Journal of Parallel and Distributed Computing 145 (2020) 34–41

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

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

Performance enhancement of a dynamic K-means algorithm through a parallel adaptive strategy on multicore CPUs

Giuliano Laccetti [a], Marco Lapegna [a,*], Valeria Mele [a], Diego Romano [b], Lukasz Szustak [c]

[a] *Department of Mathematics and Applications, University of Naples Federico II, Italy*
[b] *Institute for High-Performance Computing and Networking (ICAR), National Research Council (CNR), Italy*
[c] *Department of Computer Science, Czestochowa University of Technology, Poland*

ARTICLE INFO

Article history:
Received 30 January 2020
Received in revised form 4 May 2020
Accepted 15 June 2020
Available online 25 June 2020

Keywords:
K-means clustering
Adaptive algorithm
Unsupervised learning
Multicore CPUs

ABSTRACT

The K-means algorithm is one of the most popular algorithms in Data Science, and it is aimed to discover similarities among the elements belonging to large datasets, partitioning them in K distinct groups called clusters. The main weakness of this technique is that, in real problems, it is often impossible to define the value of K as input data. Furthermore, the large amount of data used for useful simulations makes impracticable the execution of the algorithm on traditional architectures. In this paper, we address the previous two issues. On the one hand, we propose a method to dynamically define the value of K by optimizing a suitable quality index with special care to the computational cost. On the other hand, to improve the performance and the effectiveness of the algorithm, we propose a strategy for parallel implementation on modern multicore CPUs.

1. Introduction and related works

In the last thirty years, several theories, methodologies, and tools have been introduced *to learn from data*, that is to understand, comprehensively, complex phenomena through the analysis of large structured or unstructured datasets representing real problems. This wealth of knowledge has often changed its name over the years (for example, *data mining* or *big data*), and today is commonly known as *data science* [10].

One of the most used tools in this field is a class of unsupervised learning methods known as Clustering Algorithms, whose main aim is to collect similar data in the same group according to a precise metric [29].

An extensive literature is available in the field, giving an overall picture of the clustering approaches, in sequential computing environments as well for parallel architectures [1,20,23,29,31,36,39,40]. Very often, they use different taxonomies for the description of the algorithms, but all of them consider the K-means algorithm as one of the most useful computational tools.

It can be described as follows. Given a dataset consists of N elements in the d-dimensional space $S = \{\mathbf{x}_n : \mathbf{x}_n \in \mathbf{R}^d, \quad n = 1, \ldots, N\}$, and an integer K, the K-means algorithm defines a partition $\mathcal{P}_K = \{C_k : C_k \subseteq S, \quad k = 1, \ldots, K\}$ of the elements of S in K non empty subsets C_k called *clusters*, each of them with N_k elements, and where the elements showing some similarity according to a given criterion are assigned to the same cluster.

In its classical version, the K-means algorithm identifies each cluster with a representative $\mathbf{c}_k$, called centroid, computed with the following vector operation:

$$\mathbf{c}_k = \frac{1}{N_k} \sum_{\mathbf{x}_n \in C_k} \mathbf{x}_n \qquad k = 1, \ldots, K \tag{1}$$

while the similarity of an element $\mathbf{x}_n$ with the centroids $\mathbf{c}_k$ can be measured using some metric in R^d, but usually, the Euclidean distance

$$dist(\mathbf{x}_n, \mathbf{c}_k) = \|\mathbf{x}_n - \mathbf{c}_k\|_2 \quad k = 1, \ldots, K$$

is employed. With these definitions, the K-means algorithm assigns each $\mathbf{x}_n$ to the cluster C_δ that minimizes the distance from its centroid $\mathbf{c}_\delta$, that is:

$$\|\mathbf{x}_n - \mathbf{c}_\delta\|_2 = \min_{k=1,\ldots,K} \|\mathbf{x}_n - \mathbf{c}_k\|_2 \tag{2}$$

The following Algorithm 1 then provides an outline of the Basic K-means Algorithm.

```
Algorithm 1: Basic $K$-means Algorithm
(1) Define $K$ clusters $C_k$, assigning to them $N_k$ elements $x_n \in S$.
(2) repeat
      (2.1) for each cluster $C_k$ compute the centroids $\mathbf{c}_k$ as in (1):
      (2.2) for each $\mathbf{x}_n \in S$
              (2.2.1) search the cluster $C_\delta$ as in (2)
              (2.2.2) assign $\mathbf{x}_n$ to $C_\delta$
            endfor
    until (no change in the reassignment)
```

* Correspondence to: Department of Mathematics and Applications, University of Naples Federico II, via Cintia - Monte S. Angelo, 80126 Napoli, Italy.

E-mail addresses: giuliano.laccetti@unina.it (G. Laccetti), marco.lapegna@unina.it (M. Lapegna), valeria.mele@unina.it (V. Mele), diego.romano@cnr.it (D. Romano), lszustak@icis.pcz.pl (L. Szustak).

https://doi.org/10.1016/j.jpdc.2020.06.010
0743-7315/© 2020 Elsevier Inc. All rights reserved.

A detailed study of the convergence of the K-means algorithm is reported in [34], and several papers address the mathematical properties of the algorithm. They show, in particular, that the final result strongly depends on several factors, such as the choice of the initial partition in step (1) or the number of clusters K supplied as input data.

For example, regarding the choice of the initial partition, in [30], the K clusters are defined starting from a random selection of K points of S chosen as centroids. In [2], the clusters are identified by selecting their centroids as far away as possible among them. Similarly, in [19], a dataset preprocessing is proposed to find well-separated points with a high density of elements surrounding them. In general, all the methods proposed for this problem guarantee only a convergence to a local minimum.

The choice of the value of K is one of the main challenges in the design of this algorithm. Very often, in several real and large problems, the dataset does not show sufficiently clear patterns to define such a value as input data. If it is defined too small, there is a high chance that dissimilar objects will be gathered in the same cluster, while if it is set too large, there is the risk that related elements will be scattered in separate clusters.

