.0. The size of blocks used for the experiments is 64 MB.

Clustering evaluation metrics

Purity

The purity of a cluster i measures the extent to which this contains objects of a single class and it is defined as:

$$P_i = \frac{n_i^k}{n_i}. \tag{13}$$

where n_i denotes the size of the cluster i, n_i^k is the number of instances that are correctly assigned in the cluster i, and k denotes the dominant class in the cluster i. Then, the overall purity is defined as:

$$P = \frac{1}{k}\sum_{i=1}^{k} P_i. \tag{14}$$

where k denotes the number of clusters.

Normalized mutual information

The normalized mutual information is bounded in [0, 1], and like the purity measure, it should be maximized:

$$NMI = \frac{I(P_i, P_h)}{\sqrt{E(P_i)E(P_h)}}. \tag{15}$$

where $I(P_i, P_h)$ is the mutual information of the produced partition and the ground truth partition and it is calculated as follows:

$$I(P_i, P_h) = \sum_i \sum_h \frac{n_i^h}{N} \log \frac{N n_i^h}{n_i n_h}. \tag{16}$$

where n_i and n_h are the sizes of the cluster i and the class h respectively, and n_i^h denotes the number of instances in the cluster i, which belong to the class h. $E(P_i)$ and $E(P_h)$ represent the entropy of the produced partition and the ground truth partition, and N is the total number of instances in the data set.

Adjusted rand index

The adjusted rand index [13] is also bounded in [0, 1] and it is defined as:

$$ARI(P_i, P_h) = \frac{\sum_{i,h}\binom{n_i^h}{2} - \left[\sum_i \binom{n_i}{2}\sum_h \binom{n_h}{2}\right] \Big/ \binom{N}{2}}{\frac{1}{2}\left[\sum_i \binom{n_i}{2} + \sum_h \binom{n_h}{2}\right] - \left[\sum_i \binom{n_i}{2}\sum_h \binom{n_h}{2}\right] \Big/ \binom{N}{2}}. \tag{17}$$

where P_i represents the produced partition and P_h represents the ground truth partition, n_i^h denotes the number of instances that belong to the class h in the cluster i , n_i denotes the sizes of the cluster i, n_h denotes the size of the class h, and N is the total number of instances in the data set.

Data sets description

All real-life data sets used in the experiments were retrieved from UCI Machine Learning Repository [14]. Table 1 presents a brief description of the categorical data sets used in the experiments.

Clustering results

These experiments measure the quality of the clustering results. We show in detail the partitions obtained when using the real-life data sets described above.

Table 2 contains the clustering results obtained when applying the PMR-Transitive method to the soybean data set. 5 clusters have been discovered, of which four reached the maximum purity value. It is noteworthy that the number of clusters discovered by the PMR-Transitive is very close to the number of classes (four diseases) in the soybean data set. The obtained partition contains only one incorrectly clustered instance (in cluster four). Thus, the overall purity of the clustering result is 97%.

Table 3 contains the clustering results obtained when applying the PMR-Transitive method to the zoo data set. 7 clusters have been discovered, of which two reached the

Table 1 Description of real-life data sets

Data set	Size	Number of attributes	Number of classes	Missing values
Soybean	47	35	4	No
Zoo	101	17	7	No
Mushroom	8124	22	2	Yes

Table 2 Clustering result of PMR-Transitive applied to the soybean data set

Cluster	Size	Distribution				Purity
		C_1	C_2	C_3	C_4	
1	10	10	0	0	0	1
2	10	0	10	0	0	1
3	9	0	0	9	0	1
4	7	0	0	1	6	0.85
5	11	0	0	0	11	1

Table 3 Clustering result of PMR-Transitive applied to the zoo data set

Cluster	Size	Distribution							Purity
		C_1	C_2	C_3	C_4	C_5	C_6	C_7	
1	42	41	0	1	0	0	0	0	0.98
2	5	0	4	1	0	0	0	0	0.80
3	17	0	16	0	0	0	1	0	0.94
4	17	0	0	3	13	1	0	0	0.76
5	3	0	0	0	0	3	0	0	1
6	5	0	0	0	0	0	5	0	1
7	12	0	0	0	0	0	2	10	0.83

maximum purity value. Once again, the number of clusters discovered by the proposed method corresponds to the number of classes in the zoo data set. In the obtained partition, 92 instances are correctly assigned giving 91% as overall purity.

