Hierarchical Density-Based Clustering Using MapReduce

Joelson Antônio dos Santos, Talat Iqbal Syed, Murilo C. Naldi, Ricardo J. G. B. Campello, and Joerg Sander

Abstract—Hierarchical density-based clustering is a powerful tool for exploratory data analysis, which can play an important role in the understanding and organization of datasets. However, its applicability to large datasets is limited because the computational complexity of hierarchical clustering methods has a quadratic lower bound in the number of objects to be clustered. MapReduce is a popular programming model to speed up data mining and machine learning algorithms operating on large, possibly distributed datasets. In the literature, there have been attempts to parallelize algorithms such as Single-Linkage, which in principle can also be extended to the broader scope of hierarchical density-based clustering, but hierarchical clustering algorithms are inherently difficult to parallelize with MapReduce. In this paper, we discuss why adapting previous approaches to parallelize Single-Linkage clustering using MapReduce leads to very inefficient solutions when one wants to compute density-based clustering hierarchies. Preliminarily, we discuss one such solution, which is based on an exact, yet very computationally demanding, random blocks parallelization scheme. To be able to efficiently apply hierarchical density-based clustering to large datasets using MapReduce, we then propose a different parallelization scheme that computes an approximate clustering hierarchy based on a much faster, recursive sampling approach. This approach is based on HDBSCAN*, the state-of-the-art hierarchical density-based clustering algorithm, combined with a data summarization technique called data bubbles. The proposed method is evaluated in terms of both runtime and quality of the approximation on a number of datasets, showing its effectiveness and scalability.

Index Terms—Density-based hierarchical clustering, MapReduce, big data

1 Introduction

Clustering is an unsupervised learning task that aims to decompose a dataset into distinct *clusters* of data objects that help understand how a dataset is structured [1]. Density-based clustering is a category of algorithms where the fundamental idea is that a dataset of interest represents a sample from an unknown probability density function, which describes the mechanism or mechanisms responsible for producing the observed data. Clustering with a density-based algorithm directly or indirectly involves the problem of density estimation. Clusters are somehow detected from this estimation as "high density regions separated by regions of low density", where the notions of "high" and "low" density depend on some type of density threshold [2]. In partitioning algorithms this threshold is fixed, whereas in hierarchical algorithms each hierarchical level corresponds to a different threshold.

Partitioning density-based clustering algorithms, such as DBSCAN [3], have fundamental limitations [2], [4]: (i) they require a user-defined threshold, which is a critical parameter; (ii) often, it is not possible to simultaneously detect clusters of varied densities by using a single, global density threshold; and (iii) a flat clustering solution alone cannot describe possible hierarchical relationships that may exist between nested clusters lying on different density levels. Nested clusters at varied levels of density can only be described by hierarchical density-based clustering methods [5], [6], [7], which provide more elaborated descriptions of a dataset at different degrees of granularity and resolution.

Hierarchical models are indeed able to provide richer descriptions of clustering structures than those provided by flat models, but applications in which the user also needs a flat solution are common, either for further manual analysis by a domain expert or in automated KDD processes in which the output of a clustering algorithm is the input of a subsequent data mining procedure. In this context, the extraction of a flat clustering from a hierarchy, as opposed to the extraction directly from data by a partitioning-like algorithm, can be advantageous because hierarchical models describe data from multiple levels of specificity/generality, providing a means for exploration of multiple possible solutions from different perspectives while having a global picture of the clustering structure available.

- *J.A. dos Santos is with the Department of Computer Science, University of São Paulo, São Carlos 13566-590, Brazil. E-mail: joelsonn.santos@gmail.com.*
- *T.I. Syed and J. Sander are with the Department of Computing Science, University of Alberta, Edmonton ABT6G 2R3, Canada. E-mail: {TalatIqbal, jsander}@ualberta.ca.*
- *M.C. Naldi is with the Department of Computer Science, Federal University of São Carlos, São Carlos, SP 13565-905, Brazil. E-mail: naldi@ufscar.br.*
- *R.J.G.B. Campello is with the School of Mathematical & Physical Sciences, University of Newcastle, Newcastle, NSW 2308, Australia. E-mail: campello@icmc.usp.br.*

Manuscript received 11 July 2018; revised 12 Dec. 2018; accepted 19 Mar. 2019. Date of publication 26 Mar. 2019; date of current version 1 Mar. 2021. (Corresponding author: Murilo Coelho Naldi.)
Recommended for acceptance by J. Cao.
Digital Object Identifier no. 10.1109/TBDATA.2019.2907624

However, hierarchical clustering algorithms are typically much more computationally demanding than partitioning algorithms. Indeed, the computational complexity of hierarchical clustering methods has a quadratic lower bound in the number of objects to be clustered. With very large databases being generated in an increasing number of real-world application scenarios, the analysis of such data using hierarchical clustering becomes challenging. In these scenarios, parallelization may increase computational performance and broaden the applicability of the algorithms. To achieve this goal, programming models must be used that allow large volumes of data to be analyzed in a timely manner, while guaranteeing safety against failures and scalability as the demand for data analysis as well as the database sizes grow. One such programming model, MapReduce, was presented in [8], [9]. This model is an abstraction that allows for simplified distributed and scalable programming, and it has been used by companies such as IBM, Microsoft, Dell, Yahoo!, and Amazon, among others [10], [11].

MapReduce frameworks and extensions have been developed to deploy the concept in practice—e.g., Apache Hadoop [12] and Spark [13], which have quickly spread since their creation. Special libraries have been created with algorithms to solve large-scale problems. Among those, the Apache Mahout project [14] and the Machine Learning Library (MLlib) [13] have a collection of machine learning algorithms adapted for MapReduce. However, there are only a few data clustering algorithms in these libraries, mainly because most traditional clustering algorithms were not originally designed to work in a distributed or parallel fashion, and adapting them may not be doable or worthwhile due to restrictions imposed by the MapReduce model or due to their high computational complexity. The programming model requires that the algorithms split their tasks into two main functions, called map and reduce, and imposes an execution flow on them. These functions execute parallel jobs independently on distributed parts of the dataset, thus each job cannot share information about its portion of data with other jobs in execution. This constraint can be a burden for traditional clustering algorithms, specially hierarchical ones as they are usually based on pairwise comparisons between data objects, which may require mapping the whole dataset repeatedly a number of times proportional to the square of its size. Such calculations tend to require prohibitive amounts of computer power, network load and processing time, specially for large amounts of data. For this reason, few techniques for hierarchical clustering based on MapReduce have been proposed in the literature [15], [16], mostly for Single-Linkage. However, these techniques have shortcomings that make their implementation in practice both challenging and inefficient, namely: (a) the large number of managed partitions; and (b) redundant processing, where calculations are repeated in different processing units.

In this paper, we discuss why adapting previous approaches to parallelize Single-Linkage clustering using MapReduce leads to inefficient solutions when it comes to computing density-based clustering hierarchies. In order to efficiently apply hierarchical density-based clustering to large datasets using MapReduce, we propose an alternative parallelization scheme, called MapReduce HDBSCAN* or MR-HDBSCAN*, which efficiently computes an approximate clustering hierarchy based on a recursive sampling strategy preliminarily introduced in [17]. This method is based on the state-of-the-art hierarchical density-based clustering algorithm HDBSCAN* [7], which can be seen as a generalization of Single-Linkage, combined with two possible data summarization techniques, namely, random samples and data bubbles [18]. These two variants of MR-HDBSCAN* were compared with an exact MapReduce HDBSCAN* version using the Random Blocks approach, preliminarily introduced in [17] and also presented in this paper, both in terms of clustering quality and runtime. The experiments show that, while the Random Blocks version provides exact results (i.e., the same as the centralized version of HDBSCAN* running in a single machine, if this machine could cope with all data storage and processing), the proposed approximate variants can obtain competitive results in terms of clustering quality, while being orders of magnitude faster in a distributed environment.

The remainder of this paper is organized as follows. In Section 2 we review the related work. In Section 3 we present the required background and briefly discuss an exact (yet inefficient) preliminary solution to parallelize HDBSCAN* based on the random blocks approach. In Section 4 we discuss in detail an efficient MapReduce implementation of HDBSCAN*, following a recursive sampling approach. In Section 5 we present our experimental results and analyses. Finally, in Section 6 we address the conclusions and future work.

2 Related Work

MapReduce-based strategies to parallelize hierarchical clustering algorithms have been proposed before in the literature. For instance, PArallel, RAndom-partition Based hierarchicaL clustEring (PARABLE) [15] is a two step algorithm in which the first step computes local hierarchical clusterings on distributed nodes and the second step integrates the results by a proposed *dendrogram alignment technique*. First, the dataset is partitioned into random subsamples. The sequential algorithm is then run in parallel on the resulting data subsets to form intermediate dendrograms. These intermediate dendrograms are then aligned with each other to form the final dendrogram by recursively aligning the nodes from the root to the leaves, by comparing the nodes of two given dendrograms using a similarity measure. The authors claim that the "alignment procedure is reasonable when the dendrograms to be aligned have similar structure" [15] and that a good sampling strategy can lead to local dendrograms being similar. During the dendrogram alignment technique, however, the algorithm fails to address the fact that addition of new branches in a dendrogram (addition of new data objects to the dataset) also affects the height at which mergers occur.