For the above, several researchers introduce methods to set the value of K dynamically at run time, so to realize a trade-off between efficiency (a small value of K) and accuracy (a high affinity of elements in each cluster). As an example, in [3] the authors iteratively determine the values of K by merging two clusters when the distance between the centroids is smaller than an inter-clusters tolerance or by dividing the cluster with a standard deviation of his points (that is a measure of their affinity) greater than an intra-cluster threshold. A similar approach is described in [21], where the measures of the intra and inter-cluster distances (the so-called silhouette) are averaged. Lastly, in [4], the authors introduce a method based on the maximization of a function based on the sum of squares of the distances among elements belonging to the same cluster and different clusters.

Finally, in the last years, further efforts have been addressed toward parallel implementations of the K-means algorithm in several high-performance computing environments. Significant algorithms are described, for example, in [11] for distributed memory architectures, in [22,24,33] for multicore CPUs and in [7, 8] for GPUs based systems. Almost all these studies exploit the role that a large amount of data can play in an implementation based on the data parallelism programming model.

Our work joins this research trend introducing a parallel adaptive K-means algorithm, with new features respect to the other works. On the one hand, it defines the value of K dynamically by dividing only selected clusters with non-similar elements, so to reduce the number of displacements of the items among the clusters. On the other hand, it improves the performance of the algorithm, implementing it in a multicore CPUs based computing environment by using two different parallelization strategies.

The present paper is, therefore, organized as follows: in Section 2 we introduce the new parallel adaptive K-means algorithm describing the clusters generation procedure and the parallelization strategy; in Section 3 we report the implementation details of the algorithm with special care to the data structure organization; in Section 4 we show the results obtained from various experiments aimed to validate the new algorithm; in Section 5 we discuss the results, and in Section 6 we conclude the work.

2. A new parallel adaptive K-means algorithm

This section has a double aim: from the one hand, we introduce a methodology aimed to dynamically define the number of clusters with a reduced computational cost of the algorithm, and, on the other hand, we propose a parallel implementation in multicore environments.

A widespread method to define the value of K without considering it as an input data is to execute the Basic K-means Algorithm several times, with an increasing value of K, until a given quality index, used as a measure for the goodness of the solution, satisfies the user requirements (e.g., [35]). To this aim, it is possible to find a large variety of quality indexes in the literature (see, for example, [15]). One of the most used index to determine the number of clusters existing in a data set is the Root-Mean-Square Standard Deviation (RMSSD) that represents a measure of the average affinity of the elements grouped inside the several clusters of the partition $\mathcal{P}_K$:

$$R_{\mathcal{P}_K} = \left[\frac{\sum_{k=1}^{K} \sum_{\mathbf{x}_n \in C_k} \|\mathbf{s}_n - \mathbf{c}_k\|_2^2}{d(N-K)} \right]^{1/2} \tag{3}$$

When the number of clusters K increases, the (3) initially decreases but, when a good affinity among the elements in each $C_k \in \mathcal{P}_K$ is reached, there are no further improvements. In other words, it is possible to increase the number of clusters until adding a new cluster does not give a significant reduction of $R_{\mathcal{P}_K}$. The previous approach is known as the "Elbow" method [18].

From the above it is, therefore, possible to design the following Iterative K-means Algorithm (Algorithm 2) executing the Algorithm 1 several times with the value of K increasing at each step until the $R_{\mathcal{P}_K}$ index shows no further improvements:

```
Algorithm 2: Iterative K-means Algorithm
(1) Set the number of clusters $K = 0$
(2) repeat
        (2.1) Increase the number of clusters $K = K + 1$
        (2.2) Define $K$ clusters $C_k$, assigning to them $N_k$ elements of $S$.
        (2.3) repeat
                (2.3.1) for each cluster $C_k$ compute the centroids $\mathbf{c}_k$ as in (1)
                (2.3.2) for each $\mathbf{x}_n \in S$
                        (2.3.2.1) search the cluster $C_\delta$ as in (2)
                        (2.3.2.2) assign $\mathbf{x}_n$ to $C_\delta$
                    endfor
              until (no change in the reassignment)
        (2.4) update $R_{\mathcal{P}_K}$ as in (3)
    until (the variation of $R_{\mathcal{P}_K}$ is smaller than a given threshold)
```

Let consider now the Computational Cost (CC) of the previous Algorithm 2, paying special attention to the kernels inside the iterative structure 2.3. At this regard, it is easy to show that the computation of the centroids $\mathbf{c}_k$ in the Step 2.3.1, by using the (1), needs

$$CC_{2.3.1} = \sum_{k=1}^{K} dN_k = Nd \quad \text{floating points operations}$$

whereas the (2), used for the search of the new cluster of each element $\mathbf{x_n}$ in the step 2.3.2.1, requires

$$CC_{2.3.2.1} = NKd \quad \text{floating-point operations}$$

Finally, let consider the cost of step 2.3.2.2 of Algorithm 2. It is a critical section of the procedure because it strongly depends on the initial partition $\mathcal{P}_K$ of the dataset S defined at the beginning of each iteration (step 2.2). With an inappropriate initial partition, there is a high risk that a large number of displacement of elements $\mathbf{x}_n$ among the clusters C_k in step 2.3.2.2 occurs, before the stopping criterion of the iterative structure 2.3 is satisfied. For such reason, we propose a strategy designed to control such a number.

Our idea is based on the assumption that, at the iteration K, the elements of S are already grouped according to their similarity, in the $K-1$ clusters of the partition $\mathcal{P}_{K-1}$. Many of them do not need to be displaced so that it is possible to concentrate the attention only on the clusters showing still little affinity among the elements. A traditional way to compute the similarity of a set of N_K elements is through the standard deviation:

$$V_k = \sqrt{\frac{1}{N_k - 1}\sum_{n=1}^{N_k} \|\mathbf{x}_n - \mathbf{c}_k\|_2^2}$$

Greater the value V_k is, more distant the elements $\mathbf{x}_n$ from centroid $\mathbf{c}_k$ are, and the cluster consists of dissimilar elements.

