Neurocomputing 127 (2014) 30–42

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Evolutionary *k*-means for distributed data sets

M.C. Naldi [a,*], R.J.G.B. Campello [b]

[a] Federal University of Viçosa - UFV. Rodovia BR 354 - km 310. Caixa Postal: 22 - CEP: 38.810-000 Rio Paranaíba, MG, Brazil
[b] Institute of Mathematics and Computer Sciences, University of São Paulo - USP. Av. Trabalhador São-Carlense, 400 - Centro. Caixa Postal: 668 - CEP: 13560-970 São Carlos, SP, Brazil

ARTICLE INFO

Article history:
Received 19 December 2012
Received in revised form
24 April 2013
Accepted 31 May 2013
Available online 20 August 2013

Keywords:
Distributed clustering
Evolutionary *k*-means
Distributed data mining

ABSTRACT

One of the challenges for clustering resides in dealing with data distributed in separated repositories, because most clustering techniques require the data to be centralized. One of them, *k*-means, has been elected as one of the most influential data mining algorithms for being simple, scalable and easily modifiable to a variety of contexts and application domains. Although distributed versions of *k*-means have been proposed, the algorithm is still sensitive to the selection of the initial cluster prototypes and requires the number of clusters to be specified in advance. In this paper, we propose the use of evolutionary algorithms to overcome the *k*-means limitations and, at the same time, to deal with distributed data. Two different distribution approaches are adopted: the first obtains a final model identical to the centralized version of the clustering algorithm; the second generates and selects clusters for each distributed data subset and combines them afterwards. The algorithms are compared experimentally from two perspectives: the theoretical one, through asymptotic complexity analyses; and the experimental one, through a comparative evaluation of results obtained from a collection of experiments and statistical tests. The obtained results indicate which variant is more adequate for each application scenario.

1. Introduction

The amount of data produced has grown substantially over the years. Collections of documents, images, bioinformatics and others types of data are created and increased by new technologies. There is a trend and growing need to distribute large data sets in separate repositories, known as *data sites*. In many cases, the data are naturally distributed or generated and stored in different data sites. On these grounds, computational techniques must be able to extract relevant information from distributed data with good computational performance and scalability [59,8]. However, most data mining techniques, when proposed, consider the data being centralized. The centralization of a large distributed data set implies high transference and storage costs; thus it greatly increases the overall time of the mining process. In most cases, this option is not feasible due to computational limitations related to the working memory capacity or time availability for centralized (rather than distributed) processing.

Data clustering is a fundamental conceptual problem in data mining, in which one aims at determining a finite set of categories to describe a data set according to similarities among its objects [31]. This problem has broad applicability in areas that range from image and market segmentation to document categorization, bioinformatics and distributed computing (e.g., see [32,58,8]), just to mention a few.

Many clustering algorithms have been proposed in the literature [32,58]. Among them, the *k*-means methods have been investigated for more than half a century [51]. Recently, *k*-means has been elected as one of the top 10 most influential data mining algorithms, for being simple, scalable and easy to adapt to different application domains [57]. However, *k*-means is sensitive to the selection of the initial cluster prototypes, as it may converge to suboptimal solutions if the initial prototypes are not properly chosen [32]. In addition, it requires the number of clusters, *k*, to be specified in advance. This can be quite restrictive in practice, since the number of clusters in a data set is generally unknown, especially in real-world applications involving high dimensional and/or distributed data. For such applications, *k*-means may be executed multiple times for incremental values of *k* (typically starting from two) and the best partition obtained from all executions, according to a specific cluster validity criterion, is selected as the clustering solution [18,46]. Although this ordered repetitive procedure may be efficient if the optimal number of clusters is "small", this may not be the case. In addition, even if this is the case, one hardly knows about it in advance.

An alternative repetitive approach is executing *k*-means multiple times with the number of clusters randomly drawn from a user-defined interval. In this case, the problem is twofold: On one hand,

* Corresponding author. Tel.: +55 3438559350.
E-mail addresses: murilocn@ufv.br (M.C. Naldi), campello@icmc.usp.br (R.J.G.B. Campello).

0925-2312/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.neucom.2013.05.046

if the optimal number of clusters is not within this interval, then the corresponding solution will never be discovered. On the other hand, if the interval is too much overestimated to be likely to comprise the optimal value, then the worst-case (and even the average-case) execution time may be prohibitive. This scenario may become worse if the number of clusters to be drawn require replacement due to the usual sensitivity of the k-means algorithm to the initial position of prototypes.

A number of approximation algorithms have been investigated in the literature in an attempt to circumvent the above-mentioned limitations of the systematic approaches based on multiple executions of k-means. This includes the hybridization of k-means with some sort of general purpose meta-heuristic adapted to the clustering problem [48]. Evolutionary algorithms are meta-heuristics widely believed to be able to provide satisfactory suboptimal solutions to NP-hard problems in an acceptable time. From a combinatorial optimization perspective, clustering problems can be formally classified as NP-hard [12]. Probably for this reason, several evolutionary approaches for clustering problems have been proposed in the literature (e.g., see the monograph by [12] and the survey by [27] for extensive overviews). Of special interest here are those approaches based on the use the k-means as a local search operator to refine the global search performed by the evolutionary procedure. For instance, [15,49,35,4,34,36,37,50] adopted k-means for fine-tuning of partitions produced by evolutionary operators designed to work with a fixed (user-defined) number of clusters k. Just a few papers in the literature have been devoted to evolutionary-guided k-means with a variable number of clusters [27]. In particular, the Evolutionary Algorithm for Clustering (EAC) proposed by [29] was mainly designed to evolve partitions with variable k by eliminating, splitting, and merging clusters that are systematically refined by the k-means algorithm. The use of guided mutation operators with self-adjusting application rates, among other additional features, improved the computational efficiency of EAC considerably [1]. The incorporation of those features gave rise to the Fast Evolutionary Algorithm for Clustering (F-EAC) [1], which showed (by means of extensive experiments and statistical tests) to be significantly more efficient than systematic approaches based on multiple executions of the k-means algorithm when the number of clusters in a data set is unknown [5,41]. Similar results were obtained when F-EAC was compared with other approximation algorithms [42]. Additionally, variants of the F-EAC were successfully developed for fuzzy clustering and relational data [5,25].

The present paper gives evidence that evolutionary algorithms for clustering can be successfully applied to distributed data. In particular, the distributed version of the F-EAC, here called Distributed Fast Evolutionary Algorithm for Clustering (DF-EAC), is proposed in this paper in order to obtain the exact results as the algorithm's centralized version. Additionally, the DF-EAC will be compared to a category of algorithms based on the generation and selection of k-means clustering in each data site and, after that, the combination of the obtained clusters into a single clustering solution that represents the whole data set. Algorithms of this category are called Combinations of Distributed Clustering (CDC) and were proposed in our preliminary work [40]. The algorithms are compared from two perspectives: the theoretical one, through asymptotic complexity analyses; and the experimental one, through a comparative evaluation of results obtained from a collection of experiments and statistical tests.

The remainder of this paper is organized as follows. In Section 2, a brief description of the area within which this work falls is provided. Then, in Section 3, the DF-EAC algorithm is presented, followed by a description of how it is distributed and its complexity analysis. The CDC algorithms are described in Section 4. In Section 5, DF-EAC is experimentally compared to the CDC algorithms, in order to determine which algorithms are most appropriate for each application scenario. Finally, the conclusions are addressed in Section 6.

2. Distributed data mining and clustering

According to Zaki [60], Distributed Data Mining (DDM) are techniques that involve discovering patterns or generating models from distributed data for which centralization is not feasible nor desirable. In order to solve this problem, different algorithms or different parts of one algorithm are usually applied to distributed subsets of the data and, later, the results are combined into a final solution [16]. The DDM algorithms can be categorized as exact or approximate [19]. On the one hand, exact algorithms produce a final model identical to a hypothetical model generated by a centralized algorithm having access to the full data set. On the other hand, approximate algorithms produce a model that approximates a centralized model, usually with less data transmission or computational savings.

In order to meet the increasing need for distributed computational techniques with good performance and scalability, distributed versions of classical clustering algorithms have been proposed. One of the most cited distributed versions of the k-means algorithm was proposed by Dhillon and Modha [10], later improved by Tian et al. [52] and adapted to peer-to-peer networks by Bandyopadhyay et al. [3] and Datta et al. [6]. Forman and Zhang [14] proposed a technique to parallelize algorithms based on centroids, which includes not only the algorithm k-means, but algorithms like the Expectation Maximization [39] and BIRCH [61]. Other papers proposed the distribution of hierarchical clustering algorithms, with the main objective to divide the calculation of data dissimilarity among different processing units [44,11]. Garg et al. [17] proposed a parallel version of the BIRCH algorithm that balances the computational load among processors in a cyclic manner. Like the k-means, hierarchical algorithms were also adapted to peer-to-peer networks [19]. However, distributed versions of traditional clustering algorithms inherit the limitations of their original versions, such as elevated computational costs or inability to determine the best number of clusters in a data set.

Data privacy preservation may be a concern during distributed clustering applications, especially if one or more distributed parts of the algorithm require information about data objects that they may not access. In this case, data transformation methods [43,53] may be applied to the distributed data set before and after the clustering algorithm. Such application is recommended for clustering algorithms that consider the data objects as points in an Euclidean space, which is the case of the algorithms compared in this work.

3. Distributed fast evolutionary algorithm for clustering (DF-EAC)

Essentially, DF-EAC is the exact distributed version of the F-EAC algorithm [1,41]. A description of the F-EAC algorithm is presented in Algorithm 1. Both were designed to efficiently evolve hard data partitions (from now on referenced as partitions for short) obtained from the k-means algorithm. A partition of a data set $X=\{\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_n\}$, composed of a-dimensional feature or attribute vectors $\mathbf{x}_j$, is a collection $C=\{C_1,C_2,\ldots,C_k\}$ of k non-overlapping data subsets C_i (clusters) such that $C_1\cup C_2\cup\cdots\cup C_k=X$, $C_i\neq\varnothing$ and $C_i\cap C_l=\varnothing$ for $i\neq l$ and $i,l\in[1..k]$. In F-EAC and DF-EAC, partitions are represented as *genotypes* (see Section 3.2) and evolved through mutation and selection operators. These operators use probabilistic rules to process partitions sampled from the search space. Roughly speaking, better partitions have higher probabilities

of being sampled. In other words, the evolutionary search is biased towards more promising clustering solutions. To determine the relative quality of each partition, here called *fitness*, a relative validation index [31,55] is used to assess the appropriateness of each obtained partition. When used for this purpose, the relative index is considered a *fitness function* $f(\cdot)$ (Section 3.5).

Algorithm 1. Fast Evolutionary Algorithm for Clustering (F-EAC).

