Distributed $k$-means algorithm

Gabriele Oliva, Roberto Setola, and Christoforos N. Hadjicostis

Abstract—In this paper we provide a fully distributed implementation of the $k$-means clustering algorithm, intended for wireless sensor networks where each agent is endowed with a possibly high-dimensional observation (e.g., position, humidity, temperature, etc.). The proposed algorithm, by means of one-hop communication, partitions the agents into measure-dependent groups that have small in-group and large out-group “distances”. Since the partitions may not have a relation with the topology of the network—members of the same clusters may not be spatially close—the algorithm is provided with a mechanism to compute the clusters’ centroids even when the clusters are disconnected in several sub-clusters. The results of the proposed distributed algorithm coincide, in terms of minimization of the objective function, with the centralized $k$-means algorithm. Some numerical examples illustrate the capabilities of the proposed solution.

Index Terms—$k$-means, clustering, distributed algorithms, consensus

1 INTRODUCTION

Wireless sensor networks are increasingly asserting their presence in everyday life, as a powerful tool to measure and manage spatially dispersed information in several applications, such as monitoring, robot coordination, indoor localization, rescue, etc. Dealing with huge sets of multidimensional data is, in many cases, a non-trivial issue, and there is the need to provide agile and efficient methodologies to grasp the fundamental aspects of the ongoing situation.

In the literature, several distributed techniques have been provided in order to let a network of agents reach distributed agreement (or consensus) based on iterative strategies that rely at each iteration on local observations (for instance, [1], [2], [3]).

The resulting agreement, however, is on a single value for all the agents in the network, while in many circumstances it might be of interest to obtain multiple values, as well as to identify sets of observations that are homogeneous according to some criteria. In the literature, a powerful, yet centralized, solution to the latter problem is provided by means of the so called clustering algorithms. One of the most widely adopted clustering algorithms is the $k$-means algorithm, as well as its extensions such as fuzzy $c$-means [4] or Mixture of Gaussians [5] algorithms. The above approaches are examples of unsupervised learning algorithms, where a set of observations has to be partitioned in a finite number of groups (eventually considering soft partitions in the case of fuzzy $c$-means) with small in-group and large out-group distances, or has to be explained by a combination of probability distributions (Mixture of Gaussians).

The $k$-means algorithm in particular, is an iterative procedure where a set of $k$ centroids are obtained by alternating assignment phases (i.e., each observation is assigned to the nearest current centroid) and refinement phases where the average of the observations in the same partition becomes the new centroid.

In this paper, we provide a fully distributed implementation of the $k$-means algorithm for a network of agents, each holding a possibly high-dimensional observation or piece of information.

1.1 Paper Outline

The outline of the paper is as follows: after two subsections on notation and definitions that conclude this introduction, Section 2 reviews the $k$-means clustering algorithm, while Section 3 provides a discussion on related work and on the contribution of the paper; Section 4 outlines distributed consensus algorithms; Section 5 introduces the distributed $k$-means algorithm, while section 6 provides a discussion on the correctness and complexity of the proposed algorithm; simulation results and conclusions are provided in Sections 7 and 8 respectively.

