Big Data Research 11 (2018) 44–53

Contents lists available at ScienceDirect

Big Data Research

www.elsevier.com/locate/bdr

# Variations on the Clustering Algorithm BIRCH[☆]

Boris Lorbeer[*], Ana Kosareva, Bersant Deva, Dženan Softić, Peter Ruppel, Axel Küpper

*Service-centric Networking, Telekom Innovation Laboratories, Technische Universität Berlin, Germany*

ARTICLE INFO

*Article history:*
Received 17 January 2017
Received in revised form 7 July 2017
Accepted 25 September 2017
Available online 18 October 2017

ABSTRACT

Clustering algorithms are recently regaining attention with the availability of large datasets and the rise of parallelized computing architectures. However, most clustering algorithms suffer from two drawbacks: they do not scale well with increasing dataset sizes and often require proper parametrization which is usually difficult to provide. A very important example is the cluster count, a parameter that in many situations is next to impossible to assess. In this paper we present A-BIRCH, an approach for automatic threshold estimation for the BIRCH clustering algorithm. This approach computes the optimal threshold parameter of BIRCH from the data, such that BIRCH does proper clustering even without the global clustering phase that is usually the final step of BIRCH. This is possible if the data satisfies certain constraints. If those constraints are not satisfied, A-BIRCH will issue a pertinent warning before presenting the results. This approach renders the final global clustering step of BIRCH unnecessary in many situations, which results in two advantages. First, we do not need to know the expected number of clusters beforehand. Second, without the computationally expensive final clustering, the fast BIRCH algorithm will become even faster. For very large data sets, we introduce another variation of BIRCH, which we call MBD-BIRCH, which is of particular advantage in conjunction with A-BIRCH but is independent from it and also of general benefit.

© 2017 The Authors. Published by Elsevier Inc. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

## 1. Introduction

Clustering is an unsupervised learning method that groups a set of given data points into well separated subsets. Two prominent examples of clustering algorithms are k-means, see Macqueen [10], and the *expectation maximization* (EM) algorithm, see Dempster et al. [6]. This paper addresses two issues with clustering: (1) clustering algorithms usually do not scale well and (2) most algorithms require the number of clusters (cluster count) as input. The first issue is becoming more and more important. For applications that need to cluster, for example, millions of documents, huge image or video databases, or terabytes of sensor data produced by the Internet of Things, scalability is essential. The second issue severely reduces the applicability of clustering in situations where the cluster count is very difficult to predict, such as for data exploration, feature engineering, and document clustering.

An important clustering method is *balanced iterative reducing and clustering using hierarchies*, or BIRCH, which was introduced by Zhang et al. [19] and is one of the fastest clustering algorithms available. It outperforms most of the other clustering algorithms by up to two orders of magnitude. Thus, BIRCH already solves the first issue mentioned above. However, to achieve sufficient clustering quality, BIRCH requires the cluster count as input, therefore failing to solve the second issue. This paper describes a method to use BIRCH without having to provide the cluster count, yet preserving cluster quality and speed. This is achieved as follows. We first remove the global clustering step that is done at the end of BIRCH, since this is slow and often requires extra parameters like e.g. the cluster count as input. Then, by analyzing the remaining part of the BIRCH algorithm, which we call *tree-BIRCH*, we identify three ways in which tree-BIRCH can go wrong: *cluster splitting*, *cluster combining*, and *supercluster splitting*. This knowledge then enables us to improve tree-BIRCH and compute an optimal threshold parameter from the data. With the resulting algorithm, the user has to provide neither any parameters for the final clustering step like e.g. the cluster count, since there is no final clustering step anymore, nor the threshold parameter, since this is computed automatically.

The threshold parameter is computed from two attributes of the data, the maximum cluster radius $R_{max}$ and the minimum distance $D_{min}$ between clusters. Often, those attributes are already known. For situations in which this is not the case, we describe one possible procedure of obtaining them. Following an idea in

☆ This article belongs to Big Data & Neural Network.
* Corresponding author.
*E-mail addresses:* lorbeer@tu-berlin.de (B. Lorbeer), ana.kosareva@tu-berlin.de (A. Kosareva), bersant.deva@tu-berlin.de (B. Deva), softic.dzenan@gmail.com (D. Softić), peter.ruppel@tu-berlin.de (P. Ruppel), axel.kuepper@tu-berlin.de (A. Küpper).

https://doi.org/10.1016/j.bdr.2017.09.002
2214-5796/© 2017 The Authors. Published by Elsevier Inc. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Bach and Jordan [2], we propose to learn these attributes from representative data.

Above, supercluster splitting was mentioned as one of the problems with tree-BIRCH. This led us to devise another extension of BIRCH, *MBD-BIRCH*, which is considerably reducing supercluster splitting, though at the expense of speed. However, MBD-BIRCH, too, is still much faster than most of the other clustering algorithms.

Our approach aims at datasets drawn from two-dimensional isotropic Gaussian distributions which are typical when dealing with, for example, geospatial data.

The paper is organized as follows. In Section 2 we review previous work. To fix notation, we give a short overview of the BIRCH algorithm in Section 3. The fundamental ideas of the paper are explained in Section 4. This is followed by the details of A-BIRCH, a new algorithm we developed, in Sections 5 and 6. Sections 7 and 8 focus on issues arising when dealing with very large data and how to deal with them using another new algorithm, MBD-BIRCH. The described algorithms are evaluated in Section 9. Finally, after the description of future work in Section 10 the paper is summarized in Section 11.

## 2. Related work

Clustering algorithms usually do not scale well, because often they have a complexity of $O(N^2)$ or $O(NM)$, where $N$ is the number of data points and $M$ is the cluster count. Scalability is typically achieved by parallelization of the algorithm in compute clusters, such as Mahout's k-means clustering in Owen et al. [12] or Spark's distributed versions of k-means, EM clustering, and power iteration, see Meng et al. [11]. Other parallelization attempts use the graphics processing unit (GPU). This has been done for k-means in Zechner and Granitzer [18], for EM clustering in Kumar et al. [9], and for many others. The bottleneck here is the relatively slow connection between host and device memory if the data does not fit into device memory.

The second issue we are concerned with is the identification of the cluster count. A standard approach is to use one of the clustering algorithms that require the cluster count to be input as a parameter, then run it for each count $k$ in a set of likely values. Then, the "elbow method" from Sugar [16] is used to determine the optimal number $k$. For probabilistic models, one can apply information criteria such as the *Akaike Information Criterion* AIC as described in Akaike et al. [1] or the *Bayesian Information Criterion* BIC following Schwarz [15] to rate the different clustering results, see, for example, Zhou and Hansen [20]. However, all these methods increase the computation time considerably, especially if there is not enough prior information to keep the range of possible cluster counts small. Some clustering algorithms find the number of clusters directly, without being required to run the algorithm for all possible counts. Two of the more well-known examples are *Density-based spatial clustering of applications with noise* (DBSCAN) by Ester et al. [7] and *Gap Statistic*, Tibshirani et al. [17]. Besides our approach, there are also some other attempts to improve the clustering quality of BIRCH by changing the algorithm itself, for instance in Ismael et al. [8] with non-constant thresholds, in Burbeck and Nadjm-Tehrani [3] with two different global thresholds, or by using DBSCAN on each tree level to reduce noise in Dash et al. [5]. However, while sometimes improving the quality, those approaches slow BIRCH down and still require the cluster count as input.

## 3. BIRCH

We shortly describe BIRCH, mainly to fix notations. For details, see Zhang et al. [19]. BIRCH requires three parameters: the branching factor $Br$, the threshold $T$, and the cluster count $k$. While the data points are entered into BIRCH, a height-balanced tree, the *cluster features tree*, or CF tree, of hierarchical clusters is built. Each node represents a cluster in the cluster hierarchy, intermediate nodes are superclusters and the leaf nodes are the actual clusters. The branching factor $Br$ is the maximum number of children a node can have. This is a global parameter. Every node contains the most important information of the belonging cluster, the *cluster features* (CF). From those, the cluster centers $C_i = 1/n_i \sum_j^n x_{ij}$, where $\{x_{ij}\}_{j=1}^n$ are the elements of the $i$th cluster, and the cluster radii $R_i = \sqrt{1/n_i \sum_j^n (x_{ij} - C_i)^2}$ can be computed for each cluster. Every new point starts at the root and recursively walks down the tree, always entering the subcluster with the nearest center until the walk ends at a certain leaf node.