Other parallel and distributed hierarchical clustering algorithms have been studied in the literature, but most of these studies [19], [20], [21], [22] were aimed at the Single-LINKage (SLINK) clustering algorithm [23], which can be seen as a special case of HDBSCAN* [7], [24]. In [25], the parallelization of SLINK was translated to the problem of parallelizing a Minimum Spanning Tree (MST) with data objects as vertices of a graph and the distance between objects as

edge weights connecting the respective vertices. CLUstering through MST in Parallel (CLUMP) [26] addresses the parallelized construction of an MST on distributed nodes using a collection of overlapping datasets. Different nodes are responsible for processing different subsets of data with each subset of data duplicated at least in another node. The authors use the *"Linear Representation"* (LR), which is a list of elements of a dataset $\mathbf{X}$ whose sequential order is the same as the order in which the elements are selected for constructing an MST using Prim's algorithm.

SHaRed-memory SLINK (SHRINK) [27] is a scalable algorithm to parallelize the Single-Linkage hierarchical clustering algorithm. The strategy is to partition the original dataset into overlapping subsets of data objects and compute the hierarchy for each subset using Single-Linkage, then construct the overall dendrogram by combining the solutions for the different subsets of the data. Other algorithms such as PINK [28] and DiSC [16] are based on similar ideas as CLUMP and SHRINK. While PINK is designed for a distributed memory architecture, Distributed Single-Linkage Hierarchical Clustering (DiSC) is specifically implemented using the MapReduce programming model. DiSC divides the data into overlapping subsets of data that are processed individually at different distributed nodes to form partial sub-solutions. These sub-solutions are then iteratively combined to form an overall solution for the complete dataset. The MapReduce implementation of DiSC consists of two rounds of MapReduce jobs. The first round consists of a Prim's Mapper, which builds an MST on the corresponding subset of data, and a Kruskal Reducer, which uses a K-way merge to combine and get the intermediate result. The Kruskal Reducer uses a UnionFind data structure [28] to keep track of the membership of all the connected components and filter out any cycles. The second MapReduce job, called the Kruskal-MR job, consists of a Kruskal Mapper which is just an identity mapper (rewrites the input as output) and a Kruskal Reducer, which essentially does the same work as the reducer of the first round.

The aforementioned methods have been designed and implemented for the Single-Linkage algorithm. Although Single-Linkage is a particular case of HDBSCAN* [7], [24], there are two main reasons as for why the above strategies are very inefficient in the general case [17]. First, the number of partitions to be processed in parallel increases rapidly as the value of HDBSCAN*'s density estimate smoothing parameter m_{pts} increases. In addition, the number of duplicate computations at various parallel nodes also increases as m_{pts} increases. For these reasons, the use of these strategies to parallelize HDBSCAN* is not scalable for applications involving large datasets and practical values of m_{pts} (other than 1 or 2, in which case HDBSCAN* reduces to Single-Linkage) [17]. In this context, the MapReduce framework can facilitate data management, particularly in a distributed computing environment, but it does not trivially reduce the computational effort involved in the hierarchical clustering calculations, among other reasons because many partial copies of the data may have to be transmitted from one processing unit to another and the total amount of transmitted data can be many times larger than the dataset size, unless a carefully designed algorithm and architecture is in place, as we will discuss in this paper.

MapReduce implementations of partitioning density-based clustering algorithms have been proposed in the literature, particularly for DBSCAN [29], [30], [31]. However, DBSCAN and, accordingly, its MapReduce implementations, follow a density-based model that is limited to a single density threshold and produces a "flat" clustering solution based on a global density level, rather than a tree of density-contour clusters and sub-clusters across multiple different levels. This approach has well-known fundamental limitations, namely [2], [7]: the choice of the density threshold is both difficult and critical; the algorithm may not be able to discriminate between clusters of very different densities; and, a flat clustering solution cannot describe hierarchical relationships that may exist between nested clusters at different density levels. HDBSCAN*, which constitutes the main focus of this work, does not suffer from these limitations.

3 Background and Preliminaries

3.1 HDBSCAN* and FOSC

The HDBSCAN* algorithm [7], [24] has several advantages over traditional partitioning and hierarchical clustering algorithms. It combines the aspects of density-based clustering and hierarchical clustering, producing a complete density-based clustering hierarchy from which a simplified hierarchy composed only of the most prominent clusters can be straightforwardly extracted. Its results can be visualized by means of a complete dendrogram, a simplified cluster tree, and other visualization techniques that do not require any critical parameter as input. In fact, HDBSCAN*'s only parameter, m_{pts}, is a just smoothing factor of a non-parametric density estimate performed by the algorithm, from which all DBSCAN-like solutions corresponding to any value of density threshold (radius $\epsilon \in [0, \infty)$ in DBSCAN) are automatically produced in a nested way, without the need to specify a particular threshold as input. The behavior of m_{pts} is well understood [7], [24] and methods that have an analogous parameter (e.g.,[5], [32], [33], [34]) are typically robust to it. HDBSCAN* is also flexible in that a user can choose to analyze the resulting hierarchy and cluster tree directly, or perform local cuts through this tree to automatically obtain a flat (non-hierarchical) solution that is optimal according to a given criterion.

Given a dataset $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ with n data objects, $\mathbf{x}_i$, and a particular value of the smoothing factor m_{pts}, the following definitions are used by HDBSCAN*:

Core Distance. The core distance of an object $\mathbf{x}_p \in \mathbf{X}$ w.r.t. m_{pts}, $d_{core}(\mathbf{x}_p)$, is the distance from $\mathbf{x}_p$ to its m_{pts}th nearest neighbor (incl. $\mathbf{x}_p$).

The core distance of an object $\mathbf{x}_i$ can be interpreted as the minimum radius ϵ (maximum density threshold) such that a data object $\mathbf{x}_p$ is considered a dense object, so-called *core object*, for having at least m_{pts} objects within its ϵ-neighborhood (a ball of radius ϵ centered at $\mathbf{x}_i$, i.e., $\{\mathbf{x} : d(\mathbf{x}, \mathbf{x}_i) \leq \epsilon\}$, where $d(\cdot, \cdot)$ is an arbitrary dissimilarity measure).

Mutual Reachability Distance. The mutual reachability distance between two objects $\mathbf{x}_p, \mathbf{x}_q \in \mathbf{X}$ w.r.t. m_{pts} is defined as

$$d_{m_{reach}}(\mathbf{x}_p, \mathbf{x}_q) = max\big(d_{core}(\mathbf{x}_p), d_{core}(\mathbf{x}_q), d(\mathbf{x}_p, \mathbf{x}_q)\big).$$

The mutual reachability distance between two objects $\mathbf{x}_p$ and $\mathbf{x}_q$ can be interpreted as the minimum radius ϵ such that both objects are core objects and fall within each other's ϵ-neighborhood, which is equivalent to the maximum density threshold such that both objects are dense and directly density connected.

Mutual Reachability Graph. A mutual reachability graph of a dataset $\mathbf{X}$ w.r.t m_{pts} is a (conceptual only) complete weighted graph, $G_{m_{reach}}$, in which the data objects are the vertices and the edge weight between each pair of vertices is given by the mutual reachability distance between the corresponding pair of objects.

Algorithm 1. HDBSCAN* (Main Algorithm)

Requires $\{\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}, m_{pts}\}$.
Output {HDBSCAN* hierarchy}.

1) Given a dataset $\mathbf{X}$, compute the core distances of all the data objects in $\mathbf{X}$.
2) Compute a Minimum Spanning Tree of the Mutual Reachability Graph, $G_{m_{reach}}$ (the graph does not need to be materialized, as the edge weights can be computed on demand).
3) Extend the MST, to obtain MST_{ext}, by adding a "self edge" (i.e., a loop) to each vertex, with edge weight equal to the vertex's core distance, $d_{core}(\cdot)$.
4) Extract the HDBSCAN* hierarchy as a dendrogram from MST_{ext}.
   a) All objects are initially assigned the same cluster label (the root of the cluster tree).
   b) Iteratively remove edges from MST_{ext} in decreasing order of weights.
      i) Edges with the same weight are removed simultaneously.
      ii) After removal of an edge, cluster labels are assigned to the two connected components that contain one vertex of the removed edge. A new cluster label is assigned if the component has at least one (possibly self) edge remaining in it, otherwise the object (s) in the component are labeled as noise (null label).

The main steps of HDBSCAN* are summarized in Algorithm 1. It is worth noticing that the original algorithm [7], [24] is also equipped with an *optional* parameter, m_{clSize}, which allows the user to specify the minimum size for a component to be considered a cluster, in such a way that components with fewer than m_{clSize} objects are disregarded as noise. This represents an additional, optional control that can significantly reduce the size of the resulting clustering hierarchy, if desired. By default, HDBSCAN* uses $m_{clSize} = m_{pts}$, so in practice only m_{pts} needs to be given as input.

Once the HDBSCAN* hierarchy is obtained, it is possible to perform cluster analysis, outlier detection, and data visualization using an integrated framework as presented in [7]. In particular, HDBSCAN* incorporates an optional dynamic programming-based post-processing routine, called Framework for Optimal Selection of Clusters from hierarchies (FOSC) [35], that can automatically extract a "flat" clustering solution consisting of the most prominent clusters according to an optimization criterion. This flat solution is optimally extracted from local, possibly non-horizontal cuts through the cluster tree, which may correspond to different density levels and whose clusters may not be detectable by a single, global density threshold (a traditional horizontal cut through the clustering hierarchy).