1.2 Notation

$A'$ transpose of a matrix/vector $A$; 
$\#(A)$ cardinality of a set $A$; 
$A \otimes B$ Kronecker product of matrix/vector $A$ and $B$; 
$I_a$ $a \times a$ identity matrix; 
$I_1$ $a \times 1$ vector whose elements are all equal to 1;
We set of vertices of the graph $V$; $G$ set of edges of the graph $G$; $v_i$ $i$-th vertex of the graph $G$; $(v_i, v_j)$ directed edge connecting two vertices $v_i$ (source) and $v_j$ (destination) of graph $G$; $N_i$ neighborhood of agent $i$; $T$ identifier of a step in the (distributed) $k$-means algorithm; $t$ identifier of an iteration in a distributed consensus algorithm; $n$ number of agents/observations; $d$ size of the observations; $x_i$ $d \times 1$ vector of the $i$-th observation; $k$ number of clusters for $k$-means algorithm; $S_i(T)$ $i$-th cluster at step $T$; $r_{ij}(T)$ boolean decision variable representing the assignment of observation $x_i$ to cluster $S_j(T)$ at step $T$; $c_j(T)$ $d \times 1$ vector of the centroid of cluster $S_j(T)$ at step $T$; $D(T)$ objective functional of the $k$-means algorithm at step $T$; $M$ maximum number of iterations of the $k$-means algorithm; $\Delta_{\text{max}}$ positive parameter used to terminate the $k$-means algorithm; $z_i(t)$ generic, possibly vectorial, state assumed by the $i$-th agent of a consensus algorithm at iteration $t$; $z_{i0}$ generic, possibly vectorial, initial condition assumed by the $i$-th agent of a consensus algorithm at iteration $t$; $u_i(t)$ input for the $i$-th agent in consensus algorithms at iteration $t$; $\chi(\cdot)$ function to be computed by $\chi$-consensus; $t_{\text{max}}$ maximum number of iterations for consensus algorithms; $\tau_i$ final value of the state of the $i$-th agent in finite-time consensus algorithms; $w_{ij}$ coefficients that compose the input $u_i(N_i \cup \{i\}, t)$ of average-consensus algorithm; $W$ $n \times n$ matrix containing all the terms $w_{ij}$; $z(t)$ stack vector containing the state $z_i(t)$ of each agent at iteration $t$ of a consensus algorithm; $\delta_i$ number of steps necessary to calculate the average-consensus value in finite time for the $i$-th agent; $\gamma_{ij}$ $j$-th coefficient used by the $i$-th agent to calculate the average-consensus value in finite time; $\Gamma_i$ $1 \times 1$ vector containing the terms $\gamma_{ij}$; $q_i(g)$ minimal polynomial of agent $i$; $\beta_{ij}$ coefficients used to calculate $\gamma_{ij}$; $\alpha_{ij}$ coefficients used to calculate the minimal polynomial $q_i(g)$; $(\delta_i + 1) \times 1$ vector containing the terms $\alpha_{ij}$; $\Theta_i$ Observability matrix used to calculate the minimal polynomial $q_i(g)$; $z_{*, j}(t)$ $n \times 1$ vector containing the state of the agents when the $j$-th preparatory average-consensus is executed in the finite-time consensus; $Z_{i, \delta_i}$ $n \times (\delta_i + 2)$ matrix containing up to a step $\delta_i$ the state of the agent $i$ in all the preparatory average-consensus runs executed in the finite-time consensus; $\hat{n}$ upper bound of the number of agents $n$; $i^*$ index of the leader in the distributed $k$-means algorithm; $c_{ij}(T)$ $d \times 1$ vector containing the value of the $j$-th centroid as computed by the $i$-th agent at step $T$ of the distributed $k$-means algorithm; $\overline{c}_i(T)$ $kd \times 1$ stack vector containing the terms $c_{ij}(T)$ for all $j = 1, \ldots, k$; $\gamma_{ij}$ index of the centroid selected by agent $i$ at step $T$ of the distributed $k$-means algorithm; $c_{\gamma_{ij}}(T)$ $d \times 1$ vector containing the value of the centroid chosen by agent $i$ at step $T$ of the distributed $k$-means algorithm; $\mu_i(T)$ $k \times 1$ vector used to represent the choice of a centroid for agent $i$ at step $T$ of the distributed $k$-means algorithm; $\tilde{\mu}_i(T)$ $k \times 1$ vector used to represent the choice of a centroid for agent $i$ at step $T$ of the distributed $k$-means algorithm; $\epsilon_i(T)$ $kd \times 1$ vector used as initial condition of a max-consensus algorithm in order to let each agent calculate the value of the current centroids at step $T$ of the distributed $k$-means algorithm; $N_i^c(T)$ subset of $N_i$ containing agents with the same choice of a centroid at step $T$ of the distributed $k$-means algorithm; $G^c(T)$ subgraph of $G$ induced by $N_i^c(T)$ for all $i = 1, \ldots, n$; $SCC_{jh}(T)$ $d \times 1$ vector representing the centroid of the $h$-th sub-cluster of the $j$-th cluster at step $T$ of the distributed $k$-means algorithm;
1.3 Definitions

Let $G = \{V, E\}$ be a graph, where $V$ is a set of $n$ vertices $v_1, \ldots, v_n$ and $E$ is the set of edges $(v_i, v_j)$ that allow a communication from vertex $v_i$ to vertex $v_j$. A graph is said to be undirected if $(v_i, v_j) \in E$ whenever $(v_j, v_i) \in E$ (i.e., the communication is always bidirectional), and is said to be directed otherwise. A graph $G$ is connected if for any $v_i, v_j \in V$ there is a path whose endpoints are in $v_i$ and $v_j$, without necessarily respecting the orientation of edges. A graph $G$ is strongly connected if for any $v_i, v_j \in V$ there is a path whose endpoints are in $v_i$ and $v_j$, respecting the orientation of edges.