Once arrived at a leaf, the new point is added to this leaf cluster, provided this would not increase the radius of the cluster beyond the threshold $T$. Otherwise a new cluster is created with the new point as its only member. This way, the threshold parameter controls the size of the clusters.

If the creation of a new cluster leads to more than $Br$ child nodes of the parent, the parent is split. To ensure that the tree stays balanced, the nodes further above might need to be split recursively. Once all points are submitted to BIRCH, the centers of the leaf clusters are, in the *global clustering* phase, entered into a clustering algorithm such as agglomerative clustering or k-means which is given as parameter the cluster count $k$. This last step improves the cluster quality by merging neighboring clusters.

In this paper, when it is necessary to distinguish between the BIRCH algorithm with a final global clustering phase and the one without, we will call the former *full-BIRCH* and the latter *tree-BIRCH*.

## 4. Concept

Tree-BIRCH is very fast. It clusters 100,000 points into 1000 clusters in 4 seconds, on a 2,9 GHz Intel Core i7, using scikit-learn, see Pedregosa et al. [13]. The k-means implementation of the same library needs over two minutes to complete the same task on the same architecture. Furthermore, tree-BIRCH doesn't require the cluster count as input, which in full-BIRCH is only needed for the global clustering phase. In addition, tree-BIRCH can be used as an online algorithm. Full-BIRCH cannot be used online, since it needs an end at which the global clustering could be run, but online algorithms never end. However, tree-BIRCH usually suffers from bad clustering quality. Therefore, we focus on improving the clustering quality of tree-BIRCH.

In this paper, we consider the problem of clustering datasets that are samples of a mixture of two-dimensional isotropic Gaussians. In particular, we present a method that automatically chooses an optimal threshold parameter for tree-BIRCH. As a prerequisite, we now describe three causes of erroneous clustering by tree-BIRCH and how they can, in principle, be eliminated.

Zhang et al. [19] noted that the CF-tree depends on the order in which the data is entered. If the points of a single cluster are entered in the order of increasing distance from the center, tree-BIRCH is more likely to return just one cluster than if the first two points are from opposite sides of the cluster. In the latter case, the single cluster is likely to split, a situation we will refer to as *cluster splitting* (Fig. 1A).

Next, consider two neighboring clusters. If the first two points are from opposite clusters but still near each other, they could be collected into the same cluster, given the threshold is large enough, which we refer to as *cluster combining* (Fig. 1B).

Cluster combining often co-occurs with cluster splitting. To reduce splitting of a single cluster, the threshold parameter of tree-

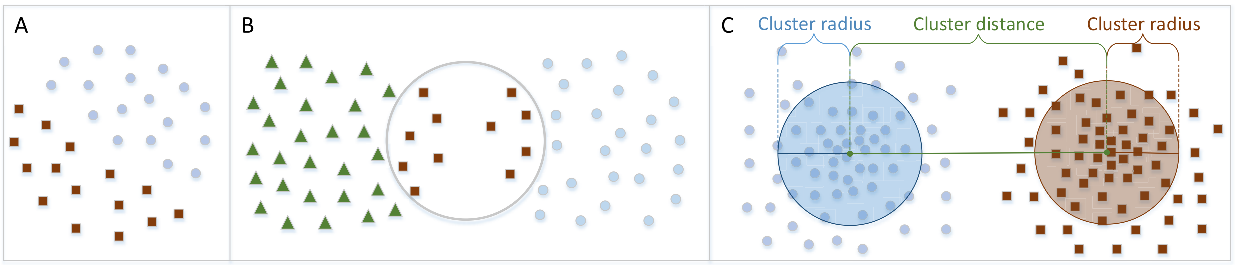

**Fig. 1.** Cluster splitting (A), cluster combining where the combining cluster is circled (B), depiction of cluster radius and cluster distance (C). Different forms and colors of the observations correspond to different clusters they belong to.

BIRCH has to be increased, whereas a decreased threshold parameter reduces cluster combining. Datasets with a large ratio of cluster distance (the distance between the cluster centers) to cluster radius are less likely to produce such errors (Fig. 1C).

The third source of deficient clustering cannot be influenced by the choice of the threshold. This happens if a cluster overlaps with two regions belonging to two non-leaf nodes in the CF-tree. We call this type of error *supercluster splitting*. It will be described in Section 7. This does not occur if the tree is flat, i.e. all leaves are children of the root. A flat tree can be obtained by choosing the branching factor $Br$ to be larger than the maximal possible number of clusters. However, the tree structure is important for the performance if the number $k$ of clusters is large, because the part of the algorithm that searches the cluster that a new point belongs to is logarithmic in $k$.

In the next section we will deduce a formula for the optimal threshold given two conditions: first we presume that the branching factor is chosen large enough for the BIRCH tree to be flat, and second, that all the clusters have approximately the same number of elements. In Sections 6, we lift the second condition, i.e., clusters can have different element counts. Then, in Section 7 and 8, we consider supercluster splitting and develop an extension of tree-BIRCH that considerably reduces it.

## 5. Automatic estimation of the tree-BIRCH threshold for clusters of same element count

As has already been pointed out, this paper focuses on the problem of clustering samples from a mixture of two-dimensional isotropic Gaussians and we want to describe a method which automatically finds the optimal threshold parameter for tree-BIRCH. In the current section this will be done for the special case that all clusters have approximately the same number of elements. Also, for now the BIRCH tree is chosen to be flat.

The goal is to obtain the optimal threshold parameter as a function of the maximum radius $R_{max}$ of the clusters and the minimum cluster distance $D_{min}$, which we both presume to exist. The underlying assumption is that those two values are either already known or easy to obtain.

One typical procedure for finding $R_{max}$ and $D_{min}$, if they are not known yet, is the following: We will presume that there exists a small but representative subset of the data that has the same maximum cluster radius $R_{max}$ and minimum cluster distance $D_{min}$ as the full dataset. On this small dataset, Gap Statistic is applied to obtain the cluster count $k$. This $k$ in turn is given to k-means to produce a clustering of the subset, which finally yields the two values $R_{max}$ and $D_{min}$.

For the determination of $R_{max}$ and $D_{min}$ one could also use any other clustering algorithm that finds the cluster centers and radii without requiring the cluster count $k$ as an input. However, Gap Statistic is chosen here, due to its high precision.

Our approach in this paper is mainly heuristic. For each scenario of interest we run tree-BIRCH many times, always under the same conditions but each time with newly sampled data. From the frequencies of cluster splitting and cluster combining we deduce their probabilities. The number of repetitions necessary to obtain a required accuracy can be obtain from the well-known theory of confidence intervals, see e.g. Casella and Berger [4]. We have used the implementation in R Core Team [14]. Thus, for our error probability estimate at 0.01 to have a confidence interval of roughly $\pm 0.002$, we need 10,000 repetitions, which is what we used in all our experiments.

**Avoiding Cluster Splitting.** We create many clusters containing the same number of elements $n$ by sampling from a single isotropic two dimensional Gaussian probability density function. The units are chosen such that the radius $R$ of this cluster will be one. Then, tree-BIRCH is applied with the same threshold $T$ to each of those datasets. After each iteration, we determine whether tree-BIRCH returns the correct number of clusters, namely one. From this, we assess if the error probability estimate for tree-BIRCH is less than 0.01, or one percent. We also investigate the impact of varying the number of elements $n$ in the cluster on the resulting error probability. Finally, we repeat all the above for several thresholds $T$. For this heuristic analysis, we use the Python library scikit-learn, Pedregosa et al. [13], and its implementation of BIRCH.

According to the results presented in Fig. 2a, there is no indication that the number of objects in the cluster impacts the error probability. However the error probability is clearly affected by the threshold parameter; the error drops below one percent when the threshold value is greater than or equal to 1.6 (with units such that $R = 1$).

**Avoiding Cluster Combining.** While cluster splitting concerns only one single cluster, cluster combining involves two clusters. So, to begin with, we use a mixture of two Gaussians with a cluster distance $D = 6$, both with equal radius $R$, again in units chosen such that $R = 1$. (If one of the clusters has radius $R < 1$, the error probabilities would be smaller, so we only consider the situation where both radii are equal.) Again, the error probabilities do not depend on the total number of data points, as shown in Fig. 3.