Let k_{max} be the maximum initial number of clusters, $\mathbf{S_{C1}}$ be the stopping criterion for k-means, $\mathbf{S_{C2}}$ be the F-EAC stopping criterion, v_{VC} be the best fitness value for the current generation and $\mathbf{g}_S$ be the corresponding (fittest) genotype, k^* be the number of clusters in $\mathbf{g}_S$, g be the generation counter, P_g be the current population, $|P|$ be the population size. Then, F-EAC can be summarized as follows:

Require k_{max}, $\mathbf{S_{C1}}$, $\mathbf{S_{C2}}$ and $|P|$;
1: $g \leftarrow 1$;
2: initialize a population P_g of $|P|$ genotypes encoding $k \in \{2, \ldots, k_{max}\}$ random clusters each;
3: **Repeat**
4: apply the k-means algorithm to each genotype in P_g until $\mathbf{S_{C1}}$ is met;
5: evaluate each genotype in P_g according to the fitness function;
6: compute v_{VC}, store its corresponding genotype $\mathbf{g}_S$;
7: $k^* \leftarrow$ number of clusters of the partition encoded into $\mathbf{g}_S$;
8: **if** $\mathbf{S_{C2}}$ is not met **then**
9: apply elitist strategy;
10: select genotypes from P_g;
11: calculate which mutation operator will be applied to each genotype;
12: **for all** selected genotypes **do**
13: select the clusters to be mutated;
14: apply mutation operators on the selected clusters to create new genotypes;
15: **end for**;
16: copy the new genotypes to the next population P_{g+1};
17: $g \leftarrow g+1$;
18: **end if**;
19: **until** $\mathbf{S_{C2}}$ is met;
20: **return** k^* and the corresponding genotype $\mathbf{g}_S$;

DF-EAC requires the number of genotypes $|P|$ to be processed during each iteration, which is known as *population size*. During the initialization step (equivalent to Step 2 of Algorithm 1), the genotypes are initialized by randomly drawing both the number of clusters (k) and the prototypes for each partition, as described in Section 3.3. This procedure favors diversity in the initial population by creating initial partitions that represent different numbers of clusters. Thus, the algorithm requires a maximum *initial* number of clusters k_{max}. It is important to note that F-EAC is robust to the choice of values for k_{max} since its evolutionary search operators are able to properly increase or decrease the number of clusters of partitions during execution [1,41,42]. DF-EAC holds these properties, as the algorithm is the exact implementation of F-EAC.

For each DF-EAC iteration (also known as *generation*), every partition is fine-tuned with the k-means algorithm (equivalent to Step 4 of Algorithm 1), as detailed in Section 3.4. Thus, DF-EAC requires at least one stopping criterion ($\mathbf{S_{C1}}$) for k-means. In this work, two stopping criteria are used as $\mathbf{S_{C1}}$: convergence or a maximum number of iterations, t. Convergence is attained when no significant difference is observed between the values of the centroids in two consecutive iterations. Further details are discussed in Section 5.2.

The difference between F-EAC and DF-EAC lies in the method of how the steps in the Algorithm 1 are implemented. For example, DF-EAC is distributed among processing *nodes* (to be described in Section 3.1), thus its codification and data structures are also distributed (as presented in Section 3.2). During the population initialization (Step 2 of Algorithm 1), DF-EAC must synchronize information about the genotypes (detailed in Section 3.3), which is not necessary in F-EAC. Additionally, DF-EAC must transmit information about the distributed data to apply k-means (Step 4), calculate the fitness of the partitions (Step 5), and apply the evolutionary operators (Steps 11–14). These steps are described in Sections 3.4–3.6, respectively.

Not all DF-EAC steps need information from the data set. This is the case in Step 9, in which an elitist strategy is adopted, copying the genotype with the highest fitness into the next generation. The remaining genotypes are selected for mutation according to proportional selection [38] (e.g., the well-known roulette wheel strategy) in Step 10. Optionally, a deterministic selection procedure can also be adopted [1]. In particular, experimental results suggest that the use of the $(\mu+\lambda)$ strategy [13] produces similar results. For further details on the F-EAC, refer to Alves et al. [1], Naldi et al. [41].

Last but not least, DF-EAC requires the stopping criteria $\mathbf{S_{C2}}$ for itself (equivalent to Step 19 of the Algorithm 1). For experimental scenarios, where the algorithm is compared to other clustering algorithms, a common practice consists of establishing a reference value v_R for the fitness of the genotypes [41,42]. In these scenarios, the algorithm stops if the fitness values of one or more genotypes are equal to or greater than v_R. For other scenarios, such as practical applications, other stopping criteria may be adopted (e.g., imposing a maximum number of generations and/or a minimum threshold for the population diversity, among others [12]). Once stopped, the algorithm returns the best evaluated partition $\mathbf{g}_S$ and its number of clusters k^*.

3.1. Distributed version of the algorithm

DF-EAC is distributed in processing nodes, which can be divided into two types:

- *Data node*: a data node is responsible for all processing that require direct access to a particular subset of the data. Each data node is responsible for a single subset of the data, i.e., no subset of the data can be shared by two or more data nodes. Data nodes can receive/transmit information from/to other data nodes or the master node.
- *Master node*: the master node organizes and processes the information from the data nodes. Its final goal consists of combining this information into a single clustering solution. Additionally, the master node executes all DF-EAC steps that do not require direct access to the data (equivalent to Steps 9, 10, 11 and 13 of Algorithm 1) and synchronizes the data nodes during the other steps.

It is important to stress the difference between a DF-EAC node and a data site. A DF-EAC node is a distributed fraction of the algorithm while a data site is a data repository. Thus, it is possible to process multiple DF-EAC nodes at the same data site if this data site stores multiple data subsets. A typical (fully connected) DF-EAC topology is presented in Fig. 1, where the arrows indicate information being transmitted between nodes. In this topology, each data site stores one of the s data subsets, which is processed by one data node and all the data nodes are coordinated by a master node. For the sake of simplicity, this topology is assumed

in the remainder of this paper. However, other topologies are possible. The only prerequisite is that all data nodes must be able to send and receive information to/from the master node, either directly or through intermediate data nodes.

3.2. Encoding scheme and data structures

As stated in Section 3.1, the DF-EAC data and master nodes have different purposes. Thus, they need different encoding schemes and data structures. In this work, the data structures are divided as following:

- *Local data structures*: located in the data nodes, these data structures store important information about the node's data subset.
- *Global data structures*: located in the master node, this data structure represents the partition of the whole data set.

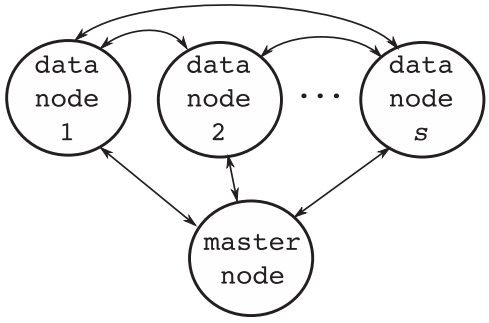

Fig. 1. DF-EAC full connected topology.

DF-EAC assumes that the data set is distributed across mutually exclusive subsets, to be processed by different data nodes. Data distribution and data partition are two different concepts: the former refers to the way the data is distributed among data nodes and the latter is the final result of the clustering process. In order to illustrate those concepts, consider a data set $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{10}\}$ in a distributed scenario with three data sites responsible for processing the data subsets: $\{\mathbf{x}_1, \mathbf{x}_2\}$, $\{\mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6\}$, and $\{\mathbf{x}_7, \mathbf{x}_8, \mathbf{x}_9, \mathbf{x}_{10}\}$. Fig. 2 illustrates the local and global data structures for this data set, when partitioned into the clusters $C_1 = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$, $C_2 = \{\mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7\}$ and $C_3 = \{\mathbf{x}_8, \mathbf{x}_9, \mathbf{x}_{10}\}$. The details will be discussed in the following.

Originally, the F-EAC has a label-based integer encoding to represent genotypes. More specifically, a genotype is an integer vector of n positions, where n is the number of data set objects. Each position corresponds to a particular object, i.e., the ith position (*gene*) represents the ith data set object. Provided that a genotype represents a partition formed by k clusters, each gene has a value over the alphabet $1, 2, 3, \ldots, k$. For example, the genotype $\mathbf{g} = [1112222333]$ represents the partition exemplified in Fig. 2, where label 1 is associated with cluster C_1 (objects $\mathbf{x}_1, \mathbf{x}_2$ and $\mathbf{x}_3$), label 2 is associated with cluster C_2 (objects $\mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6$ and $\mathbf{x}_7$) and label 3 is associated with cluster C_3 (objects $\mathbf{x}_8, \mathbf{x}_9$ and $\mathbf{x}_{10}$). In the DF-EAC algorithm, a genotype $\mathbf{g}$ is stored in the master node and consists of the concatenation of vectors $\mathbf{g}[j]$ for $j = 1, \ldots, s$, where $\mathbf{g}[j]$ contains the labels of the objects inside the jth data node. Besides the genotypes, DF-EAC stores additional information about the partitions in auxiliary data structures and variables in order to accomplish computational savings.

The first auxiliary variable is k, which stores the number of clusters. A copy of k is present at each DF-EAC node, since its content is frequently required and has a low storage cost. Vector $\mathbf{n}_C$ represents the total number of objects each cluster contains

