KD-means: clustering method for massive data based on kd-tree

Nabil El malki
Capgemini, Université de Toulouse, UT2J, IRIT(CNRS/UMR5505)
Toulouse, France
nabil.el-malki@capgemini.com

Franck Ravat
Université de Toulouse, UT1, IRIT(CNRS/UMR 5505)
Toulouse, France
franck.ravat@irit.fr

Olivier Teste
Université de Toulouse, UT2J, IRIT(CNRS/UMR 5505)
Toulouse, France
olivier.teste@irit.fr

ABSTRACT

K-means clustering is a popular unsupervised classification algorithm employed in several domains, e.g., imaging, segmentation, or compression. Nevertheless, the number of clusters k, fixed apriori, affects mainly the clustering quality. Current State-of-the-art k-means implementations could automatically set of the number of clusters. However, they result in unreasonable processing time while classifying large volumes of data. In this paper, we propose a novel solution based on kd-tree to determine the number of cluster k in the context of massive data for preprocessing data science projects or in near-real-time applications. We demonstrate how our solution outperforms current solutions in terms of clustering quality, and processing time on massive data.

1 INTRODUCTION

Up to now, data clustering, also known as cluster analysis, has been one of the most important tasks in exploratory data analysis. It is also applied in a variety of applications, e.g., web page clustering, pattern recognition, image segmentation, data compression and nearest neighbor search [12]. Various clustering algorithms have been available since the early 1950s. The goal of data clustering is to classify a set of patterns, points or objects into groups known as clusters. Data of each group are as similar as possible to one another, and different groups are as dissimilar as possible from one another [11]. In this paper, we consider one of the most widely used data clustering algorithm, i.e., k-means [6, 8]. Due to its simplicity and understandability, this algorithm has been popular for more than four decades [11][13]. Given a set of points $X = \{x1, ..., x_n\}$ where each point is defined in the d-dimensional space $\mathbb{R}^d$ and k an integer, k-means aims to partition X into a set of clusters $C = \{C1, ..., C_k\}$ by minimizing the euclidean distance between the points (data) within each cluster:

$$\sum_{j=1}^{k} \sum_{\forall x \in C_k} ||x - G_k||^2 \quad (1)$$

where G_k the centroid of the cluster C_k and $||.||$ the euclidean distance. Minimizing this function requires an iterative process that ends when centroids do not change between two iterations. This process consists of two steps:

- assigning each point to the nearest centroid using the euclidean distance;
- recalculating the centroids, i.e., the average of the point values, of the newly formed clusters.

Disadvantageously, k-means requires that the user should predefine the value of k. Thus, determining the inaccurate value of k has a direct negative impact on the clustering quality. Different solutions have been proposed to estimate the relevant number of clusters k [10, 16, 17]. Among the limitations of these solutions, we show three major ones that challenge their normal process:

- they unroll k-means several times on the same detailed data. However, k-means has difficulties in scaling because it has a temporal complexity proportional to knt with t the number of iterations [11]. Consequently, in a context of massive data, repetitive access to data caused by the repetitive use of k-means on the same data make them computational time consuming or unusable for applications that require near-real-time responses;
- they hardly supports overlaped clusters. This problem can lead to two different cases. i) overestimating the number of clusters exaggeratedly, compared to the actual number, thus increasing the processing time significantly. ii) underestimating the value of k, thus producing a very poor clustering quality;
- if we consider clusters that approach the sphere form, which k-means identifies, then these k estimation solutions tend to capture clusters of this form. While, in real data sets, there are also clusters of spherical shapes that are more or less elongated, approaching the elliptical shape and its declines. These clusters could also have different directions. We consider a cluster to be strictly spherical if it represents a perfectly or almost spherical shape. Similarly, a cluster not strictly or less spherical when it has a shape that tends towards the elliptical. All these clusters of different sphericity will be called natural clusters. Formally, when a cluster has a strictly spherical data distribution then the variances of each data dimension are equal. If the cluster has a non-strict sphericity, then the dimensions have different variances. Statistically, these clusters can be assimilated to gaussian distributions.

In this paper, we propose a solution to these problems. We introduce an algorithm, based on the use of kd-tree[3], that estimates the number of k clusters for massive data. This estimation performs in reasonable processing time, and thus it is possible to integrate such solutions into data-intensive applications, e.g., near-real-time applications, or the preprocessing stage of a data analysis project. Our solution is an algorithm, based on kd-tree(k dimentional tree), which hierarchically structures data aggregates, e.g., clusters of data points and their corresponding metadata, with different levels of precision details. Furthermore, we define several new clusters merge criteria to support more efficiently overlapping and natural clusters.

We compare our solution to known methods of estimating k from literature (x-means [16] and g-means [7, 10]). We show that our solution is faster than current methods while ensuring better clustering quality on massive data.

This paper is organized as follows. In Section 2, we study the state-of-the-art solutions. Next, in Section 3 we introduce our

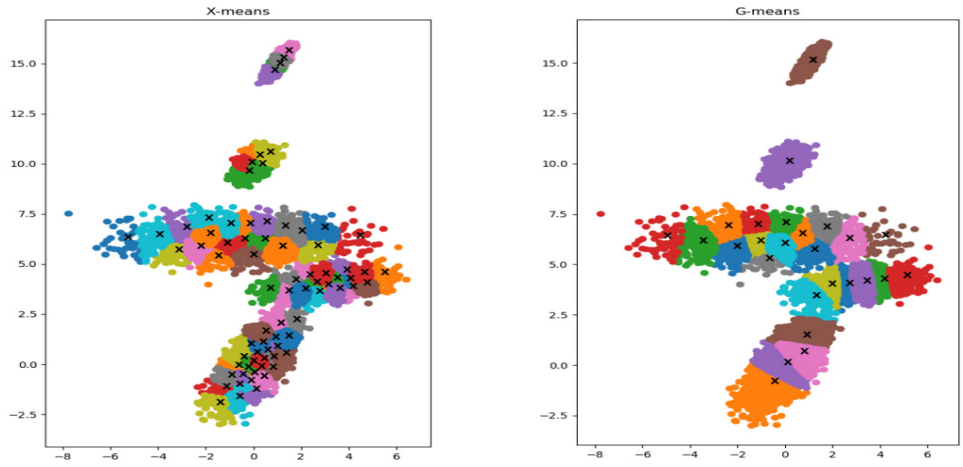


Figure 1: Estimation of k by each of the three algorithms when clusters overlap. The respective estimates of x-means and g-means are 67 and 26. The centroids are represented by black crosses. The horizontal and vertical axes refer to both dimensions of the used dataset.

contribution. In Section 4, we compare our solution with that of our competitors on data sets containing up to 4 million points. Finally we conclude in Section 5.

2 BACKGROUND

For several decades, the problem of estimating the number of clusters k remains significant in the literature. Several studies have been conducted to prevent the user from defining an inexact value of k, thus leading to counterintuitive results, and it is disparate from the expected value of k. We identify two types of clustering solutions close to k-means. Some integrate the estimation of the value of k. Others do not, as in the case of Birch [22]. The latter is a hierarchical algorithm based on a micro-clustering approach. It consists of two main phases. The first tends to produce many small spherical clusters without having to specify the number to be produced. The second uses another clustering algorithm to perform macro-clustering. We have not retained it in our study because in the second phase an algorithm should be used to apply it on the representatives of small clusters (usually centroids). The most used algorithms in the second phase and adapted for the first phase are often k-means or the hierarchical ascendant clustering [12] to which the value of k should be given a priori. We focus on two algorithms specialized in the automatic estimation of k: x-means [16] and g-means [7, 10].