As the default optimization criterion for FOSC, HDBSCAN* uses the total sum of *Stability* of the extracted clusters, which in turn is a generalization, for density-based hierarchies, of the classic notion of cluster *lifetime* in traditional dendrograms [1]. Stability is additive and local, which means that it can be decomposed as a sum of components precomputed individually and independently for each candidate cluster in the cluster tree. As computed by HDBSCAN*, the stability of a cluster $\mathbf{C}_i$ is related to the statistical notion of (relative) *excess of mass* of a mode of a density function

$$S(\mathbf{C}_i) = \sum_{\mathbf{x}_i \in \mathbf{C}_i} \left(\frac{1}{\epsilon_{min}(\mathbf{x}_j, \mathbf{C}_i)} - \frac{1}{\epsilon_{max}(\mathbf{C}_i)} \right), \quad (1)$$

where $\epsilon_{min}(\mathbf{x}_j, \mathbf{C}_i)$ is the minimum ϵ value (height of hierarchical level) for which $\mathbf{x}_j \in \mathbf{C}_i$ and $\epsilon_{max}(\mathbf{C}_i)$ is the maximum ϵ value for which cluster $\mathbf{C}_i$ exists. Further details are discussed in [7], [24].

3.2 MapReduce Programming Model and Frameworks

MapReduce is a programming model [8] to process large scale data in a massively data parallel way. MapReduce has several advantages over other existing parallel processing models. The programmer is not required to know the details related to data distribution, storage, replication and load balancing, thus hiding the implementation details and allowing the programmers to develop applications/ algorithms that focus more on processing strategies. The programmer is required to specify two functions: a *map* and a *reduce*. The programming model (detailed in [36]) is summarized as follows:

1) The map stage passes over the input file and outputs <key/value> pairs.
2) The shuffling stage transfers the mappers output to the reducers based on the key.
3) The reduce stage processes the received pairs and outputs the final result.

Due to its scalability and simplicity, MapReduce has become a widespread tool for large scale data analysis. Frameworks have been created to work with this model and facilitate its implementation—e.g., Apache Hadoop [12], which has quickly spread since its creation. A more recent framework, Apache Spark [13], exhibits additional advantages (besides the use of MapReduce), such as the use of an abstraction called Resilient Distributed Data (RDD). This abstraction allows scalable programs to carry out operations with persistent data in main memory with high fault tolerance, being particularly suitable for use in iterative algorithms [13], which is the case of several data mining algorithms. Both frameworks interleave sequential and parallel computation, consisting of several rounds. Each round transmits information among distributed systems (nodes),

which perform the required computation using the data available locally. The output of these computations is either combined to form the final result or sent to another round of computation depending on the application's requirement. However, constraints are imposed to the MapReduce workflow, as the calculations applied during each of its functions must occur independently and no information can be exchanged among jobs of the same function during their parallel execution.

3.3 Exact MapReduce HDBSCAN*—The Random Blocks Approach

MapReduce limits the algorithms to splitting their tasks into two main functions, called map and reduce, and imposes an execution flow on them. Most traditional algorithms were not originally designed to work in a distributed or parallel fashion, especially when these restrictions are imposed. Traditional hierarchical clustering algorithms are usually based on pairwise comparisons between objects of the dataset, which implies mapping the whole dataset repeatedly a number of times that grows with its size. For this reason, exact versions of these algorithms may not be doable in practice due to the aforementioned restrictions or due to their prohibitive computational requirements. The HDBSCAN* algorithm is no exception. It requires the whole dataset to be available in order to calculate the mutual reachability distances between all pairs of data objects and compute the MST of the Mutual Reachability Graph, $G_{m_{reach}}$, in order to obtain the HDBSCAN* hierarchy. This requirement makes it difficult to parallelize HDBSCAN*.

A simple approach to parallelize the computation of the exact HDBSCAN* hierarchy across multiple distributed nodes is known as the Random Blocks approach, preliminarily proposed by one of our authors in [17] and described here in this paper as a baseline for comparisons. It essentially adapts CLUMP [26] for parallel computation of the extended MST, MST_{ext}, from which the HDBSCAN* hierarchy is extracted. To parallelize and distribute an MST in the same spirit as in [26], the Random Blocks approach divides the complete dataset into k *"base partitions"*, which then have to be combined into so-called *"data blocks"*, so that for any pair of objects $(\mathbf{p}, \mathbf{q})$ the edge weight between $\mathbf{p}$ and $\mathbf{q}$ (in the complete graph upon which the MST is built) can be exactly determined within one of the data blocks. Hence, MSTs can be computed within data blocks independently and in parallel at different processing units, and these independently computed MSTs can then be combined to obtain an exact MST for the complete dataset.

To parallelize the MST for Single-Linkage computation as in [26], one has to generate $\binom{k}{2}$ data blocks in total (all k choose 2 pairwise combinations of the k base partitions). In this case, it is guaranteed that every pair of objects $(\mathbf{p}, \mathbf{q})$ is together in at least one data block, such that the exact distance between $\mathbf{p}$ and $\mathbf{q}$ can be determined. This is the edge weight between $\mathbf{p}$ and $\mathbf{q}$, needed to compute the Single-Linkage hierarchy.

HDBSCAN*, however, is not based on an MST computed in the original distance space. Instead, it is built upon an MST computed in the transformed space of mutual reachability distances, i.e., edge weights correspond to mutual reachability distances rather than original distances. In order to determine the mutual reachability distance between two objects $\mathbf{p}$ and $\mathbf{q}$, one needs to be able to compute the core distance for both $\mathbf{p}$ and $\mathbf{q}$ w.r.t. m_{pts}, as well as the distance between $\mathbf{p}$ and $\mathbf{q}$. The core distance of an object $\mathbf{p}$ is defined as the distance from $\mathbf{p}$ to its m_{pts}th nearest neighbor, including $\mathbf{p}$ itself (Definition 3.1). To determine this distance in a single data block, $\mathbf{p}$ and the other $m_{pts} - 1$ objects closest to $\mathbf{p}$ have to be present in that data block. Consequently, to determine the mutual reachability distance between $\mathbf{p}$ and $\mathbf{q}$, if their neighborhoods do not share any objects, $2 \times m_{pts}$ different objects have to be present in a single data block. In the worst case, each of those $2 \times m_{pts}$ different objects belongs to a different base partition. Consequently, given a set of k base partitions and the parameter m_{pts}, to guarantee that the mutual reachability distance between any two objects $\mathbf{p}$ and $\mathbf{q}$ can be determined in at least one data block, $\binom{k}{2 \times m_{pts}}$ unique data blocks (all k choose $2 \times m_{pts}$ combinations of base partitions) have to be generated and distributed to different processing units. In practice, this leads to an unacceptable computational burden, especially for larger values of m_{pts}, which are necessarily larger than 2 for practical purposes.[1] This burden may be tolerated for small datasets that fit the main memory of a single machine, but it is prohibitive for larger datasets, which require the data to be distributed and processed across multiple nodes.

While an exact version of the HDBSCAN* hierarchy based on Random Blocks cannot be efficiently computed in a distributed environment, in the next section we show that it is possible to efficiently compute an *approximate* version of the hierarchy using MapReduce, which trades only a small amount of accuracy for a large gain in scalability.

4 Efficient MapReduce HDBSCAN*—Recursive Sampling Approach

The way we propose to parallelize HDBSCAN* within a Map-Reduce framework is based on a *"recursive sampling"* approach. Unlike the parallelization of an MST computation, our *Recursive Sampling Approach* eliminates the processing of overlapping datasets at multiple processing units. This is achieved by an *"informed"* data subdivision that divides the data to be processed at different processing units taking into account the structure of the data themselves.

4.1 General Idea

Fig. 1 shows the flow of the Recursive Sampling Approach. The general idea of clustering data that cannot be processed entirely in a single node is to use a subsample that captures the most prominent clusters as a very coarse representation of the structure contained in the dataset. Those clusters constitute the higher levels of a cluster hierarchy. We capture the most prominent clusters in a sample by first clustering the sample using HDBSCAN* and then extracting the major clusters using the FOSC framework,

1. HDBSCAN* can be seen as a generalized, robust version of Single-Linkage that addresses its undesirable sensitivity to the *"chaining effect"*. For $m_{pts} \leq 2$, HDBSCAN* is equivalent to Single-Linkage.

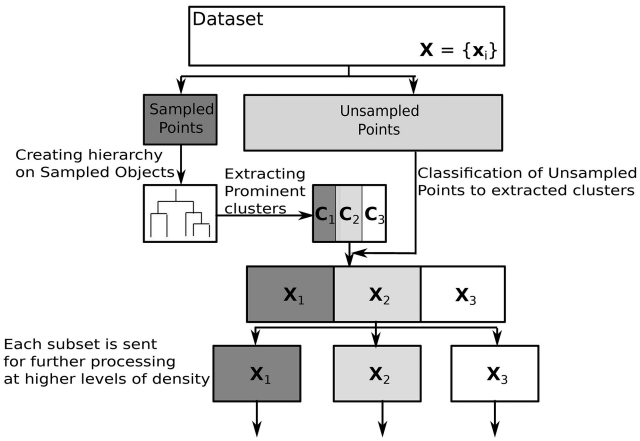

Fig. 1. Execution flow of Recursive Sampling Approach.

as described in Section 4.4. Based on the extracted partition of the sample, one can induce a partition of the whole dataset into subsets corresponding to the major clusters in the data (by assigning each data object to the same cluster as its nearest sample representative), and each of these subsets is then sent to different processing units for further refinement. Depending on the capacity of the processing units, these subsets will be recursively partitioned into smaller subsets representing the next levels of the clustering hierarchy, by applying the same strategy, until the data blocks are of sizes that can be processed completely by a processing unit to obtain exact MSTs for these subsets of the data. The edges that connect these local MSTs hierarchically are determined during the recursive partitioning, so that at the end a spanning tree of the whole data is obtained, which is "locally minimal" for the subsets that are small enough to be processed at a single node. This spanning tree can be used by the HDBSCAN* algorithm in the same way as an MST in order to obtain a clustering hierarchy (as a dendrogram or simplified cluster tree). Implementation details are discussed in Section 4.3.