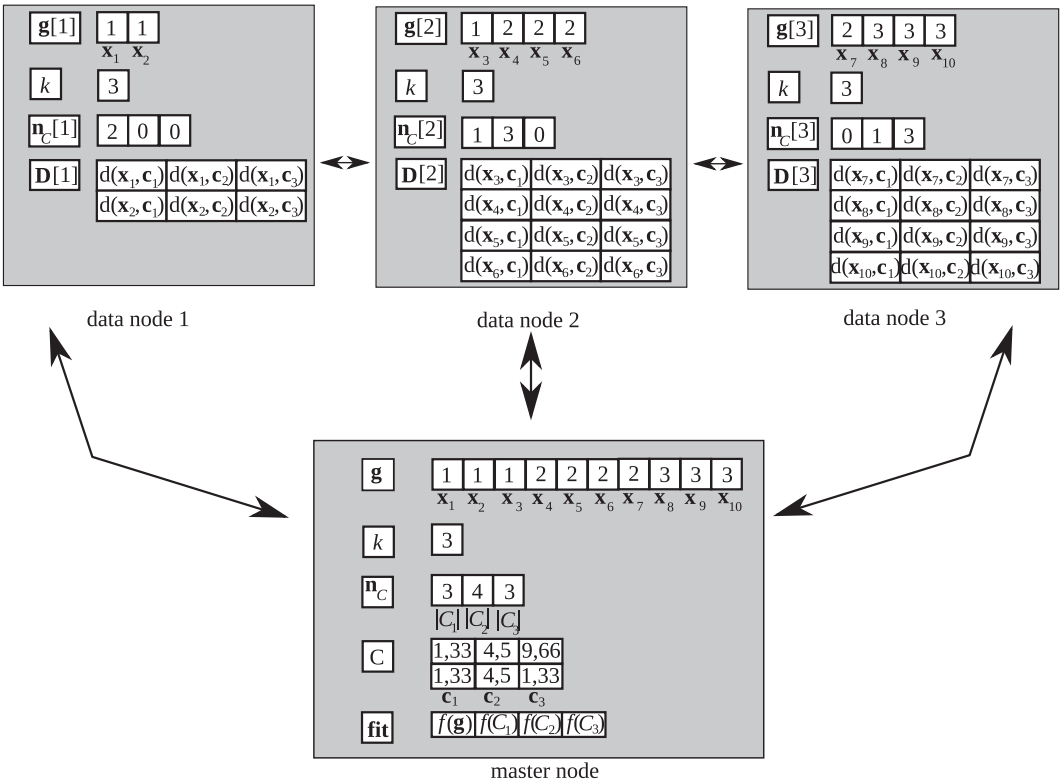

Fig. 2. Example of data structures for a genotype $\mathbf{g}$ in a DF-EAC population: $\mathbf{g}$ stores the genotype, $\mathbf{g}[j]$ stores the labels of the objects in the jth data node, k stores the number of cluster, $\mathbf{n}_C$ stores the number of objects in each cluster, $\mathbf{n}_C[j]$ stores the number of objects in the jth data node for each cluster, $\mathbf{C}$ stores the centroids of the clusters, **fit** stores the fitness $f(\cdot)$ for the partition $\mathbf{g}$ and each cluster C_l, $\mathbf{D}[j]$ stores the dissimilarities $d(\mathbf{x}_i, \mathbf{c}_l)$ between the object $\mathbf{x}_i$ and the centroid $\mathbf{c}_l$ of cluster C_l.

and is stored in the master node. Similarly, $\mathbf{n}_C[j]$ stores the number of objects which are in the jth data node assigned to each cluster. Thus, the information in $\mathbf{n}_C$ is calculated as the sum $\sum_{j=1}^{s}\mathbf{n}_C[j]$ computed over all s data nodes.

Kept in the master node, matrix $\mathbf{C}$ stores the centroids of each cluster. This matrix is transmitted, completely or partially, to the data nodes when information about the centroids is needed. Depending on the available memory, a copy of $\mathbf{C}$ may be stored in each of the data nodes in order to achieve computational savings.

Some auxiliary structures are unique to the master node or the data nodes. As presented in Section 3, the fitness is calculated by means of a function $f(\cdot)$ for the genotype $\mathbf{g}$ and each cluster C_l, for $l=1\ldots k$. These values are stored in vector **fit**, accessed in the master node only. Differently, matrix $\mathbf{D}[j]$ stores the dissimilarities $d(\mathbf{x}_i,\mathbf{c}_l)$ between each object $\mathbf{x}_i$ in the jth data node and the centroid $\mathbf{c}_l$ of the lth cluster, for $l=1\ldots k$. These dissimilarities are directly related to the data and, for this reason, they are stored in the data nodes.

3.3. Initialization

Unlike the F-EAC initialization (Step 2 of Algorithm 1), DF-EAC demands the synchronization of the data nodes with the master node. In this step, each data node loads the corresponding data subset and informs the master node about its number of objects and attributes. Then, a population with $|P|$ genotypes and auxiliary data structures is initialized in the master node. Each genotype represents a partition with k clusters randomly drawn in the interval $\{2,\ldots,k_{max}\}$. In F-EAC, the initial cluster prototypes are usually randomly drawn among the objects of the data set [41,42]. This procedure is replicated in the DF-EAC master node and the k chosen objects are requested to their corresponding data nodes. In return, the data nodes that contain the chosen objects transmit them to the master node. During this step, the k value is transmitted and stored in every data node. There is no need to update the genotypes and the rest of the auxiliary data structures in the initialization step, since this will be made in the local search step after the master node receives the prototypes.

3.4. Local search

During each generation, DF-EAC partitions are fine tuned by the k-means algorithm (equivalent to Step 4 of Algorithm 1), in order to search for higher quality partitions. Essentially, the k-means algorithm has three main steps:

1. Calculate the dissimilarity between objects and centroids.
2. Assign each object to the cluster for which the centroid has the lowest dissimilarity to this object.
3. Update the cluster centroids.

DF-EAC uses an adapted version of the distributed k-means algorithm by Dhillon and Modha [10]. In the first step, the master node transmits the centroids (matrix $\mathbf{C}$) for all data nodes. After that, the dissimilarity between objects of each data subset and the centroids is calculated and stored in $\mathbf{D}[j]$, for the jth data node and $j=\{1,\ldots,s\}$. Based on the dissimilarities stored in $\mathbf{D}[j]$, each object is assigned to the cluster which has the centroid with the lowest dissimilarity to this object, characterizing k-means' second step. For the jth data node, this assignment is made by updating every object label in $\mathbf{g}[j]$, which allows the calculation of the number of objects in each cluster and the storage of these calculated values in $\mathbf{n}_C[j]$. In the third step, the assigned objects are summed for each cluster in every data node. Although the result of this sum is not explicitly stored as part of the auxiliary data structures, we will refer to the sum of all objects in the jth data node assigned to the lth cluster as $\mathbf{s}_{C_l}[j]=\sum_{\mathbf{x}_i[j]\in C_l}\mathbf{x}_i[j]$, where $\mathbf{x}_i[j]$ is an object of the lth cluster in the jth data node. Then, the sums $\mathbf{s}_{C_l}[j]$ are transmitted along with $\mathbf{g}[j]$ and $\mathbf{n}_C[j]$ to the master node, for $l=\{1,\ldots,k\}$ and $j=\{1,\ldots,s\}$. The master node then calculates $\mathbf{n}_C$ as follows:

$$\mathbf{n}_C=\sum_{j=1}^{s}\mathbf{n}_C[j]=[|C_1||C_2|\cdots|C_k|] \tag{1}$$

Once $\mathbf{n}_C$ is obtained, the cluster centroids are straightforwardly calculated by the following equation:

$$\mathbf{c}_l=\sum_{j=1}^{s}\frac{\mathbf{s}_{C_l}[j]}{|C_l|} \tag{2}$$

where $\mathbf{c}_l$ is the centroid of the lth cluster, $\mathbf{s}_{C_l}[j]$ is the sum of all objects in the jth data node assigned to the lth cluster and $|C_l|$ is the total number of objects assigned to the lth cluster. An overview of the distributed local search procedure described in this section is presented in Fig. 3.

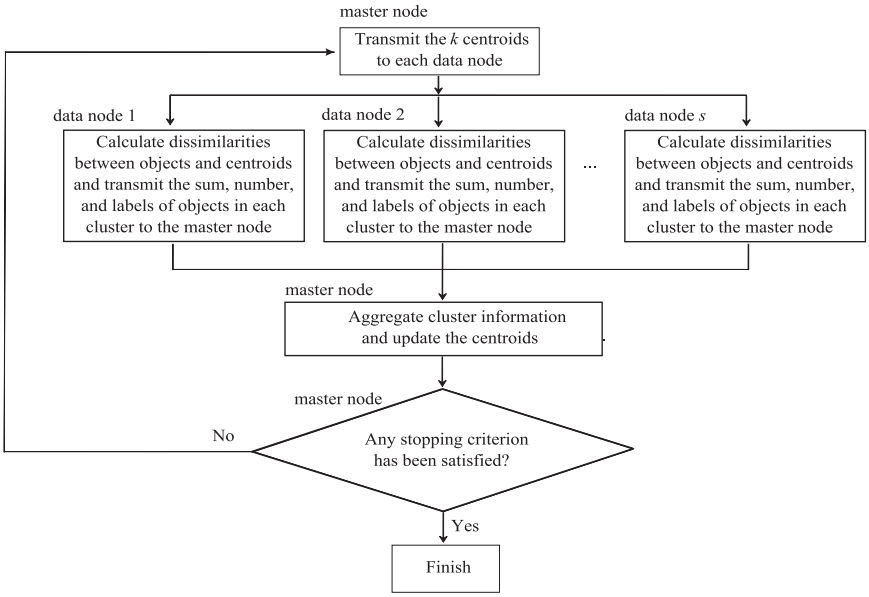

Fig. 3. Distributed local search overview.

3.5. Fitness calculation

The fitness function (equivalent to the Step 5 of Algorithm 1) must be able to assess the quality of partitions and individual clusters. One such a function, proposed by Hruschka et al. [28,29], is a simplified (computationally much more efficient) version of the well-known relative validation index Silhouette Width Criterion [33]. The Simplified Silhouette index scored the best among 40 indexes in a recent comparative study [54,55]. It measures how compact and separate the clusters of a given partition are. In this sense, there is a synergy between this index and the k-means algorithm, as both seek the same kind of partition. In order to explain this index, let us consider an object $\mathbf{x}_i$ belonging to cluster C_l. The dissimilarity between $\mathbf{x}_i$ and the centroid $\mathbf{c}_l$ of C_l is denoted by $a(\mathbf{x}_i)$, whereas the dissimilarity between $\mathbf{x}_i$ and the centroid $\mathbf{c}_m$ of another cluster C_m is termed $d(\mathbf{x}_i, \mathbf{c}_m)$. After computing $d(\mathbf{x}_i, \mathbf{c}_m)$ for all clusters $C_m \neq C_l$, the lowest one is retained and termed $b(\mathbf{x}_i)$, i.e., $b(\mathbf{x}_i) = \min_{m \neq l} d(\mathbf{x}_i, \mathbf{c}_m)$. This value represents the dissimilarity between $\mathbf{x}_j$ and its nearest neighboring cluster. Once $a(\mathbf{x}_j)$ and $b(\mathbf{x}_i)$ have been introduced, the Simplified Silhouette $s(\mathbf{x}_i)$ can thus be defined as

$$s(\mathbf{x}_i) = \frac{b(\mathbf{x}_i) - a(\mathbf{x}_i)}{\max\{a(\mathbf{x}_i), b(\mathbf{x}_i)\}} \tag{3}$$

It is easy to verify that $0 \leq s(\mathbf{x}_i) \leq 1$ if the objects are always assigned to the cluster with the closest prototype. The higher $s(\mathbf{x}_i)$ the better the assignment of object $\mathbf{x}_i$ to a given cluster. If $s(\mathbf{x}_i)$ is equal to zero, then it is not clear whether the object should have been assigned to its current cluster or to a neighboring one. Finally, if cluster C_l is a singleton, then $s(\mathbf{x}_i)$ is not defined and the most neutral choice is to set $s(\mathbf{x}_i) = 0$ [33].