Note that our criterion for correct clustering is whether the right cluster count has been detected. We have found this criterion to be sufficient and reliable for the problem that is considered here.

Fig. 3 pertains only to the cluster distance 6.0. To understand the situation for different cluster distances, consider Fig. 4. Here, we see the dependence of the error probability on the threshold for several different cluster distances. For small thresholds ($T < 1.7$) we witness cluster splitting which results in higher cluster counts and a higher error. This can also be deduced from Fig. 3, where for the small threshold $T = 1.5$ we see many cluster counts of three and four. With $T = 2.0$ less splitting occurs and the error probability decreases. If the threshold continues growing ($T = 3.0$), cluster combining occurs more frequently, which increases the cluster counts of three and therefore increases the

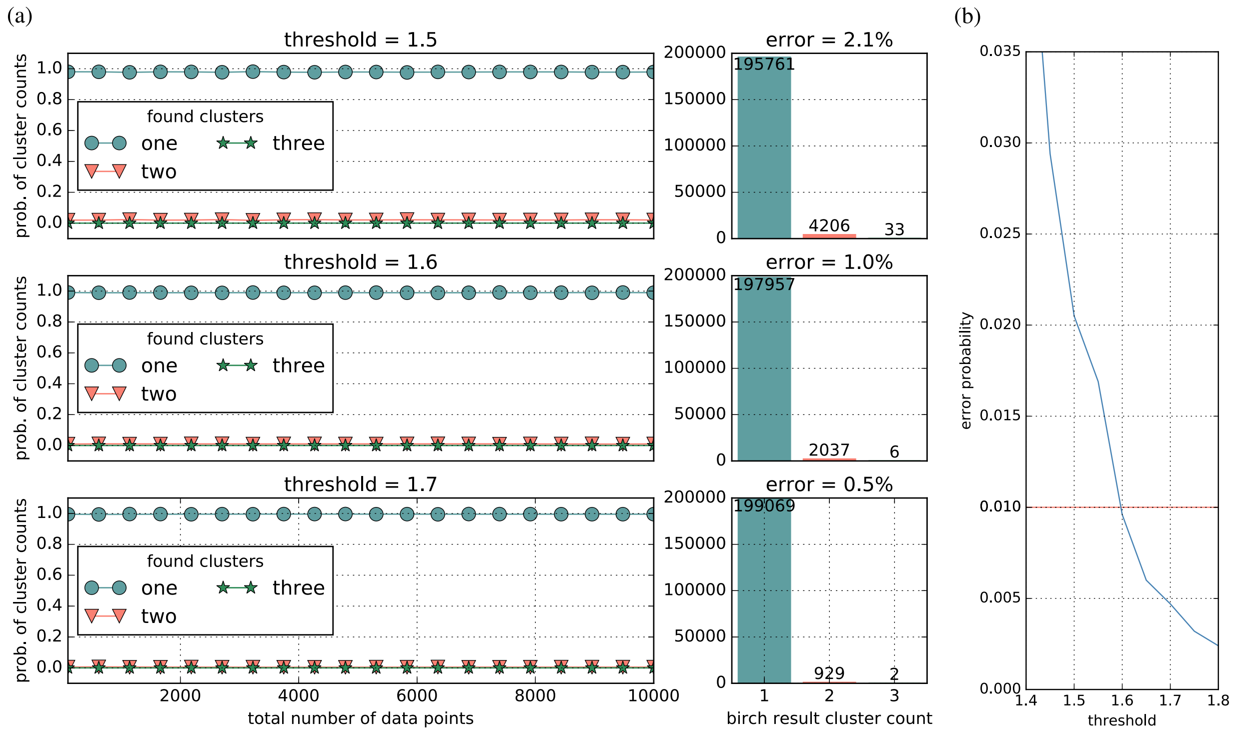

**Fig. 2.** (a) For each pair $(n, T)$ of total number $n$ of objects, running from 100 to 10,000, and threshold $T \in \{1.5, 1.6, 1.7\}$, we sampled $n$ elements from an isotropic Gaussian of radius $R = 1$, applied tree-BIRCH with threshold $T$, and recorded the cluster count. Every count different from 1 is an error. For each pair $(n, T)$ this was repeated 10.000 times to approximate the probabilities of cluster counts 1, 2, and 3. (b) For each threshold we sample 500 points from a single Gaussian of radius $R = 1$, apply tree-BIRCH and record how often it returns the right number of clusters. This is repeated 10,000 times for each $T$ to obtain an estimate for the error probability.

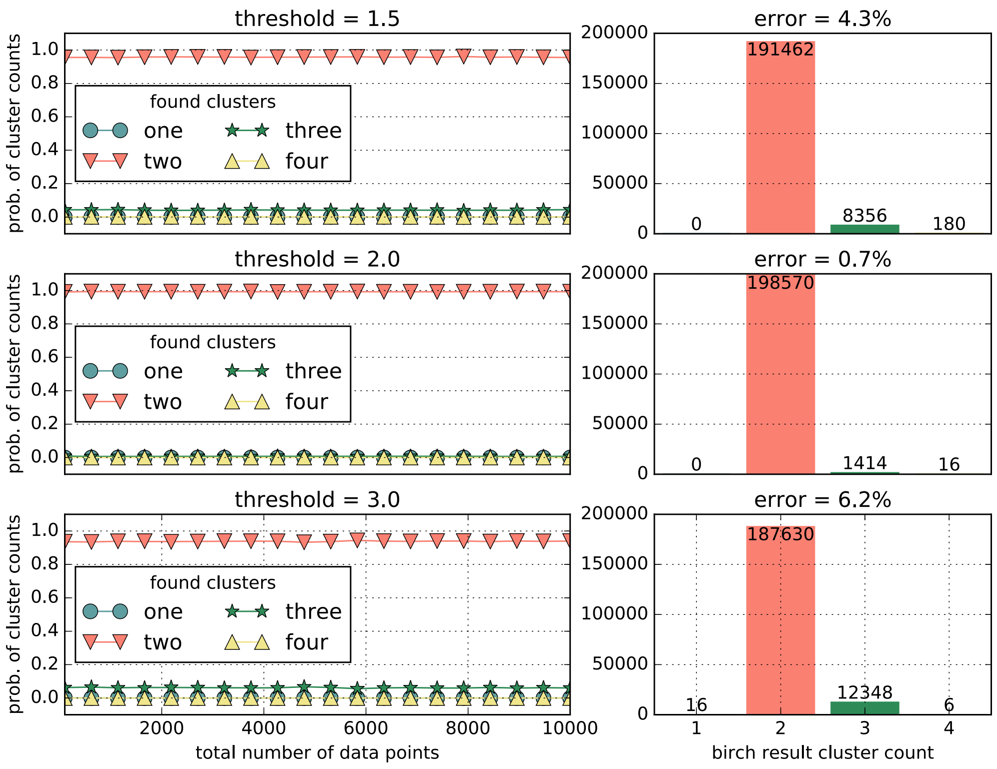

**Fig. 3.** For each pair $(n, T)$ of total number $n$ of objects, running from 100 to 10,000, and thresholds $T \in \{1.5, 2.0, 3.0\}$, we sampled $n$ elements from a mixture of two isotropic Gaussians, both of radius $R = 1$ and distance 6.0, applied tree-BIRCH with threshold $T$, and recorded the cluster count. Every count different from 2 is an error. For each pair $(n, T)$ this was repeated 10.000 times to approximate the probabilities of cluster counts 1, 2, 3, and 4.

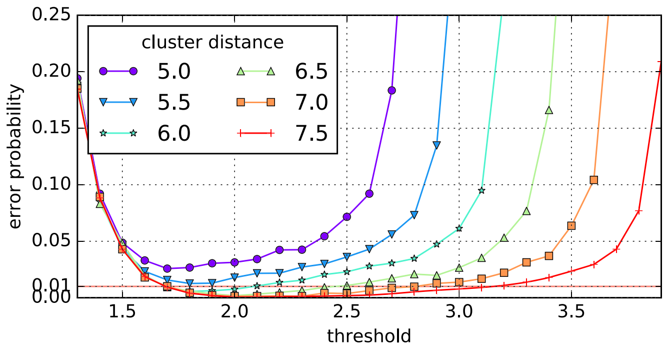

**Fig. 4.** For each pair $(D, T)$ of cluster distance $D$ and threshold $T$, we sampled 10,000 times 500 elements from a mixture of two Gaussians of radius $R = 1$ and distance $D$. Each time we applied tree-BIRCH with the threshold set to $T$ and computed the error probabilities.