4.2 Using Data Summarizations Instead of a Sample

Fig. 1 illustrates the Recursive Sampling Approach using just simple samples. While this is a possible approach, it has been observed that applying density-based hierarchical clustering to samples can lead to inferior results when the sample size is small compared to the whole dataset, due to the fact that (1) distances between "representative" sample objects represent poorly the distances between the objects that they represent, and (2) density estimates using only a sample of the data approximates poorly the density in the whole dataset [37]. To alleviate these problems, the concept of "data bubbles" has been introduced in [37], and it has been shown to vastly improve clustering quality of the density-based hierarchical clustering algorithm OPTICS. In this paper, we focus on data bubbles for points in an euclidean vector space, although data bubbles have been also extended for data from general metric spaces (see [18]).

Constructing data bubbles starts with drawing a small sample $\mathbf{S}$ of size m of the whole dataset. Conceptually, each sample point $\mathbf{o}$ "represents" all objects in the dataset that are closer to $\mathbf{o}$ than to any other sample point in $\mathbf{S}$. This corresponds conceptually to a (Voronoi) partition of the dataset into m subsets. In a single sequential scan of the whole data, for each of those m subsets, $\mathbf{X_b}$, sufficient statistics of the form (n, LS, SS) are computed, where n is the number of objects in the subset $\mathbf{X_b}$, $LS = \sum_{\mathbf{x}_i \in \mathbf{X_b}} \vec{\mathbf{x}_i}$ is the Linear Sum of the points in $\mathbf{X_b}$, and $SS = \sum_{\mathbf{x}_i \in \mathbf{X_b}} \vec{\mathbf{x}_i}^2$ is the Sum of Squares of the points in $\mathbf{X_b}$.

For clustering, each of the m subsets $\mathbf{X_b}$ is then represented by a data bubble, which is a tuple $B_{\mathbf{X_b}} = (rep, n, extent, nnDist)$, where $rep = LS/n$ is the mean of the objects in $\mathbf{X_b}$, n is the number of objects in $\mathbf{X_b}$, $extent = \sqrt{\frac{\sum_{i=1,\ldots,n}\sum_{j=1,\ldots,n}(\mathbf{x}_i - \mathbf{x}_j)^2}{n(n-1)}} = \sqrt{\frac{2 \times n \times SS - 2 \times LS^2}{n(n-1)}}$ is the "radius" of the subset $\mathbf{X_b}$, and $nnDist$ is an estimate of average k-nearest neighbor distances within the set of objects $\mathbf{X_b}$ for values of $k = 1, \ldots, m_{pts}$. Assuming a uniform distribution within the subset $\mathbf{X_b}$, this estimate is given by $nnDist(k) = \frac{k}{n}^{\frac{1}{d}} \times extent$, where d is the data dimension.

The above representation allows: (1) the definition of an appropriate distance between two data bubbles B and C, $dist(B, C)$, as described in detail in [37]; (2) the definition of an appropriate core distance, $d_{core}(B)$, for a data bubble B by estimating the m_{pts}-nearest neighbor distance of its representative rep, using function $nnDist$ (and if B contains less than m_{pts} many points, the distance to the next closest data bubble that would the include the m_{pts}-nearest neighbor of rep); (3) the definition of mutual reachability distance, required by HDBSCAN*, analogously to the mutual reachabiltiy distance between points, as $d_{m_{reach}}(B, C) = max(d_{core}(B), d_{core}(C),\ dist(B, C))$. The core distance of a data bubble also acts as a core distance for all the points in the bubble, and the *excess of mass* of a cluster in the cluster tree from a sample can thereby be better estimated by taking into account the estimated mass of all the points in the cluster, which improves the FOSC extraction of the most prominent clusters (Section 4.4).

It has been shown in [37] that data bubbles allow recovering the clustering structure of large datasets from extremely low samples, where clustering just the sample would generate meaningless results. The experiments in [37] were based on the algorithm OPTICS. We will show in our experimental evaluation that similar conclusions can be draw from the use of data bubbles alongside with HDBSCAN*.

4.3 Implementation Using MapReduce

HDBSCAN* can be implemented in a parallel and distributed way using smart mapping and aggregation strategies, provided by the MapReduce framework. Here, we call our implementation MapReduce HDBSCAN* or MR-HDBSCAN*. MR-HDBSCAN* is composed of three main steps: first, the data is partitioned until each partition fits into a single processing unit capacity (represented by τ) and an MST is calculated for each part; subsequently, all MSTs and inter-cluster edges are combined into a single MST_{ext}; finally, the HDBSCAN* hierarchy is extracted as a dendrogram from MST_{ext}. These steps are described below and

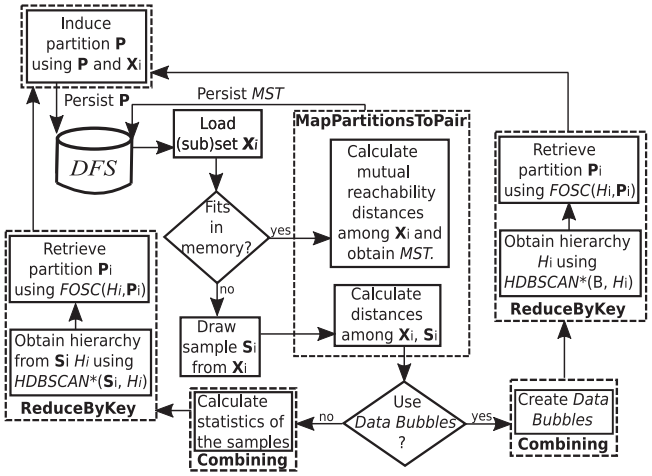

Fig. 2. Illustration of the first step of MR-HDBSCAN*.

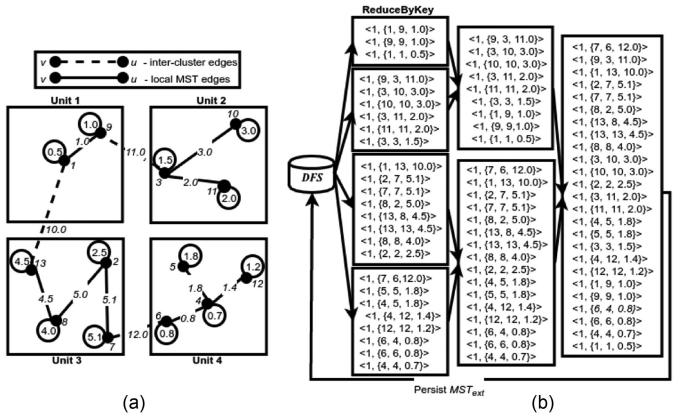

Fig. 3. Example of combining persisted MSTs from data partitions and inter-cluster edges to obtain an overall extended spanning tree. (a) MSTs and inter-cluster edges distributed across four processing units. (b) *ReduceByKey* edges combining scheme.

further detailed in the Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TBDATA.2019.2907624.

The first step of MR-HDBSCAN* is based on the Recursive Sampling Approach, presented in Section 4.1. A truly recursive implementation of this approach would require a function to call itself to recursively partition the data, until the data size is small enough to be processed by a single processing unit. However, such a recursive function cannot be directly parallelized by the MapReduce model.For this reason, we have instead developed an alternative implementation of this sampling process, based on an iterative partitioning of the data.

The first step of MR-HDBSCAN* partitions the data until it fits a single processing unit. This step is divided into four parts: a *MapPartitionsToPair* function, designed to build a local MST over the space of mutual reachability distances or, if the data is too big to be processed locally, calculate the nearest sample representative of each data object; *Combining* functions, applied over the samples to create data bubbles and update information; a *ReduceByKey* function, responsible for partitioning the data bubbles/samples with HDBSCAN* and FOSC; and a *Mapper* function, which maps the data objects represented by the bubbles/samples into the previously obtained partition.

An overview of the first step of MR-HDBSCAN* is illustrated in Fig. 2. It assumes that the data resides in the Distributed File System (DFS) of the MapReduce framework. The algorithm starts with a single cluster containing the whole dataset $\mathbf{X}$. At each iteration, every cluster in the DFS is loaded to be processed in parallel and independently by a MapPartitionsToPair function. This function receives a partition of the data and maps the objects of each cluster $\mathbf{X}_i$ into <key/value> pairs. If $\mathbf{X}_i$ is small enough to be processed by a single processing unit, the function computes an MST (w. r.t. mutual reachability distances) for $\mathbf{X}_i$, extends it with self-edges, and stores (persists) all edges in the DFS. If $\mathbf{X}_i$ is too big to fit a single processing unit, a sample $\mathbf{S}_i$ is drawn from $\mathbf{X}_i$ and the distances between the remaining objects of $\mathbf{X}_i$ and the sampled objects in $\mathbf{S}_i$ are calculated. These distances are sent to a Combining function, which calculates the statistics (n, LS, SS) for each sampled object, as described in Section 4.2, and can optionally build data bubbles over the samples.[2]

The Combining function sends the combined statistics of the samples or data bubbles to a ReduceByKey function, which applies HDBSCAN* to construct an MST and, from that MST, it builds the cluster hierarchy H_i. Next, FOSC with the Excess of Mass (EOM) as quality measure is used to extract the most prominent clusters $(\mathbf{C}_{i1}, \ldots, \mathbf{C}_{ik})$ and noise $(\mathbf{N_i})$. Each noise sample/bubble of $\mathbf{N_i}$ is inserted into the nearest prominent cluster, resulting in the local partition $\mathbf{P}$ which is sent to a Mapper function.