The Simplified Silhouette of a given collection of data objects is defined as the arithmetic mean of the individual values $s(\mathbf{x}_i)$ computed for each object. This way, it is possible to compute the index either for individual clusters (C_l) or for an entire data partition (C) by calculating the mean of $s(\mathbf{x}_i)$ over the corresponding objects, as follows:

$$f(C_l) = \sum_{\mathbf{x}_i \in \mathbf{C}_l} \frac{s(\mathbf{x}_i)}{|C_l|} \tag{4}$$

$$f(C) = \sum_{i=1}^{n} \frac{s(\mathbf{x}_i)}{n} \tag{5}$$

where $|\cdot|$ stands for the cardinality of a set.

In order to calculate the fitness of the DF-EAC partitions and clusters, the master node transmits the centroids to every data node. Having the centroids updated, the data nodes calculate the simplified silhouette value for each object of the data subsets, by using Eq. (3). After that, the simplified silhouette of the objects are summed for each cluster and transmitted to the master node, i.e., a vector $\mathbf{s}_{fit}[j] = [\mathbf{s}_{fit_1}[j]\ \mathbf{s}_{fit_2}[j]\ \cdots\ \mathbf{s}_{fit_k}[j]]$ is calculated and transmitted, where $\mathbf{s}_{fit_l}[j]$ is the sum of the simplified silhouette values of the objects $\mathbf{x}_i[j]$ in the jth data node belonging to the lth cluster:

$$\mathbf{s}_{fit_l}[j] = \sum_{\mathbf{x}_i[j] \in C_l} s(\mathbf{x}_i[j]). \tag{6}$$

Being $\sum_{j=1}^{s} \mathbf{s}_{fit_l}[j] = \sum_{\mathbf{x}_i \in \mathbf{c}_l} s(\mathbf{x}_i)$ and $\sum_{j=1}^{s} \sum_{l=1}^{k} \mathbf{s}_{fit_l}[j] = \sum_{i=1}^{n} s(\mathbf{x}_i)$, the master node calculates the simplified silhouette value for every partition and cluster[1] by Eqs. (4) and (5).

3.6. Mutation operators

DF-EAC has two mutation operators. The first mutation operator (MO_1) eliminates one or more selected clusters, assigning each of their objects to the cluster for which the centroid is the least dissimilar to this object. MO_1 can only be applied to genotypes that encode more than two clusters and at least two clusters must remain after the application. The second mutation operator (MO_2) is only applied to clusters formed by at least two objects. It splits one or more selected clusters into two new clusters each. To do so, an object from the selected cluster is randomly chosen. This object will then be used as seed to generate the new cluster, whereas the farthest object from the randomly chosen object will also be used as seed to generate another cluster. Each of the remaining objects of the split cluster will be assigned to the new generated cluster for which the seed is the least dissimilar to this object.

Both DF-EAC operators adopt a (probabilistic) informed search strategy for mutation purposes. In particular, it has been hypothesized that the better the cluster, the smaller should be its probability of being mutated. Thus, good clusters tend to be maintained during the evolutionary process, whereas bad clusters are more likely to be mutated, hopefully improving the encoded partition. The informed search is guided by the fitness of the clusters (calculated as described in Section 3.5) and probabilistic selection may be applied to chose the clusters to be mutated, e.g., the roulette wheel strategy [7]. In this case, a linear normalization [7] is recommended before the selection, to avoid premature convergence and balance the evolutionary pressure, as performed in the original F-EAC [1] and predecessors [28,29]. This step is equivalent to Step 13 of Algorithm 1.

Apart from the genotypes chosen by the elitist strategy (Step 9 of Algorithm 1), every genotype selected by proportional selection[2] in Step 10 of Algorithm 1 is modified by one of the mutation operators. In F-EAC, the mutation operator (to be applied) is proportional to its performance during the evolutionary search. From this perspective, if a particular mutation operator provides better results in a given generation, then it will be applied with higher probability in the next generation. One simple way of accomplishing this is to consider the performance of the operators individually for each genotype. If the use of an operator generated a genotype with fitness higher than its predecessor, this operator will be chosen to mutate the generated genotype afterwards. Otherwise, the other operator will be chosen. If the genotype belongs to the initial population or was selected by elitism, both operators have the same chance of being chosen. This method was successfully applied in [24,25] and will also be adopted in the DF-EAC at the step equivalent to Step 11 of Algorithm 1.

A computationally efficient implementation of MO_1 was proposed in [41], where the objects of the eliminated clusters remain unassigned until the next local search step (Section 3.4). This implementation does not require access to the data information and can be executed in the master node solely.[3] After that, the result of the MO_1 is transmitted from the master node to the data nodes in order to allow the update of the genotype and auxiliary data structures.

The original MO_2 needs multiple transmissions between nodes in order to select seeds for the new clusters and to reassign the objects. To reduce the number of transmissions in DF-EAC, a MO_2 variant is proposed and used here. In this variant, the master node randomly chose two objects from each of the eliminated clusters to become the seeds of new clusters. Then, the master node informs the data nodes

[1] At this point, singleton clusters have their simplified silhouette assumed to be 0 by convention [33].

[2] The well-known roulette wheel strategy [38] was applied.

[3] For this reason, the efficient implementation of the MO_1 will be adopted in this work.

which clusters were eliminated and requests for the objects chosen to be seeds. After the seeds are received, they become cluster prototypes in the master node and the rest of the objects of the eliminated clusters remain unassigned until the next local search step (Section 3.4). When compared with the original MO_2 operator, the proposed variant showed results of equivalent quality and presented a significant reduction in data transfer.

3.7. DF-EAC complexity analysis

In this work, the complexity analysis of the DF-EAC algorithm is threefold: it considers the computational complexity of the algorithm, the storage cost of its data structures, and the amount of data transmitted between nodes.

As far as the computational complexity is concerned, the application of k-means to each genotype represents the main computational burden of DF-EAC. The k-means algorithm computational complexity is assessed as $O(t \cdot \hat{k}_{max} \cdot n)$, where n is the number of objects in the distributed data set, $\hat{k}_{max}$ is the highest number of clusters ever encoded by a genotype during the evolutionary search and t is the maximum number of k-means iterations (see Section 3). Considering that the k-means algorithm is applied to every genotype over all generations, the overall computational cost of DF-EAC is (over) estimated as $O(g_{max} \cdot |P| \cdot t \cdot \hat{k}_{max} \cdot n)$ for g_{max} generations and population size $|P|$. However, if the distributed parts of the algorithm are executed in parallel, this computational cost is proportionally divided by the number of objects in each data node. Assuming n_{max} as the size of the larger data subset, the DF-EAC has computational complexity estimated as $O(g_{max} \cdot |P| \cdot t \cdot \hat{k}_{max} \cdot n_{max})$ when the data nodes execute the algorithm in parallel. If the objects are evenly distributed among the data nodes, i.e., $n_{max} \approx n/s$ where s is the number of data subsets, the algorithm computational cost can be assessed as $O(g_{max} \cdot |P| \cdot t \cdot \hat{k}_{max} \cdot n/s)$.

Concerning the storage cost, the DF-EAC largest data structures are the $\mathbf{D}[j]$ matrices, which store the dissimilarity between the objects of the jth data node and the centroids. The storage cost of this data structure is estimated as $O(\hat{k}_{max} \cdot n_{max})$ in the worst case. Thus, the DF-EAC storage cost is assessed as $O(|P| \cdot \hat{k}_{max} \cdot n_{max})$, which can be considered as $O(|P| \cdot \hat{k}_{max} \cdot n/s)$ if $n_{max} \approx n/s$. Although it is possible to reduce the DF-EAC storage cost by reducing the population size $|P|$, the number of generations tends to be inversely interdependent on the population size, in such a way that decreasing one of them typically increases the other [41].

Finally, the largest amount of data transmitted between nodes is matrix $\mathbf{C}$, which stores the centroids of the clusters. This data structure is transmitted during every k-means iteration and its asymptotic complexity is calculated as $O(g_{max} \cdot |P| \cdot t \cdot \hat{k}_{max} \cdot a \cdot s)$, where a is the number of attributes of the objects.

It is important to note that the amount of data transmitted by the DF-EAC algorithm is independent of the number of objects of the data set n, which makes DF-EAC data transmission scalable with respect to this critical quantity.

4. Combinations of distributed clustering (CDC)

The Combinations of Distributed Clustering (CDC) is a category of algorithms proposed in our preliminary work [40] and aimed at combining distributed information about the data set into a final clustering solution. The algorithms are distributed in data and master nodes like the DF-EAC, as described in Section 3.1. Generally speaking, the CDC algorithms have two main steps: the first step is the independent generation of clustering models in the data nodes and the second step combines, in the master node, the models obtained in the first step.

The first step of the CDC algorithms consists of clustering the subsets of the distributed data set, i.e., generating data clusters for each of the data nodes separately. In the present work, the F-EAC (Algorithm 1) will be used in this step in order to independently generate partitions for the subsets in the data nodes. After receiving the resulting centroids and the number of objects in each cluster from every data node, the master node generates a combined partition of the data set in the second step of the CDC algorithms. Because the centroids share the same feature space (as the objects of the data set), it is possible to cluster these centroids with clustering algorithms in order to obtain a meta-partition. Let $MX = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_c\}$ be the set of centroids from all data nodes, resulting from the first step of a CDC algorithm, where c is the total number of centroids. A meta-partition is a collection $M\pi = \{MC_1, MC_2, \ldots, MC_{mk}\}$ of meta-clusters MC_i for which $MC_1 \cup MC_2 \cup \cdots \cup MC_{mk} = MX$, $MC_i \neq \emptyset$ and $MC_i \cap MC_l = \emptyset$ for $i \neq l$. The meta-partition can be converted into a global partition of the original data set, replacing each centroid by the objects it represents.

In order to obtain the global partition, it is possible to compute the centroid of a meta-cluster as the mean of the objects it represents. To do so, a meta-centroid is calculated as the mean of the centroids in the meta-cluster weighted by the number of objects that these centroids represent. Thus, the jth meta-centroid is calculated as presented in the following equation:

$$\mathbf{mc}_j = \frac{\sum_{\mathbf{c}_i \in MC_j} \mathbf{c}_i |C_i|}{\sum_{\mathbf{c}_i \in MC_j} |C_i|} \tag{7}$$

where $|C_i|$ is the cardinality of the cluster for which $\mathbf{c}_i$ is the centroid.

In this work, three methods were chosen to combine (cluster) the set of centroids MX thus giving rise to three different variants of CDC: F-EAC (Algorithm 1) and the hierarchical algorithms single-link and average-link [32]. As the hierarchical clustering is a nested sequence of hard partitions, each level of the hierarchy is evaluated with the Simplified Silhouette index (the same used as the fitness function for F-EAC) and the best level (partition) is chosen as the final solution. In CDC algorithms, the calculation of the Simplified Silhouette index is exact, i.e., it obtains the same result as the centralized version of the index. This is done by transmitting the meta-centroids from the master node to the data nodes and retrieving the Simplified Silhouette (Eq. (3)) value of each object from the data nodes. A more detailed discussion about the CDC algorithms can be found in [40].