The key choice of our strategy is then based on the reuse of the partition $\mathcal{P}_{K-1}$ defined in the previous iteration $K-1$, avoiding to start with a generic partition in step 2.2. More precisely, let $C_\gamma \in \mathcal{P}_{K-1}$ be the cluster with the largest standard deviation at the iteration $K-1$, that is:

$$C_\gamma \quad \text{such that} \quad V_\gamma = \max_{k=1,\dots,K-1} V_k \tag{4}$$

the proposed algorithm divides only this cluster into two new subclusters C_α and C_β, and defines the initial partition $\mathcal{P}_K$ as follows:

$$\begin{array}{lll} K=0 & \mathcal{P}_0 = \{C_0\} & \text{where} \quad C_0 \equiv S \\ K \geq 1 & \mathcal{P}_K = \mathcal{P}_{K-1} - \{C^*_{K-1}\} \cup \{C_\alpha, C_\beta\} & \end{array} \tag{5}$$

In this regard, we remark that the strategy of reorganizing only the subsets of partitions showing poor values of some quality index is an assessed approach in the design of the so-called adaptive algorithms in other scientific computing areas. For example, several adaptive algorithms for numerical integration [26,27] or computational fluid dynamics [17,38] refine only the subdomains of a partition where the discretization error is significant. Similar adaptive approaches are used to develop modified versions also of the K-means algorithm (e.g., [5]).

From what has been said, we, therefore, propose the following Adaptive K-means Algorithm:

```
Algorithm 3: Adaptive K-means Algorithm
(1) Set the number of clusters $K = 0$
(2) repeat
    (2.1) Increase the number of clusters $K = K + 1$
    (2.2) find the cluster $C_\gamma$ as in (4)
    (2.3) define the new partition of clusters $\mathcal{P}_K$ as in (5)
    (2.4) repeat
        (2.4.1) for each cluster $C_k$ compute the centroids $\mathbf{c}_k$ as in (1)
        (2.4.2) for each $\mathbf{x}_n \in S$
            (2.4.2.1) search the cluster $C_\delta$ as in (2)
            (2.4.2.2) assign $\mathbf{x}_n$ to $C_\delta$
        endfor
    until (no change in the reassignment)
    (2.5) update $R_{\mathcal{P}_K}$ as in (3)
until (the variation of $R_{\mathcal{P}_K}$ is smaller than a given threshold)
```

Today, all the most powerful high-performance computing systems are based on multicore CPUs (e.g., www.top500.org) so that an efficient implementation of algorithms on these devices can be considered as the first level of a complex software stack, able to solve real-world problems in these environments with high performances [37]. In a multicore CPU, P computing units (the cores) share the same main memory, each of them has a private set of registers so that the operating system can dispatch on them concurrent threads, allowing the implementation of the algorithm based on the shared memory Multiple Program Multiple Data paradigm.

An analysis of Algorithm 3 shows that the K-means algorithm is well suited for implementations on high-end parallel computing environments based on multicore CPUs. More precisely, we observe that it is possible to identify in it at least two forms of parallelism:

parallelism at clusters level. In this case, the number of clusters K determines the degree of parallelism. More precisely the clusters C_k with $k = 1, \dots, K$ are distributed among the P threads. This approach finds a natural application in step 2.4.1.

parallelism at elements level. In this case, the number of elements N determines the degree of parallelism. More precisely the elements $\mathbf{x}_n$ with $n = 1, \dots, N$ are distributed among the P threads. This approach finds a natural application in step 2.4.2.

The critical task in Algorithm 3 is then the step 2.4.2.2. In such instruction, some elements $\mathbf{x}_n \in S$ are assigned to new clusters taking into account their distance from the centroids defined in (2). This displacement is the cause of a high risk of race condition when different threads attempt to access the data structures that are used to describe the clusters (see the next section for more details on the implementation issues). Such a step is then the only sequential task of Algorithm 3, and it may be the reason for the decay of the algorithm efficiency.

Let T_P and T_s respectively be the running time of the algorithm with P threads and the time required to run the sequential sections of the algorithm. Then the *Serial Fraction* is defined as $\alpha = T_s/T_1$, and the Amdhal law states that the Speed-up S_P and the Efficiency E_P can be respectively represented as follows:

$$S_P = \frac{T_1}{T_P} = \frac{1}{\alpha + (1-\alpha)/P} \qquad E_P = \frac{T_1}{PT_P} = \frac{1}{\alpha P + (1-\alpha)} \tag{6}$$

From (6), we observe that also for a moderate value of α, the Speed-up and the Efficiency can be strongly degraded, so that it is of paramount importance to start with a suitable partition in the step 2.3 designed to reduce the total number of the displacements of the elements in the step 2.4.2.2.

3. Implementation details

To better describe the new parallel Adaptive K-means Algorithm, in this section, we report some implementation details, with particular attention to the data structures that are used to describe the management of the clusters (see also Fig. 1).

To store the N elements $\mathbf{x}_n$, our implementation uses a 2-dimensional $N \times d$ static array S, where each row represents a d-dimensional element of the dataset. This choice is due to the greater efficiency in accessing the elements compared to other implementations based on dynamic data structures such as lists or trees. In any case, we recall that one of the most expensive steps of the algorithm is the displacement of the elements among the clusters (step 2.4.2.2). Each displacement has a not negligible cost of $O(d)$ memory accesses with a severe impact on the performance so that our algorithm leaves the physical order of the rows of S unchanged to reduce such a cost.