X-MEANS. X-means [16] is a k-means-based algorithm that consists of searching for the set of centroids that best fit the data. Starting from a minimum k (k_{min}), generally equal to one, the algorithm iteratively adds new centroids if necessary until the maximum value of k (k_{max}) is reached. At each iteration, the algorithm identifies the subset of centroids that must be divided in two. This identification is done using a statistical criterion called bayesian information criterion (BIC). This is used for model selection and is based on the likelihood function.

The x-means process consists of two operations that are iterated until completion: Imporve_param and improve_structure. Imporve_param is only the execution of k-means with the current k as parameter. Improve_structure researches where new centroids must be added. For each centroid and its associated region of points, three sub-operations follow each other:

- the corresponding bic is calculated
- k-means is executed with k=2, two children's centroids are obtained
- the bic of the same region is calculated but taking into account the two children's centroids instead of their parent centroid.

If the previous Bic is smaller than the next Bic, then the parent centroid is replaced by its two children's centroids or the parent centroid is retained.

G-MEANS. This algorithm [10] wraps k-means and adopts almost the same approach as x-means. Instead of the BIC criterion the gaussianity test (at the level of α confidence defined by the user) on clusters is used to decide whether to divide a centroid in two.

It starts from a set of centroids of size k_{min} and it increases the set size during the process. The value of k_{min} is initialized to 1 because usually we have no a priori knowledge of the possible values of k. In this case, the first centroid corresponds to the artihmetic mean of the values of all data points. A list is defined to contain the centroids whose sets of points surrounding them are gaussians so as not to be processed in subsequent iterations.

At each iteration, the centroids not stored in the list called parent centroids, are divided into two children's centroids c1 and c2. This division is operated via the principal component analysis (pca). The pca method is a time complexity of $O(nd^2 + d^3)$ [21] with d the data point dimension and n the size of the dataset. So the authors recommended using accelerated versions of pca such as pca based on the power method [4]. The children's centroids are then refined through the execution of 2-means (k=2). Then the gaussianity test, via the Anderson-Darling test [19] (only works on points defined in one dimension), is performed on points around the parent centroid. To do this, the points around the parent centroid are first projected on the vector $v = c1 - c2$ linking the two children centroids to obtain points defined on a single dimension. If these points follow a gaussian distribution then the centroid is added to the list of centroids that will no longer be processed in the following iterations. Otherwise, it is replaced by its two children's centroids. The value assigned to α by the authors [10] is 0.0001 to test their solution on synthetic and real data.

LIMITATIONS. The limitations of the above methods are related to the difficulty of dealing with overlapping clusters, the

Figure 2: Overall solution for estimating the value of k.

quality of clustering (and therefore also the value of k) they produce as well as the execution time to provide a result.

The both methods are of different computational complexity and each can identify only a limited number of more or less spherical cluster types. X-means is suitable for data whose clusters are strictly spherical [10]. If other forms of clusters are present, then the estimate of k could be overestimated. On the other hand, g-means could identify clusters whose spherical shape is less strict than that identified by x-means. If the clusters are well spaced apart, g-means could provide the relevant value of k [10]. In addition to this distribution, the strict sphericity requirement must be added to the clusters for x-means to have the same performance. If the clusters overlap then none of them estimates the correct value of k. Under these conditions x-means and g-means tend to overestimate k.

These cases are illustrated by Figure 1. There are 5 gaussian clusters that have been generated but with a more or less different sphericity. Two clusters are well separated from each other. The other three overlap. All three algorithms were run on this set of 20,000 data. The k_{max} has been set at 140. G-means identified 26 clusters but detected the 2 clusters well separated from the others. However, there is overfitting on the remaining three clusters, an overestimation of 24 clusters instead of only estimating 3. X-means estimated k at 76. Overestimation is slightly higher for overlapping clusters than for separate clusters.

In terms of execution time, the two algorithms are not suitable when the data are massive and it is even more accentuated when the estimated value of k tends to be large.

3 CONTRIBUTION

In this section, we will define our solution that addresses the problem of estimating k in a context of massive data and overlapping clusters. Figure 2 illustrates the different steps of our solution.

The solution is composed of two main parts; i) the storage and organization of the X data provided by the user in a data structure, and ii) the processing is done on this structure to estimate k (Merging task and Cluster regularization).

We opted for the kd-tree[3] data structure to fulfill the roles of storage and data organization. A kd-tree is a binary tree that represents a hierarchical subdivision of space using splitting planes that are orthogonal to the coordinate dimensions (do not confuse the k of kd-tree which is just the dimension of the data stored in the tree and the k of k-means which corresponds to the number of clusters). In Subsection 3.1 we discuss the advantages of kd-tree as well as the method that builds the kd-tree in 3.1.2.

Concerning the processing part, first of all we proceed to the estimation of k clusters in each of the leaves (see 3.1.4). This operation results, in each leaf, in the constitution of several clusters. Then the clusters of the nodes are merged recursively from the leaves to the root according to rules defined in Subsection 3.2.2. These rules are built to manage overlapping clusters. The final step is a regularization phase (Subsection 3.3). It decides whether the small cluster is a separate cluster or should be merged with another cluster. This phase avoids having an overestimation of k due to small clusters. Note that the nodes are processed according to the post-fixed path, i.e., each node is processed after each of its children is processed. This path avoids exploring a node unnecessarily (time consuming task because it involves conditional tests) in case it will not be processed because none of its children are processed yet.

3.1 KD-TREE

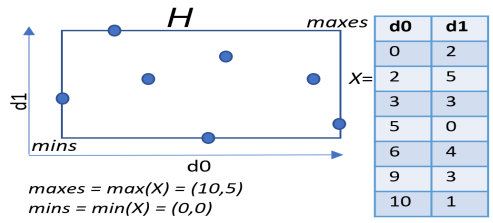
Kd-tree puts points that are spatially close together in the same node. This property is exploited by several machine learning methods to accelerate calculations related to point space. One of the best known cases is the k-closest neighbors algorithm [5]. This property is advantageous to us in two cases. First of all, it partly addresses the problem of k-means, which is to group together the most similar points possible in the same cluster. Second, it provides an optimized spatial subdivision of space to accelerate data processing. It recursively splits the whole dataset into partitions, in a manner similar to a decision tree acting on numerical datasets [9]. The root node is the whole data space, while the leaves are the smallest possible partitions of the data space. A node is associated with several closed rectangular regions of space, called hyper-rectangles. Traditionally, in the literature, the hyper-rectangle is only used to represent the subspace that the node represents. In addition, we will also use it for other purposes in 3.1.1 because of its advantages in terms of calculation and representation of sets.

3.1.1 HYPER-RECTANGLE. Formally, it corresponds to the smallest possible rectangle (generalized to n dimensions) covering all the data points of a set. It is defined by the following equation:

$$H = \{X | X_{mins} \leqslant X_i \leqslant X_{maxes} \forall i\} \tag{2}$$

where, *mins* and *maxes* respectively represent the lower and upper limits of the hyper-rectangle. Indeed, *mins*(*maxes*) is a vector corresponding to the minimum (maximum) values of each dimension of the X data set (see Figure 3).

In our case, we consider a hyper-rectangle as a geometric object to approximate the overall spatial distribution of a set of points contained in the node. In other words, each of the clusters of a node is geometrically represented by a hyper-rectangle. From this representation results, in our solution, a time saving on the calculations that involves sets (a set is a cluster of points). For example, to calculate a distance between two sets or perform a set operation involving at least two sets, their corresponding hyper-rectangles could be used. Therefore, instead of visiting all the data points of the sets, it is only necessary to use the *mins* and *maxes* vectors of the hyper-rectangles for the above calculations. This greatly saves a lot of time on large amounts of data.



d0	d1
0	2
2	5
3	3
5	0
6	4
9	3
10	1

Figure 3: A 2-dimensional hyper-rectangle H and the associated bounds *maxes* and *mins* located on the corners (upper right and lower left). Note that X is the dataset with two dimensions (d0 and d1) and the circles represent the points of X. The $max(min)$ function returns the maximum (minimum) of both dimensions.

