A Recursive k-means Initialization Algorithm for Massive Data

Marco Capó[1], Aritz Pérez[1], and José A. Lozano[1,2]

Basque Center for Applied Mathematics, BCAM. Bilbao, Spain
University of the Basque Country UPV/EHU. San Sebastian, Spain
mcapo@bcamath.org, aperez@bcamath.org, ja.lozano@ehu.eus

Abstract. Due to the progressive growth of the amount of data available in a wide variety of scientific fields, it has become more difficult to manipulate and analyze such information. Even as datasets have grown in size, the *k-means* algorithm remains one of the most popular clustering methods, in spite of its dependency on the initial settings. In this work, we propose an efficient initialization method, intended for massive data, that deals with this problem by reducing the entire dataset into a small set of representatives. Preliminary experimental results indicate that this method outperforms well known initialization techniques, such as the *k-means++*, in both number of distance calculations and data scans.

1 Introduction

The exponential increase of the data volumes that scientists, from different backgrounds, face on a daily basis implies the development of simple yet scalable tools that eases the analysis and characterization of such information [6]. One of the most relevant analysis is data clustering. This process consists of grouping a given dataset into a predetermined amount of disjoint sets, called clusters. This is done in a way such that intra-cluster similarity is high and the inter-cluster similarity is low. Furthermore, clustering is a basic task of many areas, such as artificial intelligence, machine learning, data mining and pattern recognition [10].

Even when there exist a wide variety of clustering methods, the k-means algorithm remains one of the most popular ones [4],[8]. In fact, it has been identified as one of the top 10 algorithms in data mining [17].

1.1 k-means

Given a set of n data points (instances) $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ in $\mathbb{R}^d$ and an integer k, the clustering problem is to determine a set of k centroids $\zeta = \{c_1, \ldots, c_k\}$ in $\mathbb{R}^d$, so as to minimize the following error function:

$$E(\zeta) = \sum_{\mathbf{x} \in D} \min_{i=1,\ldots,k} \|\mathbf{x} - c_i\|^2, \tag{1}$$

This optimization problem is known to be NP-hard [1].

The k-means has two stages: Initialization, in which we set the starting set of centroids, and an iterative stage, called *Lloyd's algorithm* [13]. Additionally,

Lloyd's algorithm also consists of two steps: The assignment step, in which each instance is assigned to its closest centroid, and the centroid's update step. The time required for the assignment step is $O(nkd)$, while the centroid's update step and the computation of the error function is $O(nd)$. Hence, the assignment, or distance computation, stage is the most time consuming phase of the algorithm. Conveniently, both phases of the k-means can be easily parallelized [18]. When dealing with massive data applications, parallel implementation of the algorithm is a major key to meet the scalability of the problem [17]. Therefore, in practice, the method is appealing.

In terms of the initialization, it is widely reported in the literature that the performance of the Lloyd's algorithm highly depends upon this stage [14]. Hence, one might need several re-initializations before achieving a solution of acceptable quality. In addition, a poor initialization could lead to an exponential running time in the worst case scenario [16]. All these features are major downsides and show the importance of defining an appropriate initialization strategy.

There exist several approaches to initialize the k-means. One of the earliest, and most commonly used reference, was proposed by Forgy in 1965 [5]. It consists of defining the initial centroids set as k randomly selected instances from the dataset. The intuition behind this approach is that, by choosing the prototypes randomly, we are more likely to choose a point near an optimal cluster center, since such points tend to be where the highest density points are located [15]. The main disadvantage of this approach is that there is no guarantee that two, or more, of the selected seeds will not be near the center of the same cluster [15].

We also consider an alternative initialization procedure that is based on simple probabilistic seeding procedures. In particular, the *k-means++* method, proposed by Arthur and Vassilvitskii in [3], consists of randomly selecting only the first centroid from the dataset. Each subsequent initial centroid is chosen with a probability proportional to the distance with respect to the previously selected set of centroids. The key idea of this cluster initialization technique is to preserve the diversity of seeds while being robust to outliers. The k-means++ algorithm leads to an $O(\log k)$ approximation of the optimal error after the initialization [3]. The drawback of this approach is referred to its sequential nature, as well as to the fact that it requires k scans of the entire dataset, therefore it has a complexity of $O(nkd)$. Moreover, the sequential nature of this initialization technique hinders its parallelization.