To describe the structure of each cluster C_k, we used a pointers array PT, where each component points to an element of the dataset. In the array PT, each cluster is therefore defined as a set of contiguous items, each of them pointing to a row of S representing an element $\mathbf{x}_n$ of the cluster C_k. The new partition $\mathcal{P}_K$ in step 2.4.2.2, is therefore achieved by exchanging only components of PT, each of them with a cost of $O(1)$ memory accesses. We remark that the array PT is shared among all threads so that any change in it has to be carried out in a critical section

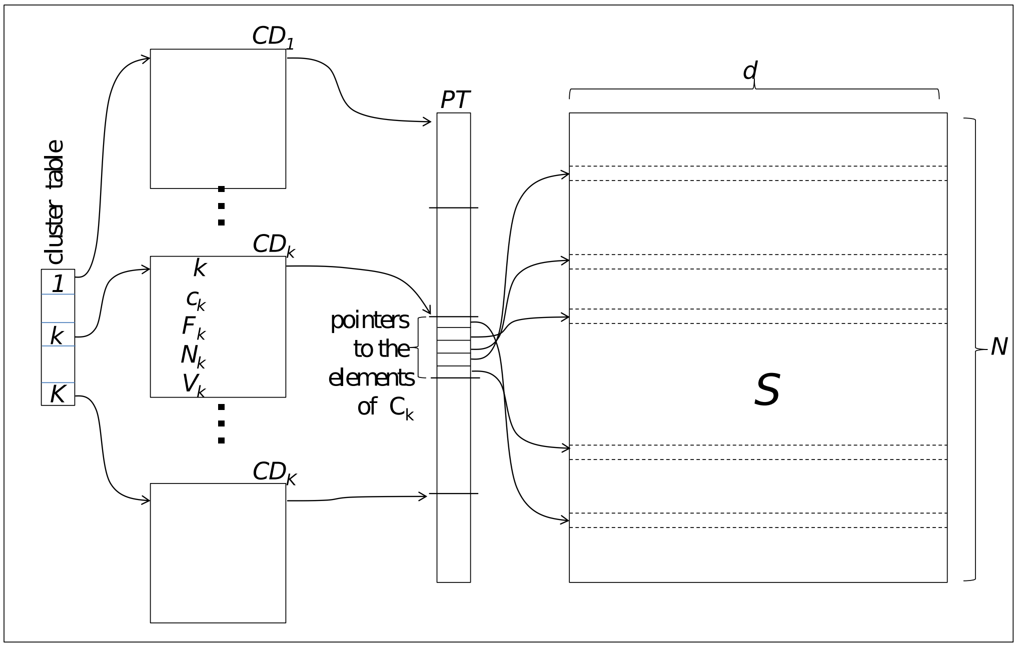

Fig. 1. Organization of the main data structures in the parallel adaptive K-means algorithm.

with exclusive access by each thread. As the number of memory accesses in the step 2.4.2.2 decreases, then the value of α in (6) is reduced, with an improvement of the speed-up and efficiency of the algorithm.

In our implementation, each cluster C_k consists of N_k rows of S referenced by contiguous pointers in PT, and it is fully described by a Cluster Descriptor (CD_k). It is a data structure containing:

- k: a cluster identifier
- $\mathbf{c}_k$: the centroids of the cluster
- F_k: a pointer to the first of the contiguous items of PT referencing the cluster elements
- N_k: the number of cluster elements
- V_k: the standard deviation of the elements of the cluster

Finally, a table of pointers called Cluster Table CT provides direct access to the Cluster Descriptors CD_k.

With the described data structures organization, it is then possible to efficiently manage all information required in Algorithm 3.

4. Experimental results

We tested the accuracy and the efficiency of the proposed Adaptive K-means Algorithm running several experiments using the HPC cluster available at the Department of Science and Technologies of the University of Naples Parthenope. This facility is a computing environment where each node is equipped with two Intel Xeon 16-core 5218 CPUs running at 2.3 GHz for a total of 32 computing cores per node, and 192 Gbytes of main memory. In this system, we implemented the Algorithm 3 in C language using the POSIX thread library for the thread management.

As a benchmark for our experiments, we used the following datasets with different dimension and from different application areas, taken from the University of California (UCI) Machine Learning Repository [12]:

Iris [13]. This problem is based on a quite small but very popular dataset used for multivariate classification. The dataset contains $N = 150$ instances of iris flowers classified in $K = 3$ classes of $N_k = 50$ instances, each of them representing different types of iris plants. The items are described by $d = 4$ attributes representing respectively the width and length of petals and sepals.

Letters [14]. In this case, the problem is represented by a larger dataset with $N = 20\,000$ black-and-white rectangular images, each of them representing one of the $K = 26$ capital letters of the English alphabet. Each letter is described by $d = 16$ attributes (e.g., the dimension of the character, the number of the edges, the position of the black pixels in the image, and other graphical features).

Wines [6]. This dataset contains $N = 4898$ instances of Portuguese wines described by $d = 11$ attributes (e.g., acidity, sugar, sulfate, alcohol, and other organoleptic features). The wines are grouped in $K = 11$ classes according to their quality (from 0 to 10).

Banknotes [16]. This problem is described by a dataset containing $N = 1372$ instances representing images that were taken from genuine and forged banknote-like specimens. Some mathematical transformations were used to extract $d = 4$ features from images, that are classified in $K = 2$ clusters (true or false).

Cardio [9]. In this dataset, $N = 2126$ fetal cardiotocographic reports (CTGs) have been processed and $d = 21$ diagnostic features measured (e.g., acceleration, pulsation, short and long term variability, and other physiological features.). We use the dataset to classify the elements concerning $K = 10$ morphologic patterns.

Clients [32]. In this dataset, data are related to the direct marketing campaigns of a banking institution. The marketing campaigns were based on $N = 45\,211$ phone calls to access if a bank deposit would be or not subscribed (that is $K = 2$ clusters). We classify the elements according to $d = 16$ attributes (age, job, education, marital status, and other individual features).

The first set of experiments is aimed to evaluate the effectiveness of the Adaptive K-means Algorithm (Algorithm 3) with respect to the Iterative K-means Algorithm (Algorithm 2), through

Table 1
Performance of Algorithm 2 and Algorithm 3.