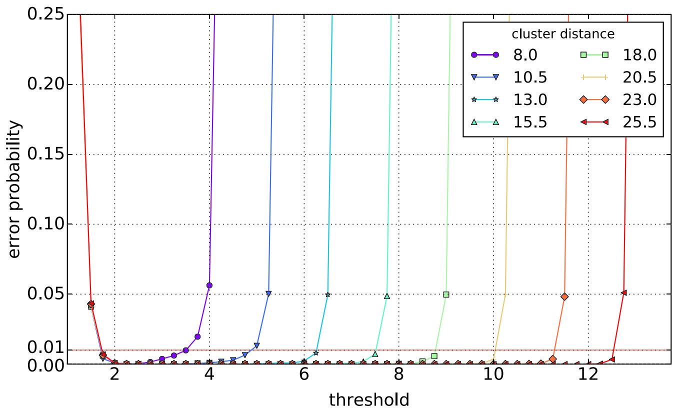

**Fig. 5.** For each pair $(D, T)$ of cluster distance $D$ and threshold $T$, we sampled 10,000 times 500 elements from a mixture of two isotropic Gaussians of radius $R = 1$ and distance $D$. Each time we applied tree-BIRCH to compute the error probabilities.

error probability. The fact that the graphs in Fig. 4 are dropping below one percent later than in Fig. 2b is due to cluster combining, which was not possible with just one cluster. For $D \geq 6.0$ there are thresholds where the error probability drops below one percent. More precisely, from Figs. 3 and 4 it can be deduced that, if $D \geq 6.0$, a threshold of:

$$T = 2.0 \cdot R_{max} \tag{1}$$

would ensure an error probability of less than one percent for each pair of neighboring clusters in the dataset.

Of course, if $D_{min}$ is clearly larger than $6.0 \cdot R_{max}$, it would be beneficial to increase the threshold beyond (1). While the lower bound on the threshold is nearly the same for all cluster distances, the upper bound increases linearly, roughly with half the increase of the distance (Fig. 5). This is intuitively clear since two times the threshold should fit comfortably between two clusters to avoid cluster combining. To place the threshold in the middle between lower and upper bound, we choose $\frac{1}{4}$ as the ratio of $\Delta D_{min}$ and $\Delta T$. We then fit an intercept of 0.7, which yields the following expression for choosing the threshold (in arbitrary units):

$$T = 0.2 \cdot D_{min} + 0.8 \cdot R_{max}, \tag{2}$$

provided

$$D_{min} \geq 6.0 \cdot R_{max}. \tag{3}$$

**A-BIRCH with parallel Gap Statistic.** Above, we have mentioned one possibility of obtaining the values $R_{max}$ and $D_{min}$ in case they are not yet already known. That involves the use of Gap Statistic. But Gap Statistic is very slow compared to full-BIRCH, possibly ruining the overall performance. Therefore, we developed a parallel version of Gap Statistic. Note that Gap Statistic runs k-means for each cluster count $k \in \{1, \ldots, k_{max}\}$ not only on the dataset itself, but also on many Monte Carlo simulations (the R and MATLAB implementations choose 100 simulations as default value). Therefore, we parallelized the loop over the Monte Carlo reference simulations. The distribution of work and collection of the results are performed by Apache Spark.

Again we want to emphasize that this is not actually part of A-BIRCH, it is just one possibility to obtain $R_{max}$ and $D_{min}$ if they are not yet known. The proposed approach is summarized in Algorithm 1.

**Algorithm 1:** A-BIRCH: Automatic threshold for tree-BIRCH using Gap Statistic for clusters with equal element count.

```
Data: N 2-dimensional data points $\{X_i\}$, $k_{max}$, number of Monte Carlo
      simulations $B$
Result: CF-tree
begin
    make sure the data $\{X_i\}$ is well shuffled
    $k^* \leftarrow$ parallel Gap Statistic( subsample($\{X_i\}$), $k_{max}$, $B$ )
    labels $\leftarrow$ k-means( subsample($\{X_i\}$), $k^*$ )
    compute the maximum radius $R_{max}$ and the minimal distance $D_{min}$ from
    the clustered data
    if condition (3) is violated then
        Warning: the clusters are too close – tree-BIRCH result might be
        inaccurate
    $T \leftarrow$ equation (2)
    CF-tree $\leftarrow$ tree-BIRCH( $\{X_i\}$, $T$, $Br = \infty$ )
```

## 6. Automatic estimation of the tree-BIRCH threshold for clusters of different element count

In this section the condition that all clusters have the same number of elements is lifted. In the previous section, the mixture of Gaussians has been uniformly weighted, i.e. if a normal distribution with mean $\mu$ and covariance matrix $\Sigma$ is written as $\mathcal{N}(\mu, \Sigma)$, the data was sampled from a mixture:

$$p(x) = \sum_{i=1}^{k} \frac{1}{k} \mathcal{N}(\mu_i, \alpha_i \mathbb{1}), \tag{4}$$

where $\alpha_i \in \mathbb{R}, \alpha > 0$ and the uniform weight is $\frac{1}{k}$. Now, this constraint is lifted and the weights can be arbitrary positive real numbers $\{w_i\}_{i=1}^{k}$ that sum to one:

$$p(x) = \sum_{i=1}^{k} w_i \mathcal{N}(\mu_i, \alpha_i \mathbb{1})$$
$$\sum_{i=1}^{k} w_i = 1, \; w_i > 0 \, (i = 1, \ldots, k). \tag{5}$$

Again, tree-BIRCH is applied repeatedly to the case of two isotropic clusters. As before, it suffices to consider the situation where both clusters have radius one. However, this time the weight ratio is different from one. First, the weight ratio 1:10 is considered (Fig. 6). In this case, too, the simulation indicates that the probabilities of the cluster counts are independent of the total number of data points. However, contrary to the uniformly weighted case, an increasing threshold very quickly leads to a wrong cluster count of one. Intuitively this can be understood as the smaller weighted cluster not being "strong" enough to prevail; it is more likely to be "swallowed" up by the heavier cluster.

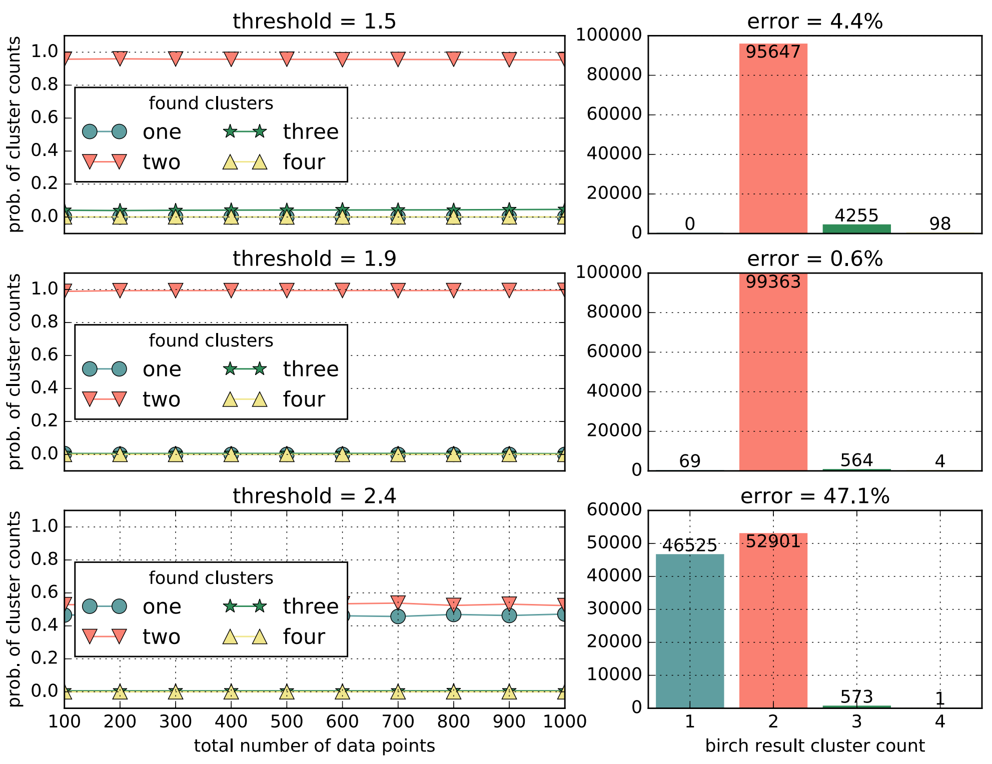

**Fig. 6.** For each pair $(n, T)$ of total number $n$ of objects, running from 100 to 1,000, and thresholds $T \in \{1.5, 1.9, 2.4\}$, we sampled $n$ elements from a mixture of two isotropic Gaussians with weight ratio 1:10, both of radius $R = 1$ and distance 6.0, applied tree-BIRCH with threshold $T$, and recorded the cluster count. Every count different from 2 is an error. For each pair $(n, T)$ this was repeated 10.000 times to approximate the probabilities of cluster counts 1, 2, 3, and 4.