Bahmani et al [4] proposed a parallel version of the k-means++, called *k-means* $\|$. The idea behind this method is to sample $O(k)$ points per iteration, instead of one single instance, as proposed in the k-means++. This process is repeated for, approximately, $O(\log n)$ rounds. At the end of this step, we remain with $O(k \log n)$ points, which are then reclustered into k initial centers. Hence, this approach outperforms k-means++ in both sequential and parallel settings [4].

In this work, we propose a parallel initialization algorithm based on a recursive data partitioning process that reduces the number of distance calculations and data scans. The rest of this article is organized as follows: In Section 2, we

describe the idea behind our initialization method. In Section 3, we show a series of experiments in which we analyze the effect of different factors such as the size of the dataset, n, the dimension of the instances, d, and the number of clusters, k, over the performance of our algorithm. Finally, in Section 4, we define the next steps and possible improvements to our current work.

2 Recursive k-means initialization algorithm

We propose a novel, iterative initialization strategy for the k-means algorithm that is based on a sequence of recursive partitions of the dataset. The objective is to approximate the solution of the k-means for the full dataset by recursively applying a weighted version of the k-means over a growing, yet small, number of representatives of the dataset.

In the first step of the iterative procedure, the dataset is partitioned into a number of disjoint subsets, called blocks, which are contained in equally sized hypercubes. Each block is then characterized by a representative and its corresponding weight. Finally, a weighted version of Lloyd's algorithm is applied over the set of representatives. From one iteration to the next, a more refined partition is constructed by dividing each hypercube into 2^d equally sized hypercubes. Such partition allows subsets of the previous iteration to reallocate in a different cluster, in order to reduce the overall error. This process is repeated until a certain number of iterations is achieved or when the approximation, in two consecutive iterations, changes slightly.

2.1 Recursive Partitions

As we commented before, our algorithm successively partitions the dataset into blocks contained in hypercubes of smaller volume. Each non-empty block, or *active block*, is characterized by a representative (centroid of the block) and its associated weight (cardinality of the block).

In order to generate the m-th partition, $\mathcal{P}_m$, we divide each dimension into 2^m equally sized intervals, i.e., 2^{dm} hypercubes. Furthermore, the j-th block of the partition $\mathcal{P}_m$ is denoted by $\mathcal{B}_j^m$ and the active set of a partition $\mathcal{P}_m$ as $\mathcal{A}_m = \{\mathcal{B}_j^m : |\mathcal{B}_j^m| > 0\}$. Observe that $|\mathcal{A}_m| \leq \min\{n, 2^{md}\}$. For each partition $\mathcal{P}_m$, we also compute the set of its corresponding weight $W_m = \{w_j^m = |\mathcal{B}_j^m| : \mathcal{B}_j^m \in \mathcal{A}_m\}$ and the set of representatives $\mathcal{C}_m = \{\mu_j^m = \sum_{x \in B_j^m} x/w_j^m : \mathcal{B}_j^m \in \mathcal{A}_m\}$. The entire partition process has an $O(nd)$ time complexity.

Our algorithm generates a sequence of partitions $\mathcal{P}_{m_{min}}, \ldots, \mathcal{P}_{m_{max}}$, where m_{min} is the smallest integer such that $|\mathcal{A}_{m_{min}}| \geq k$ and $m_{max} \geq m_{min}$. The partition $\mathcal{P}_{m+1}$ is generated after dividing the previous partition, $\mathcal{P}_m$: One block of the m-th partition is divided into 2^d blocks of the $(m+1)$-th partition, see Fig.1.

2.2 Initialization Algorithm

In this section, we present our initialization algorithm: **Recursive Partition Based Initialization** or just the **RPI** algorithm. This algorithm is mainly based on a weighted version of Lloyd's algorithm, called *weighted Lloyd's algorithm*,

which is applied over the set of representatives of a given partition, taking into consideration the weight associated to each block.