The Mapper function partitions the objects in $\mathbf{X}_i$ according to the clusters of $\mathbf{P}$ extracted by the ReduceByKey function, by assigning each object to the cluster that contains the representative data bubble, or the closest sample representative if data bubbles are not adopted. This process is repeated iteratively for each level of the hierarchy H_i. In the end, the Mapper stores (persists) relevant information about the mapped partition $\mathbf{P}_i$ into the distributed file system, including the inter-clusters and intra-cluster edges.

The second step of MR-HDBSCAN* is used to combine all edges from the stored partial MSTs and inter-cluster edges into a global extended spanning tree (MST_{ext}), i.e, a spanning tree that connects all objects extended with self-edges. Each edge is represented as a vector $[u, v, d_{reach}(u, v)]$, where u and v are the vertices connected by the edge, and $d_{reach}(u, v)$ is the mutual reachability distance between these vertices. This combination is made by a *ReduceByKey* function, illustrated in Fig. 3, where the edges are paired with the key "1" to be reduced to a single result. The edges are combined in descending order of weight (d_{reach}) and stored (persisted) in the DFS. In particular, in Fig. 3, four local MSTs were obtained in different processing units and combined by the *ReduceByKey* task into a single global MST_{ext}, to be stored in the DFS.

After MST_{ext} is constructed, MR-HDBSCAN* builds a hierarchy (as a dendrogram) over all objects of the dataset.

2. The use of data bubbles is optional and user defined in this work, i.e., MR-HDBSCAN* can work directly with samples instead of data bubbles.

This third step can be implemented using one of two approaches: a top-down approach, which is the MapReduce version of the original method described in [24], or a bottom-up approach, based on the Union-Find algorithm [38].

4.3.1 Hierarchy Construction

The idea behind the HDBSCAN* hierarchy construction using the divisive (or top-down) approach [24] consists of assuming that initially all objects are members of the same cluster and then iteratively building the next hierarchical level by splitting clusters until all data is considered noise. Clusters are split by removing from MST_{ext} the edge(s) with the highest weight (d_{reach}) and detecting the resulting connected components as clusters at the new level of the hierarchy.

In a distributed environment, searching for the connected components is challenging, since the vertices of MST_{ext} may be scattered across different processing units. Thus, the top-down approach proposed in this work uses a sophisticated MapReduce method, known as CC-MR [39], to detect connected (sub)components in large scale graphs. The basic idea of the CC-MR method [39] is to iteratively alter growing local parts of the graph until each connected component is presented by a star-like subgraph [40], where all vertices are connected to the central vertex. In order to do so, each node receives an unique ID $\in \mathbb{Z}$. In each iteration, edges are added and deleted such that vertices with larger IDs are assigned to the reachable vertex with smallest ID. In the end, all vertices of the connected components form a star subgraph, with the vertex with the smallest ID as center.

The connected components found by CC-MR are used to determine the clusters at each hierarchical level of MR-HDBSCAN*. When the optional parameter m_{clSize} is considered, a connected component is a valid cluster if its size is equal to or larger than m_{clSize}. If not valid, the connected component is a set of noise objects. When removing the edges with the highest weight, the original cluster can (a) be divided into two new smaller valid clusters, (b) undergo "spurious" fragmentation (shrinking) so it "survives" at the next level of the hierarchy, or (c) become noise. The following information about each sub-component is maintained: (a) the number of vertices (objects), and (b) if it is a spurious component (a set of noise objects), an existing cluster that has "shrunk", or a new cluster that has resulted from a true cluster split [7], [24].

With the connected components, the hierarchy is updated with a new level using a map and a reduce function. The *Mapper* function receives an edge of the connected components, the size of the component it belongs to and whether that component comes from a valid split. Additionally, a data structure with the information of the clusters assessed so far is consulted, i.e., the labels of the clusters, their stability and size, the level of their appearance and disappearance in the hierarchy, their relationship and relevant information [24]. Then, it maps each vertex of the components to the appropriate cluster or labels it as noise, returning the information needed to update the hierarchy. The *ReduceByKey* function combines the information mapped from the vertices and objects to define what happens with the clusters of the current level at the next hierarchical level. Such information allows the function to assess the behavior of each cluster: it may be divided into new clusters, undergo spurious fragmentation (shrinking) or be considered noise. This information is used to build the next level of the hierarchy and connect it to the current level. One important information that is assessed at this point is the stability of the clusters of the current level, which is used by FOSC when one wants to extract the most prominent clusters from the hierarchy. The whole process is repeated to assess the next levels of the hierarchy, until all the connected components resulting from the removals of edges are noise.

Although the top-down approach follows the original proposal of HDBSCAN* [24], constructing each hierarchical level according to this method using MapReduce may be computationally expensive. As the CC-MR transforms the MST_{ext} into star-graphs, a duplicate of the hierarchies must be made and used as input to CC-MR at each level. Additionally, the CC-MR works with digraphs, i.e., edges from the MST_{ext} must be duplicated to represent edges in both directions between connected vertices. Thus, the top-down approach requires additional computing power and storage for these functionalities.

On the other hand, the general idea of the bottom-up (or agglomerative) approach is to consider all vertices of the distributed MST_{ext} as independent (sub)trees and merge them according to an increasing order of weight (d_{reach}) of their connecting edges. Each merger results in a new cluster in a higher hierarchical level. The process is performed iteratively until a single cluster containing every object is obtained at the top of the hierarchy. If the optional parameter $m_{clSize} > 1$ is used, all objects are initially considered as noise. If the number of merged objects is lower than m_{clSize}, the result is considered to be noise, otherwise, a valid cluster is obtained. The latter may arise from noise, from merging two valid clusters, or from "expanding" a valid cluster with the insertion of former "spurious" data points.

In this work, the MapReduce implementation of our bottom-up method is based on the *ReduceByKey* function. This reducer receives all edges of the local MSTs along with the inter-cluster edges and sorts them in ascending order of mutual reachability distances d_{reach}. The main idea behind sorting algorithms for MapReduce frameworks is to sort the edges "locally" in parallel, and then apply MergeSort [41] to the locally sorted edges to obtain a global solution. As an example, one sorting algorithm suited for this type of application is the TimSort [42]. After sorting, each edge in ascending order of d_{reach} is used to build the global hierarchy, by iteratively merging the clusters of the objects it connects at the d_{reach} level of the hierarchy. During the merging process, the stabilities of the clusters are calculated as described in Equation (1).

A computationally efficient way of building and merging MSTs is by using a disjoint set data structure [38]. These data structures manage disjoint sets of data objects, which may represent a "flat" partition of the dataset, allowing the fast merging of these sets using union-find operations. In particular, the parallel union-find data structure introduced in [43] can be adapted to our proposal.

4.4 Cluster Extraction by FOSC

Once the HDBSCAN* hierarchy is obtained, the extraction of the most prominent clusters is performed using the FOSC framework [7], [35], instantiated with the *excess of mass* in

TABLE 1
Characteristics of Artificial and Real Datasets Used in the Experiments

Datasets	Number of objects	Number of attributes	Number of clusters
Gauss1	1×10^6	10	20
Gauss2	3×10^6	10	30
Gauss3	5×10^6	10	50
YearPrediction	515,345	90	89
Poker	1,025,010	11	10
HT Sensor	919,438	11	3
Skin	245,057	4	2
Yellow1	1,600,000	17	6
Yellow2	8,000,000	17	6
HEPMASS	10,500,000	28	2
HIGGs	11,000,000	28	2

TABLE 2
Parameters Setup of the Algorithms Random Blocks, MR-HDBSCAN*$_{bubbles}$ and MR-HDBSCAN*$_{sampling}$

Datasets	Shared Parameters	Random Blocks	MR-HDBSCAN*
	$m_{pts} = m_{clSize}$	p	s (%)
Gauss1	50	10,000	0.9
Gauss2	50	30,000	0.3
Gauss3	50	50,000	0.2
YearPrediction	30	2,783	1.9
Poker	50	10,251	0.9
HT Sensor	30	5,506	1
Skin	40	1,961	4
Yellow1	20	4,194	0.3
Yellow2	20	33,968	0.08
HEPMASS	50	105,000	0.09
HIGGs	30	66,000	0.02

Equation (1) as quality measure. This is a bottom-up dynamic programming optimization procedure. The general idea consists of traversing the cluster hierarchy bottom-up starting from the leaves, comparing the quality (stability) of parent clusters against the aggregated quality of the respective children (sub-clusters), carrying the optimal choices upwards until the root is reached. The method is comprehensively presented and discussed in [7], [35]. Details of its distributed implementation using MapReduce are provided in the Appendix, available in the online supplemental material.