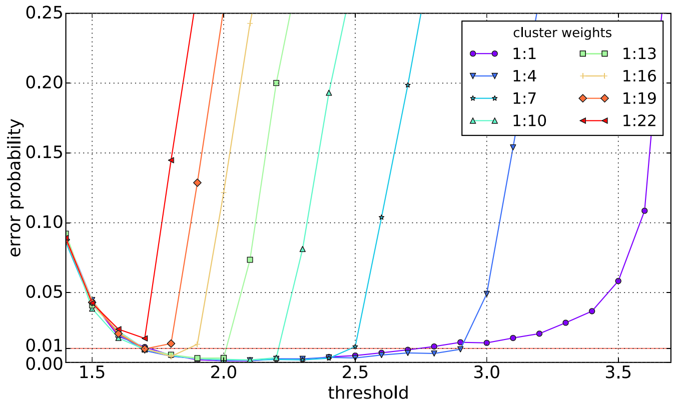

**Fig. 7.** For each pair $(wr, T)$ of weight ratio $wr$ and threshold $T$, we sampled 10,000 times 500 elements from a mixture of two isotropic Gaussians of radius $R = 1$, distance 7 and weight ratio $wr$. Each time we applied tree-BIRCH to compute the error probabilities.

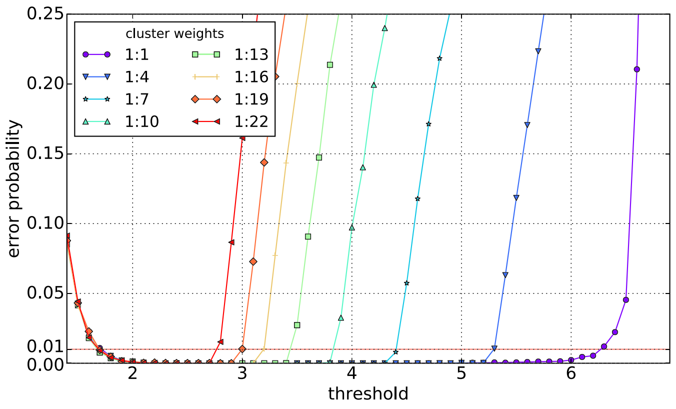

**Fig. 8.** For each pair $(wr, T)$ of weight ratio $wr$ and threshold $T$, we sampled 10,000 times 500 elements from a mixture of two isotropic Gaussians of radius $R = 1$, distance D = 13 and weight ratio $wr$. Each time we applied tree-BIRCH to compute the error probabilities.

Next, the error probability as function of the threshold is computed from simulations where, for a fixed cluster distance, the weight ratio is varied. As examples, cluster distances set to the values 7 and 13 are shown (Figs. 7 and 8). The results show that with increasing weight ratio, the interval of thresholds with error probability below 0.01 shrinks.

The next step is to obtain an expression for the optimal threshold, the one with smallest error probability, as a function of cluster distance and weight ratio. To this end, we conducted simulations of the two-cluster scenario for various combinations of those two parameters, each time recording the optimal threshold. This data then suggested to fit a nonlinear function that is linear in the cluster distance $D$ with a slope given by the inverse of a linear function of the weight ratio $wr$:

$$T = \frac{1}{a \cdot wr + b} D + c \tag{6}$$

where $a, b$, and $c$ are parameters to be learned. We used the nonlinear least square function `nls` provided by the statistics computing environment R, see R Core Team [14]. The fitted parameters are (see also Fig. 9):

$$\begin{aligned} a &= 0.3 \\ b &= 4.5 \\ c &= 0.8. \end{aligned} \tag{7}$$

The correlation of the measured and fitted values is 0.9984.

Furthermore, we would like to know when the threshold from this formula leads to an error not greater than one percent. To this end, we have, again, run tree-BIRCH on many cases of two

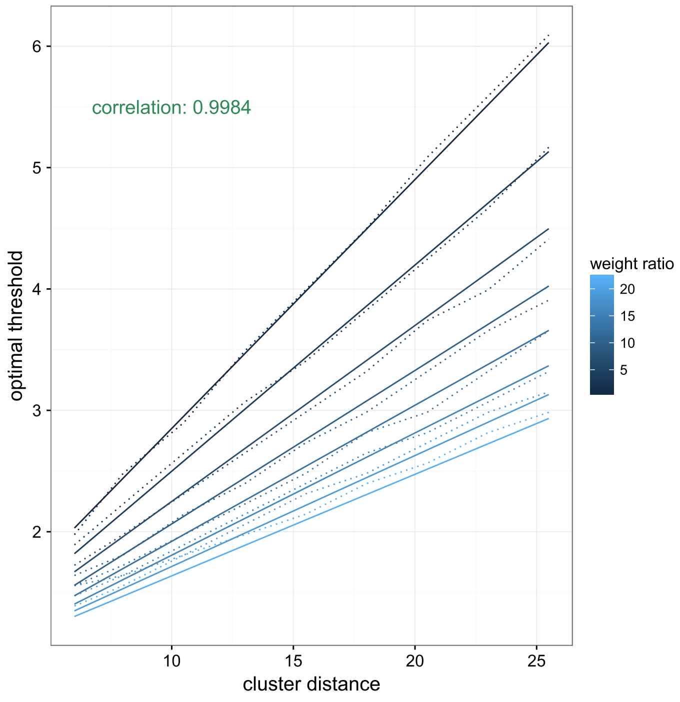

**Fig. 9.** The dotted lines connect the optimal thresholds from the simulations, one line for each weight ratio, and the solid lines are the fitted values.

**Algorithm 2:** A-BIRCH: Automatic threshold for tree-BIRCH for clusters with different element counts.

```
Data: $N$ 2-dimensional data points $\{X_i\}$, $k_{max}$, number of Monte Carlo
      simulations $B$
Result: CF-tree
begin
    make sure the data $\{X_i\}$ is well shuffled
    $k^* \leftarrow$ parallel Gap Statistic( subsample($\{X_i\}$), $k_{max}$, $B$ )
    labels $\leftarrow$ k-means( subsample($\{X_i\}$), $k^*$ )
    compute the maximum radius $R_{max}$, the minimal distance $D_{min}$, and the
    maximum weight ratio of neighboring clusters from the clustered data
    if conditions (10) are not satisfied then
        Warning: the clusters are too close – tree-BIRCH result might be
        inaccurate
    $T \leftarrow$ equation (9)
    CF-tree $\leftarrow$ tree-BIRCH( $\{X_i\}$, $T$, $Br = \infty$ )
```

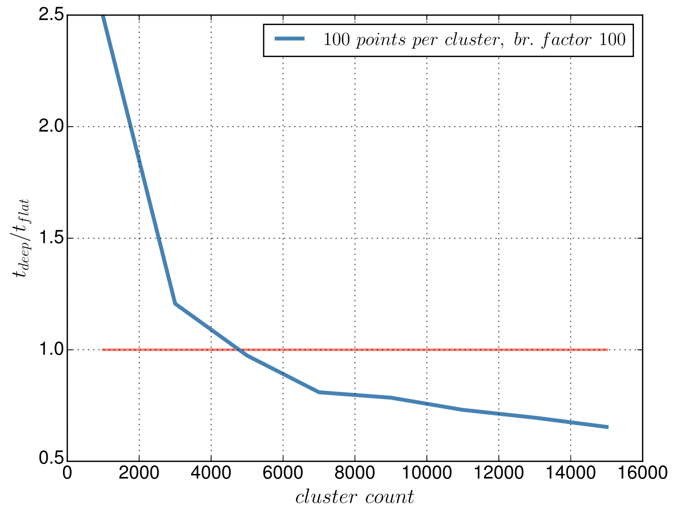

**Fig. 10.** Comparison of runtime between a flat tree and one with branching factor 100, computed with the BIRCH implementation of scikit-learn by Pedregosa et al. [13].

neighboring clusters as described above. This time, cluster distance and weight ratio have been chosen from a grid of points in the plane spanned by those two parameters. We restricted the grid to cluster distances in the interval [6, 11]. Those points are separated into two regions, the one where the optimal threshold has an error probability of less then one percent and the region where it is above one percent. The boundary between those two regions was approximated with a line. This line is given by:

$$wr = 8.8 \cdot D - 42.1, \tag{8}$$

where $D$ is the cluster distance in units of $R_{max}$. For all pairs of cluster distance and weight ratio to the right of this line, the optimal threshold computed with (6) and (7) will have an error probability of less than one percent.