3.1.2 SPILITTING METHOD. The partitioning of a data space of an internal node (i.e., not leaf) is performed mainly based on two cutting elements that must be specified: a given data dimension (sd) of the node space and a value of this dimension (sv). Thus the points whose value at the dimension sd is less (resp. greater) than sv then they will be assigned to the child's node which is called *lesschild* (resp. *greaterchild*). This process is carried out recursively from the root to the leaves.

Several rules for splitting nodes are proposed to choose the optimal dimension and associated value for correct data separation. Among them, we opted for *sliding midpoint splitting rule* [18] because it provides an optimized data organization than other classical rules [14]. This performance on others is explained because it does not produce empty nodes or nodes whose data space is very sparse (i.e., a very large space, specifically at the sides, compared to the data it represents when it should be small). In addition, the classical rules choose the median as the cutting value sv while the sliding midpoint splitting rule chooses the middle of the points ($(max + min)/2$) which is less expensive. The dimension sd chosen is the one that is the longest ($max - min$).

Note that theoretically there is no guarantee on the limit of the depth that kd-tree can have, the trees could be very deep (so the time of tree construction is increased). As a result, we have added two stopping conditions to avoid deep trees. These two conditions are performed at the beginning of the method that partitions the node in two. If one of the conditions is verified then the node will not be divided and will be considered as a leaf:

- the depth of the leaf is limited to $log(n)$ to save construction time compared to the normal time taken if the depth is not limited. The root has a depth of 1;
- each node is limited by a minimum number of points, ψ. It is not interesting to have leaves that have a number of points close to one in a context of massive data. This would result in a very long kd-tree construction time and therefore make our solution unsuitable for near real-time applications. In addition, if the sum of the number of points of the node is less than $2 * \psi$ then it will be considered as a leaf. Indeed, two cases are possible if this limit is not defined. Either the two children will be leaves if their number of points (size) is less than ψ. Either the size of one will be greater than ψ and therefore the other is a leaf. In the latter case a difference in depth will occur which brings a certain imbalance of the tree. This condition limits the number of small leaves, limits the depth of the tree and contributes to the balance of the tree.

3.1.3 NODE COMPONENTS. Each node contains a set of hyper-rectangles. We call this set LH. Each hyper-rectangle is associated with a list of indices of the data points that it represents, the arithmetic mean of these data and the sum of the squared differences (sse) between each point and the arithmetic mean. The data points are not stored in the tree but are accessed via the indices. The internal nodes have in addition the splitting dimension index sd as well as the associated value sv.

3.1.4 INITIALIZATION OF LEAVES. During the tree building and more precisely during the instantiation of the leaf, we estimate the number of clusters present in the subset of the data represented by this leaf (see Algorithm 1). At the same time the clustering of these data is carried out, which results in a set of clusters. When clusters are close to each other or overlap, they could overestimate the value of k in an exaggerated way. For this, the value of k was limited to a maximum value ($leaf\_kmax$) so that g-means (this version of g-means is called *estimateK* in Algorithm 1) cannot return more than a limited number of clusters. This number is called k_t with $t \in \mathbb{N}*$ the i_th iteration of *estimateK*. Indeed, at each iteration t we test if $k_t >= leaf\_k_{max}$. The value of k_t could be up to 2^t. This case only occurs if all the centroids have been split in two because they have not been evaluated as gaussian. Note that even if the total number of clusters in all leaves exceeds k_{max} then our merging algorithms will approach this number towards the real k of the dataset.

Let w be the number of leaves that kd-tree has, $leaf\_kmax$ is then defined as follows:

$$leaf\_k_{max} = \begin{cases} \sqrt{k_{max}}, & \text{if } w >= \sqrt{k_{max}} \\ \frac{k_{max}}{w}, & \text{otherwise} \end{cases} \quad (3)$$

If we assume that in Equation 3 only the second condition exists and $leaf\_kmax$ will always be equal to the result of this same condition then this would induce $leaf\_kmax$ to tend towards a value of 1 or 0. Indeed for a given value of k_{max}, the greater w is the greater the $leaf\_kmax$ tends towards 1 if $w <= k_{max}$. When $w > k_{max}$ then $leaf\_kmax$ tends towards 0. These cases could make the solution ineffective because they would underestimate the number of clusters in the leaf and by extension underestimate the number of clusters in the tree dataset. The first condition is essential to maintain a balance between having an underestimation of k if there is only the second condition and having an overestimation of k if there is no condition that would limit the value of k.

After the estimation of k, each cluster of data points is represented by a hyper-rectangle as well as the associated aggregated calculations (arithmetic mean (G) and the difference between the points and G (sse)).

Algorithm 1: InitializeLeaf

input : k_{min},k_{max},$data$,$leaf$
output: LH

```
LD ← data_{leaf.indexes}
G, k, I ← estimateK(LD, k_min, k_max)
for j = 1 to j = k do
    LH_j.indexes ← I_j
    LH_j.G ← G_j
    LH_j.maxes ← max(LD_{I_j})
    LH_j.mins ← min(LD_{I_j})
```

3.2 NODE MERGING

At the end of node initialization, clusters are obtained in each node that are considered as subclusters of the final clusters. These latter are located in the root and are considered to be the closest to real clusters. The merging of clusters is necessary to achieve the final clusters. In this subsection Algorithms 3 and 2 for merging clusters will be presented. But first, we will list the functions and definitions that will be used in these algorithms.

3.2.1 DEFINITIONS.

Definition 3.1. Let LH be a set of hyper-rectangles, the *maxes* (resp. *mins*) vector associated with LH is the maximum (resp. minimum) of each dimension of the *maxes* (resp. *mins*) vectors of the hyper-rectangles of LH:

$$getBounds(LH) = \begin{cases} mins = min(\{LH_h.mins | h \in [1; card(LH)]\}) \\ maxes = max(\{LH_h.maxes | h \in [1; card(LH)]\}) \end{cases} \tag{4}$$

Definition 3.2. Let $P = \{(u,v) | u,v \in \mathbb{N} \cap [1;k] \text{ and } u \neq v\}$ a list of index pairs of hyper-rectangles of a node. We consider $(u,v) = (v,u)$ and consequently P contains only one of them. The average of the pairwise distances is calculated by the following function:

$$apd(LH) = \frac{\sum_{(u,v)\in P} ||LH_u.G - LH_v.G||}{card(P)} \tag{5}$$

Lemma 3.3. *Given two datasets $U \in \mathbb{R}^d$ and $P \in \mathbb{R}^d$. n_1 and n_2 are respectively the number of points contained in U and P. If the sum of squared errors of U and P are as follow:*

$$SSE_U = \sum_{i=1}^{n_1} ||u_i - \bar{u}||^2$$

$$SSE_P = \sum_{i=1}^{n_2} ||p_i - \bar{p}||^2$$

then the the sum of squared errors of $V = U \cup P$ is:

$$SSE_V = SSE_U + SSE_P + n_1||\bar{U} - \bar{V}||^2 + n_2 * ||\bar{P} - \bar{V}||^2 \tag{6}$$

where $\bar{V}$ can computed as the average of the weighted averages

$$\frac{n_1\bar{U} + n_2\bar{P}}{n_1 + n_2} \tag{7}$$

Proof. We are trying to calculate the sum of two sse:

$$SSE_V = \underbrace{\sum_{i=1}^{n_1} ||U_i - \bar{V}||^2}_{SSE_{UV}} + \underbrace{\sum_{i=1}^{n_2} ||P_i - \bar{V}||^2}_{SSE_{PV}}$$