Problem	*K*	Algorithm 2		Algorithm 3	
		Disp	*Time*	*Disp*	*Time*
Iris	3	67	5.4e−4	59	5.8e−4
Letters	26	1 001 349	55.9	130 801	23.8
Wines	11	81 095	2.6	18 507	9.1e−1
Banknote	2	682	2.7e−3	682	2.9e−3
Cardio	10	25 158	4.3e−1	6553	2.5e−1
Clients	2	30 734	2.3	30 734	2.3

Table 2a
Performance of Algorithm 3 on the Iris, Letters and Wines problems.

P	Iris			Letters			Wines		
	Time	S_P	E_P	*Time*	S_P	E_P	*Time*	S_P	E_P
2	3.4e−4	1.70	0.85	13.22	1.80	0.90	5.2e−1	1.74	0.87
4	2.5e−4	2.32	0.58	7.00	3.48	0.87	3.3e−1	2.72	0.68
8	2.9e−4	2.00	0.25	4.31	5.52	0.69	2.5e−1	3.60	0.45
16	3.3e−4	1.76	0.11	3.31	7.20	0.45	2.2e−1	4.16	0.26
32	3.6e−4	1.60	0.05	3.23	7.36	0.23	2.0e−1	4.48	0.14

Table 2b
Performance of Algorithm 3 on the Banknotes, Cardio and Clients problems.

P	Banknotes			Cardio			Clients		
	Time	S_P	E_P	*Time*	S_P	E_P	*Time*	S_P	E_P
2	1.7e−3	1.70	0.85	1.4e−1	1.74	0.87	1.35	1.70	0.85
4	1.2e−3	2.44	0.61	9.2e−2	2.60	0.65	9.4e−1	2.44	0.61
8	1.3e−3	2.16	0.27	7.2e−2	3.44	0.43	9.9e−1	2.32	0.29
16	1.4e−3	2.08	0.13	6.7e−2	3.68	0.23	9.5e−1	2.40	0.15
32	1.3e−3	2.24	0.07	6.5e−2	3.84	0.12	8.9e−1	2.56	0.08

a comparative analysis of the total execution time with $P = 1$ thread. Table 1 shows such results reporting, for each test problem, the total number of elements displaced among the clusters (*disp*) and the total elapsed time in seconds (*time*) for the generation of the reported number of clusters K.

The second set of experiments is aimed to evaluate the performance of Algorithm 3 through the values S_P and E_P given by (6) up to 32 computing cores. For these experiments, we remark that, although each core can run more than one thread with the Intel Hyperthreading Technology, we only used one thread for each core. Tables 2a and 2b report the total running time in seconds (*time*), the Speed-up (S_P), and the Efficiency (E_P) by using $P = 2, 4, 8, 16$ and 32 threads for the six test problems.

For a complete analysis, we also compared the performance of the Adaptive K-means Algorithm (Algorithm 3) with the Iterative K-means Algorithm (Algorithm 2) and other implementations of the k-means algorithm on multicore CPUs:

Pkmeans, described in [33]. It is based on a parallel strategy at Cluster Level similar to that implemented in our Algorithm 3, but with a fixed value for the number of cluster K. The algorithm has been run on a problem with $N = 1\,000\,000$ items with $d = 2$ attributes. The number of generated clusters is $K = 10$.

McKmeans, described in [24]. It is based on the concept of transactional memory to guarantee thread safety indirectly. The algorithm has been run on a problem with $N = 100\,000$ items with $d = 500$ attributes. The number of generated clusters is $K = 20$.

Algorithm 2 and Algorithm 3 are run on the "Letters" problems ($N = 40\,000$ items, $d = 16$ attributes and $K = 26$ clusters). Table 3 reports the results of such an experiment.

Finally, the last set of experiments studies the reduction of the Root-Mean-Square Standard Deviation (3) when the number of clusters K increases. This trial represents a critical test because it allows the evaluation of the quality of the partition $\mathcal{P}_K$ generated by Algorithm 3, compared to that of the traditional K-means algorithm. More precisely, we are interested in verifying that a smaller number of displacements in Algorithm 3 does not produce a distribution of the elements among the clusters of lower quality than that provided by Algorithm 2. To this aim, we compared the Root-Mean-Square Standard Deviations $R_{\mathcal{P}_K}$ of both Algorithm 2 and Algorithm 3 for the six selected test problems with a variable value of K. The graphs in Fig. 2 report such results.

Table 3
Performance comparison of Algorithm 3 with other parallel implementation of the K-means algorithm.

P	Pkmeans [33]		McKmeans [24]		Algorithm 2		Algorithm 3	
	S_P	E_P	S_P	E_P	S_P	E_P	S_P	E_P
2	1.78	0.89	1.76	0.84	1.68	0.84	1.80	0.90
4	3.22	0.80	1.98	0.62	2.50	0.62	3.48	0.87
8	n.a.	n.a.	2.30	0.43	3.43	0.43	5.52	0.69
16	n.a.	n.a.	n.a.	n.a.	3.85	0.24	7.20	0.45
32	n.a.	n.a.	n.a.	n.a.	4.64	0.15	7.36	0.23

5. Discussion

From Table 1, we observe better effectiveness of the Algorithm 3 with respect to Algorithm 2, measured in terms of the number of elements displaced among the clusters and the total execution time, mainly for large values of K (that is the datasets related to the Letters, Wines, and Cardio problems). More precisely, when the number of iterations is large, we can better appreciate the effects of the adaptive strategy aimed to reuse the partition already defined at the previous iterations, with a smaller number of elements that need to be moved among the clusters. For a little value of K (namely Clients, Iris, and Banknotes problems), the experiments do not report significant benefits for Algorithm 3. In these cases, we register a similar number of displacements and about the same execution time because the adaptive strategy is applied only for a small number of iterations.

Parallel performance reported in Tables 2a and 2b shows a strong dependence of the Speed-up and the Efficiency on the number of clusters. The higher the value of K, the higher the Efficiency we get. This aspect can be explained through the two forms of parallelism, at clusters level and items level, we introduced in Algorithm 3. A significant K increases the concurrency and allows full usage of the p computing units. On the other hand, because the degree of concurrency is independent of the number of attributes, we do not observe significant differences among Speed-up and Efficiency for problems with different values for d.