In summary (see Algorithm 2), we propose to use high-performance clustering with Gap Statistic on a subset of the data as described above, to obtain the maximum radius $R_{max}$, the minimum cluster distance $D_{min}$, and the maximum weight ratio $wr_{max}$ of neighboring clusters. Those parameters are then used to compute the optimal threshold using (with arbitrary units):

$$T = \frac{1}{0.3 \cdot wr_{max} + 4.5} \cdot D_{min} + 0.8 \cdot R_{max}. \tag{9}$$

This formula gives the optimal threshold parameter $T$. Using it, tree-BIRCH will have an error probability of less then one percent for each neighboring cluster pair, provided the following conditions are satisfied:

$$\begin{aligned} &6.0 \cdot R_{max} \leq D_{min} \\ &\quad wr_{max} \leq 8.8 \cdot \frac{D_{min}}{R_{max}} - 42.1. \end{aligned} \tag{10}$$

Note that those conditions are only sufficient, not necessary.

However, from Fig. 5, 7, and 8, it can be seen that with a threshold of $T = 2 \cdot R_{max}$ the error probability is always clearly below one percent, as long as the cluster distance and weight ratio are not too extreme. Thus, we can also formulate a rule of thumb:

If the minimal cluster distance is greater than six times the maximum radius and the weight ratio of neighboring clusters is not greater than ten, a threshold $T = 2 \cdot R_{max}$ is a decent choice, with error probability smaller than one percent.

## 7. Supercluster splitting

The tree structure of BIRCH makes it possible to use BIRCH in situations with many thousands of clusters. When a new data point enters the BIRCH tree at the root, it descends the tree to its belonging cluster in one of the leaves. For a flat tree, this search is of order $O(k)$, with $k$ being the current cluster count. For a tree with branching factor $Br$, this is of order $O(Br \cdot \log_{Br}(k))$. So it is important to lift the condition that the branching factor is infinite, i.e. the tree will no longer be flat. In Fig. 10, we see a comparison of runtime for two different choices of branching factor (Python implementation in scikit-learn by Pedregosa et al. [13]). From now on, we will call BIRCH with a flat tree, i.e. the tree consists only of the root node and its direct children, which are all leaves, *flat BIRCH*. Otherwise, we call it *deep BIRCH*.

The downside of using deep BIRCH is that a new kind of error is introduced. Suppose the BIRCH tree consists of three layers and that the root has two children. The nodes at the lowest, third, layer, i.e. the leaves, represent the actual clusters, while the children of the root node in the middle layer represent *superclusters*. Furthermore, presume that right in the middle between the centers of those two child nodes of the root, there is a cluster so that half of the points in this cluster are nearer to the center of the first child and the other half is nearer to the center of the second child.

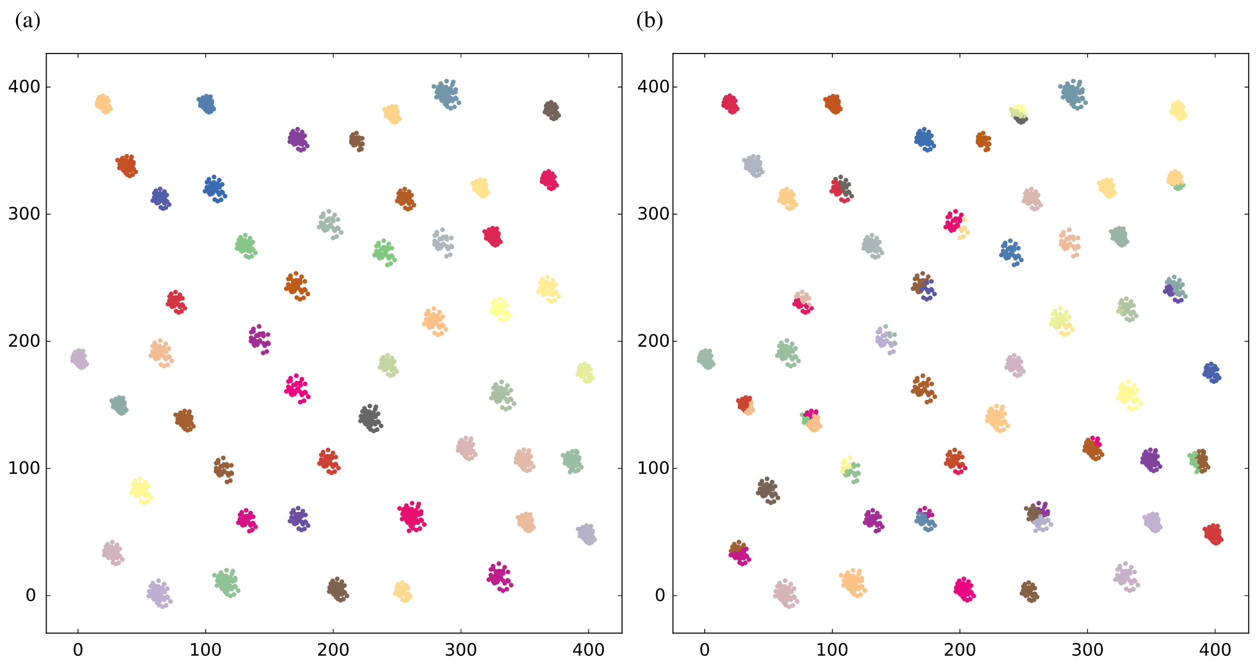

**Fig. 11.** (a) Flat BIRCH finds the right clusters. (b) The same dataset as in (a) is used here. The algorithm is deep BIRCH with a branching factor of 5. The right cluster count is 50, but the algorithm finds 73. The reason for this is supercluster splitting.

That means that this middle cluster will already be split at this supercluster level in the BIRCH tree. This problem only arises with deep BIRCH, it does not occur when using flat BIRCH. Obviously, this effect is completely independent of the size of the threshold parameter. We call this effect *supercluster splitting*.

An example of how supercluster splitting affects the clustering quality can be seen in Figs. 11a and 11b. There, the same dataset is clustered both with flat (Fig. 11a) and with deep (Fig. 11b) BIRCH. The thresholds are the same, and the data enters BIRCH in the same order, so the difference in clustering performance is only due to the difference in the tree structure. The dataset consists of 50 clusters and flat BIRCH properly recognizes exactly all of them. However, deep BIRCH, with a branching factor of 5, introduces supercluster splitting and finds 73 clusters.

In full BIRCH, i.e. when tree-BIRCH is followed by a global clustering phase, this global clustering usually alleviates supercluster splitting to a certain extend, but with tree-BIRCH this effect is clearly noticeable.

## 8. Multiple branch descent

As was shown above, tree-BIRCH is especially susceptible to supercluster splitting. So an algorithm is needed that eliminates or at least considerably reduces this effect without resort to a final global clustering phase. We suggest the following modification of tree-BIRCH.

As was shown in Section 7, supercluster splitting happens because once a new point arrives at the second to last level, i.e. at the level of parents of the leaves, there is only a small portion of all the clusters, namely the children of the current parent, the point could possibly be assigned to. In unfavorable situations, the nearest cluster might not be among them. With flat trees, however, since all clusters are children of the same parent, the root, all clusters are considered when searching for the nearest one, so the closest one is always found. Our suggestion is to compromise between those two situations, by considering not only the leaves of a single parent, but also the leaves of parents nearby.