5 Experimental Evaluation

The main goal of the experiments presented in this section is to compare the exact MapReduce version of HDBSCAN*, based on the Random Blocks method, against the two approximate versions proposed in this work, namely, MR-HDBSCAN* with sampling and MR-HDBSCAN* with data bubbles. The source codes for MR-HDBSCAN* are available at www.dc.ufscar.br/∼naldi/source. We compare these three variants based on the quality of their results as well as their overall runtime. The former is assessed by comparing a reference clustering solution (a ground truth) against the optimal non-hierarchical clustering solutions extracted from the hierarchies produced by the compared algorithms, using the FOSC extraction method discussed in Section 4.4 [7]. To do so, each algorithm is run to build an MST in the transformed space of mutual reachability distances w.r.t. m_{pts}, and the resulting MST gives rise to a clustering hierarchy by using the top-down method described in Section 4.3.1, from which a flat solution is extracted by FOSC. For the sake of simplicity, the exact version of HDBSCAN* will be referred to hereafter as *Random Blocks*.

5.1 Datasets

Three artificial and eight real datasets were used in the experiments presented in this section. The three artificial datasets were created using Gaussian mixtures, provided by the MixSim R package [44], with increasing number of objects and clusters. Most of the real datasets are available from the UCI machine learning repository [45], and they are from different domains. Two additional datasets are the *Yelow* 1 and 2 datasets, collected from the yellow taxi trip records of January and February 2018, respectively, both available at the New York City website.[3] The main characteristics of the datasets are summarized in Table 1.

5.2 Experimental Setup

HDBSCAN* has a single compulsory parameter m_{pts}, as well as an *optional* additional parameter, m_{clSize}. Parameter m_{pts} can be shown to be a smoothing factor of the density estimates performed by the algorithm (defined in Section 3), whereas m_{clSize} is a minimum threshold for the number of objects that characterize a valid cluster. When this optional threshold is used, the default setting as proposed by the original authors is $m_{pts} = m_{clSize}$ [7], [24], which is also adopted in this paper. The values for m_{pts} (and m_{clSize}) are determined empirically for each dataset, and presented in Table 2. Those values will also be considered for the top-down method adopted to build the hierarchies.

The parameter p of Random Blocks represents the number of data split partitions for constructing $\binom{p}{2 \times m_{pts}}$ non-overlapping data blocks to be processed in parallel by each processing unit [17]. The size of the data blocks must be small enough to be processed smoothly by the processing unit capacity (τ). Thus, the number of data partitions (p) can be calculated as $p = \frac{n}{n_p}$, where n represents the number of objects in the whole dataset and n_p represents the maximum number of objects any of these partitions should have, which is calculated as $n_p = \frac{\tau}{2 \times m_{pts}}$.

The parameter s represents the number of sampled objects from the data subset used to build any local HDBSCAN* model during MR-HDBSCAN* iterations. The values of s were determined based on the processing capacity (τ) and the sizes of the data subsets to be processed by MR-HDBSCAN*$_{bubbles}$ and MR-HDBSCAN*$_{sampling}$, so that the number of sampled objects does not exceed the capacity of the processing unit.

All experiments were executed in a computer cluster with 10 machines interconnected by a local Gigabyte Ethernet network. Each computer has 8 GB of working memory (RAM) with an AMD FX(tm)-6100 six-core processor and 1 terabyte (1T) of secondary memory. The cluster was configured with Apache Spark 2.1.1 with Hadoop 2.7 over Linux.

3. www.nyc.gov

TABLE 3
Clustering Quality Comparison—*ARI* Values for All Algorithms and *p*-Values of the t-Test between MR-HDBSCAN*$_{bubbles}$ and MR-HDBSCAN*$_{sampling}$

Datasets	Random Blocks	Sampling	Data Bubbles	t-test
	ARI	mean(std) ARI	mean(std) ARI	p-value
Gauss1	**0.881**	0.690(0.001)	0.864(0.000)	5.83e-86
Gauss2	**0.820**	0.588(0.001)	0.759(0.002)	1.40e-81
Gauss3	**0.801**	0.602(0.006)	0.777(0.002)	7.77e-64
YearPrediction	**0.403**	0.301(0.005)	0.388(0.004)	2.95e-48
Poker	**0.310**	0.196(0.005)	0.297(0.001)	1.08e-58
HT Sensor	**0.359**	0.287(0.001)	0.330(0.000)	1.22e-62
Skin	**0.441**	0.360(0.004)	0.425(0.002)	3.43e-51
Yellow1	**0.328**	0.250(0.011)	0.290(0.014)	7.60e-11
Yellow2	**0.426**	0.362(0.024)	0.407(0.011)	1.38e-14
HEPMASS	**0.546**	0.408(0.005)	0.529(0.000)	8.70e-62
HIGGS	**0.235**	0.210(0.008)	0.227(0.006)	2.70e-12

TABLE 4
Runtimes (in Minutes) of All Compared Algorithms and *p*-Values of the t-Test between MR-HDBSCAN*$_{bubbles}$ and MR-HDBSCAN*$_{sampling}$

Datasets	Random Blocks	Sampling	Data Bubbles	t-test
	time	mean(std) time	mean(std) time	p-value
Gauss1	13312.24	**67.35(18.11)**	82.75(31.48)	0.0042
Gauss2	28393.65	**162.07(30.50)**	225.40(23.45)	7.09e-13
Gauss3	1+ month	**115.39(0.076)**	182.05(0.077)	5.75e-13
YearPrediction	11622.61	**106.57(17.62)**	109.89(21.30)	0.2147
Poker	28955.89	**52.81(5.84)**	97.06(35.32)	7.23e-11
HT Sensor	31450.89	**42.54(4.74)**	82.07(25.89)	7.23e-14
Skin	1743.93	**21.14(4.99)**	60.19(26.00)	2.13e-12
Yellow1	1+ month	**106.44(8.44)**	265.50(17.15)	2.03e-45
Yellow2	1+ month	**474.69(43.26)**	776.67(7.40)	2.95e-38
HEPMASS	1+ month	**385.67(26.62)**	695.29(28.01)	1.82e-40
HIGGS	1+ month	**752.17(48.05)**	887.54(46.18)	2.71e-19

5.3 Quality Evaluation

Each dataset chosen for our experiments has a "reference solution", i.e., a flat partition based on the known categories of the dataset, which can be used as ground truth to evaluate results from clustering algorithms. These reference partitions can then be compared with the flat clustering solutions extracted from the hierarchies by the compared algorithms (FOSC extraction—see Section 4.4). For the comparison of the extracted solutions against the ground truth partitions, we use the well known *Adjusted Rand Index* (ARI) [46]; objects left unclustered as noise in the evaluated clustering solutions are considered as singletons (a cluster with a single object) during the ARI computation. The maximum ARI value is 1, which is only achieved when the result under evaluation is identical to the ground truth partition. The expected value of the ARI for a random solutions is 0 (due to adjustment for chance).

The clustering quality comparison was conducted for each dataset described in Section 5.1. Random blocks is the exact MapReduce version of HDBSCAN*, which has deterministic behavior, resulting in a single hierarchical clustering for a given dataset. However, MR-HDBSCAN*$_{sampling}$ and MR-HDBSCAN*$_{bubbles}$ are based on random sampling of the data, which may incur variations in their results. For this reason, these approximations were run 45 times for each dataset, resulting in 45 flat partitions extracted from the corresponding clustering hierarchies. The ARI value of the partition from Random blocks and the mean and standard deviation (in brackets) of the ARI evaluations over the MR-HDBSCAN*$_{sampling}$ and MR-HDBSCAN*$_{bubbles}$ are presented in Table 3. The best results are shown in bold. Additionally, the paired t-test was applied to statistically assess the differences in the results from MR-HDBSCAN*$_{sampling}$ and MR-HDBSCAN*$_{bubbles}$, with 95 percent confidence level [47]. The p-values resulting from the tests are also shown in Table 3. We can see from the very low p-values that the null hypothesis is rejected in all datasets, and this would still be the case even if a much higher confidence level was adopted. The test supports the conclusion that MR-HDBSCAN*$_{bubbles}$ achieves superior quality results when compared to MR-HDBSCAN*$_{sampling}$ for all the chosen datasets, but its ARI values are lower than those of the "exact" Random Blocks approach.

The results in Table 3 show that Random Blocks achieves the best quality results, which is not surprising as it is an exact version of the original HDBSCAN*. In contrast, whenever sampling or summarization is used, information loss occurs. However, the use of bubbles as a more sophisticated data summarization technique partially mitigates this issue, incurring much less noticeable losses. Indeed, MR-HDBSCAN*$_{bubbles}$ produces results that are very close to the exact results obtained by Random Blocks. When comparing the ordinary sampling performed by MR-HDBSCAN*$_{sampling}$ with the data bubbles summarization performed by MR-HDBSCAN*$_{bubbles}$, the latter preserved most of the information and structure of clusters. It also incurred less variability of the results, which shows as smaller values of standard deviation for most datasets.

In absolute terms, the variability of results observed from both approximate versions of MR-HDBSCAN* are very small, with standard deviations of ARI values lower than 0.025 for the sampling variant and lower than 0.015 for the data bubbles variant. Since these variants are orders of magnitude faster than the exact method, they are still computationally advantageous even if one wants to perform multiple runs of the algorithms from which some unsupervised model selection procedure could be applied to select the best out of these runs. However, the small variability in the results observed from Table 3 suggests that this procedure may not be required in practice.