A graph $G$ can be represented by an $n \times n$ adjacency matrix $A$, whose elements $a_{ij} = 1$ if $(v_i, v_j) \in E$ and are zero otherwise. Let the in-neighborhood $N_i^{in}$ of a vertex $v_i$ be the set of vertices $\{v_j : (v_j, v_i) \in E\}$, while $N_i^{out}$ is the out-neighborhood of vertex $v_i$, i.e., the set of vertices $\{v_j : (v_i, v_j) \in E\}$; for undirected graphs the in-neighborhood and the out-neighborhood coincide and are referred to as the neighborhood $N_i$. In the remainder of this paper, where not explicitly stated, we will assume the graphs to be connected and undirected.

In the following we denote by $T$ the index of a step in the (distributed) $k$-means algorithm, while we denote by $t$ the index of an iteration of average-consensus or max-consensus algorithms executed within each step of the distributed $k$-means algorithm.

2 $K$-MEANS ALGORITHM

Consider a set of $n$ observations $x_1, \ldots, x_n$, where each observation $x_i$ is a vector in $\mathbb{R}^d$. Suppose we want to partition the $n$ observations into $k$ ($k \leq n$) sets or clusters $S_1, \ldots, S_k$. Specifically, we want to find a set of centroids $c_1, \ldots, c_k$, each associated to a cluster, and we want to solve the following optimization problem:

$$\min D = \sum_{i=1}^{n} \sum_{j=1}^{k} r_{ij} \|x_i - c_j\|^2$$

Subject to

$$\begin{align*}
\sum_{j=1}^{k} r_{ij} &= 1 \quad \forall i = 1, \ldots, n \\
r_{ij} &\in \{0, 1\} \quad \forall i = 1, \ldots, n; \forall j = 1, \ldots, k
\end{align*}$$

where $r_{ij} = 1$ if observation $x_i$ is assigned to the set $S_j$ and $r_{ij} = 0$ otherwise, and $c_j \in \mathbb{R}^d$ is the centroid of the observations within the set $S_j$.

Problem 1 is hard to solve exactly when $n$ and $k$ are large (it is NP-hard in general Euclidean space $\mathbb{R}^d$, even for 2 clusters [6] and for a general number of clusters $k$, even in the plane [7]), and in the literature several heuristic algorithms have been proposed. Among others, the $k$-means algorithm [8] proved its effectiveness.

The $k$-means algorithm starts with a random set of $k$ centroids $c_1(0), \ldots, c_k(0)$, and alternates at each step between an assignment and a refinement phase.

During the assignment phase, each observation $x_i$ is assigned to the set characterized by the nearest centroid, i.e.:

$$r_{ih}(T) = \begin{cases} 1 & \text{if } h = \arg\min_j \|x_i - c_j(T)\|^2 \\ 0 & \text{else} \end{cases}$$

During the refinement phase each centroid $c_j$ is updated as the centroid of the observations associated to $S_j(T)$, i.e.:

$$c_j(T + 1) = \frac{\sum_{i=1}^{n} r_{ij}(T) x_i}{\sum_{i=1}^{n} r_{ij}(T)}$$

The two steps are iterated until convergence or up to a maximum of $M$ iterations.

Figure 1 is an example of execution of the algorithm for a set of $n = 12$ observations in $\mathbb{R}^2$ and for $k = 3$. Specifically, Figure 1(a) shows with circles the initial centroids, Figure 1(b) and Figure 1(c) report the assignment and refinement phases for the first step, while Figure 1(d) depicts the assignment phase for the second step. The $k$-means algorithm is known to converge to a local optimum value, while there is no guarantee to converge
to the global optimum. However, given the complexity of the problem at hand, k-means algorithm is de facto the most diffused heuristic algorithm: indeed “ease of implementation, simplicity, efficiency, and empirical success are the main reasons for its popularity.”