This is done as follows. Currently, at each node, the distance of the new point to the centers of all the children of this node are computed and the new point descends into the nearest child. This is modified by having the new point not only descend into the nearest child, but also into other children, that are not much further away from the new point than the nearest child. That way, the new point performs a multiple branch descend into the tree. As a result, the new point will be compared against more leaves and the probability that the nearest one is amongst them, increases.

Of course, one now has to specify a criterion deciding which children, besides the nearest one, the new point has to descend into. We have opted for a straight forward method: the user provides a parameter $s$, and all the children of the current node, whose distance from the new point is less than the distance of the new point to the nearest cluster center plus $s$, will be descended into. To be more precise, if the new point is $p$, the metric is given by $d(\cdot,\cdot)$, the children $\{n\}_{i=1}^{N}$ of the current node have centers $C(n_i)$, and the child with center nearest to the new point is $n^*$, then the set $\Omega$ of children to descend into is given by:

$$\Omega = \{n_i \mid d(C(n_i), p) - d(C(n^*), p) < s\}. \qquad (11)$$

We call this algorithm *multiple branch descent BIRCH* (MBD-BIRCH). The higher the parameter $s$, the less supercluster splitting there will be. On the other hand, a higher $s$ will also slow down the algorithm, since the number of branches to descend into and the number of clusters to compare will increase.

## 9. Evaluation

First, we evaluated the accuracy of A-BIRCH for clusters with approximately the same element count as discussed in Section 5. That means, we use the threshold estimation as stated in Equation (2). A-BIRCH performs correctly with different sizes of $D_{min}$ and different numbers of clusters. The evaluation datasets contain samples from two-dimensional isotropic Gaussian distributions with $D_{min} \geq 6.0 \cdot R$, which is the requirement from (3) (see Fig. 12).

In an additional step, we evaluated the scalability of A-BIRCH. As already stated before, tree-BIRCH itself is very fast. So if $R_{max}$ and $D_{min}$ are known attributes of the data, A-BIRCH with the computed threshold from (2) or (9) will be extremely fast and scales with a complexity less than $O(nk)$, where $n$ is the number of

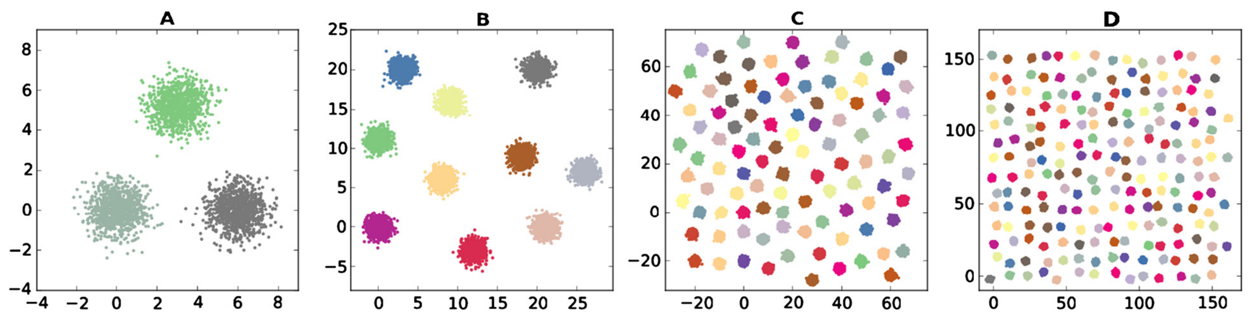

**Fig. 12.** The datasets A, B, C and D contain 3, 10, 100 and 200 clusters, respectively. Each cluster consists of 1000 elements, the radius of the clusters is $R = 1$, and the $D_{min}$ is in all cases larger than 6: in A – 6.001, in B – 7.225, in C – 6.025, in D – 6.410.

**Table 1**
Speedup of Gap Statistic by parallelization on Spark.

| | Sequential | Spark: 4 workers | Spark: 8 workers |
|---|---|---|---|
| $B = 100$, $k_{max} = 20$ | 1775 s | 349 s | 197 s |
| $B = 100$, $k_{max} = 40$ | 7114 s | 1425 s | 795 s |
| $B = 500$, $k_{max} = 20$ | 8803 s | 1470 s | 725 s |
| $B = 500$, $k_{max} = 40$ | 35242 s | 5953 s | 2909 s |

points and $k$ the number of clusters. For the situations in which $R_{max}$ and $D_{min}$ are not yet known, we discussed above the use of a subsample to determine those parameters, using a parallel version of Gap Statistic. We have tested the parallelized implementation of Gap Statistic on an Apache Spark cluster on Microsoft Azure. Two compute cluster configurations have been evaluated, each with two master nodes and with four and eight workers, respectively, each of which running on virtual machines of type `Standard_D3`. They currently provide four CPU cores and 14GB of memory, running the Linux operating system. The parallelization has been implemented using the Spark Python API (PySpark). The computation of Gap Statistic was run on a dataset containing 10 clusters, each consisting of 1000 two-dimensional data points. The computation times for varying numbers $B$ of reference datasets and maximal number of clusters $k_{max}$ are shown in Table 1.

The results show that the parallelized implementation of Gap Statistic with Spark is scalable as the computation times decrease linearly with an increasing number of worker nodes. Although the Gap Statistic phase is considered computationally expensive, it provides us with the parameters $R_{max}$ and $D_{min}$ that are needed for A-BIRCH to increases the correctness of tree-BIRCH significantly and does not require any prior knowledge on the dataset.

Next, we evaluated A-BIRCH with varying weight ratios. In Fig. 13 we have a data set with 100 clusters that has differing weights, and varying cluster radii, and the clusters are at times that near to each other that condition (10) is actually violated. Nevertheless, A-BIRCH succeeds in clustering the data set correctly.

In Section 8 MBD-BIRCH has been described as an alternative in cases where the cluster count is too large for flat A-BIRCH to satisfy given speed requirements. It has also been pointed out that the choice of the parameter $s$ represents a trade-off between speed and quality. This is exemplified in Fig. 14. Here we have used a data set consisting of 400 clusters, each sampled from a Gaussian with radius 3, each containing 50 points. The inter-cluster distance is never smaller than 24. This data set has been clustered with branching factor 10 and various $s$ values. We see that with branching factor 10 and $s = 0$, i.e. no multiple branch descent, supercluster splitting leads to a cluster count of almost twice the correct value. But when the $s$ value is increased, the detected cluster count approaches the correct value. Although the runtime increases with increasing $s$ value, the correct cluster count is achieved with a runtime that is still clearly smaller than the runtime of flat BIRCH.

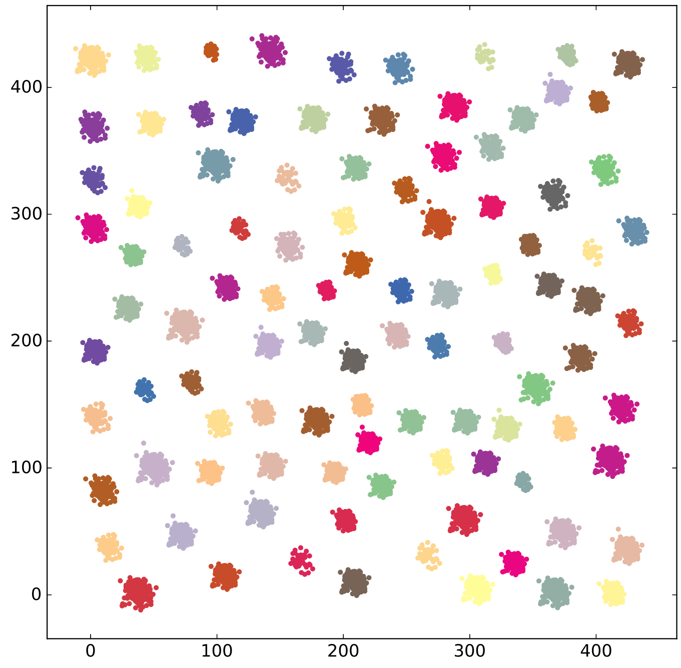

**Fig. 13.** A-BIRCH clustering accuracy for a data set with 100 clusters with points per cluster ranging from 30 to 300, cluster radius ranging from 4 to 7, and with a maximum radius of 7 and a minimum cluster distance of 30. Even though condition (10) is clearly violated, A-BIRCH succeeds in clustering the data set correctly.