Finally, it is worth remarking that, except for the Gauss1 dataset, all partitions extracted using the optimal extraction method FOSC discussed in Section 4.4 contain clusters from multiple levels of the corresponding HDBSCAN* hierarchies. Those partitions cannot be obtained by partitioning methods, such as DBSCAN and other related algorithms.

5.4 Runtime Evaluation

As previously discussed, an exact MapReduce version of HDBSCAN* is computationally burdensome due to pairwise distance calculations. One of the goals of this work is to show how much data abstractions can improve the computational performance of HDBSCAN* in MapReduce architectures. In Table 4 we present the computational time needed to execute Random Blocks as well as the mean computational time (followed by standard deviation) of MR-HDBSCAN*$_{sampling}$ and MR-HDBSCAN*$_{bubbles}$ executions. Additionally, in order

TABLE 5
Runtimes (in Minutes) of Different Hierarchy Construction Methods ($\mathrm{div}_{hierarchy}$ and $\mathrm{agg}_{hierarchy}$)

Datasets	$\mathrm{div}_{hierarchy}$ mean (std)	$\mathrm{agg}_{hierarchy}$ mean (std)
Gauss1	206.87 (28.57)	**136.38 (10.59)**
Gauss2	356.00 (21.53)	**266.95 (2.36)**
Gauss3	520.17 (17.52)	**402.20 (46.51)**
YearPrediction	106.59 (5.68)	**67.65 (4.52)**
Poker	212.72 (11.35)	**135.37 (12.94)**
HT Sensor	188.90 (5.80)	**124.82 (8.61)**
Skin	51.21 (2.80)	**32.95 (2.11)**
Yellow1	505.00 (33.46)	**486.00 (38.32)**
Yellow2	778.09 (6.87)	**734.00 (12.85)**
HEPMASS	692.87 (27.18)	**532.63 (8.73)**
HIGGS	938.60 (3.23)	**875.00 (14.18)**

to assess the statistical significance of the differences between the runtimes of MR-HDBSCAN*$_{sampling}$ and MR-HDBSCAN*$_{bubbles}$, a paired t-test was applied with 95 percent confidence level [47]. The p-value results are also shown in Table 4. Clearly, sampling and data bubbles allowed the approximate algorithms to be much faster (up to 700 times, 240 times on average) than Random Blocks. Although the standard deviation of their runtimes is relatively large when compared to the mean value, it is still very low when compared with the runtime of the Random Blocks approach. MR-HDBSCAN*$_{sampling}$ has the lowest average execution times, closely followed by MR-HDBSCAN*$_{bubbles}$. Experiments using the random blocks approach that executed for more than one month in our original cluster, described at the beginning of Section 5.2, were interrupted and re-executed in a system with a far superior computational power to achieve results in an acceptable running time for this publication. These results are described as "1+ month" in Table 4.

Considering the t-test results, the only dataset for which the null hypothesis was not rejected with 95 percent of significance was *YearPrediction*, where both algorithms had similar performance. In the majority of the datasets, building data bubbles significantly increased the computational times in relative terms. When compared to the runtimes of the exact, Random Blocks approach, however, MR-HDBSCAN*$_{bubbles}$ still is much much faster, at the price of only a small loss in quality of the result, suggesting that this algorithm represents the best trade-off between quality and computational performance for many practical application scenarios involving large datasets.

5.5 Top-Down and Bottom-Up Hierarchy Construction Evaluation

The compared MapReduce variants of HDBSCAN* build an MST in the transformed space of mutual reachability distances, which must be translated into a clustering hierarchy of the dataset. Here, we compare the top-down approach (denoted as $\mathrm{div}_{hierarchy}$) and the bottom-up approach (denoted as $\mathrm{agg}_{hierarchy}$) to build the HDBSCAN* hierarchy. Since both approaches build the very same hierarchy, the comparison is made in terms of computational time only. They were both applied to the 45 MSTs produced by MR-HDBSCAN*$_{bubbles}$. The mean and standard deviation of their runtimes are presented in Table 5.

Note that, on average, $\mathrm{agg}_{hierarchy}$ outperformed $\mathrm{div}_{hierarchy}$ for all analyzed datasets. This is because, unlike the top-down method, the bottom-up approach does not need to perform a number of iterations, until convergence, to find connected sub-components at each hierarchical level. If the process to find the sub-components requires several iterations, the execution time of $\mathrm{div}_{hierarchy}$ can increase considerably. In contrast, $\mathrm{agg}_{hierarchy}$, rather than performing several iterations to find sub-components at each given hierarchical level, joins the sub-components in a single iteration per hierarchical level using efficient *union-find* strategies. Another aspect that increases the runtime of $\mathrm{div}_{hierarchy}$ is related to many duplicated edges to find connected sub-components. In summary, the bottom-up approach ($\mathrm{agg}_{hierarchy}$) should be preferred to the top-down approach ($\mathrm{div}_{hierarchy}$).

6 Conclusions and Future Work

In this paper, we described an exact as well as two computationally efficient, approximate MapReduce implementations of HDBSCAN*. By definition, the exact version, which is based on a random blocks approach, is capable of constructing in a distributed and parallel fashion the same hierarchy that would result from a centralized HDBSCAN* implementation running in a single machine with access to the whole dataset. As expected, this is the result with the highest quality in our experiments. The exact MapReduce computation of this result is, however, computationally prohibitive. As for the approximate variants, which are based on a recursive sampling scheme, our experiments show that applying density-based hierarchical clustering directly over data subsamples may lead to inferior quality, although the computational time improved vastly (hundreds of times, for the datasets in our experiments). To mitigate quality loss, we applied data bubbles as a data pre-processing step.

With data bubbles, the quality of the obtained results became competitive with the exact version of the algorithm, while the runtimes were much lower, comparable to those obtained by using only subsamples. These results allow to conclude that approximate versions of HDBSCAN* are a viable choice for large scale distributed frameworks and datasets.

In addition, two different approaches for building the HDBSCAN* hierarchy were compared. Although the top-down approach may be more intuitive, the bottom-up approach is a more natural choice for the MapReduce framework, as map functions can deal with a large granularity of objects and reduce functions work analogously to merge functions. Such characteristics reflected in the superior performance obtained by the bottom-up implementation.

As future work, the parallelization of HDBSCAN* for shared memory architectures may be an interesting choice for applications involving amounts of data that could be handled in the main memory of a single machine with multiple processing units. In this case, a MapReduce implementation may be too burdensome as the redundancies and safety mechanisms may not be required, so

improvements in the computational time of the original HDBSCAN* may be achieved.

ACKNOWLEDGMENTS

The authors would like to thank NSERC (Discovery Grant Program), CAPES, CNPq (PQ Grant) 304137/2013-8, CNPq (PVE CSF Grant) 400772/2014-0, FAPESP and FAPEMIG for the grants and research fundings.

REFERENCES

[1] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988.

[2] H.-P. Kriegel, P. Krger, J. Sander, and A. Zimek, "Density-based clustering," *Wiley Interdisciplinary Rev.: Data Mining Knowl. Discovery*, vol. 1, no. 3, pp. 231–240, 2011. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.30

[3] M. Ester, H. Peter Kriegel , J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. 2nd Int. Conf. Knowl. Discovery Data Mining*, 1996, pp. 226–231.

[4] V. Kumar, P.-N. Tan, and M. Steinbach, *Introduction to Data Mining*, 2nd ed., London, U.K.: Pearson, 2013.

[5] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "OPTICS: Ordering points to identify the clustering structure," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1999, pp. 49–60. [Online]. Available: http://doi.acm.org/10.1145/304182.304187

[6] S. Brecheisen, H.-P. Kriegel, P. Krger, and M. Pfeifle, "Visually mining through cluster hierarchies," in *Proc. SIAM Int. Conf. Data Mining*, 2004, pp. 400–411. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9781611972740.37

[7] R. J. G. B. Campello, D. Moulavi, A. Zimek, and J. Sander, "Hierarchical density estimates for data clustering, visualization, and outlier detection," *ACM Trans. Knowl. Discov. Data*, vol. 10, no. 1, pp. 5:1–5:51, Jul. 2015. [Online]. Available: http://doi.acm.org/10.1145/2733381

[8] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. 6th Conf. Symp. Opearting Syst. Des. Implementation - Volume 6*, 2004, pp. 10–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251254.1251264

[9] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[10] *Companies & Organizations Powered by Hadoop*, [Online]. Available: https://wiki.apache.org/hadoop/PoweredBy. Accessed on: Jan. 18, 2017.

[11] *Companies & Organizations Powered by Spark*, [Online]. Available: http://spark.apache.org/powered-by.html. Accessed on: Jan. 18, 2017.

[12] T. White, *Hadoop: The Definitive Guide*, 4th ed., Sebastopol, CA, USA: O'Reilly Media, Inc., 2015.

[13] M. Hamstra, H. Karau, M. Zaharia, A. Konwinski, and P. Wendell, *Learning Spark: Lightning-Fast Big Data Analytics*. Sebastopol, CA, USA: O'Reilly Media, Incorporated, 2015. [Online]. Available: http://books.google.com.br/books?id=9IPUlgEACAAJ

[14] S. Owen, R. Anil, T. Dunning, and E. Friedman, *Mahout in Action*. Greenwich, CT, USA: Manning Publications Co., 2011.