Algorithm 1: RPI Algorithm

Input: Dataset D, number of clusters k, integers $\{m_{min}, m_{max}\}$ and threshold ϵ.
Output: Initial set of centroids.
for $m = m_{min} : m_{max}$ **do**
  Step 1 Construct the partition $\mathcal{P}_m$.
  Step 2 Define the local dataset $\mathcal{C}_m$ and its respective weights set W_m.
  Step 3 Update the centroid's set approximation, $\zeta_m = \{c_j^m\}_{j=1}^k$:
  $\zeta_m =$ *Weighted_ Lloyd*$(\mathcal{C}_m, W_m, k, \zeta_{m-1})$
  Step 4 Compute the centroid's set displacement measure:
  $\mathbf{v}(\zeta_{m-1}, \zeta_m) = \max_{j=1,\dots,k} \|c_j^m - c_j^{m-1}\|^2$
  if $\mathbf{v}(\zeta_{m-1}, \zeta_m) \leq \epsilon$ **then**
    Return ζ_m
  end
end
Return ζ_M

As can be seen in Algorithm 1, at each iteration of the RPI algorithm, we first construct the partition $\mathcal{P}_m$ (Step 1) and then we determine the corresponding set of representatives $\mathcal{C}_m$ and weights W_m (Step 2). Later on, we update the centroids approximation by applying the weighted Lloyd's algorithm over these sets, taking as initial centroids the approximation for the previous iteration, ζ_{m-1} (Step 3). In the first iteration, we set ζ_{m-1} as k random points of $\mathcal{C}_m$ (Forgy's type initialization). The algorithm iterates until the stopping criteria is met. This occurs when $m = m_{max}$ or when the centroids' set approximation, in two consecutive iterations, ζ_{m-1} and ζ_m, does not change significantly (Step 4).

Regarding the time complexity at the m-th iteration, for Step 1, we observe that, if we first construct the partition $\mathcal{P}_M$, where $M = m_{max}$, and then use it to generate $\mathcal{P}_m$, this process has an $O(|\mathcal{A}_M|d)$ time cost. For Step 2, we just perform $O(|\mathcal{A}_m|d)$ computations, while the time required for the *weighted Lloyd's algorithm* (Step 3) is $O(|\mathcal{A}_m|kd)$. Finally, the last step of the algorithm just performs $O(kd)$ computations. Hence, the overall complexity of the RPI algorithm is $O(|\mathcal{A}_M|kd)$.

2.3 Graphical Example

In the following example, we consider a set of 10000 points generated from a mixture of three $2D$ Gaussians. We compute, as a reference, the solution for $k = 3$ by the means of the k-means++ method. After ten runs, we obtained, on average, an error of 11393.45 with a standard deviation of 4.69, the number of Lloyd iterations was, on average, 21.40 with a standard deviation of 4.22. Next, we show the evolution of the RPI algorithm, for $M = 6$. In Fig.1, the red circles represent the initial set centroids, the yellow diamonds the final set of centroids and the blue points the set of block representatives, $\mathcal{C}_m$, for each iteration.

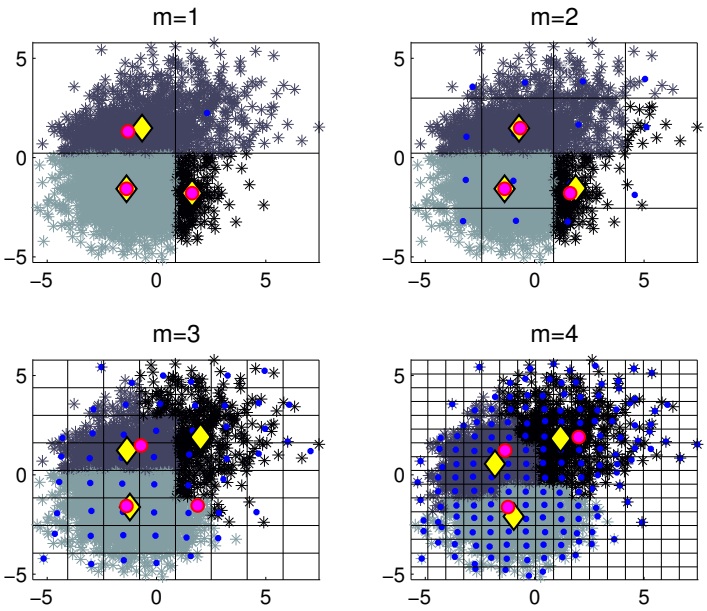

Fig. 1: Best clustering obtained for $m = \{1, 2, 3, 4\}$

m	It	Act Blocks	Cost Funct
1	2	4	14050.06
2	2	15	14024.38
3	9	53	12350.41

m	It	Act Blocks	Cost Funct
4	8	173	11424.24
5	3	528	11408.40
6	4	1361	11389.54

Table 1: Information for the recursive method

We observe that, even with $m = 4$, which in this case implies 173 active blocks, we have a fairly good approximation of the average best solution found by the k-means++ algorithm for the entire 10000 points. Furthermore, as we increase m, the associated cost function converges to such solution. The intuition behind this method is to transform a random initial set of centroids into a competitive one, by using small groups of representatives, instead of the entire dataset. Then, we consider higher values of m to refine such an approximation.