This example reflects correctly what we have witnessed in multiple simulations. However, the influence of the choice of $s$ on the clustering quality as well as on the runtime, depending on the branching factor and the attributes of the data set, needs to be investigated more thoroughly. Similarly, finding a method to *automatically* choose an, in some appropriate sense, optimal $s$ value, is a task that deserves more research.

## 10. Future work

This paper has focused only on two-dimensional data sets. In future work, the findings discussed here will be extended to data sets with more than two dimensions.

In Sections 5 and 6 we have fitted models for the optimal threshold in an ad hoc manner. It would be worthwhile to use methods of model selection to obtain more accurate solutions. Also, the parameter regions which result in error probabilities less than 1 percent, given by (3) and (10) could be improved, especially in regions with small cluster distance.

Most of the results in this paper have been achieved by investigating simulated samples from mixtures of Gaussians. It would be interesting to see how much could be achieved exclusively with mathematical reasoning. Take, for example, the splitting probability

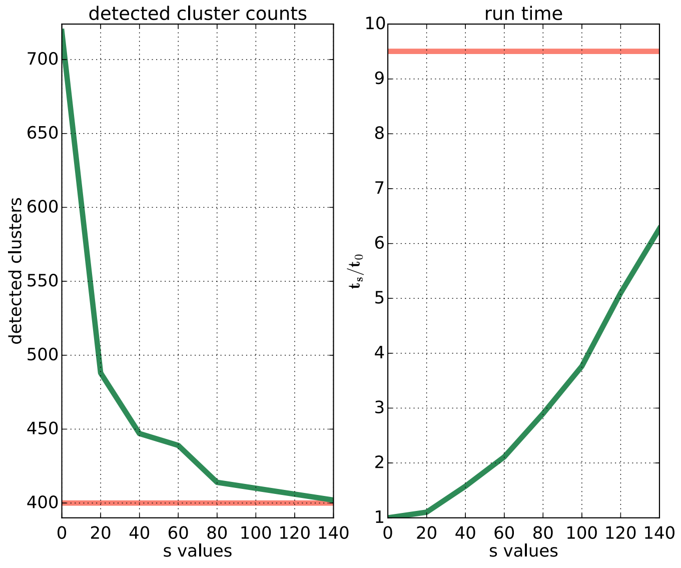

**Fig. 14.** A dataset with 400 clusters has been clustered with MBD-BIRCH with branching factor 10 and various $s$ values. On the left, the detected cluster count is plotted. The horizontal line signifies the correct cluster count. On the right, the ratio between the runtime for the given $s$ value and the runtime for $s = 0$ is plotted. The horizontal line denotes the time needed by the flat tree.

of a single cluster and start with just two points. It is not difficult to compute the probability of those two points to belong to the same cluster for a given threshold $T$. Then, generalize this formula to the case of arbitrarily many points. Next, try to find a mathematical formula, or a reasonable approximation, for the error probability in the case of two neighboring clusters. Also, the conditions 3 and 10 might benefit from a more thorough mathematical description of the problem.

In Section 8 we have pointed out that in the choice of the optimal parameter $s$ there is a trade-off between speed and accuracy. In future work it would be important to make clear what exactly *optimal* means in this situation and to find an *automatic* way of deriving this optimal parameter $s$ from appropriate attributes of the data set.

Last but not least, all the evaluations have been done with synthetic data. It is important to see how A-BIRCH and MBD-BIRCH will perform on real world data.

## 11. Conclusion

In this paper we introduced A-BIRCH, a parameter-free variant of BIRCH, and MBD-BIRCH, an extension of tree-BIRCH that improves the accuracy when BIRCH uses deep trees.

Choosing the correct parameters for clustering algorithms is often difficult as it requires information about the dataset, which is often not available. This is also true for BIRCH, which requires the cluster count k as well as a threshold T in order to compute the clusters correctly. For this reason, we removed the global clustering phase, thus rendering the cluster count parameter $k$ unnecessary, restricted the BIRCH trees to be flat, thus removing the possibility of supercluster splitting, and proposed a method that automatically estimates the threshold T from the attributes $R_{max}$ and $D_{min}$ of the data that are more likely to be already known or at least can be obtained more easily than T. Moreover, as an example how one could go about obtaining $R_{max}$ and $D_{min}$ in case they are not yet known, we described a method to obtain those parameters that involved analyzing a representative subset using a parallelized version of Gap Statistic. The evaluation proved the applicability of our approach in a very robust manner for two-dimensional mixtures of isotropic Gaussians.

This version works well for data sets of tens of thousands of clusters. However, for data with even more clusters, it is advantageous to use tree-BIRCH with deep trees. This introduces supercluster splitting, and we have developed MBD-BIRCH, an extension of tree-BIRCH, which reduces or even completely removes supercluster splitting while still being faster than tree-BIRCH with flat trees.

## References

[1] H. Akaike, B. Petrov, F. Csaki, Information Theory and an Extension of the Maximum Likelihood Principle, 1973.
[2] F.R. Bach, M.I. Jordan, Learning spectral clustering, in: Advances in Neural Information Processing Systems, 2004, pp. 305–312.
[3] K. Burbeck, S. Nadjm-Tehrani, Adaptive real-time anomaly detection with incremental clustering, Inf. Secur. Tech. Rep. 12 (1) (2007) 56–67.
[4] G. Casella, R.L. Berger, Statistical Inference, vol. 2, Duxbury, Pacific Grove, CA, 2002.
[5] M. Dash, H. Liu, X. Xu, '$1 + 1 > 2$': merging distance and density based clustering, in: Database Systems for Advanced Applications, Proceedings of the Seventh International Conference on Database Systems for Advanced Applications, DASFAA 2001, April 2001, pp. 32–39.
[6] A.P. Dempster, N.M. Laird, D.B. Rubin, Maximum likelihood from incomplete data via the em algorithm, J. R. Stat. Soc. B 39 (1) (1977) 1–38.
[7] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in: E. Simoudis, J. Han, U.M. Fayyad (Eds.), Second International Conference on Knowledge Discovery and Data Mining, AAAI Press, 1996, pp. 226–231.
[8] N. Ismael, M. Alzaalan, W. Ashour, Improved multi threshold birch clustering algorithm, Int. J. Artif. Intell. Appl. Smart Devices 2 (1) (2014) 1–10.
[9] N.S.L.P. Kumar, S. Satoor, I. Buck, Fast parallel expectation maximization for Gaussian mixture models on GPUs using CUDA, in: 11th IEEE International Conference on High Performance Computing and Communications, June 2009, pp. 103–109.
[10] J.B. Macqueen, Some methods for classification and analysis of multivariate observations, in: Proceedings of the Fifth Berkeley Symposium on Math, Statistics, and Probability, vol. 1, University of California Press, 1967, pp. 281–297.
[11] X. Meng, J.K. Bradley, B. Yavuz, E.R. Sparks, S. Venkataraman, D. Liu, J. Freeman, D.B. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M.J. Franklin, R. Zadeh, M. Zaharia, A. Talwalkar, MLlib: machine learning in apache spark, CoRR, 2015.
[12] S. Owen, R. Anil, T. Dunning, E. Friedman, Mahout in Action, Manning Publications Co., 2011.
[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: machine learning in python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[14] R Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, 2015, https://www.R-project.org/.
[15] G. Schwarz, Estimating the dimension of a model, Ann. Stat. 6 (2) (1978) 461–464.
[16] C.A. Sugar, Techniques for Clustering and Classification with Applications to Medical Problems, Stanford University, 1998.
[17] R. Tibshirani, G. Walther, T. Hastie, Estimating the number of clusters in a data set via the gap statistic, J. R. Stat. Soc., Ser. B, Stat. Methodol. 63 (2) (2001) 411–423.
[18] M. Zechner, M. Granitzer, Accelerating k-means on the graphics processor via cuda, in: First International Conference on Intensive Applications and Services, 2009, pp. 7–15.
[19] T. Zhang, R. Ramakrishnan, M. Livny, BIRCH: a new data clustering algorithm and its applications, Data Min. Knowl. Discov. 1 (2) (1997) 141–182.
[20] B. Zhou, J. Hansen, Unsupervised audio stream segmentation and clustering via the Bayesian information criterion, in: Proceedings of ISCLP-2000: International Conference of Spoken Language Processing, 2000, pp. 714–717.