From Table 3, it is interesting to observe that, with the proposed adaptive strategy, Algorithm 3 shows significantly better Speed-up and Efficiency with respect to a parallel version of Algorithm 2. About this aspect, we have previously stated in Sections 2 and 3, that the displacements of the items $\mathbf{x}_n$ among the clusters are implemented through a rearrangement of the pointers in the array *PT* occurring in a sequential section of the algorithm. This step is, therefore, the cause of a significant Serial Fraction α in the (6), so that it has a disruptive impact on S_P and E_P. With a smaller computational cost for this step in Algorithm 3, we then also reduce the Serial Fraction, with a significant improvement of the Speed-up and Efficiency.

Furthermore, regarding the performance comparison with other algorithms reported in the same Table 3, it is first evident that the Adaptive K-means Algorithm (Algorithm 3) outperforms the McKmeans algorithm based on the transactional memory. We believe that the bad performance of the McKmeans algorithm is motivated by the sequential access to shared memory with high idle time. In essence, the transactional memory model is primarily

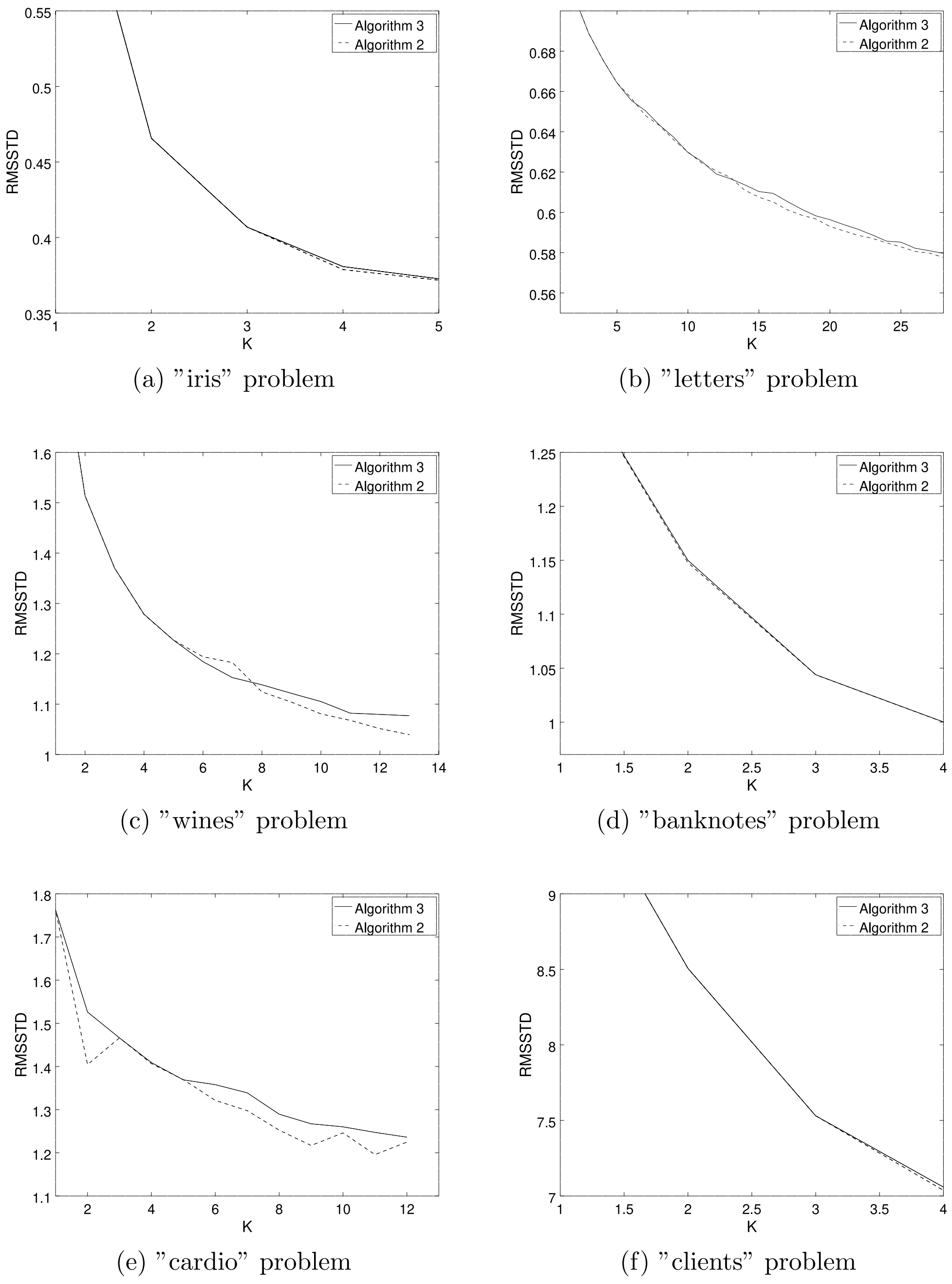

Fig. 2. $R_{\mathcal{P}_K}$ varying the number of iteration for Algorithm 2 (dashed line) and Algorithm 3 (solid line), for the six test problems.

designed to avoid race conditions, but it is not a guarantee of high performance. The Pkmeans algorithm, instead, uses a parallelization strategy at Cluster Level similar to those implemented in our Algorithm 3, with comparable performance values. In any case, we emphasize that the Pkmeans algorithm requires a fixed number of cluster K, while our Algorithm 3 overcomes this problem with the adaptive procedure described in Section 3.

Finally, from the analysis of the values of the Root-Mean-Square Standard Deviation (3) reported in 2, we register a very similar behavior of the Iterative K-means Algorithm (Algorithm 2) and the Adaptive K-means Algorithm (Algorithm 3), very often with almost identical graphs. More precisely, in our experiments, we measured a difference of no more than 5% between the Root-Mean-Square Standard Deviation $R_{\mathcal{P}_K}$ related to the partitions generated by the two algorithms, asserting the effectiveness of the Adaptive K-means Algorithm proposed in Section 2.