Let focus on the first term of this equation:

$$SSE_{UV} = \sum_{i=1}^{n_1} ||U_i - \bar{V}||^2 = \sum_{i=1}^{n_1} ||(U_i - \bar{U}) + (\bar{U} - \bar{V})||^2$$

$$= \sum_{i=1}^{n_1} ||U_i - \bar{U}||^2 + 2\left(\sum_{i=1}^{n_1}(U_i - \bar{U})\right)(\bar{U} - \bar{V}) + n_1||\bar{U} - \bar{V}||^2$$

But:

$$\sum_{i=1}^{n_1}(U_i - \bar{U}) = \sum_{i=1}^{n_1} U_i - \sum_{i=1}^{n_1} \bar{U} = n_1\bar{U} - n_1\bar{U} = 0$$

So $SSE_{UV} = SSE_U + n_1||\bar{U} - \bar{V}||^2$. If we repeat the same calculation for the second term SSE_{PV} then we obtain Equation 6.

The minimum distance between two hyper-rectangles x and y is calculated as follows:

$$minH(x,y) = ||0 - max(0, max(x.mins - y.maxes, y.mins - x.maxes))|| \tag{8}$$

where the function $max()$ returns the maximum of the two numbers given as parameters.

Algorithm 2: Local merger

input : LH, τ, $lenght$, th_evo
output : LH

```
merged ← True
it ← 0
old_card_lh ← card(LH)
while merged = True do
    x ← LH_it
    T ← {it}
    tmp_indexes ← {0, ..., old_card_lh}
    tmp_indexes ← tmp_indexes\{it}
    for i in tmp_indexes do
        y ← LH_i
        ΔG ← ||x.G − y.G||
        ratio ← ΔG/lenght
        if ratio < τ /ϵ then
            ΔC ← ||((x.mins + x.maxes)/2) −
              ((y.mins + y.maxes)/2)||
            e ← False
            if ΔG < ΔC then
                minDistanceXY ← minH(x, y)
                if minDistanceXY = 0 and
                  ΔG < ΔC ∗ λ then
                    e ← True
                if e = True and evo <= th_evo and
                  minDistanceXY/lenght <= τ /ϵ then
                    e ← True
                ‘if e = True then
                    T ← T ∪ i
        if card(T) > 1 then
            LH ← merge(T)
            //The new hyper-rectangles are placed at
              the end of the list
        new_card_lh ← card(LH)
        if old_card_lh < new_card_lh then
            old_card_lh ← new_card_lh
            it ← 0
        else
            if it + 1 < new_card_lh then
                it ← it + 1
            else
                merged ← False
```

3.2.2 PROPOSED MERGE ALGORITHMS.

It cannot be objectively confirmed that two clusters naturally correspond to the same cluster (i.e., they must form a single cluster) because this confirmation implies the use of clustering related concepts such as close, similar or complementary whose definition is also subjective. Consequently, we focus on the detection of clusters that k-means could detect (i.e., clusters close to strict sphericity) but also less spherical clusters as present in real data. So our goal is,

first, to capture natural clusters as defined in Section 1. Secondly, to identify these clusters even if they overlap with each other. As a result, a natural cluster is a cluster with a certain sphericity and at the same time it has low connectivity with another cluster if it overlaps with it. By connectivity between two clusters we mean of how close two clusters are in form. If two clusters of different shapes overlap considerably, they could be considered as two separate clusters. The distance between clusters alone is not enough. Note that considering only the distance between clusters to conclude that two clusters form a single cluster could lead to the following situation: if two clusters that do not naturally correspond to the same cluster (i.e., they have low connectivity) and whose distance is zero or partially overlapping, could be considered as a single cluster. This could perhaps contradict the reality of the meaning given to the data points. By taking into account the distance and connectivity between two clusters, we consider that if two clusters are of the same shape and overlap only partially then they are two different clusters that could not be merged. So it is necessary to take into account the connectivity between clusters in addition to distance when deciding to merge two clusters.

We define a set of criteria to evaluate how much two clusters correspond to the same cluster in order to decide whether two clusters can be merged in accordance with the definition of natural clusters that we have mentioned. These criteria involve measures of proximity and connectivity between clusters. The fusion algorithms 3 and 2 incorporate these criteria.

Note that merging several clusters signifies merging the associated hyper-rectangles. It consists of calculating the weighted mean of the means of the point values contained in the hyper-rectangles already calculated, concatenating the lists of the point indices, calculating the *sse* based on Equation 6 and to calculate the *mins* and *maxes* vectors according to Equation 4.

Algorithm 2 (l̂ocal merger) allows us to merge clusters whose representative hyper-rectangles are included in the LH set. LH hyper-rectangles are associated with a list of ascending ordered numerical indices. Thus the indices of the first and last elements of LH are respectively 0 and $card(LH) - 1$. We consider x to be a temporary variable that runs through the LH elements. The algorithm process is as follows. First we assign the first element of LH to x. This one is now a hyper-rectangle representing a cluster. We then check if all the other clusters represented by hyper-rectangles y are mergeable with the cluster represented by x. If x is not mergeable to any of them then the next hyper-rectangle in LH is assigned to x. On the other hand, if it is mergeable to at least one hyper-rectangle, then the concerned hyper-rectangles are merged resulting in a new hyper-rectangle (merging operated by the *merge* function). In this case the first item of freshly reconstituted LH is reassigned to x. This procedure is repeated until x is the second last element of LH and there are no more mergers. During the fusion test between two hyper-rectangles x and y, several criteria, formed from inequalities and equations, must be verified. The first two criteria only select the hyper-rectangles candidates y for merger and at the same time they reduce the number of hyper-rectangles to be processed in the other two criteria. While the last two decide whether to merge x and y or not.

CRITERION 1. *Let mins and maxes the limits of a defined data space* $\in \mathbb{R}^d$ *and LH the list of hyper-retangles belonging to this space then two clusters represented by the hyper-rectangles* $x \in LH$ *and* $y \in LH$ *are possibly mergeable if:*

$$ratio < \frac{\tau}{\epsilon} \tag{9}$$

where $\tau = \frac{apd(LH)}{||maxes - mins||}$, $ratio = \frac{||x.G - y.G||}{||maxes - mins||}$ *and* $\epsilon \in \mathbb{N}^*$.

Criterion 1 is based on the proximity measure. It checks whether two hyper-rectangles are close enough to be possibly considered mergeable. It is based on the average of the pairwise distances between centroids (*apd*) to approximate the average distance between clusters of a node without calculating the distances between all point pairs of all clusters. The pairwise distances reduces the bias brought by the very distant centroids from the majority of other centroids. The values of $ratio$ and τ of the inequality are normalized between 0 and 1 by the maximum length of the data space to be compared. The value of ϵ controls the boundary between the qualifiers "distant" and "close" when referring to the distance between clusters of x and y. If this criterion is verified then other criteria must be verified to confirm the fusion between x and y. Note that the higher the value of ϵ is, the fewer hyper-rectangles validate the criterion. Moreover, if $\epsilon = 1$, then a significant amount of hyper-rectangles will validate the criterion, which could lead to further tests and probably lead to a non fusion result because if two clusters are very distant then there will be no fusion. The value of ϵ is to be adjusted according to the strictness level of the "enough close" notion required by the criterion including this parameter.