Table 4 contains the clustering results obtained when applying the PMR-Transitive method to the mushroom data set. 16 clusters have been discovered, of which 13 reached the maximum purity value. As reported in [3], the number of clusters discovered by the Transitive heuristic is 14 for this data set. The clusters 6, 12, and 13 in Table 4 are also observed in the result of the original method.

Table 5 shows the evaluation of clustering results of PMR-Transitive according to the purity, the NMI, and the ARI metrics. When the results are assessed using the ARI measure, the number of clusters must meet the number of classes in the clustering partition recognized as the ground truth in order to maximize the value of ARI. This is not applicable for the PMR-Transitive heuristic since it automatically detects the number of clusters. This explains why the values of ARI decrease somewhat.

Table 4 Clustering result of PMR-Transitive applied to the mushroom data set

Cluster	Size	Distribution		Purity
		C_1	C_2	
1	2010	1937	73	0.96
2	768	768	0	1
3	307	307	0	1
4	89	48	41	0.54
5	185	185	0	1
6	192	192	0	1
7	17	17	0	1
8	48	48	0	1
9	963	706	257	0.73
10	1719	0	1719	1
11	291	0	291	1
12	36	0	36	1
13	1296	0	1296	1
14	7	0	7	1
15	8	0	8	1
16	188	0	188	1

Table 5 Evaluation of clustering results of PMR-Transitive according to the purity, NMI, and ARI metrics

Data set	Purity	NMI	ARI
Soybean	0.97	0.95	0.78
Zoo	0.91	0.85	0.88
Mushroom	0.96	0.82	0.44

When the MMR [8], the ROCK [15], and the Squeezer [16] algorithms are applied to the mushroom data set, they produce 20 clusters, 21 clusters, and 24 clusters, respectively, which are fairly large. While the proposed method outperforms these clustering algorithms in term of the number of clusters discovered compared to the quality of clustering.

Figure 3 presents a comparison of the overall purity obtained by applying the proposed method using the real-life data sets described above. For comparison purposes, we present some results reported in [8] and [9] concerning, respectively, the performance of the MMR algorithm and some enhanced k-modes versions applied to the same data sets used to assess the quality of results produced by the proposed method.

The original *Transitive* heuristic outperforms its new parallel version, PMR-Transitive, as to the mushroom data set. This can be explained by the fact that the *Transitive* heuristic is a multiple scan method, while the PMR-Transitive is a single-scan method. However, it must not be forgotten that the results produced by the original method represent the best in 100 runs, since the random start of the *Transitive* heuristic produces a different solution for each run, while the proposed method reaches good quality results, stable, and reproducible.

In general, the performance of the proposed method and its predecessor is better than the results obtained with the MMR and the k-modes algorithms, except for the soybean

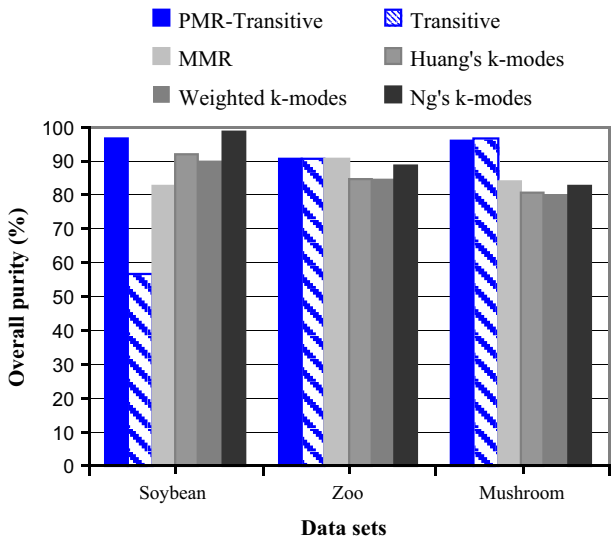

Fig. 3 Comparison of PMR-Transitive with transitive, MMR, and k-modes algorithms on real-life data sets

data set, where the enhanced Ng's k-modes algorithm produces a high-quality result reaching 99% of purity.