6. Conclusions

This paper describes our studies aimed to improve the performance of the K-means algorithm in case the number of clusters K is not available as input data. This issue is a common situation in real applications so that traditional approaches are based on several runs of the algorithm with different values of K attempting to optimize some quality index, with a high risk to increase the computational cost. The method we introduced is based on an adaptive procedure aimed to minimize the number of displacements of the elements of the dataset among the clusters, preserving, at the same time, the clusters of the partition with small values of the standard deviation. Furthermore, the paper addresses the problem of the algorithm implementation in a multicore CPU based system, giving special attention to the reduction of the Serial Fraction so to improve Speed-up and Efficiency. Several experiments confirm these expectations: the proposed algorithm achieves better performance and efficiency with respect to other approaches, mainly for large values of the number of clusters K, showing, at the same time, similar values of the Root-Mean-Square Standard Deviation, used as a measure of the global quality of the generated partition.

We already planned future works aimed to integrate, in a single hybrid implementation, the Adaptive K-means Algorithm with other algorithms designed for different advanced computing environments, such as GPUs based or distributed memory computing systems [25,28].

CRediT authorship contribution statement

Giuliano Laccetti: Conceptualization, Supervision, Resources. **Marco Lapegna:** Conceptualization, Methodology, Investigation, Writing - review & editing. **Valeria Mele:** Data curation, Software, Writing - original draft. **Diego Romano:** Software, Validation, Writing - original draft. **Lukasz Szustak:** Software, Validation, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported by institutional funding provided by the universities of the researchers.

References

[1] A. Ahmad, S.S. Khan, Survey of state-of-the-art mixed data clustering algorithms, IEEE Access 7 (2019) 31883–31902.
[2] D. Arthur, S. Vassilvitskii, K-means++: The advantages of careful seeding, in: Proc. of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2007, pp. 1027–1035.
[3] G.H. Ball, D.J. Hall, ISODATA, A Novel Method of Data Analysis and Pattern Classification, Technical Report, DTIC Document, 1965.
[4] T. Calinski, J. Harabasz, A dendrite method for cluster analysis, Comm. Statist. Theory Methods 3 (1974) 1–27.
[5] H. Chen, X. We, J. Hu, Proc. SPIE 6788, MIPPR 2007: Pattern Recognition and Computer Vision, 67882A, 2007.
[6] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, J. Reis, Modeling wine preferences by data mining from physicochemical properties, in: Decision Support Systems, Vol. 47, Elsevier, 2009, pp. 547–553.
[7] S. Cuomo, V. De Angelis, G. Farina, L. Marcellino, G. Toraldo, A GPU-accelerated parallel K-means algorithm, Comput. Electr. Eng. 75 (2019) 262–274.
[8] S. Cuomo, P. De Michele, E. Di Nardo, L. Marcellino, Parallel implementation of a machine learning algorithm on GPU, Int. J. Parallel Program. 46 (2018) 923–942.
[9] A. de Campos, et al., SisPorto 2.0 a program for automated analysis of cardiotocograms, J. Matern. Fetal Neonatal Med. 5 (2000) 311–318.
[10] V. Dhar, Data science and prediction, Commun. ACM 56 (2013) 64–73.
[11] I.S. Dhillon, D.S. Modha, A data-clustering algorithm on distributed memory multiprocessors, in: M.J. Zaki, C.T. Ho (Eds.), Large-Scale Parallel Data Mining, in: Lecture Notes in Computer Science, vol. 1759, Springer, Berlin, Heidelberg, 2002.
[12] D. Dua, C. Graff, UCI Machine Learning Repository, University of California, School of Information and Computer Science, Irvine, CA, 2017, http://archive.ics.uci.edu/ml.
[13] R. Duda, P.E. Hart, Pattern Classification and Scene Analysis, John Wiley & Sons, 1973, (Q327.D83).
[14] P.W. Frey, D.J. Slate, Letter recognition using Holland-style adaptive classifiers, Mach. Learn. 6 (1991) 161–182.
[15] D.G. Gan, C. Ma, J. Wu, Data Clustering: Theory, Algorithms, and Applications, in: ASA-SIAM Series on Statistics and Applied Probability, SIAM, Philadelphia, ASA, Alexandria, VA, 2007.
[16] S. Glock, E. Gillich, J. Schaede, V. Lohweg, Feature extraction algorithm for banknote textures based on incomplete shift invariant wavelet packet transform, in: DAGM-Symposium 2009, pp. 422–431.
[17] W. Haase, K. Misegades, M. Naar, Adaptive grids in numerical fluid dynamics, Numer. Methods Fluids 5 (1985) 515–528.
[18] M. Halkidi, Y. Batistakis, M. Vazirgiannis, On clustering validation techniques, J. Intell. Inf. Syst. 17 (2001) 107–145.
[19] J.A. Hartigan, M.A. Wong, Algorithm AS 136: A k-means clustering algorithm, J. R. Stat. Soc. Ser. C Appl. Stat. 28 (1979) 100–108.
[20] A. Joshi, R. Kaur, A review: Comparative study of various clustering techniques in data mining, Int. J. Adv. Res. Comput. Sci. Softw. Eng. 3 (2013) 55–57.
[21] L. Kaufman, P.J. Rousseeuw, et al., Finding Groups in Data: An Introduction to Cluster Analysis, J. Wiley & Sons, 1990.
[22] K. Kerdprasop, N. Kerdprasop, Parallelization of k-means clustering on multi-core processors, in: Proc. 10th WSEAS International Conference on Applied Computer Science, ACS'10, World Scientific and Engineering Academy and Society (WSEAS), 2010, pp. 472–477.
[23] W. Kim, Parallel clustering algorithms. Survey, CSC 8530 Parallel Algorithms, spring, 2009.
[24] J.M. Kraus, H.A. Kestler, A highly efficient multi-core algorithm for clustering extremely large datasets, BMC Bioinformatics 11 (2010) art. 169.
[25] G. Laccetti, M. Lapegna, V. Mele, A loosely coordinated model for heap-based priority queues in multicore environments, Int. J. Parallel Program. 44 (2016) 901–921.
[26] G. Laccetti, M. Lapegna, V. Mele, D. Romano, A study on adaptive algorithms for numerical quadrature on heterogeneous GPU and multicore based systems, in: Proc. 10th International Conference on Parallel Processing and Applied Mathematics, PPAM 2013, in: Lecture Notes in Computer Science, vol. 8384, 2013, pp. 704–713.
[27] G. Laccetti, M. Lapegna, V. Mele, D. Romano, A. Murli, A double adaptive algorithm for multidimensional integration on multicore based HPC systems, Int. J. Parallel Program. 40 (2012) 397–409.
[28] G. Laccetti, M. Lapegna, R. Montella, A scalable unified model for dynamic data structures in message passing (clusters) and shared memory (multicore CPUs) computing environments, in: Proc. 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2018, 2018, pp. 599–608.
[29] R.S.M. Lakshmi Patibandla, N. Veeranjaneyulu, Survey on Clustering Algorithms for Unstructured Data, Springer Verlag, Singapore, 2018.
[30] J. MacQueen, Some methods for classification and analysis of multivariate observations, in: Proc. of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Vol. 1, Berkeley, CA, USA, 1967, pp. 281–297.
[31] A. Mohebi, S. Aghabozorgi, T. Ying Wah, T. Herawan, R. Yahyapour, Iterative big data clustering algorithm: a review, Softw. Pract. Exp. 46 (2016) 107–129.
[32] S. Moro, P. Cortez, P. Rita, A data-driven approach to predict the success of bank telemarketing, Decis. Support Syst. 62 (2014) 22–31.
[33] I.K. Savvas, D. Tselios, Combining distributed and multi-core programming techniques to increase the performance of k-means algorithm, in: Proc. IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE, 2017, pp. 95–100.
[34] S.Z. Selim, M.A. Ismail, K-means-type algorithms: A generalized convergence theorem and characterization of local optimality, IEEE Trans. Pattern Anal. Mach. Intell. 6 (1984) 81–87.
[35] A. Shafeeq, K.S. Hareesha, Dynamic Clustering of Data with Modified k-means algorithm, in: 2012 International Conference on Information and Computer Networks, Vol. 27, ICICN 2012, IACSIT Press, Singapore, 2012, IPCSIT.
[36] F. Shah, H. Doshi, M. Shah, M. D'silva, A comparative study on data mining clustering algorithms, Int. J. Res. Edu. Sci. Methods 6 (2018) 1–5.
[37] L. Szustak, P. Bratek, Performance portable parallel programming of heterogeneous stencils across shared-memory platforms with modern Intel processors, Int. J. High-Perform. Comput. Appl. 33 (2019) 507–526.
[38] J.F. Thompson, A survey of dynamically-adaptive grids in the numerical solution of partial differential equations, Appl. Numer. Math. 1 (1985) 3–27.
[39] X. Wang, A survey of clustering algorithms based on parallel mechanism, in: Computer Modeling, Simulation and Algorithm, CMSA 2018, in: Advances in Intelligent Systems Research Series, Atlantis Press, 2018.
[40] D. Xu, Y. Tian, A comprehensive survey of clustering algorithms, Ann. Data Sci. 2 (2015) 165–193.