CRITERION 2. *Let* x *and* y *two hyper-rectangles,* ΔC *the distance between the centers of* x *and* y *and* ΔG *the distance between the centroids of the* x *and* y *clusters. If* $\Delta G < \Delta C$ *then* x *and* y *are possibly mergeable.*

Criterion 2 refines the set of hyper-rectangles from the first criterion. It estimates how strong the connectivity between two clusters is. Knowing that a hyper-rectangle corresponds to the smallest rectangular envelope covering all the data points of a cluster, then it could be ruled that the center of the hyper-rectangle would roughly correspond to the centroid of the cluster if the points of the cluster are uniformly distributed with a certain sphericity. So the cluster with these characteristics is probably a natural cluster apart. As a result the closer the distance between centroids (ΔG) is to the distance between centers (ΔC) the smaller the probability of merging x and y. This probability is considered null when $\Delta G >= \Delta C$.

The first two criteria make it possible to identify relatively close clusters but also to consider that two clusters are distant if their centroids are also distant even if the two clusters are contiguous. The latter could occur if the majority of points surround the centroid while a minority are far from the centroid. However, these criteria are based only on the distances applied to centroids and centers. Two clusters could be considered as close, but one or more clusters could be located between them. In this case they cannot be merged directly. They could only be if at least one cluster between them merges with them. The following two criteria require stricter distances and stronger connectivity to avoid directly merging two nearby clusters separated by other clusters.

CRITERION 3. *Let be* x *and* y *two hyper-rectangles. We merge* x *and* y *when Criteria 1 and 2 are verified and if* $minH(x, y) = 0$ *and* $\Delta G < \Delta C * \lambda$ *with* $\lambda \in]0; 1]$

Criterion 3 only deals with contiguous or overlapping hyper-rectangles. Indeed $minH(x, y)$ returns 0 if the two hyper-rectangles overlap or if their distance is zero. Also, a connectivity constraint

is added. It results in a constraint on the proximity between centroids: for two hyper-rectangles x and y the distance between centroids must be less than $\Delta C * \lambda$. The smaller λ, the more the notion of "close enough" between clusters to the point of matching the same cluster is strict.

CRITERION 4. *Let x and y two hyper-rectangles and $\epsilon \in \mathbb{N}^*$. We merge x and y when Criteria 1 and 2 are verified and if evo <= th_evo and $minH(x, y)$ normalized is less than τ/ϵ with evo the sum of the squared errors (sse) of the x and y data points union.*

Unlike Criterion 3, to validate Criterion 4 hyper-rectangles are not necessary to be contiguous but a maximum distance is required. The normalized distance $minH(x, y)$ must be less than τ/ϵ. It may be that two clusters are not contiguous but almost when the distance is as small as it does not allow a cluster to interpose itself between two clusters concerned by the calculations of the criterion. In addition to this, the criterion adds a requirement on the compactness of x and y clusters. To do this, it uses the homogeneity measure called sum of square error (*sse*). In this criterion we have the choice to limit the quantity of *sse* for possible mergers. Indeed, two clusters are merged only if the *sse* of their data is less than the limit is *th_evo*. Its value is entered by the user. However, *th_evo* is adapted according to the value of Δk provided Equation 10. The latter evaluates the difference between the value of k_{max} and the total sum of the values of k specific to each leaf ($\sum k_\vartheta$). The purpose of this adaptation is to approach the estimated value k on the entire dataset at k_{max} and avoid an overestimation of k compared to k_{max}. If $\delta k >= 1$ it means $\sum k_\vartheta$ is at least equal to k_{max} and therefore *th_evo* will be unchanged. Otherwise, if $k_{max} > \sum k_\vartheta$ then *th_evo* is recalculated according Equation 11.

$$\delta k = 1 - \frac{k_{max}}{\sum k_\vartheta} \tag{10}$$

$$th\_evo = \begin{cases} th\_evo, & \text{if } \delta k >= 1 \\ th\_evo + \delta k, & \text{otherwise} \end{cases} \tag{11}$$

Algorithm 3: Global merger

input : *internal*,*th_evo*
output: *LH*

```
LH ← internal.less.LH ⋃ internal.greater.LH
mins, maxes ← getBounds(LH)
lenght ← ||maxes − mins||
if card(internal.greater.LH)>1 then
    τ ← apd(internal.greater.LH)/lenght
    internal.greater.LH ←
      localMerger(internal.greater.LH, τ, lenght, th_evo)
if card(internal.less.LH)>1 then
    τ ← apd(internal.less.LH)/lenght
    internal.less.LH ←
      localMerger(internal.less.LH, τ, lenght, th_evo)
LH ← internal.less.LH ⋃ internal.greater.LH
τ ← apd(LH)/lenght
LH ← localMerger(LH, τ, lenght, th_evo)
```

The *global merger* fusion algorithm can be seen as a layer that envelops the algorithm *local merger*. Intuitively, for a given internal and parent node, a good part of the clusters of one of its children are closer to each other than to the other child's clusters. So we process the clusters of a child node locally but in the data space of the parent node. Hence the proposal of the *global merger* algorithm. It performs the merging of clusters node by node using the *local merger* algorithm. It starts first with the children of the "*internal*" node and then concatenates the hyper-rectangles of the both children to form the list of hyper-rectangles of the *internal* node. Finally, *local merger* is applied to *internal*. Before the children node is processed, the *apd* function is normalized by the maximum length of the parent node hyper-rectangle. This normalization allows to have the same distance ratio in the three nodes (*internal*,*greater* and *less*). On the other hand, the value of *apd* changes from one node to another because it depends only on the clusters of the concerned node. Consequently, if two clusters have not been merged into one of the two children nodes, they could be merged into the parent node. Note that the processing of the children nodes can even be parallelized because they are independent.

3.3 CLUSTER REGULARIZATION

This subsection discusses the strategy we have defined to identify clusters that should not be as such but rather be part of another cluster. The algorithms of the previous step could produce small (in number of points) but compact clusters. A small cluster is located in three cases: either it is a natural cluster different from the others, or it is part of an agglomeration of small clusters that represent a single cluster, or it is part of a large cluster. The definition of a small cluster has no objective purpose. We consider the definition of the small cluster resulting from the work [1] as a cluster with a number of points less than $\sqrt{n}$ with n the size of the cluster.

We have developed an algorithm that makes it possible to match a small cluster to one of the three cases. The goal of the algorithm is to get as many natural clusters as possible. Knowing that the final result of our solution is in the root of the tree then this algorithm is unrolled only in this one. The algorithm consists of two parts that are executed consecutively once. The first identifies the small clusters and then merges those that are close to each other. The second re-identifies small clusters from the new list of small clusters from the first part and affects those that are close to large clusters. If a small cluster has not undergone a merger operation in both parts then it will be considered as a separate natural cluster.

One of the reasons for the presence of small clusters is due to Criteria 3 and 4 where the limit of the minimum distance between hyper-rectangles is very strict. Indeed, if the minimum distance between two hyper-rectangles, at least one of which is a small cluster, is greater than this limit, then they cannot be merged. This limit is respectively equal to 0 in Criterion 3 and τ/ϵ in Criterion 4. The τ/ϵ limit is more flexible than the first one, it was used in both parts of the algorithm to define the boundary between "close" and "distant" regarding the distance between hyper-rectangles.

4 EXPERIMENTAL ASSESSMENTS

In this section, we carried out experiments to study the performance of our solution, to demonstrate its effectiveness compared to the algorithms listed in Section 2(x-means and g-means) and to give recommendations on the values that the parameters of our solution could take (the minimum size of an internal node

ψ and limit th_evo of Equation 11). In the experiments, real data and synthetic data were used.