4.1. Complexity of the CDC algorithms

Assuming that the ith data node has $n[i]$ objects, the first step of the CDC algorithms presents a computational complexity equivalent to that of F-EAC [41] summed over the multiple data nodes, i.e., $O(g_{max} \cdot |P| \cdot t \cdot \hat{k}_{max} \cdot (n[1]+n[2]+\cdots+n[s]))$, which is equivalent to $O(g_{max} \cdot |P| \cdot t \cdot \hat{k}_{max} \cdot n)$ where g_{max} is the F-EAC maximum number of generations, $|P|$ is the population size, t is the limit of k-means iterations, $\hat{k}_{max}$ is the maximum number of clusters codified by a genotype during the evolutionary search, s is the number of data nodes, n is the number of objects in the whole (distributed) data set and $n[j]$ is the number of objects in the jth data node. If executed in parallel, this complexity is reduced to $O(g_{max} \cdot |P| \cdot t \cdot \hat{k}_{max} \cdot n_{max})$, where n_{max} is the maximum number of objects in a data node. Moreover, if $n_{max} \approx n/s$, then the complexity of the CDC's first step is $O(g_{max} \cdot |P| \cdot t \cdot \hat{k}_{max} \cdot n/s)$. The second step of the CDC algorithms presents the computational complexity of the clustering algorithm adopted. For this estimation, the number of clustered objects is the total number of centroids which resulted from the first step (c). If hierarchical algorithms are adopted, the

second step has the computational complexity $O(c^2 \cdot \log\ c)$ [32]. If F-EAC is adopted, the computational cost of the second step is $O(g_{max} \cdot |P| \cdot t \cdot \hat{k}_{max} \cdot c)$ [41].

The memory allocation cost in the first step is $O(\hat{k}_{max} \cdot |P| \cdot n_{max})$. In the second step, this cost is $O(\hat{k}_{max} \cdot |P| \cdot c)$ if the F-EAC algorithm is adopted. Otherwise, if the hierarchical algorithms are used, the dissimilarities between centroids lead to a memory allocation cost of $O(c^2)$.

The amount of data transferred among the data nodes and the master node by the CDC algorithms is also related to the clustering algorithm used in their second step. To apply the Simplified Silhouette index, the meta-centroids must be transmitted to the data nodes. If the hierarchical algorithms are adopted, the total number of meta-centroids is $2+3+\cdots+k_{max} \approx k_{max}(k_{max}+1)/2$, which requires the transmissions amounts of order $O(k_{max}^2 \cdot a \cdot s)$. If F-EAC is adopted, then the data transmission is estimated as $O(g_{max} \cdot |P| \cdot \hat{k}_{max} \cdot a \cdot s)$ in the worst case. Additionally, it is important to note that the transmission cost of the algorithm does not depend on the number of objects in the data set, which makes the algorithm scalable in relation to this aspect.

5. Experiments

In [42,41], F-EAC has shown to be more computationally efficient than procedures to circumvent k-means limitations based on repetitive executions. Similar results were obtained in [5,25] for fuzzy clustering and relational data. This efficiency is caused by the evolutionary search through mutation and selection operators, which are biased towards more promising clustering solutions. Unlike the F-EAC, the repetitive procedures do not make use of the information on the quality of previously assessed partitions to generate potentially better partitions. Thus, the repetitive procedures usually result in a higher number of k-means executions than F-EAC. As presented in Section 3, DF-EAC is the exact distributed version of the F-EAC algorithm, i.e., it produces a clustering model identical to the one produced by F-EAC. Thus, the DF-EAC inherits the computational efficiency of the F-EAC and is expected to behave similarly when compared to these repetitive procedures. As far as data transmission is concerned, the distributed k-means presents the highest transmission cost for DF-EAC and procedures based on k-means repetitive executions. As these repetitive procedures usually require more k-means executions than DF-EAC, they tend to transfer a higher amount of data and, therefore, result in a worse performance. For these reasons, procedures based on k-means repetitive executions will not be experimentally compared in this work.

Differently from the DF-EAC, approximate algorithms produce a model that approximates (but may not be identical to) the type of result sought by the F-EAC algorithm, aiming at less data transmission and potential computational savings. This is the case of CDC presented in Section 4. In order to evaluate the two different types of distributed algorithms that were studied, comparative experiments between the DF-EAC and the CDC algorithms were executed, taking into account three aspects: the quality of the resulting partitions, the execution time, and the total amount of data transferred between data nodes.

The quality of the resulting partitions is measured with the well-known Corrected Rand (CR) external index [30] in relation to the known clusters or "golden truth". The index returns values in the interval $[-1, 1]$ and the higher the CR index value is, the better the quality of the evaluated partition gets.

The execution time and amount of data transmitted was measured using Matlab software in computers with quad core 3.0 GHz processors and 12 GB of RAM memory, one node per computer. For these measurements, the data nodes executed in parallel and the data were transmitted concurrently. It is important to note that the experiments presented in this section aim at comparing each of those aspects independently, as the execution and transmission time are machine dependent.

5.1. Data sets

A collection of artificial data sets analyzed in [20] was chosen for the experiments presented here. These data sets contain clusters generated from multivariate normal distributions and their main features are:

- Overlapped clusters, with higher variance (elongated) in an arbitrary direction.
- Varied numbers of clusters, k, where $k \in \{4, 10, 20, 40\}$.
- The number of objects in each cluster is randomly drawn from the interval $[50, 500]$ for data sets where $k \in \{4, 10\}$. For data sets where $k \in \{20, 40\}$, this number is drawn uniformly from the interval $[10, 100]$.
- Data objects are described by either $a=2$ or $a=10$ attributes.
- Ten sampling replications were generated for each combination of k and a, producing a total of 80 data sets variations.

Originally, the chosen data sets are centralized, i.e., they are not naturally distributed [20]. Thus, every data set had to be distributed across data subsets, one for each data node, according to the following characteristics:

- Uniform data distribution among the data nodes: the data sets were distributed into subsets of equal size, i.e., $n_{max} \approx n/s$, where n_{max} is the size of the larger data subset, n is the number of objects in the data set and s is the number of data subsets (nodes).
- Different numbers of data nodes: to assess the impact of s on the algorithms, the experiments were executed with $s =$ 5, 20 and 80 data nodes for every data set.
- Different types of cluster balancing: for the ith known cluster, the proportion of the number of objects is calculated as $p_{C_i} = |C_i|/n$, where $|C_i|$ is the cardinality of cluster C_i. The proportion of the ith known cluster in the jth data node is $p_{C_i}[j] = |C_i[j]|/n[j]$, where $C_i[j]$ is the number of objects of the ith known cluster in the jth data node and $n[j]$ is the number of objects in the jth data node. When the objects distribution is *balanced*, the proportion of objects in the data nodes is similar to the proportion in the data set, i.e., $p_{C_i} \approx p_{C_i}[j]$, $\forall i \in \{1, \ldots, k\}, j \in \{1, \ldots, s\}$. However, this balancing is not guaranteed in practical applications. For this reason, additionally to the balanced distribution, the objects were distributed with *unbalanced* proportions, i.e., $p_{C_i} \neq p_{C_i}[j]$, $\forall i \in \{1, \ldots, k\}, j \in \{1, \ldots, s\}$. Generally speaking, the method adopted to unbalance the clusters proportions consists of choosing two clusters for each data node. After that, the proportion of objects of the first chosen cluster is increased while the proportion of the second chosen cluster is decreased. This method is described in detail in the supplementary material. The main objective is to verify how much the quality of the approximate algorithms' outcome varies[4] with the type of balance adopted during the objects distribution.

[4] DF-EAC, as an exact algorithm, always produces the same results that would be produced if the data were centralized, no matter how the data are distributed across nodes.

5.2. Compared algorithms and stopping criteria

Comparisons involving computational efficiency of clustering algorithms should not neglect the quality of the partitions achieved by these algorithms. Bearing this in mind, it is reasonable to assume that fair comparisons can be achieved if the algorithms being analyzed are capable of finding partitions of equivalent quality, which must be as high as possible. Therefore, an experimental methodology developed by [5] and applied in [41] will be adapted for this work. This methodology consists of using a clustering validity index to determine whether the partitions found by different algorithms have similar qualities. In order to do so, it is necessary to set an acceptable reference value for the validity index in such a way that: (i) it is as high as possible; and (ii) the algorithms under investigation may be capable of finding it.

The algorithms compared in this work fine tune partitions with k-means. Thus, to estimate the reference value, the following repetitive procedure was applied for each data set described in Section 5.1 and each distributed subset:

1. Initialize the iterator counter $i=1$.
2. k-means is applied to the data set (or subset) 100 times with random initializations of cluster prototypes and k value as the known number of clusters.
3. The Simplified Silhouette (Section 3.5) is used to evaluate every partition resulting from Step 2 and the best value out of the 100 available values is stored in v_i.
4. Return to Step 2 and increase i by one if $i \leq 30$.
5. Calculate the reference value for this data set (or subset) as $v_R = \sum_{i=1}^{30} v_i/30$.

The purpose of these steps is to make all the compared algorithms able to find partitions of equivalent quality, from different configurations of their parameters. Thus, the reference value v_R is used here as a guide to find reasonable partitions, i.e., partitions evaluated with values equal or superior to v_R. When such partition is obtained, the algorithm stops. In this work, this stopping criterion is adopted by DF-EAC and also by the F-EAC routine used as part of the CDC algorithms. Although this stopping criterion may not have, for obvious reasons, a practical appeal, it allows the comparison of different algorithms based on partitions of reasonable quality. For practical application scenarios, other stopping criteria may be used, such as imposing a maximum number of generations and/or a minimum threshold for population diversity [12].

As the CDC algorithms are approximate algorithms, it is not possible to ensure that these algorithms are capable of resulting in partitions with evaluation values equal to or higher than v_R. Thus, the reference value cannot be assumed as the only stopping criterion for CDC. For this reason, a limit of 100 generations is adopted as an additional stopping criterion for the F-EAC routine used as part of the CDC algorithms.

DF-EAC and F-EAC initial populations are composed of partitions with the number of clusters k randomly chosen in the interval $[2, n^{1/2}]$, a commonly used rule of thumb [47,45,5]. From the results for different population sizes reported in [41], it is possible to observe that larger populations tend to make F-EAC converge to good solutions in fewer generations, which, by their turn, are more computationally costly than those related to smaller populations. These results suggest that the algorithm is reasonably robust to the choice of the population size $|P|$, especially if one considers that its effectiveness in solving the clustering problem is barely affected by this choice. Experiments in [42,26,24,41] show that $|P|=10$ is a robust blind choice across different data sets and, for this reason, will also be adopted here. When adopted for local search strategy (Section 3.4), k-means convergence is attained when no significant difference is observed between the values of the centroids in two consecutive iterations, for which a threshold of 10^{-3} is adopted in this work. Additionally, given that the evolutionary search performed by F-EAC favors a cumulative refinement of the data partitions,[5] a maximum number of iterations t is imposed to k-means. Empirical evidence suggests that $t=5$ or less repetitions ordinarily will suffice [2,41,42]. Therefore, this value is adopted here.