It should be noted that *Transitive* and its new parallel design are different from other methods of clustering categorical data, including, MMR and k-modes, regarding some relevant points. First, PMR-Transitive is a free parameter method, which determines automatically the number of clusters. Second, the proposed method operates on the entire data set and does not use any kind of data sampling or any data preprocessing.

Performance results

In this section, we evaluate the performance of the proposed method. For this purpose, we have generated synthetic data sets, which their sizes vary from 1 to 10 MB.

Figure 4 presents a comparison of the end-to-end running time of the Transitive heuristic and its proposed parallel design. The PMR-Transitive method achieved a speedup up to 8× over the original sequential method on large data sets.

As described in "MapReduce-based implementation of transitive", PMR-Transitive is designed with multiple parallel Map tasks and a single Reduce task that may appear as a bottleneck. However, Fig. 5 shows that more the size of the data set is large, more the time consumed by the Reduce task decreases.

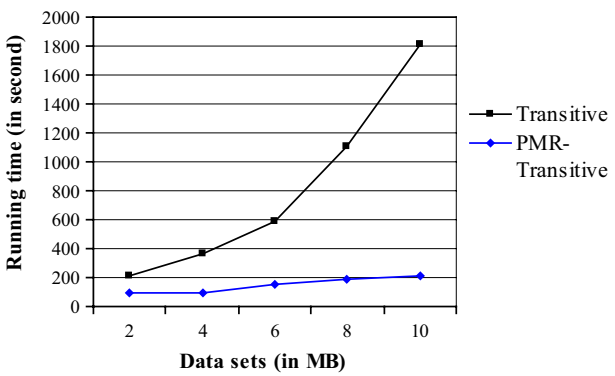

Fig. 4 Running time comparison between the proposed method and the Transitive heuristic

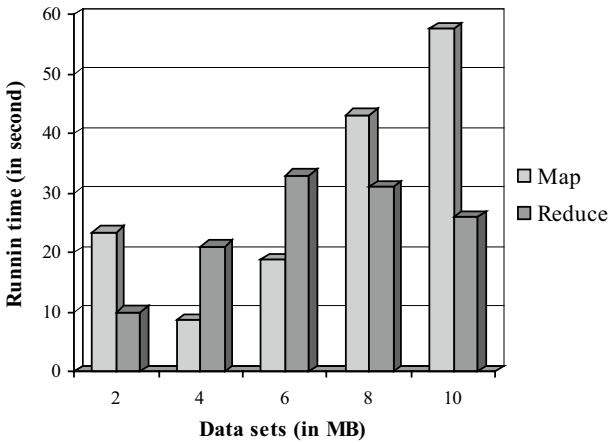

Fig. 5 Running times of the Map and the Reduce phases for different synthetic data sets

Conclusions

In this paper, we presented a new parallel clustering design of a recently appeared method, named Transitive heuristic. The proposed method processes large categorical data based on the relational analysis approach. We described in detail Transitive heuristic and its new parallel design, named PMR-Transitive, and then we discussed the adjustments that were brought to the original heuristic in order to adapt it to the MapReduce framework. Finally, we evaluated the quality of the clustering results produced while using real-life data sets. The results demonstrate that both methods, transitive and PMR-Transitive, are efficient and produce clusters of good quality.

In our future work, we plan to extend the PMR-Transitive heuristic to manage multiple data types and to treat outliers while keeping its characteristics and benefits. We also hope to test this method on a multi-nodes cluster environment in order to evaluate its performance with respect to Big Data.