4.1 EXPERIMENTAL PROTOCOL

Our solution and the compared algorithms were implemented in python 3.6. All of them use the Elkan version of k-means [6]. It corresponds to an exact version of standard k-means but is suitable for massive data in execution time. Moreover, k-means++ [2] is used for initializing the centroids before applying Elkan algorithm.

Different values are assigned to parameters k,d, n, th_evo and ψ to study the sensitivity of our algorithm to these parameters. All combinations of the values of these parameters have been tested:

- $k \in [10, 32, 80]$
- $d \in [4, 6, 8]$
- $n \in [1 \times 10^6, 2 \times 10^6, 4 \times 10^6]$
- $th\_evo \in [0.05, 0.10, 0.15, 0.20, 0.25, 0.30]$
- $\psi \in [4, 8, 10, 12]$

An experiment is structured as follows:

- first of all a combination of the above-mentioned parameters is defined. Parameter k represents the real number of clusters in a dataset. It should be noted that parameters k, d and n above refer only to synthetic data. The values of these parameters are defined in the characteristics of the real datasets,
- then follows the data generation in the case of synthetic data, otherwise the real data is loaded,
- finally, the estimation of k is performed by the three algorithms.

For each combination of these parameters the experiment is conducted five times.

We set the values of ϵ and λ in the different criteria as follows:

- ϵ has been defined as 2 and 10 respectively in Criteria 1 and 4. The value 2 is the least strict while 10 is the most strict;
- in the cluster regularization algorithm, $\epsilon = 8$ in the first part of the algorithm. If there are still small clusters left, then the second part is executed. In this case $\epsilon = 4$;
- in Criterion 3, $\lambda = 0.75$. As a result, ΔG must be less than 75% of ΔC.

We allow our algorithm and x-means to search up to $k_{max} = 5k$ centroids.

4.1.1 SYNTHETIC DATA. In order to get closer to the real data, natural and overlapping clusters were generated. These synthetic data have the following properties:

- a cluster is a set of data that follows a normal distribution. To generate this set, a semi positive covariance matrix and an average (a vector) are generated at random. Indeed, the values of the covariance matrix allow to define a cluster shape that is not isotropic (strictly spherical) and that can have different variances separately on an arbitrary basis of directions and not necessarily on those of the dimensions;
- there are at least two overlapping clusters;
- if the centroid of one cluster is in the hyperectangle of another cluster then we consider that the maximum degree of overlap has been exceeded. So as a result at least one of the clusters concerned is replaced by a new cluster;
- clusters do not have the same number of points. The cardinalities are chosen at random so that their sum is equal to the total number of points n defined previously.

4.1.2 REAL DATA. The real data comes from three known sources: Openml, UCI and Kaggle. They are all of a numerical type. These data are used in other research projects to perform benchmarking machine learning methods [20]. The values of real k, n and d are given in Table 2. The data sets used are diverse and they can be organized into several groups:

- **black and white or grayscale images**; clustering is applied in this case directly to the pixels. *Emnist* and *Fashion−mnist* contain images respectfully on the first 10 digits and clothing and shoes. Each image was flattened to form a single vector. And each vector has been assigned its corresponding class (label);
- **4-band multispectral images**; they characterize different types of soil. The corresponding dataset is *satimage*;
- **sounds**; the *JapaneseVowels* dataset is a set of digitized Japanese vowel sounds. Each sound is associated with a speaker;
- **historization of people's activities**; the *ldpa* dataset is a collection of the positions of sensors present on people in order to identify the movement they perform at a given time. The *walking* dataset is a set of people's activities designed to determine the authors;
- **images represented by characteristics extracted from the object to be identified**; for example, in the *Fourier* dataset each record is a set of coefficients of the character (one of the first 10 digits) shapes; similarly in *Zernike* each instance of a digit is of rotation invariant Zernike moments of the digit (between 0 and 9); in *Pendigits* each individual is a positions sequence characterizing a digit; in *Letter* each instance of a letter of the English alphabet contains statistical moments and edge counts of the given letter; in *Vehicle* dataset a data point is just a set of geometric characteristics of a given vehicle.

4.1.3 METRICS FOR EVALUATING EXPERIMENTAL RESULTS. To evaluate the results of the algorithms we are comparing, three metrics were used. First, the execution time (Δt) of the algorithm, the relative difference (Δk) between the actual value of k and the estimated value of k and finally the distance between the actual partitioning and the partitioning proposed by the algorithm.

The relative difference is calculated as follows:

$$\Delta k = \frac{|k_{real} - k_{estimated}|}{k_{real}} \tag{12}$$

The variation of the information (vi) [15] was used as a distance between two partitionings. It measures the amount of information gained and lost for a dataset passing from a partition A to another partition B. It respects the three properties of triangular inequality. So the smaller the vi is, the closer the partitionings are to each other.

4.2 RUNTIME ANALYSIS

If we consider the performance of the three algorithms according to the metric Δt, our algorithm takes the least time to provide a clustering result. This is true in both synthetic and real data and regardless of the value of k, d and n. Our algorithm is 5.5 to 12.9 times faster than g-means and 15.4 to 75.2 times faster than x-means in synthetic data, respectively. The same trend occurs in real data, but from 16.3 to 1566.6 times compared to g-means

			KD-means			G-means			X-means		
k	d	n	k(Δk)	vi	Δt	k(Δk)	vi	Δt	k(Δk)	vi	Δt
10	4	1×10^6	8.4(0.17)	0.4	39.7	74.0(6.4)	2.2	344.3(8.7)	45.7(3.57)	2.5	816.2(20.6)
		2×10^6	8.3(0.21)	0.5	65.4	**91.0(8.1)**	2.4	774.1(11.8)	45.7(3.57)	2.5	1420.5(21.7)
		4×10^6	8.3(0.18)	0.5	117.4	87.0(7.7)	2.3	**1517.3(12.9)**	44.6(3.46)	2.4	2203.6(18.8)
	6	1×10^6	8.6(0.15)	0.4	41.9	69.2(5.92)	2.2	396.4(9.5)	46.0(3.6)	2.3	1101.5(26.3)
		2×10^6	8.1(0.2)	0.5	72.9	89.0(7.9)	2.5	926.9(12.7)	46.1(3.61)	2.3	1869.9(25.7)
		4×10^6	8.0(0.23)	0.5	133.7	79.1(6.91)	2.2	1628.0(12.2)	45.9(3.59)	2.3	3061.4(22.9)
	8	1×10^6	7.4(0.39)	0.8	45.9	63.2(5.32)	2.6	386.0(8.4)	45.9(3.59)	1.5	789.4(17.2)
		2×10^6	6.9(0.37)	0.8	81.2	71.3(6.13)	**2.8**	823.1(10.1)	45.5(3.55)	1.6	**1253.5(15.4)**
		4×10^6	6.0(0.41)	0.9	153.6	78.9(6.89)	**2.8**	1727.2(11.2)	46.2(3.62)	1.6	2378.8(15.5)
32	4	1×10^6	35.4(0.38)	0.9	77.6	99.3(2.1)	1.9	535.6(6.9)	148.4(3.64)	2.9	2817.2(36.3)
		2×10^6	31.4(0.35)	0.8	132.5	114.1(2.57)	2.0	1152.8(8.7)	147.6(3.61)	2.9	4831.0(36.5)
		4×10^6	27.3(0.36)	1.0	233.7	132.1(3.13)	2.1	2468.4(10.6)	145.2(3.54)	2.9	7628.1(32.6)
	6	1×10^6	**40.2(0.47)**	0.9	79.2	101.0(2.16)	1.8	586.7(7.4)	150.2(3.69)	2.7	3689.9(46.6)
		2×10^6	34.6(0.44)	0.9	146.9	122.5(2.83)	2.1	1375.6(9.4)	149.1(3.66)	2.7	7880.7(53.7)
		4×10^6	29.1(0.44)	1.0	260.2	107.9(2.37)	1.7	2485.1(9.5)	135.9(3.25)	2.7	7901.8(30.4)
	8	1×10^6	35.7(0.36)	1.0	82.2	115.2(2.6)	2.2	665.7(8.1)	149.0(3.66)	1.9	2491.6(30.3)
		2×10^6	37.2(0.42)	1.0	154.9	126.8(2.96)	2.3	1383.3(8.9)	148.1(3.63)	1.8	3740.0(24.2)
		4×10^6	36.4(0.46)	0.9	299.5	136.5(3.27)	2.6	2883.7(9.6)	133.8(3.18)	1.8	5037.7(16.8)
80	4	1×10^6	90.5(0.24)	0.9	141.5	144.1(0.8)	1.5	**781.9(5.5)**	368.2(3.6)	3.3	7807.2(55.2)
		2×10^6	88.3(0.29)	1.0	283.2	170.4(1.13)	1.8	1673.6(5.9)	373.1(3.66)	3.3	14147.0(50.0)
		4×10^6	83.0(0.3)	**1.2**	570.8	186.9(1.34)	2.0	3417.8(6.0)	366.6(3.58)	**3.4**	21887.1(38.3)
	6	1×10^6	96.2(0.37)	0.9	148.3	167.0(1.09)	1.6	893.0(6.0)	376.1(3.7)	3.2	10385.8(70.1)
		2×10^6	97.4(0.39)	0.9	285.4	186.2(1.33)	1.8	1892.4(6.6)	375.0(3.69)	3.2	22403.6(78.5)
		4×10^6	92.8(0.4)	1.1	578.0	200.4(1.51)	1.7	4039.2(7.0)	363.3(3.54)	2.9	37716.6(65.3)
	8	1×10^6	57.5(0.28)	**1.2**	151.5	207.8(1.6)	2.0	1016.0(6.7)	377.2(3.71)	2.2	7834.6(51.7)
		2×10^6	98.6(0.43)	0.9	308.2	150.2(0.88)	1.4	**1708.6(5.5)**	373.6(3.67)	3.2	21824.6(70.8)
		4×10^6	**99.2(0.47)**	1.0	**626.4**	166.9(1.09)	1.5	**3581.8(5.7)**	**377.7(3.72)**	3.2	**47130.8(75.2)**