[15] S. Wang and H. Dutta, "PARABLE: A PArallel RAndom-partition Based HierarchicaL ClustEring Algorithm for the MapReduce Framework," *The Comput. J.*, vol. 16, pp. 30–34, 2011.

[16] C. Jin, M. M. A. Patwary, A. Agrawal, W. Hendrix, W.-K. Liao, and A. Choudhary, "DISC: A distributed single-linkage hierarchical clustering algorithm using MapReduce," in *Proc. 4th Int. SC Workshop Data Intensive Comput. Clouds*, 2013, pp. 1–10.

[17] T. I. Syed, "Parallelization of Hierarchical Density-Based Clustering using MapReduce," Master's thesis, Department of Computing Science, Univ. Alberta, Edmonton, Alberta, 2014.

[18] J. Zhou and J. Sander, "Data bubbles for non-vector data: Speeding-up hierarchical clustering in arbitrary metric spaces," in *Proc. 30th Int. Conf. Very Large Data Bases*, 2003, pp. 452–463.

[19] C. F. Olson, "Parallel algorithms for hierarchical clustering," *Parallel Comput.*, vol. 21, pp. 1313–1325, 1995.

[20] H.-R. Tsai, S.-J. Horng, S.-S. Lee, S.-S. Tsai, and T.-W. Kao, "Parallel hierarchical clustering algorithms on processor arrays with a reconfigurable bus system," *Pattern Recognit.*, vol. 30, no. 5, pp. 801–815, 1997.

[21] C.-H. Wu, S.-J. Horng, and H.-R. Tsai, "Efficient parallel algorithms for hierarchical clustering on arrays with reconfigurable optical buses," *J. Parallel Distrib. Comput.*, vol. 60, no. 9, pp. 1137–1153, 2000.

[22] S. Rajasekaran, "Efficient parallel hierarchical clustering algorithms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 6, pp. 497–502, Jun. 2005.

[23] R. Sibson, "SLINK: An optimally efficient algorithm for the single-link cluster method," *The Comput. J.*, vol. 16, pp. 30–34, 1973.

[24] R. J. G. B. Campello, D. Moulavi, and J. Sander, "Density-based clustering based on hierarchical density estimates," in *Proc. Pacific-Asia Conf. Knowl. Discovery Data Mining*, 2013, pp. 160–172.

[25] J. L. Bentley, B. W. Weide, and A. C. Yao, "Optimal expected-time algorithms for closest point problems," *ACM Trans. Math. Softw.*, vol. 6, no. 4, pp. 563–580, Dec. 1980. [Online]. Available: http://doi.acm.org/10.1145/355921.355927

[26] V. Olman, F. Mao, H. Wu, and Y. Xu, "Parallel clustering algorithm for large data sets with applications in bioinformatics," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 6, no. 2, pp. 344–352, Apr.–Jun. 2009.

[27] W. Hendrix, M. Ali Patwary, A. Agrawal, W. Keng Liao, and A. Choudhary, "Parallel hierarchical clustering on shared memory platforms," in *Proc. 19th Int. Conf. High Perform. Comput.*, Dec. 2012, pp. 1–9.

[28] W. Hendrix, D. Palsetia, M. Ali Patwary, A. Agrawal, W. keng Liao, and A. Choudhary, "A scalable algorithm for single-linkage hierarchical clustering on distributed-memory architectures," in *Proc. IEEE Symp. Large-Scale Data Anal. Vis.*, Oct. 2013, pp. 7–13.

[29] Y. He, H. Tan, W. Luo, H. Mao, D. Ma, S. Feng, and J. Fan, "MR-DBSCAN: An efficient parallel density-based clustering algorithm using MapReduce," in *Proc. IEEE 17th Int. Conf. Parallel Distrib. Syst.*, Dec. 2011, pp. 473–480.

[30] Y. Kim, K. Shim, M.-S. Kim, and J. S. Lee, "DBCURE-MR: An efficient density-based clustering algorithm for large data using MapReduce," *Inf. Syst.*, vol. 42, pp. 15–35, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306437913001634

[31] R. Gulati, "Efficient parallel DBSCAN algorithms for bigdata using MapReduce," Master's thesis, Comput. Sci. Eng. Dept., Thapar Univ., Patiala, Punjab, 2016.

[32] G. Gupta, A. Liu, and J. Ghosh, "Automated hierarchical density shaving: A robust automated clustering and visualization framework for large biological data sets," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 7, no. 2, pp. 223–237, Apr. 2010. [Online]. Available: http://dx.doi.org/10.1109/TCBB.2008.32

[33] T. Pei, A. Jasra, D. Hand, A.-X. Zhu, and C. Zhou, " DECODE: A new method for discovering clusters of different densities in spatial data," *Data Mining Knowl. Discovery*, vol. 18, no. 3, pp. 337–369, 2009. [Online]. Available: http://dx.doi.org/10.1007/s10618-008-0120-3

[34] W. Stuetzle and R. Nugent, "A generalized single linkage method for estimating the cluster tree of a density," *J. Comput. Graph. Statist.*, vol. 19, no. 2, pp. 397–418, 2010. [Online]. Available: https://doi.org/10.1198/jcgs.2009.07049

[35] R. J. G. B. Campello, D. Moulavi, A. Zimek, and J. Sander, "A framework for semi-supervised and unsupervised optimal extraction of clusters from hierarchies," *Data Mining Knowl. Discovery*, vol. 27, no. 3, pp. 344–371, 2013.

[36] H. Karloff, S. Suri, and S. Vassilvitskii, "A model of computation for MapReduce," in *Proc. 21st Annu. ACM-SIAM Symp. Discr. Algorithms*, 2010, pp. 938–948. [Online]. Available: http://dl.acm.org/citation.cfm?id=1873601.1873677

[37] M. M. Breunig, H. P. Kriegel, P. Kröger, and J. Sander, "Data bubbles: Quality preserving performance boosting for hierarchical clustering," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2001, pp. 79–90. [Online]. Available: http://dx.doi.org/10.1145/375663.375672

[38] Z. Galil and G. F. Italiano, "Data structures and algorithms for disjoint set union problems," *ACM Comput. Surv.*, vol. 23, no. 3, pp. 319–344, Sep. 1991. [Online]. Available: http://doi.acm.org/10.1145/116873.116878

[39] T. Seidl, B. Boden, and S. Fries, "CC-MR – Finding connected components in huge graphs with MapReduce," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*, 2012, pp. 458–473.

[40] C. Godsil and G. F. Royle, *Algebraic Graph Theory*. Berlin, Germany: Springer Science & Business Media, 2013, vol. 207.
[41] R. Sedgewick and K. Wayne, *Algorithms*, 4th ed. Boston, MA, USA: Addison-Wesley Professional, 2011.
[42] P. McIlroy, "Optimistic sorting and information theoretic complexity," in *Proc. 4th Annu. ACM-SIAM Symp. Discr. Algorithms*, 1993, pp. 467–474. [Online]. Available: http://dl.acm.org/citation.cfm?id=313559.313859
[43] F. Manne and M. M. A. Patwary, *A Scalable Parallel Union-Find Algorithm for Distributed Memory Computers*. Berlin, Germany: Springer, 2010, pp. 186–195. [Online]. Available: https://doi.org/10.1007/978-3-642-14390-8_20
[44] V. Melnykov, W.-C. Chen, and R. Maitra, "MixSim: An R package for simulating data to study performance of clustering algorithms," *J. Statistical Softw.*, vol. 51, no. 12, pp. 1–25, 2012. [Online]. Available: http://www.jstatsoft.org/v51/i12/
[45] K. Bache and M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml
[46] L. Hubert and P. Arabie, "Comparing partitions," *J. Classification*, vol. 2, no. 1, pp. 193–218, 1985.
[47] M. J. Panik, *Testing Statistical Hypotheses*. Hoboken, NJ, USA: John Wiley and Sons, Inc., 2012, pp. 184–216. [Online]. Available: http://dx.doi.org/10.1002/9781118309773.ch10

Joelson Antônio dos Santos received the bachelor's degree in information systems from the Federal University of Viçosa, Rio Paranaba - MG campus, 2014, and the MSc degree in computer science and computational mathematics from the Institute of Mathematical and Computer Sciences (ICMC), University of São Paulo - USP, 2018. He is currently a development analyst at CSDbr in São Carlos-SP.

Talat Iqbal Syed received the bachelor of technology from Anna University, MIT Campus, India, 2009, and the MSc degree in computing science from the Department of Computing Science, University of Alberta, Canada, 2015. He is currently an applied machine learning researcher at Alberta Machine Intelligence Institute, Canada.

Murilo C. Naldi received the BSc, MSc and PhD degrees in computer science from the University of São Paulo (USP), Brazil, in 2004, 2006, and 2011, respectively. He is currently an associate professor with the Department of Computer Science, Federal University of São Carlos. His main research interests in data mining, data science, machine learning, and evolutionary computation.

Ricardo J. G. B. Campello received the BSc degree in electronics engineering from the State University of São Paulo, Brazil, in 1994, and the MSc and PhD degree in electrical engineering from the State University of Campinas, Brazil, in 1997 and 2002, respectively. He is currently a full professor with the School of Mathematical and Physical Sciences, University of Newcastle, Australia. His current research interests include data mining, data science, machine learning, and computational intelligence.

Joerg Sander received the MS and PhD degree in computer science from the University of Munich, Germany. He is currently a full professor in computing science at the University of Alberta, Canada. His research interests include knowledge discovery in databases, clustering, spatial data mining, similarity search.