Since there is a strong dependency on the initial choice of the centroids, a common practice is to execute the algorithm several times, each time with different initial conditions, and select the best solution.

Note that for each step, each of the $n$ observations, and each of the $d$ components of the observations, the algorithm calculates the difference with each of the $k$ centers; hence the computational complexity is $O(dknM)$.

Remark 1: A center that is not chosen in the first step by any agent has a chance to be never chosen, e.g., when a random center is chosen far from the other centers and observations. In this case a possible solution is to replace the center with a new random center. A more sophisticated approach, namely $k$-means++ algorithm, selects one initial centroid at random among the observations, and the remaining ones are chosen in order to try to maximize the inter-centroid distance. As the density of observations grows, however, the risk of this singular case becomes less and less likely.

Remark 2: The observations can be scaled in order to weigh the different components and obtain a clustering where some components have more importance than others. This can be done by using the following norm:

$$\|x\|_A = \langle Ax, Ax \rangle$$

where $A$ is a $d \times d$ diagonal matrix whose entries $a_{ii}$ represent the weights of the different components. As a result the following function is minimized instead of the one in Eq. 1:

$$D(T) = \sum_{i=1}^{n} \sum_{j=1}^{k} r_{ij}(T) \|x_i - c_j(T)\|^2_A.$$  \hspace{1cm} (5)

For what concerns the termination of the algorithm, to the best of our knowledge there is no available result about a bound for the number of iterations $M$. Notice that the k-means algorithm can show slow convergence on particular instances. Such particular cases do not seem to arise in practice, as witnessed by the fact that the k-means algorithm has smoothed polynomial complexity.

A possible practical solution to terminate the algorithm before the maximum number of iterations $M$ (which can be quite high) is reached, is when one of the following conditions is verified:

1) $M$ iterations are completed
2) there is no change in the assignment variables $r_{ij}(T)$ with respect to variables $r_{ij}(T-1)$;
3) $D(T) - D(T-1) < \Delta_{\text{max}}$, for some $\Delta_{\text{max}} > 0$.

3 RELATED WORK AND CONTRIBUTION

3.1 Related Work

In the literature, parallel and distributed k-means implementations have been motivated by the need to categorize huge data sets of high-dimensional observations. In the context of wireless sensor networks there are several attempts to implement distributed clustering schemes, especially with respect to probabilistic approaches: in [23] an incremental scheme is given; in [24] the distributed approach is based on a gossip algorithm; in [25] distributed consensus algorithms are used to reach consensus on a Mixture of Gaussian distribution; a methodology that requires the graph to be a tree is provided in [26], and a scheme based on the method of multipliers is given in [27]; in [28] a decentralized clustering algorithm is provided for a hierarchical sensor network, composed of a static backbone of cluster heads and a set of low-end sensors; in [29] a distributed clustering algorithm for a sensor network is given assuming agents with different power levels. In [22] distributed clustering schemes are provided for a sensor network considering a consensus clustering approach.

The above works, unfortunately, are limited to a parameter estimation or to specific network topologies and configurations, or require each agent to handle an amount of information which is comparable to the centralized approach.

3.2 Contribution

In this paper, we assume that each agent has access to some (possibly high-dimensional) measurements (e.g., position, temperature, humidity, etc.).

We provide a distributed algorithm to cluster such an information when the agents exchange information exclusively with their one-hop neighbors. Moreover, in spite of previous works, here we assume that each agent has very limited information about of the structure and the topology of the network.
Specifically, each agent
1) has a unique identifier;
2) can exchange directly information only with his one-hop neighbors;
3) can act synchronously.
4) knows an upper bound \( \bar{n} \) for the number \( n \) of agents in the network.

Moreover, the network is described by a graph \( G \). Consequently, any two agents can use a multi-hop strategy to exchange information, as will be discussed later in the paper.

According to the circumstances, there may be the need to group the agents into clusters that are homogeneous according to the agents’ positions or to the sensor measurements. If this task has to be performed by agents without any centralized supervision, we need to implement a distributed \( k \)-means clustering algorithm.