Authors' contributions
SCS first proposed the design of the original serial method, which is called *Transitive*. YL carried out its implementation and evaluation. Additionally, she suggested the parallel implementation of the *Transitive* heuristic as a single-pass method using the MapReduce model. Both authors read and approved the manuscript.

Acknowledgements
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Availability of data and materials
The real-life data sets supporting the conclusions of this article are available in [UCI Machine Learning Repository. http://archive.ics.uci.edu/ml].

The synthetic data sets supporting the conclusions of this article were produced using the generator available in [The datgen Dataset Generator. http://www.datasetgenerator.com]

Consent for publication
Not applicable.

Ethics approval and consent to participate
Not applicable.

Funding
Not applicable.

Publisher's Note

Received: 10 August 2017 Accepted: 18 September 2017
Published online: 22 September 2017

References

1. Dean J, Ghemawat S. Mapreduce: simplified data processing on large clusters. In: Proceedings of the 6th conference on symposium on operating systems design and implementation: 06–08 December 2004; San Francisco. Berkeley: USENIX Association. 2004. p. 137–50.
2. Berkhin P. A survey of clustering data mining techniques. In: Kogan J, Nicholas C, Teboulle M, editors. Grouping multidimensional data. Heidelberg: Springer; 2006. p. 25–71.
3. Slaoui SC, Lamari Y. Clustering of large data based on the relational analysis. In: Proceedings of the international conference on intelligent systems and computer vision. 25–26 March 2015; Fez. Washington: IEEE Computer Society. 2015. p. 1–7.
4. Michaud P, Marcotorchino JF. Modèles d'optimisation en analyse des données relationnelles. Math Sci Hum. 1979;67:7–38.
5. Zhao W, Ma H, He Q. Parallel k-means clustering based on mapreduce. In: Proceedings of the first international conference on Cloud Computing. 1–4 December 2009; Beijing. Heidelberg: Springer-Verlag. 2009. p. 674–79.
6. He Y, Tan H, Luo W, Mao H, Ma D, Feng S, Fan J. Mr-dbscan: an efficient parallel density-based clustering algorithm using mapreduce. In: Proceedings of the 17th international conference on parallel and distributed systems: 7–9 December 2011; Tainan. Washington: IEEE Computer Society. 2011. p. 473–80.

7. Kim Y, Shim K, Kim M-S, Lee JS. Dbcure-MR: an efficient density-based clustering algorithm for large data using mapreduce. Inf Syst. 2014;42:15–35.
8. Parmar D, Wu T, Blackhurst J. MMR: an algorithm for clustering categorical data using rough set theory. Data Knowl Eng. 2007;63(3):879–93.
9. Bai L, Liang J. The k-modes type clustering plus between-cluster information for categorical data. Neurocomputing. 2014;133:111–21.
10. Ah-Pine J, Marcotorchino JF. Overview of the relational analysis approach in data-mining and multi-criteria decision making. In: Usmani ZUH, editor. Web intelligence and intelligent agents. Rijeka: InTech; 2010. p. 325–46.
11. Michaud P. Condorcet—a man of the avant-garde. Appl Stoch Models Data Anal. 1987;3(3):173–89.
12. Manning CD, Raghavan P, Schütze H. Evaluation in information retrieval. In: Introduction to information retrieval. New York: Cambridge University Press. 2008. p. 151–75.
13. Hubert L, Arabie P. Comparing partitions. J Classif. 1985;2(1):193–218.
14. UCI Machine Learning Repository: data sets. https://archive.ics.uci.edu/ml/datasets.html.
15. Guha S, Rastogi R, Shim K. Rock: a robust clustering algorithm for categorical attributes. Inf Syst. 2000;25(5):345–66.
16. He Z, Xu X, Deng S. Squeezer: an efficient algorithm for clustering categorical data. J Comput Sci Technol. 2002;17(5):611–24.