Table 1: Comparison of the three algorithms (our KD-means algorithm, g-means and x-means) executed on synthetic data. Note that Δt is expressed in seconds. In the Δt sub-column of the g-means and x-means columns, the number in brackets represents how many times our algorithm is faster than the compared algorithm.

				KD			G-means			X-means		
dataname	k	d	n	k(Δk)	vi	Δt	k(Δk)	vi	Δt	k(Δk)	vi	Δt
vehicle	4	18	1×10^6	3.0(0.28)	2.9	38.6	**2226(555.5)**	11.8	**60530.8(1566.6)**	16.0(3.0)	5.3	281.5(7.3)
satimage	6	36	1×10^6	**28.9(3.81)**	2.7	54.5	1177(195.17)	8.8	34595.7(635.3)	28.7(3.78)	3.8	1151.0(21.1)
japaneseVowels	9	14	1×10^6	9.4(0.83)	4.5	75.6	3377(374.22)	**12.5**	55220.1(730.5)	32.0(2.56)	7.7	534.8(7.1)
pendigits	10	16	1×10^6	29.1(1.91)	3.6	31.0	2574(256.4)	9.1	40806.3(1316.9)	32.0(2.2)	4.0	759.2(24.5)
fourier	10	76	1×10^6	39.2(2.92)	3.0	91.6	1309(129.9)	8.0	38210.3(417.0)	**57.3(4.73)**	3.9	2791.2(30.5)
fashion-mnist	10	784	70000	10.9(0.11)	3.3	58.6	533(52.3)	7.1	**954.1(16.3)**	32.0(2.2)	4.0	697.0(11.9)
ldpa	11	5	164860	32.5(1.96)	6.3	12.5	1766(159.55)	10.8	5996.6(480.7)	47.2(3.3)	7.1	132.3(10.6)
walking	22	4	149332	39.6(1.35)	6.7	14.7	1257(56.14)	10.0	3254.1(220.8)	100.7(3.58)	8.4	**93.7(6.4)**
letter	26	16	999999	59.7(1.3)	**7.2**	57.8	1581(59.81)	11.1	22502.6(389.5)	117.0(3.5)	**8.6**	**6272.4(108.6)**
zernike	47	47	1×10^6	86.9(1.08)	5.3	118.1	1157(23.62)	9.0	35710.8(302.3)	128.0(1.72)	6.4	5583.1(47.3)
emnist	62	784	697932	45.6(0.26)	6.0	**948.6**	3073(48.56)	8.6	46224.8(48.7)	256.0(3.13)	6.7	**36029.7(38.0)**

Table 2: Comparison of the three algorithms executed on real data.

and from 6.4 to 108.6 times compared to x-means. Maximum execution times were reached for all three algorithms at $d = 8$, $n =$ and $k = 80$ in the synthetic data. In this combination, the elapsed execution times are 10.43 minutes for our algorithm, 59.41 minutes for g-means and 13h05 for x-means respectively. In the real data, the maximums are reached in the configuration $d = 784$, $n = 697932$ and $k = 62$ for x-means and KD-means. That is 15.48 minutes for KD-means and 10 hours for x-means. In the case of g-means, the maximum is even higher than that of the others, it is reached at 16.48 hours ($k = 4$, $d = 18$, $n = 1 \times 10^6$).

4.3 CLUSTER QUALITY ANALYSIS ($\Delta k/vi$)

KD-means is better than other algorithms in estimating k with good clustering quality in synthetic and real data.

In the synthetic data, KD-means does not exceed 0.47 in Δk and 1.2 in vi. X-means even reaches $\Delta k = 3.72$ and $vi = 3.4$. The same for g-means with $\Delta k = 8.1$ and $vi = 2.8$. The decrease in Δk g-means when k increased in the synthetic data is due to the presence of many gaussian clusters that are well separated from each other. KD-means has a better estimate of good quality than the others. In terms of vi, it is on average 2.46 and 3.05 better than g-means and x-means. Same performance in Δk, better on average by 2.83 (g-means) and 4.29 (x-means). It could be pointed out that g-means is better than x-means in terms of quality (vi) in several combinations of (k, d, n).

In real data, the KD maxima do not exceed $\Delta k = 3.81$ and $vi = 7.2$. They are higher for x-means and g-means, they reach respectively ($\Delta k = 4.73$, $vi = 8.6$) and ($\Delta k = 555.5$, $vi = 12.5$). The values of Δk and vi are higher in real data than in synthetic data. Indeed, the distributions of clusters in real data are more complex than synthetic clusters (gaussian). As a result, these complex representations of clusters are difficult to capture completely point by point. This increase is much higher for g-means because it tends to identify gaussian clusters in particular. However, clusters in real data are not necessarily gaussian because they are not constituted by gaussian distribution generators. As a result,

	Synthetic data			Real data		
th_evo	Δk	vi	Δt	Δk	vi	Δt
0.05	0.3	0.8	213.2	1.7	4.9	132.5
0.10	0.3	0.8	203.1	1.6	4.8	144.5
0.15	0.3	0.8	198.2	1.5	4.7	134.0
0.20	0.3	0.8	193.5	1.4	4.7	141.1
0.25	0.3	0.9	188.6	1.2	4.5	136.5
0.30	0.4	1.0	183.8	1.3	4.6	130.5

Table 3: Performance of our algorithm as a function of the value of th_evo

	Synthetic data			Real data		
ψ	Δk	vi	Δt	Δk	vi	Δt
4	0.5	1.1	195.0	1.3	4.2	131.1
8	0.3	0.8	201.5	1.2	4.4	141.3
10	0.3	0.7	200.0	1.4	4.8	136.1
12	0.3	0.8	190.5	1.9	5.3	137.5

Table 4: Performance of our algorithm as a function of the value of ψ

g-means makes an execessive overestimation of k (therefore Δk high) compared to KD. The KD-means algorithm is the least affected by the gaussianity of clusters because in its operation gaussianity is not explicitly sought.