3 Preliminary Experimentation

In this section, we perform a series of experiments so as to understand the properties of our algorithm: Reduction in the number of distance computations, monotone decrease of the error of the initialization until convergence to the best found solution for the entire dataset, as we increase M, etc. In this section, we generate the dataset as a d-dimensional mixture of k Gaussians with a component overlap of 0.20. In particular, we set $k = \{4, 8\}$, $d = 5$, and $n = \{1000, 10000, 100000\}$. For each setting, we generate 10 replicates of the dataset.

We compare the performance of the k-means++ with respect to the RPI algorithm. In particular, we focus on two factors: The relation between the number of distance computations, for both RPI and k-means++ algorithms, with

respect to the dataset size, n. Additionally, we analyze the quality of the initialization for different values of M with respect to the average error obtained for the k-means++ algorithm, E_{K++}.

In Fig.2, we observe the number of distance computations for the different stages the RPI algorithm for $M = \{1, \ldots, 6\}$ and the k-means++ method.

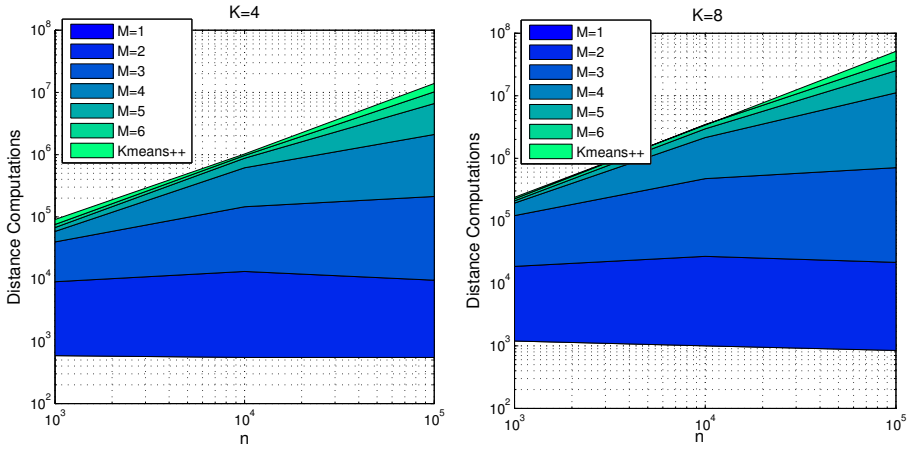

Fig. 2: Distance computations with respect to n

Notice that the number of distance calculations for small values of M, i.e., $M < 3$, does not necessarily increase with respect to the dataset size, n. This is plausible since, for small values of M, the number of active blocks for the different partitions are similar, independently of the number of instances, see Fig.3. Evidently, as we consider greater values of M, the number of active blocks will increase and so will the number of distance computations. Observe that, in this example, k does not affect the number of active blocks.

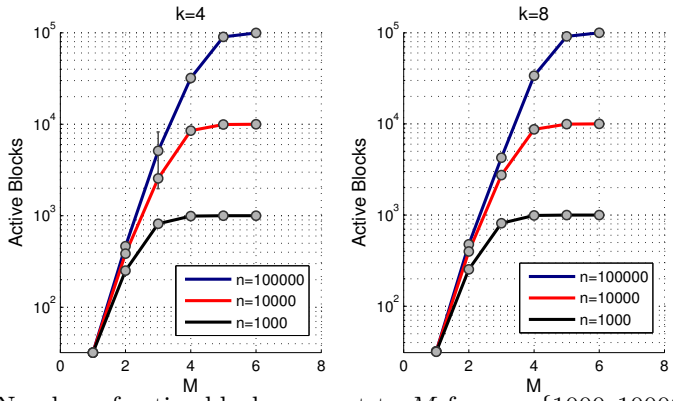

Fig. 3: Number of active blocks respect to M for $n = \{1000, 10000, 100000\}$

Moreover, after our initialization step, the Lloyd's algorithm requires a significantly smaller number of iterations over the entire dataset. In particular, after initializing with the k-means++, it needed, on average, 24, 31 and 44 iterations for $n = 1000, 10000, 100000$, respectively. Meanwhile, for the RPI initialization,

it only required, on average, 2, 2 and 3 iterations for $n = 1000, 10000, 100000$, respectively. This last increase is expected since, as we can see in Fig.4, for $M = 3$ and $n = 1000$, the RPI algorithm roughly generates as many active blocks as the number of instances. However, for $n = 10000$ and $n = 100000$, it needs at least $M = 5$ and $M = 6$, respectively, in order to approach the size of the dataset.