Each agent, therefore, has to solve an assignment problem with respect to a set of centroids (i.e., finding the nearest one) as well as to contribute to the refinement of the centroids, interacting with the other agents.

The procedure is executed by means of one-hop communication and the result is proven to be the same as the centralized \( k \)-means algorithm in terms of minimization of the objective function. This is done by resorting to a combination of distributed consensus algorithms such as average-consensus [1], [2] and max-consensus [3].

![Flowchart](image)

**Fig. 2.** Conceptual flow-chart of one step of the proposed distributed \( k \)-means algorithm.

Figure 2 shows a conceptual scheme of one step of the proposed algorithm. Specifically, the proposed algorithm takes advantage of a max-consensus algorithm for spreading the information on current centroids across the network (phase 1 in Figure 2). Such an information is then used locally by each agent to select the nearest centroid, thus segmenting the network into communities (i.e., clusters) (phase 2). Notice that there is, in general, no guarantee that the communities obtained during phase 2 will be connected, and therefore, in order to update the centroids, there is the need to gather meta-information on the sub-clusters that compose each cluster, i.e., their size and their centroid (phase 3); this is done by means of a combination of max-consensus and average-consensus algorithms. Once such an information has been gathered, the agents are able to update the centroids in a local manner (phase 4).

A typical application scenario of the proposed methodology is a wireless setting where each agent is equipped with a low range communication device, and is able to share information only with those agents that fall in its communication range; however the proposed approach is more general and applies to any connected undirected graph; moreover, it applies also to strongly connected and balanced graphs, and is easily extended to those directed graphs which can be balanced using weight balancing techniques like those presented in [30], [31], [32]. For the sake of simplicity, however, we limit the scope of the paper to undirected graphs.

### 4 Distributed Consensus Algorithms

In order to provide a distributed implementation of the \( k \)-means algorithm, we need a mechanism to diffuse information among the agents and to refine the centroid of a subset of agents in the network.

The max-consensus [3] and average-consensus [1], [2] algorithms proved their effectiveness in composing local observations by means of one-hop communication. Let a set of \( n \) agents, each associated to a vertex in the graph \( G \) and a discrete-time dynamic equation in the form:

\[
\chi(t+1) = u_i(N_i \cup \{i\}, t)
\]

where \( u_i(N_i \cup \{i\}, t) \) is a function of the state of agent \( i \) and of the agents \( j \) that belong to the neighborhood of agent \( i \) and \( \chi(t) \in \mathbb{R}^d \) for all \( i \).

Let \( \chi(z_{i0}, \ldots, z_{in}) \in \mathbb{R}^d \) be any function of the initial conditions of all the agents; the \( \chi \)-consensus problem consists in finding an input \( u_i(N_i \cup \{i\}, t) \) such that \( \lim_{t \to \infty} \chi(t) = \chi(z_{i0}, \ldots, z_{in}) \) for all \( i = 1, \ldots, n \).

Among several different problems that can be solved via consensus algorithms, we next review the max-consensus and average-consensus problems.

#### 4.1 Max-consensus

In the max-consensus problem, the agents have to converge to the maximum of the initial conditions, i.e., \( \chi(\cdot) \) is the component-wise maximum of its arguments. The problem, for connected undirected graphs or strongly connected directed graphs, is known to have a solution in finite time [3] if the following control law is chosen:

\[
u_i(N_i \cup \{i\}, t) = \max_{j \in N_i \cup \{i\}} z_j(t)\]
of the graph (i.e., the maximum among the minimum paths for every possible pair of agents, respecting the orientations of the links if the graph is directed). Again, since the agents do not know \( m \) nor \( n \), for practical applications a maximum number of iterations \( t_{\text{max}} > n \) has to be selected.

In the following we will denote by

\[
\tau_i = \text{max-consensus}(z_i(0), z_j(0) | j \neq i, G, t_{\text{max}})
\]

the execution of \( t_{\text{max}} \) iterations of the max-consensus procedure by the \( i \)-th agent in a network \( G \), starting from its own initial condition \( z_i(0) \) and the “unknown” initial conditions of the other agents, while \( \tau_i \) is the state of the \( i \)-th agent at iteration \( t_{\text{max}} \). Such a formalism just represents the execution of the max-consensus by the \( i \)-th agent, and we assume that all other agents are executing the same algorithm in a synchronous manner, each with its own initial condition.