In the experiments to be presented in Section 5.3, each node (data or master) is executed exclusively in a single computer. Four variants of the CDC algorithms were compared:

1. CDC-sl: hierarchical single-link as the 2nd step of CDC.
2. CDC-al: hierarchical average-link as the 2nd step of CDC.
3. CDC-FEAC: F-EAC as the 2nd step of CDC.
4. CDC-FEAC (10g): similar to CDC-FEAC, but this variant has a third stopping criterion for F-EAC during the combination of clusters (CDC second step). The algorithm stops after 10 consecutive generations without improvement on the fitness value of the best partition of the population.

As addressed in Section 5.1, the data sets were distributed with different types of balance. Each of the four CDC variants was applied to the balanced and unbalanced distributed data sets. Results from CDC variants applied to the unbalanced data set will be distinguished from other results with the symbol (U) in the tables presented in Section 5.3. It is important to note that the DF-EAC results are identical to those obtained by F-EAC for centralized data. Thus, the DF-EAC results are independent of the type of balance when data are distributed and will not be applied to unbalanced distributed data sets.

5.3. Results

In order to assess the significance of the experimental results, hypothesis tests were adopted. The ANalysis Of VAriance (ANOVA) test [56] assumes that the compared samples are drawn from populations with normal distributions and similar variances [9]. As these requirements are not ensured here, we applied the (non-parametric) Friedman test [23] with 95% confidence. When the null hypothesis was rejected, indicating that there is statistical evidence to support that the compared results are different, the Bonferroni procedure [22,21] was applied (using Matlab®) to the critical values to compensate for the multiple comparisons and maintain the actual level of statistical confidence at 95%.

Table 1
Mean and standard deviation (within parentheses) values obtained by the compared algorithms when applied to the collection of data sets distributed across five data nodes.

Algorithm	CR	Time (s)	Transmission (kB)
1 CDC-sl	**0.8451 (0.1361)**	3.14 (2.07)	**255.44 (183.42)**
2 CDC-sl (U)	0.8091 (0.1929)	4.91 (7.69)	**212.05 (183.89)**
3 CDC-al	**0.8532 (0.1494)**	3.19 (2.08)	**255.44 (183.42)**
4 CDC-al (U)	0.8284 (0.1852)	4.94 (7.69)	**212.05 (183.89)**
5 CDC-FEAC	0.8326 (0.1325)	15.28 (15.60)	2055.53 (3235.86)
6 CDC-FEAC (U)	0.8149 (0.1582)	26.28 (22.06)	2882.62 (3628.17)
7 CDC-FEAC (10g)	0.8194 (0.1390)	8.05 (5.05)	865.79 (939.65)
8 CDC-FEAC (10g) (U)	0.7967 (0.1630)	11.03 (9.34)	948.26 (974.23)
9 DF-EAC	0.8392 (0.1116)	**1.69 (1.91)**	3348.15 (6149.13)

[5] Refer to [26] for a comprehensive experimental study related to this issue.

Table 2
Comparison of algorithm performances for data sets distributed into five nodes. Δ means that the algorithm in the line is faster than the algorithm in the column and ▽ means the opposite. Symbol $>$ followed by a number indicates that the data transmission rate (in kB/s) should be higher than this number for the algorithm in the line to have a better performance than the algorithm in the column, whereas symbol $<$ indicates the opposite.

	1	2	3	4	5	6	7	8	9
1	–	>24.58	Δ	>24.14	Δ	Δ	Δ	Δ	<2123.24
2	<24.58	–	<25.20	Δ	Δ	Δ	Δ	Δ	<973.33
3	▽	>25.20	–	>24.74	Δ	Δ	Δ	Δ	<2060.95
4	<24.14	▽	<24.74	–	Δ	Δ	Δ	Δ	<963.73
5	▽	▽	▽	▽	–	Δ	▽	▽	<95.07
6	▽	▽	▽	▽	▽	–	▽	▽	<18.93
7	▽	▽	▽	▽	Δ	Δ	–	Δ	<390.03
8	▽	▽	▽	▽	Δ	Δ	▽	–	<256.75
9	>2123.24	>973.33	>2060.95	>963.73	>95.07	>18.93	>390.03	>256.75	–

All compared algorithms were executed 30 times for each data set and the mean values over these executions are presented in Tables 1, 3 and 5, for data distributions in 5, 20, and 80 data nodes, respectively. The best mean value and the values without statistically significant difference with respect to the best one are highlighted in bold.

As presented in Table 1, every compared algorithm was able to find partitions of reasonable quality in experiments with five data nodes, i.e., partitions with CR index mean values above 0.7. Additionally, CR index mean values were very similar for all algorithms. The best CR mean values were obtained by CDC-al, followed by CDC-sl (with no statistically significant difference) and DF-EAC.

When computational time is regarded, DF-EAC obtained the lower mean value, followed by CDC-sl and CDC-al. On the other hand, the amount of data transmitted by DF-EAC is higher than that of any other compared algorithm, specially the CDC variants that apply hierarchical algorithms in the second step. Thus, CDC can obtain good quality partitions with some data transmission savings, but requires more computational time than DF-EAC.

Being an approximated algorithm, CDC-FEAC was not able to find partitions with evaluation values higher than or equal to the reference value v_R in all executions, which made the algorithm halt after the 100 generation limit. The adoption of the 10 generation without improvement limit (10g) caused a significant reduction in the computation time and amount of data transmitted by the algorithm, with only a small loss in the mean quality of the resulting partitions.

The unbalanced distribution caused a minor reduction in the quality of the partitions obtained by the CDC algorithms and a small increase in their computational times.

Table 1 presents comparisons between the evaluated algorithms regarding their execution time and amount of data transmitted, but not both at the same time. If a specific hardware configuration is assumed, i.e., a specific relation between computational power and capacity of data transmission among nodes is adopted, it is possible to assess which algorithm has better performance considering both aspects together. Based on the mean computational time and the mean amount of data transmitted, a performance comparison among algorithms is estimated in Table 2. Symbol Δ in position (i,j) indicates that the algorithm of the ith line is faster than the algorithm in the jth column and symbol ▽ indicates the opposite, regardless of the data transmission rate. In other cases, the result is dependent on the data transmission rate and, in these cases, symbol $>$ followed by a number x in position (i,j) indicates that the algorithm in the ith line should have a better performance than the algorithm in the jth column if the data transmission rate is higher than x. Otherwise, symbol $<$ followed by a number x indicates that the transmission rate must be lower than x if the algorithm in the ith line should have a better performance than the algorithm in the jth column. It is important to note that the performance estimation presented here considers that all computers have the same configuration of the computer described in this work and each computer must be associated with a single node (master or data). Additionally, transmissions are assumed to be executed concurrently, which is the worst application scenario.

The results presented in Table 2 obtained with data sets distributed into five nodes indicate that the CDC variants with hierarchical clustering (numbers 1–4) have similar computational performance, which are superior to the CDC variants with F-EAC (numbers 5–8). DF-EAC may outperform the CDC algorithms if the transmission rate is high enough. For instance, transmission rates over 1000 kB/s (which are common in local networks) allow DF-EAC to have a performance better than most CDC variants.

The results obtained with data sets distributed into 20 nodes are presented in Tables 3 and 4.

The comparison of the CR index values between Tables 1 and 3 reflects a reduction in the main partition quality of the CDC variants with hierarchical algorithms (numbers 1–4), when the number of data nodes increases. This quality reduction did not occur with the CDC-FEAC variants (numbers 5–8) and the DF-EAC algorithm. However, CDC-FEAC variants required the highest computational time to execute and, when applied to unbalanced data, the algorithm had the largest amount of data transmitted. The adoption of the 10 generation without improvement limitation (10g) considerably reduced both computational time and transmission costs, with minor reductions in the CR index mean values as a tradeoff. Thus, this limitation showed to be an interesting choice for applications on systems with low data transmission rates.

DF-EAC is once again the algorithm with the lowest mean computational time. Its mean CR index values are similar in Tables 1 and 3, which results from the fact that the algorithm is the exact version of F-EAC.[6]

Additionally, the results presented in Table 3 indicate that the unbalanced distribution did not jeopardize the quality of partitions for the majority of the compared algorithms.

Similar to Table 2, a performance comparison among algorithms is estimated in Table 4 for data sets distributed into 20 data nodes.

The estimation presented in Table 4 indicates that the CDC variants with hierarchical algorithms (numbers 1–4) have similar performances, which are superior to the performances of other CDC variants. However, CDC-al and CDC-sl have the lower quality partitions for data sets distributed into 20 nodes (Table 3). If the transmission rate is above 8000 kB/s, DF-EAC may outperform all the CDC variants. Even a transmission rate above 500 kB/s may suffice for DF-EAC to outperform the CDC-FEAC variants (numbers 5–6) and its mean CR index value is close to the best one presented in Table 3. For systems with low transmission rates, CDC-FEAC

[6] The CR results in Tables 1 and 3 are not exactly the same for DF-EAC because this is a randomized (non-deterministic) algorithm.

(10g) showed a good relation between performance and partition quality (Tables 3 and 4).

The results obtained with data sets distributed into 80 nodes are presented in Tables 5 and 6.

The results in Tables 1, 3 and 5 reflect a degradation of the CR index values for the CDC variants with hierarchical algorithms (numbers 1–4) when the number of data nodes increases. One of the main reasons for this degradation is the overlap between clusters in the data sets, which becomes more evident as the number of data nodes increases. The increase in the number of data nodes generated a larger amount of cluster centroids in the first step of the CDC algorithms. The larger the amount of centroids, the more information about the overlap between clusters is present in the second step of the CDC algorithms. For example, a data set used in this experiment and the centroids obtained from the first step of the CDC algorithms are presented in Fig. 4 for distributions into 5, 20 and 80 data nodes. This example illustrates how the overlap between clusters becomes more evident as the number of centroids increases. The hierarchical algorithms used by CDCs 1–4, especially single-link, do not properly handle data with overlapping clusters.

Additionally, the second step of the CDC algorithms must be able to combine clusters that, altogether, have enough evidence to characterize the underlying structure of the data set. The partitions resulting from the CDC first step have CR index mean value of 0.76 over the data sets distributed among five nodes. This value decreases to 0.74 and 0.56 when the data sets are distributed into 20 and 80 nodes, respectively, which indicates that the increase in the number of data nodes (and the consequent reduction in the number of objects per node) results in partitions in the nodes less similar to the structure of the data sets.

Unlike the hierarchical variants, the CR index values obtained from the CDC-FEAC variants (numbers 5–8) were slightly affected by the increase of the number of nodes (Tables 1, 3, and 5). Moreover, the limit of 10 generations without improvement (10g) reduced the computation time and the amount of data transmitted by the algorithm, causing only a small reduction in the mean quality of the resulting partitions.