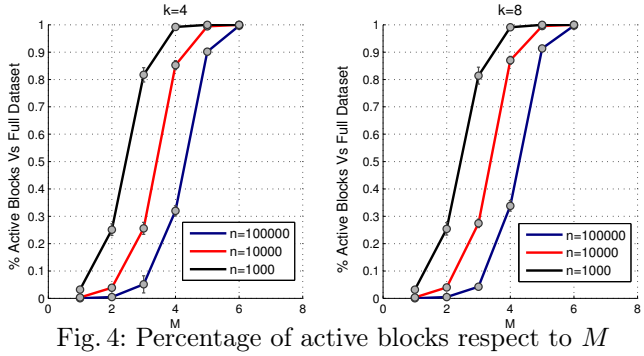

Fig. 4: Percentage of active blocks respect to M

As we commented before, we also want to analyze the quality of the solution for different values of M. To do so, we consider the quality rate $\rho(E_M) = \frac{E_{K++} - E_M}{E_{K++}}$, where E_M is the error obtained at the iteration M of the RPI algorithm. This quotient measures the percentage of improvement (if $\rho(E_M) > 0$) for the M-th iteration with respect to the average error obtained after 10 runs of the *k-means++* method, see Fig.5.

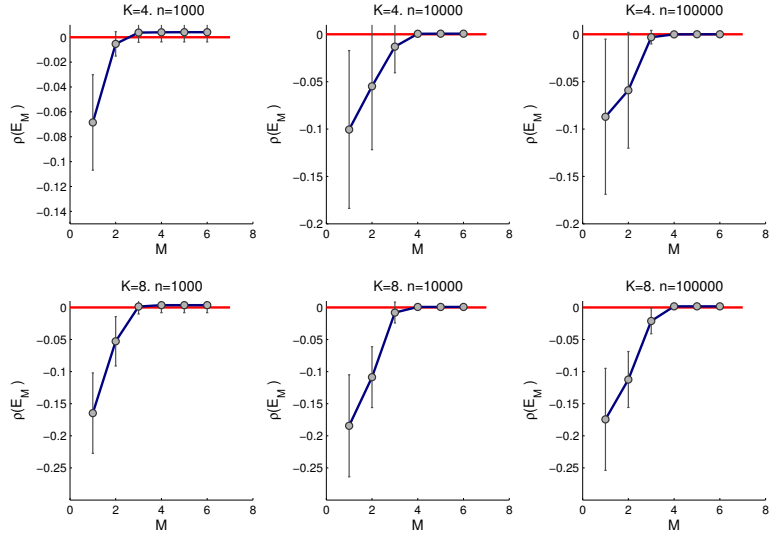

Fig. 5: Quality of the approximation for $M = \{1, 2, 3, 4, 5, 6\}$

Notice that, as M increases, the quality rate of the centroids found by the RPI algorithm increases monotonically until converging to the best found solution. With $M = 3$, for all the cases, the initialization has an error smaller than 2% with respect to the best solution. Moreover, in some cases, even with $M = 4$,

our initialization already had a smaller error than the optimal solution obtained with the k-means++ method. This is a remarkable improvement since, as can be seen in Fig.2, for $M = 3$ and $M = 4$, our approach required, on average, over $O(10^1)$ and $O(10^2)$ times fewer distance calculations, respectively.

Finally, we want to analyze the effect of the dimension, d, over our algorithm. We fix $n = 10000$, $k = 4$, $M = 4$ and $d = \{5, 7, 9\}$. The experiment is replicated 10 times.