Giuliano Laccetti is a full professor of computer science at the University of Naples Federico II, Italy. He received his Laurea degree (cum laude) in Physics from the University of Naples. His main research interests are Mathematical Software, High-Performance Architecture for Scientific Computing, Distributed Computing, Grid, and Cloud Computing, Algorithms on emerging hybrid architectures (CPU+GPU, ...), Internet of Things. He has been organizer and chair of several Workshops joint to larger International Conferences. He is the author (or co-author) of about 100 papers published in refereed international Journals, international books, and International Conference Proceedings.

Marco Lapegna received a Ph.D. in Applied Mathematics and Computer Science in 1991 from the University of Naples Federico II. He worked from 1991 until 2001 as Assistant Professor, and now he is an Associate Professor of Computer Science at the University of Naples Federico II. His research activity is aimed at the development of high performance distributed and parallel algorithms for computational mathematics for advanced architecture environments. He participated in projects funded by Italian and international institutions, and he is author of several scientific publications. His teaching activity concerns computer programming, operating systems, and distributed/parallel computing.

Valeria Mele today is a Researcher at the University of Naples Federico II (Naples, Italy). Degree in Informatics and Ph.D. in Computational Science. Her research activity has been mainly focused on development and performance evaluation of parallel algorithms and software for heterogeneous, hybrid, and multilevel parallel architectures, from multicore to GPU-enhanced machines and modern clusters and supercomputers. After attending the Argonne Training Program on ExtremeScale Computing (ATPESC) and visiting the Argonne National Laboratory (ANL, Chicago, Illinois, USA) several times, she is now mainly working on the designing, implementation and performance prediction/evaluation of software with/for the PETSc library.

Diego Romano was awarded a M.S. in Mathematics in 2000, and a Ph.D. degree in Computational and Computer Sciences from the University of Naples Federico II, Italy, in 2012. He obtained a permanent position as a researcher at the Italian National Research Council (CNR) in 2008, where he is currently employed at the Institute for High-Performance Computing and Networking (ICAR). His research interests include the performance and design of GPU Computing algorithms. Within this field, he works, for instance, on the Global Illumination problem in Computer Graphics, and mathematical models for performance analysis.

Lukasz Szustak received a D.Sc. Degree in Computer Science in 2019 and a Ph.D. granted by the Czestochowa University of Technology in 2012. His main research interests include parallel computing and mapping algorithms onto parallel architectures. His current work is focused on the development of methods for performance portability, scheduling, and load balancing, including the adaptation of stencil-based computations to modern HPC architectures.