The results presented in Table 5 indicate that the unbalanced distribution did not jeopardize the quality of any algorithm compared.

As expected from an exact algorithm, the quality of the DF-EAC partitions is robust to different types of data distributions, which is indicated by the mean CR values in Tables 1, 3, and 5. Additionally, for data sets distributed into 80 nodes, the algorithm has the best CR index values and the best mean computational time at the same time, with a transmission rate lower than the CDC variant with the best partition quality (number 6).

Analogous to Table 2, a performance comparison among algorithms is estimated in Table 6 for data sets distributed into 80 data nodes.

Due to the number of data nodes and, hence, the amount of data transmissions, algorithms with low data transmissions resulted in the best performances for data sets distributed into 80 data nodes. Exceptions may occur if the transmission rates are high, as indicated in Table 6.

6. Conclusions

From the experiments presented in this paper, it is concluded that DF-EAC is robust to different types of data distribution. This is due to the fact that the algorithm is the exact version of F-EAC, i.e., its results over distributed data are analogous to those obtained by the centralized version of the algorithm. Additionally, DF-EAC has a linear computational complexity in relation to its main parameters and resulted in the lower mean computational time for all experiments presented in this paper. However, the algorithm

Table 3
Mean and standard deviation (within parentheses) values obtained by the compared algorithms when applied to the collection of data sets distributed across 20 data nodes.

Algorithm	CR	Time (s)	Transmission (kB)
1 CDC-sl	0.6310 (0.2930)	1.66 (0.92)	**1290.41 (840.20)**
2 CDC-sl (U)	0.6408 (0.3020)	2.07 (2.16)	**1283.34 (836.64)**
3 CDC-al	0.7394 (0.2881)	2.26 (1.49)	**1290.41 (840.20)**
4 CDC-al (U)	0.7322 (0.2953)	2.18 (2.22)	**1283.34 (836.64)**
5 CDC-FEAC	**0.8321 (0.1414)**	9.76 (10.44)	7857.03 (11479.87)
6 CDC-FEAC (U)	**0.8389 (0.1550)**	14.89 (12.97)	15 154.84 (19411.24)
7 CDC-FEAC (10g)	0.8228 (0.1484)	5.27 (3.55)	3686.41 (3802.90)
8 CDC-FEAC (10g) (U)	0.8251 (0.1604)	5.57 (3.81)	4351.37 (4639.09)
9 DF-EAC	**0.8418 (0.1048)**	**0.89 (0.95)**	12 843.76 (21723.35))

Table 5
Mean and standard deviation (within parentheses) values obtained by the compared algorithms when applied to the collection of data sets distributed across 80 data nodes.

Algorithm	CR	Time (s)	Transmission (kB)
1 CDC-sl	0.2310 (0.2971)	**0.97 (0.65)**	**5127.95 (3342.77)**
2 CDC-sl (U)	0.2480 (0.3080)	**0.91 (0.61)**	**5124.81 (3338.31)**
3 CDC-al	0.5753 (0.3236)	3.76 (3.58)	**5127.95 (3342.77)**
4 CDC-al (U)	0.6196 (0.3121)	2.50 (2.05)	**5124.81 (3338.31)**
5 CDC-FEAC	0.7729 (0.2117)	18.41 (15.49)	61 525.95 (74502.38)
6 CDC-FEAC (U)	0.8123 (0.1780)	18.07 (14.31)	68 201.59 (80820.47)
7 CDC-FEAC (10g)	0.7609 (0.2177)	5.53 (3.60)	16 158.76 (16939.77)
8 CDC-FEAC (10g) (U)	0.8021 (0.1786)	5.30 (3.28)	17 386.12 (18348.24)
9 DF-EAC	**0.8403 (0.1027)**	**0.84 (0.82)**	52 939.23 (86617.73)

Table 4
Comparison of algorithm performances for data sets distributed into 20 nodes. Δ means that the algorithm in the line is faster than the algorithm in the column and ∇ means the opposite. Symbol $>$ followed by a number indicates that the data transmission rate (in kB/s) should be higher than this number for the algorithm in the line to have a better performance than the algorithm in the column, whereas symbol $<$ indicates the opposite.

	1	2	3	4	5	6	7	8	9
1	–	>17.25	Δ	>13.62	Δ	Δ	Δ	Δ	<14977.03
2	<17.25	–	Δ	Δ	Δ	Δ	Δ	Δ	<9788.75
3	∇	∇	–	∇	Δ	Δ	Δ	Δ	<8404.61
4	<13.62	∇	Δ	–	Δ	Δ	Δ	Δ	<8958.63
5	∇	∇	∇	∇	–	Δ	∇	∇	<562.20
6	∇	∇	∇	∇	∇	–	∇	∇	∇
7	∇	∇	∇	∇	Δ	Δ	–	Δ	<2092.21
8	∇	∇	∇	∇	Δ	Δ	∇	–	<1815.08
9	>14977.03	>9788.75	>8404.61	>8958.63	>562.20	Δ	>2092.21	>1815.08	–

Table 6
Comparison of algorithm performances for data sets distributed into 80 nodes. Δ means that the algorithm in the line is faster than the algorithm in the column and ▽ means the opposite. Symbol $>$ followed by a number indicates that the data transmission rate (in kB/s) should be higher than this number for the algorithm in the line to have a better performance than the algorithm in the column, whereas symbol $<$ indicates the opposite.

	1	2	3	4	5	6	7	8	9
1	–	▽	Δ	>2.05	Δ	Δ	Δ	Δ	<376315
2	Δ	–	Δ	Δ	Δ	Δ	Δ	Δ	<657872
3	▽	▽	–	▽	Δ	Δ	Δ	Δ	<16376
4	<2.05	▽	Δ	–	Δ	Δ	Δ	Δ	<28845
5	▽	▽	▽	▽	–	<19510	▽	▽	▽
6	▽	▽	▽	▽	>19510	–	▽	▽	▽
7	▽	▽	▽	▽	Δ	Δ	–	<5167	<7835
8	▽	▽	▽	▽	Δ	Δ	>5167	–	<7977
9	>376315	>657872	>16376	>28845	Δ	Δ	>7835	>7977	–

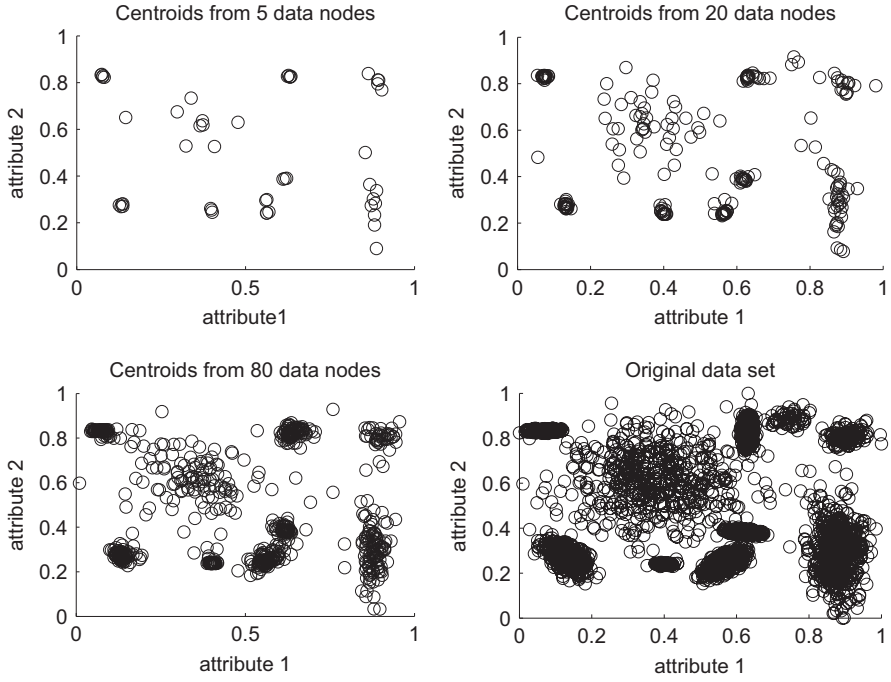

Fig. 4. Data set and centroids from the first step of the CDC algorithms for 5, 20 and 80 data nodes.

interactively transmits data between master and data nodes, which demands considerable transmission speed. Thus, the algorithm is recommended for systems with average or high transmission rates.

Although the experimental results presented in this paper indicate that the CDC variants which use hierarchical algorithms significantly reduced the amount of data transmitted, the quality of their partitions tends to decrease with the growth of the number of data nodes. The quality of the partitions of the CDC-FEAC variants for an increasing number of data nodes was not as much affected as the quality of the CDC hierarchical variants. Also, this quality was not significantly affected by unbalanced distributions with respect to the clusters known to exist in the data sets. However, a limit of generations without improvement should be adopted for the CDC-FEAC to obtain significant savings in the transmission costs. Thus, the CDC-FEAC variant with the generation limitation is recommended for systems with low transmission rates.

Acknowledgments

The authors acknowledge the Brazilian Research Agencies CNPq, FAPESP and FAPEMIG for financial support.

Appendix A. Supplementary material

Supplementary data associated with this article can be found in the online version at http://dx.doi.org/10.1016/j.neucom.2013.05.046.

References

[1] V. Alves, R. Campello, E. Hruschka, Towards a fast evolutionary algorithm for clustering, in: IEEE Congress on Evolutionary Computation, 2006, Vancouver, Canada, 2006, pp. 1776–1783.
[2] M.R. Anderberg, Cluster Analysis for Applications, Academic Press Inc, 1973.
[3] S. Bandyopadhyay, C. Giannella, U. Maulik, H. Kargupta, K. Liu, S. Datta, Clustering distributed data streams in peer-to-peer environments, Information Sciences 176 (14) (2006) 1952–1985. (Streaming Data Mining).
[4] S. Bandyopadhyay, U. Maulik, An evolutionary technique based on k-means algorithm for optimal clustering in R^n, Information Sciences 146 (1–4) (2002) 221–237.
[5] R.J.G.B. Campello, E.R. Hruschka, V.S. Alves, On the efficiency of evolutionary fuzzy clustering, Journal of Heuristics 15 (1) (2009) 43–75.
[6] S. Datta, C. Giannella, H. Kargupta, Approximate distributed k-means clustering over a peer-to-peer network, IEEE Transactions on Knowledge and Data Engineering 21 (10) (2009) 1372–1388.
[7] L. Davis, Handbook of Genetic Algorithms, International Thomson Computer Press, 1996.
[8] F. de Vega, E. Cantú-Paz, Parallel and Distributed Computational Intelligence, Studies in Computational Intelligence, vol. 269, Springer, Berlin, Heidelberg, 2010.
[9] J. Demšar, Statistical comparisons of classifiers over multiple data sets, Journal of Machine Learning Research 7 (2006) 1–30.