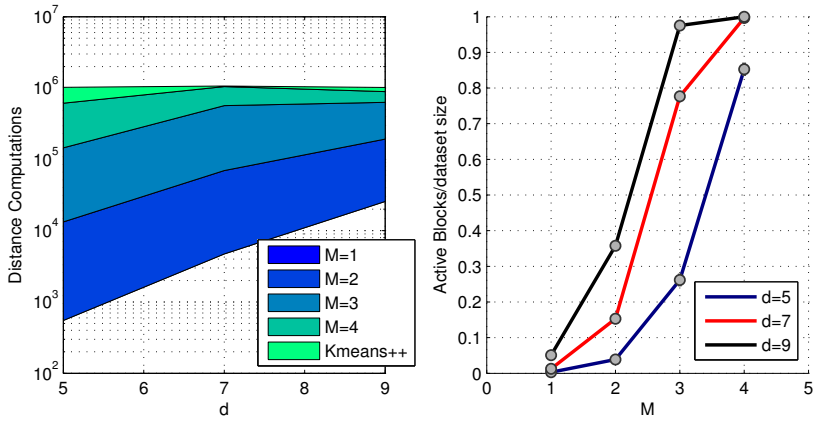

Fig. 6: Distance Computations respect to d

In the earlier stages of our algorithm, i.e., $M < 3$, the number of distance calculations increases as we increase the dimension. The reason is that, for larger values of d, we will generate a significantly larger amount of active blocks, see Fig.6. In contrast, in the latter stages of the algorithm, the number of distance calculations can be smaller, even if the dimension is increased: Independent of the dimension, the number of active blocks is closer to the number of instances, n. in particular, for $d = 9$ and $M = 3$, we already have over 90% of the data points n, hence solving the k-means problem at this stage is similar to solving the problem associated with $M = 4$. Since we are using the best solution obtained for $M = 3$ as the initial guess for $M = 4$, we will then need a smaller amount of iterations and, therefore, of distance calculations.

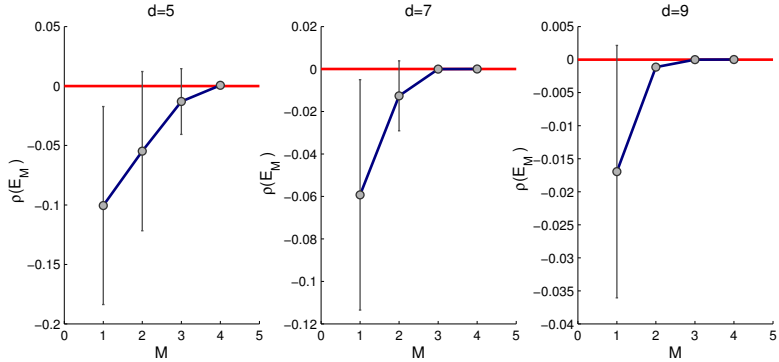

Fig. 7: Quality of the approximation for $M = \{1, 2, 3, 4\}$

In Fig.7, we observe that the error behavior is similar to that shown in Fig.5, i.e., the error in the approximation seems to decrease monotonically with respect to the increase of M. However, the only difference is that the percentage of the error is smaller for larger values of d. This was expected, since we have a greater amount of active blocks in this case.

4 Final Comments

In general, we can see that the RPI algorithm can generate competitive initializations, even for small values of M, by using a small set of representatives in comparison to the full dataset. In most of the examples, we observe a dramatic reduction in the number of distance computations, as well as in the number of Lloyd's iterations over the entire dataset.

Furthermore, at the earlier stages of the RPI (small M), the size of the dataset, n, does not have a relevant impact on the number of iterations or distance computations for the associated weighted k-means problem. Thus, the number of computations, especially for massive data application, is utterly reduced. However, at this stage, increasing the dimension implies the generation of exponentially more active blocks and, hence, the increment of distance computations. For this reason, we might need to consider different approaches on how to generate the subdivisions of a given partition, especially when dealing with larger dimensions.

In particular, one possible approach consists of characterizing the active blocks that lie on a cluster boundary, i.e., blocks that are close to two or more clusters [9]. Such a characterization controls the number of active blocks and, therefore, reduces the computational cost of the algorithm. We are currently partitioning the data in a recursive midpoint fashion that might generate several active blocks with a small probability of changing their current cluster affiliation, see Fig.1. In this sense, it might be more valuable to perform such cuts in areas that are more likely to have subsets associated with different clusters. For this reason, we plan to define a low computational cost algorithm to determine the frontiers at each iteration. The blocks in this area will have a greater priority when selecting the regions that we want to partition in the next iteration.