4.3.1 PARAMETER SENSITIVITY ANALYSIS. We analyze the three metrics (Δt, vi and Δk) according to th_evo and ψ. This is done in both data contexts (real and synthetic).

In Tables 3 and 4, th_evo and ψ do not have a particular influence on the execution time of KD-means. Indeed, the range (difference between the maximum and minimum values) of Δt does not exceed 10.2 seconds with respect to ψ. It is about 30 seconds for th_evo. If we take into account the Δt magnitude of the three algorithms, these differences are negligible.

In the synthetic data, concerning ψ, the values of Δk are identical except for $\psi = 4$ where the value is slightly higher than the others by 0.2. The values of vi have a maximum difference of 0.4. This difference is 0.1 when $\psi \in [8, 10, 12]$ for which vi is minimal. At $\psi = 4$ Δk is slightly higher by 0.3 compared to the rest. In real data, the range is 1.1 for vi but at 0.2 if $\psi \in [4, 8]$. This range is 0.7 for Δk but drops to 0.2 for $\psi \in [4, 8, 10]$.

Concerning th_evo in the synthetic data, the values of Δk and vi have a range of 0.2. In real data, the respective ranges of Δk and vi are 0.5 and 0.4. They drop to 0.2 when $\psi \in [0.20, 0.25, 0.30]$ for Δk and $\psi \in [0.15, 0.20, 0.25, 0.30]$ for vi.

The observed ranges of the two metrics vi and Δk are lower than the values of the same metrics of the competitors algorithms in the real and synthetic data. But if we are stricter by just accepting ranges less than or equal to 0.2 then the optimal values, when the data are gaussian, are $\psi \in [8, 10, 12]$ and th_evo taking all the values tested. When the data is real (not necessarily gaussian clusters), the optimal values are $\psi \in [4.8]$ and $th\_evo \in [0.20, 0.25, 0.30]$.

5 CONCLUSION AND PERSPECTIVES

The estimation of the number of clusters (k), that k-means must find, is a major problem in the literature. And it is more significant when data is massive and clusters overlap. We provided a solution, based on kd-tree, that automatically estimates the value of k on massive and high dimensional data. It is robust even if clusters overlap. Four criteria were defined to guide the procedure for estimating k. Through experiments, we have shown that our solution is very competitive compared to the known x-means and g-means algorithms, which have proven to be unsuitable for scaling up and overlapping clusters. We plan to improve our solution in several areas:

- implement a strategy that manages nested indexing. Because the indices of a node are also in the parent node;
- use random kd-tree and multiple kd-tree in case the number of dimensions is even larger to have more very good performance in searching for information stored in kd-tree or to have the best data separation. This performance will help our solution to make better decisions on cluster merging.

ACKNOWLEDGMENTS

These studies are supported by the ANRT and Capgemini funding under CIFRE-Capgemini partnership.

REFERENCES

[1] Nir Ailon, Yudong Chen, and Huan Xu. 2013. Breaking the Small Cluster Barrier of Graph Clustering. *CoRR* abs/1302.4549 (2013). arXiv:1302.4549
[2] David Arthur and Sergei Vassilvitskii. 2007. k-means++: the advantages of careful seeding. In *ACM-SIAM symposium on Discrete algorithms*. 1027–1025. arXiv:1212.1121
[3] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18 (September 1975), 509–517. Issue 9. https://doi.org/10.1145/361002.361007
[4] Gianna M. Del Corso. 1997. Estimating an Eigenvector by the Power Method with a Random Start. *SIAM J. Matrix Anal. Appl.* 18, 4 (Oct. 1997), 913–937. https://doi.org/10.1137/S0895479895296689
[5] T. Cover and P. Hart. 2006. Nearest Neighbor Pattern Classification. *IEEE Trans. Inf. Theor.* 13, 1 (Sept. 2006), 21–27. https://doi.org/10.1109/TIT.1967.1053964
[6] Charles Elkan. 2003. Using the Triangle Inequality to Accelerate K-means. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning (ICML'03)*. AAAI Press, 147–153.
[7] Aislan G. Foina, Judit Planas, Rosa M. Badia, and Francisco Javier Ramirez-Fernandez. 2011. P-means, a parallel clustering algorithm for a heterogeneous multi-processor environment. In *Proceedings of the 2011 International Conference on High Performance Computing and Simulation, HPCS 2011*. IEEE, 239–248.
[8] E.W. Forgy. 1965. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics* (1965).
[9] Guojun Gan, Chaoqun Ma, and Jianhong Wu. 2007. *Data Clustering: Theory, Algorithms, and Applications (ASA-SIAM Series on Statistics and Applied Probability)*. Society for Industrial and Applied Mathematics.
[10] Greg Hamerly and Charles Elkan. 2003. Learning the K in K-means. In *Proceedings of the 16th International Conference on Neural Information Processing Systems (NIPS'03)*. MIT Press, 281–288.
[11] Anil K. Jain. 2010. Data clustering: 50 years beyond K-means. *Pattern Recognition Letters* 31, 8 (jun 2010), 651–666.
[12] A K Jain, M N Murty, and P. J. Flynn. 1999. Data clustering: a review.
[13] Chuanren Liu, Tianming Hu, Yong Ge, and Hui Xiong. 2012. Which distance metric is right: An evolutionary k-means view. In *Proceedings of the 12th SIAM International Conference on Data Mining, SDM 2012*.
[14] Songrit Maneewongvatana and David M. Mount. 1999. It's Okay to Be Skinny, If Your Friends Are Fat. In *Center for Geometric Computing 4th Annual Workshop on Computational Geometry*.
[15] Marina Meilă. 2003. Comparing Clusterings by the Variation of Information. In *Learning Theory and Kernel Machines*, Bernhard Schölkopf and Manfred K. Warmuth (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 173–187.
[16] Dau Pelleg and Andrew Moore. 2000. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In *In Proceedings of the 17th International Conf. on Machine Learning*. Morgan Kaufmann, 727–734.
[17] D T Pham, S S Dimov, and C D Nguyen. 2005. Selection of K in K-means clustering. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science* 219, 1 (jan 2005), 103–119.
[18] Hanan Samet. 2005. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
[19] M. A. Stephens. 1974. EDF Statistics for Goodness of Fit and Some Comparisons. *J. Amer. Statist. Assoc.* 69, 347 (1974), 730–737.
[20] Jan N. van Rijn, Geoffrey Holmes, Bernhard Pfahringer, and Joaquin Vanschoren. 2014. Algorithm Selection on Data Streams. In *Discovery Science*, Sašo Džeroski, Panče Panov, Dragi Kocev, and Ljupčo Todorovski (Eds.). Springer International Publishing, 325–336.
[21] Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems* 2, 1 (1987), 37 – 52. Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists.
[22] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. 1996. BIRCH: An Efficient Data Clustering Method for Very Large Databases. *SIGMOD Record* 25, 2 (jun 1996), 103–114.