[10] I.S. Dhillon, D.S. Modha, A data-clustering algorithm on distributed memory multiprocessors, in: Revised Papers from Large-Scale Parallel Data Mining, Workshop on Large-Scale Parallel KDD Systems, SIGKDD, Springer-Verlag, London, UK, 2000, pp. 245–260.
[11] Z. Du, F. Lin, A novel parallelization approach for hierarchical clustering, Parallel Computing 31 (5) (2005) 523–527.
[12] E. Falkenauer, Genetic Algorithms and Grouping Problems, John Wiley & Sons, 1998.
[13] D. Fogel, Evolutionary Computation: Toward a New Philosophy of Machine Intelligence, IEEE Press, New York, USA, 1995.
[14] G. Forman, B. Zhang, Distributed data clustering can be efficient and exact, ACM SIGKDD Explorations Newsletter 2 (2) (2000) 34–38.
[15] P. Fränti, J. Kivijärvi, T. Kaukoranta, O. Nevalainen, Genetic algorithms for large scale clustering problems, The Computer Journal 40 (1997) 547–554.
[16] A.A. Freitas, S.H. Lavington, Mining Very Large Databases with Parallel Processing, Kluwer Academic Publishers, Norwell, MA, USA, 1997.
[17] A. Garg, A. Mangla, N. Gupta, V. Bhatnagar, Pbirch: a scalable parallel clustering algorithm for incremental data, in: Database Engineering and Applications Symposium, 2006, IDEAS '06, 10th International, 2006, pp. 315–316.
[18] M. Halkidi, Y. Batistakis, M. Vazirgiannis, On clustering validation techniques, Intelligent Information Systems Journal 17 (2–3) (2001) 107–145.
[19] K. Hammouda, M. Kamel, Hierarchically distributed peer-to-peer document clustering and cluster summarization, IEEE Transactions on Knowledge and Data Engineering 21 (5) (2009) 681–698.
[20] J. Handl, J. Knowles, An evolutionary approach to multiobjective clustering, IEEE Transactions on Evolutionary Computation 34 (2007) 56–76.
[21] Y. Hochberg, A sharper Bonferroni procedure for multiple tests of significance, Biometrika 75 (4) (1988) 800–802.
[22] Y. Hochberg, A.C. Tamhane, Multiple Comparison Procedures, John Wiley & Sons, 1987.
[23] M. Hollander, D.A. Wolfe, Nonparametric Statistical Methods, Wiley-Interscience, 1999.
[24] D. Horta, R.J.G.B. Campello, Fast evolutionary algorithms for relational clustering, in: The International Conference on Intelligent Systems Design and Applications, Pisa, Tuscany, Italy, 2009, pp. 1456–1462.
[25] D. Horta, R.J.G.B. Campello, Evolutionary clustering of relational data, International Journal of Hybrid Intelligent Systems 7 (4) (2010) 261–281.
[26] D. Horta, M.C. Naldi, R.J.G.B. Campello, E.R. Hruschka, A.C.P.L.F. Carvalho, Evolutionary fuzzy clustering: an overview and efficiency issues, in: Bio-Inspired Data Mining: Theoretical Foundations and Applications, Foundations of Computational Intelligence, vol. 4, Springer-Verlag, 2009, pp. 167–195.
[27] E. Hruschka, R.J.G.B. Campello, A.A. Freitas, A.C. Ponce Leon F. de Carvalho, A survey of evolutionary algorithms for clustering, IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 39 (2) (2009) 133–155.
[28] E.R. Hruschka, R.J.G.B. Campello, L.N. de Castro, Evolutionary algorithms for clustering gene-expression data, in: Proceedings of IEEE International Conference on Data Mining, Brighton, England, 2004, pp. 403–406.
[29] E.R. Hruschka, R.J.G.B. Campello, L.N. de Castro, Evolving clusters in gene-expression data, Information Sciences 176 (13) (2006) 1898–1927.
[30] L.J. Hubert, P. Arabie, Comparing partitions, Journal of Classification 2 (1985) 193–218.
[31] A. Jain, R. Dubes, Algorithms for Clustering Data, Prentice Hall, 1988.
[32] A.K. Jain, M.N. Murty, P.J. Flynn, Data clustering: a review, ACM Computing Surveys 31 (3) (1999) 264–323.
[33] L. Kaufman, P. Rousseeuw, Finding Groups in Data: An Introduction to Cluster Analysis, Wiley Series in Probability and Statistics, 2005.
[34] J. Kivijärvi, P. Fränti, O. Nevalainen, Self-adaptive genetic algorithm for clustering, Journal of Heuristics 9 (2) (2003) 113–129.
[35] K. Krishna, N. Murty, Genetic k-means algorithm, IEEE Transactions on Systems, Man and Cybernetics 29 (3) (1999) 433–439.
[36] Y. Lu, S. Lu, F. Fotouhi, Y. Deng, S. Brown, Incremental genetic k-means algorithm and its application in gene expression data analysis, BMC Bioinformatics 28 (5) (2004) 172.
[37] Y. Lu, S. Lu, F. Fotouhi, Y. Deng, S.J. Brown, Fgka: a fast genetic k-means clustering algorithm, in: SAC '04: Proceedings of the 2004 ACM Symposium on Applied Computing, ACM, New York, NY, USA, 2004, pp. 622–623.
[38] M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1998.
[39] T. Mitchell, Machine Learning, McGraw Hill, 1997.
[40] M.C. Naldi, R.J.G.B. Campello, Combining information from distributed evolutionary *k*-means, in: Proceedings of the Brazilian Symposium on Neural Networks, IEEE Computer Society, Curitiba, Brazil, 2012, pp. 43–48.
[41] M.C. Naldi, R.J.G.B. Campello, E.R. Hruschka, A.C.P.L.F. Carvalho, Efficiency issues of evolutionary *k*-means, Applied Soft Computing 11 (2) (2011) 1938–1952.
[42] M.C. Naldi, A. Fontana, R.J.G.B. Campello, Comparison among methods for k estimation in k-means, in: The 9th International Conference on Intelligent Systems Design and Applications (ISDA), 2009, pp. 1006–1013.
[43] S.R.M. Oliveira, O.R. Zaïne, Privacy preserving clustering by data transformation, in: Proceedings of the 18th Brazilian Symposium on Databases, 2003, pp. 304–318.
[44] C.F. Olson, Parallel algorithms for hierarchical clustering, Parallel Computing 21 (8) (1995) 1313–1325.
[45] M. Pakhira, S. Bandyopadhyay, U. Maulik, A study of some fuzzy cluster validity indices, genetic clustering and application to pixel classification, Fuzzy Sets and Systems 155 (2) (2005) 191–214.
[46] M.K. Pakhira, S. Bandyopadhyay, U. Maulik, Validity index for crisp and fuzzy clusters, Pattern Recognition 37 (3) (2004) 487–501.
[47] N. Pal, J. Bezdek, On cluster validity for the fuzzy c-means model, IEEE Transactions of Fuzzy Systems 3 (3) (1995) 370–379.
[48] V. Rayward-Smith, Metaheuristics for clustering in kdd, in: The 2005 IEEE Congress on Evolutionary Computation, 2005, vol. 3, Edinburgh, UK, 2005, pp. 2380–2387.
[49] P. Scheunders, A genetic c-means clustering algorithm applied to color image quantization, Pattern Recognition 30 (6) (1997) 859–866.
[50] W. Sheng, X. Liu, A hybrid algorithm for k-medoid clustering of large data sets, in: Proceedings of the 2004 IEEE Congress on Evolutionary Computation, Portland, IEEE Press, USA, 2004, pp. 77–82.
[51] D. Steinley, K-means clustering: a half-century synthesis, British Journal of Mathematical and Statistical Psychology 59 (34) (2006) 1–34.
[52] J. Tian, L. Zhu, S. Zhang, L. Liu, Improvement and parallelism of k-means clustering algorithm, Tsinghua Science & Technology 10 (3) (2005) 277–281.
[53] J. Vaidya, C. Clifton, Privacy-preserving data mining: why, how, and when, Security & Privacy, IEEE 2 (6) (2004) 19–27.
[54] L. Vendramin, R.J.G.B. Campello, E.R. Hruschka, On the comparison of relative clustering validity criteria, in: SIAM International Conference on Data Mining, Sparks, USA, 2009, pp. 733–744.
[55] L. Vendramin, R.J.G.B. Campello, E.R. Hruschka, Relative clustering validity criteria: a comparative overview, Statistical Analysis and Data Mining 3 (4) (2010) 209–235.
[56] R.E. Walpole, R. Myers, S.L. Myers, Probability & Statistics for Engineers & Scientists, Macmillan, 2006.
[57] X. Wu, Top Ten Algorithms in Data Mining, Taylor & Francis, 2009.
[58] R. Xu, D.I. Wunsch, Survey of clustering algorithms, IEEE Transactions on Neural Networks 16 (3) (2005) 645–678.
[59] M. Zaki, Parallel and Distributed Data Mining: An Introduction, Springer, Berlin, Heidelberg, 2000, pp. 804–827 (Chapter 1).
[60] M.J. Zaki, Parallel and distributed data mining: a survey, IEEE Concurrency 7 (4) (1999) 14–25.
[61] T. Zhang, R. Ramakrishnan, M. Livny, BIRCH: an efficient data clustering method for very large databases, in: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Canada, 1996, pp. 103–114.

Murilo C. Naldi received the B.Sc., M.Sc. and Ph.D. degrees in Computer Science in, respectively, 2004, 2006, and 2011, from the University of São Paulo (USP), São Carlos, Brazil. Currently, he is an Assistant Professor at Federal University of Viçosa - Campus Rio Paranaíba. His main interests are Data Mining, Machine Learning, and Evolutionary Computation.

Ricardo J.G.B. Campello received the B.Sc. degree in Electronics Engineering from the State University of São Paulo (Unesp), Ilha Solteira/SP, Brazil, in 1994, and the M.Sc. and Ph.D. degrees in Electrical Engineering from the School of Electrical and Computer Engineering of the State University of Campinas (Unicamp), Campinas/SP, Brazil, in 1997 and 2002, respectively. In 2002 he was a postdoctoral fellow at the Laboratoire D'Informatique, Signaux et Systèmes de Sophia Antipolis, Université de Nice - Sophia Antipolis (UNSA), France. From 2003 to 2006 he was an invited researcher of the School of Electrical and Computer Engineering of the State University of Campinas. He has been a merit scholar of the Brazilian National Research Council (CNPq) since 2005. Since 2007 he is with the Department of Computer Sciences of the University of São Paulo (USP) at São Carlos/SP, Brazil, currently as an Associate Professor (from 2011 onwards). He is currently in a sabbatical leave as a visiting professor at the Department of Computing Science of the University of Alberta (UofA), in Edmonton/AB, Canada. His current research interests fall primarily into the areas of Data Mining, Machine Learning, and Computational Intelligence.