One last, but not least, advantage of the RPI algorithm is the fact that its parallelization is direct. The RPI algorithm mainly depends on two steps: The data partition process and the application of the weighted version of Lloyd's algorithm. In the first step, each point and/or sub-block can independently decide to which block it belongs to, hence the construction of the set of representatives and weights can be done in a parallel manner. Analogously, for the second phase, given a set of prototypes, each data point can separately decide to which cluster it belongs to and the update of the centroid can be simply computed by averaging the points [7,18]. For this reason, we also plan to implement the RPI algorithm on a parallel framework such as *Apache Spark*.

Acknowledgments

Marco Capó and Aritz Pérez are partially supported by the Basque Government through the BERC 2014-2017 program and by the Spanish Ministry of Economy

and Competitiveness MINECO: BCAM Severo Ochoa excelence accreditation SVP-2014-068574 and SEV-2013-0323. José A. Lozano is partially supported by the Basque Government (IT609-13), and Spanish Ministry of Economy and Competitiveness MINECO (BCAM Severo Ochoa excellence accreditation SEV-2013-0323 and TIN2013-41272P).

References

1. Aloise D., Deshpande A., Hansen P., Popat P.: NP-hardness of Euclidean sum-of-squares clustering. Machine Learning, 75, 245 − 249 (2009).
2. Alsabti K., Ranka S., Singh V.: An Efficient K-means Clustering Algorithm, In: Proc. First Workshop High Performance Data Mining (1998).
3. Arthur D., Vassilvitskii S.: k-means++: the advantages of careful seeding. In: Proceedings of the 18^{th} annual ACM-SIAM Symp. on Disc. Alg, pp. 1027 − 1035 (2007).
4. Bahmani B., Moseley B., Vattani A., Kumar R., Vassilvitskii S.:Scalable K-means++. In: Proceedings of the VLDB Endowment (2012).
5. Forgy E., "Cluster analysis of multivariate data: Efficiency vs. interpretability of classifications". Biometrics, 21, 768 − 769 (1965).
6. Committee on the Analysis of Massive Data, Committee on Applied and Theoretical Statistics, Board on Mathematical Sciences and Their Applications, Division on Engineering and Physical Sciences, National Research Council: Frontiers in Massive Data Analysis, In:The National Academy Press (2013). (Preprint).
7. Dean J. and Ghemawat S.: MapReduce: Simplified data processing on large clusters. Communications of the ACM, 51(1), 107 − 113 (2008).
8. Dubes R., Jain A.: Algorithms for Clustering Data, Prentice Hall, Inc. (1988).
9. Hung M., Wu J., Chang J. and Yang D.: An Efficient k-Means Clustering Algorithm Using Simple Partitioning, Jour. of Info. Sci. and Eng., 21, 1157 − 1177 (2005).
10. Jain, A. K., Murty, M. N., Flynn, P. J.: Data clustering: a review . ACM Computing Surveys 31, 264 − 323 (1999).
11. Judd D., McKinley P. and Jain A.: Large-scale parallel data clustering, IEEE Transactions on Pattern Analysis and Machine Intelligence, 20, 871 − 876 (1998).
12. Kanungo T., Mount M., Netanyahu N., Piatko C., Silverman R. and Wu A.: An Efficient k-Means Clustering Algorithm: Analysis and Implementation, IEEE Transactions on Pattern Analysis and Machine Intelligence, 24, 881 − 892 (2002).
13. Lloyd S.P.: Least Squares Quantization in PCM, IEEE Trans. Information Theory. 28, 129 − 137 (1982).
14. Peña J.M., Lozano J.A., Larrañaga P.: An empirical comparison of four initialization methods for the k-means algorithm. Pattern Recognition Letters, 20(10), 1027 − 1040 (1999).
15. Redmond S., Heneghan C.: A method for initialising the K-means clustering algorithm using kd-trees, Journal Pattern Recognition Letters, 28(8), 965 − 973 (2007).
16. Vattani A.: K-means requires exponentially many iterations even in the plane, Discrete Computional Geometry, 45(4), 596 − 616 (2011).
17. Wu X., Kumar V., Ross J., Ghosh J., Yang Q., Motoda H., McLachlan J., Ng A., Liu B., Yu P., Zhou Z., Steinbach M., Hand D., Steinberg D.: Top 10 algorithms in data mining. Knowl. Inf. Syst., 14, 1 − 37 (2007).
18. Zhao W., Ma H. and He Q.: Parallel K-Means Clustering Based on MapReduce, Cloud Computing Lecture Notes in Computer Science, 5931, 674 − 679 (2009).