### 4.2 Average-consensus

In the average-consensus problem the agents are required to converge to the component-wise average of their initial conditions.

Let each agent choose

\[
u_i(N_i \cup \{i\}, t) = w_i z_i(t) + \sum_{j=1}^{n} w_{ij} z_j(k)
\]

(8)

where \( w_{ij} = 0 \) if \((v_i , v_j) \notin E\). The update strategy for the entire system can be represented as

\[
z(t + 1) = (W \otimes I_d)z(t)
\]

where matrix \( W \) contains the terms \( w_{ij} \), \( I_d \) is the \( d \times d \) identity matrix and \( z(t) \) is the stack vector containing the terms \( z_i(t) \).

According to [33], with this choice of \( u_i(N_i \cup \{i\}, t) \) the state of each agent asymptotically converges to the component-wise average of the initial states if and only if:

1) \( W \) has a simple eigenvalue at 1 and all other eigenvalues have magnitude strictly less than 1;

2) the left and right eigenvectors of \( W \) corresponding to eigenvalue 1 are \( 1_n \) and \( \frac{1}{n} 1_n \), respectively, where \( 1_n \) is a vector composed of \( n \) elements, all equal to one.

A possible choice, assuming that the underlying graph is undirected and connected and that each agent knows \( n \) (or an upper bound for \( n \)), is that each agent \( i \) chooses independently the terms \( w_{ij} \) as

\[
w_{ij} = \begin{cases} 
\frac{1}{n}, & \text{if } v_j \in N_i \\
0, & \text{if } v_j \notin N_i \\
1 - \sum_{l=1}^{l} w_{il}, & \text{if } i = j
\end{cases}
\]

resulting in a matrix \( W \) that satisfies the conditions in [33]. Several other choices that yield asymptotic consensus are possible (e.g., see [34]).

**Remark 3:** Combining the max-consensus and the average-consensus algorithm, it is possible to calculate the number of agents \( n \) in the network in a distributed way [35]. Specifically, suppose a leader is elected via max-consensus (e.g., each agent has an identifier and the agent with the maximum identifier is elected as leader via max-consensus). Now, let the agents execute a scalar (i.e., \( d = 1 \)) average-consensus algorithm with \( z_i(0) = 1 \) if agent \( v_i \) is the leader and \( z_i(0) = 0 \) otherwise; then, average-consensus yields for all \( i = 1, \ldots, n \)

\[
\lim_{t \to \infty} z_i(t) = \frac{1}{n}.
\]

Regarding the computational complexity of both max and average-consensus algorithms, for each agent \( i \) we have that \#(\( N_i \)) \( \leq n \), hence for vectorial initial conditions \( z_i(0) \in \mathbb{R}^d \), the complexity is \( O(d t_{\text{max}}) \) for each agent, where \( t_{\text{max}} \) is the total number of iterations done.

Knowing an upper bound \( \tilde{n} \) for \( n \), it is possible to choose \( t_{\text{max}} = \tilde{n} \) in the case of max-consensus; in the average-consensus algorithm, the solution is reached asymptotically, hence to obtain a good approximation of the solution the agents stop after a large number of iterations or when particular conditions are met [36], [37].

The next subsection is devoted to reviewing a finite time average-consensus algorithm that overcomes such an issue.

### 4.3 Finite Time Average-consensus

In the literature, several extensions of the average-consensus algorithm have been provided to grant convergence in finite time [38], [39], [40], [41], [42], as well as some distributed stopping approaches that allow the agents to detect when they have converged within some \( \epsilon \) of the asymptotic consensus value [36], [37]. In this paper we will resort to the schema proposed in [39], because, knowing an upper bound \( \tilde{n} \) of \( n \), the number of steps required to reach consensus phase is known.

Suppose the agents perform an average-consensus algorithm using the update rule \( u_i(N_i \cup \{i\}, t) \) of Eq. (8); in the following we will describe the algorithm with respect to a scalar setting (i.e., \( d = 1 \)), as the extension to vectorial states is trivial. The approach in [39] allows each agent \( i \) to calculate the consensus value as a linear combination of \( z_i(0), \ldots, z_i(\delta) \) for a finite \( \delta > 0 \) which may not be the same for each agent. In other terms, each agent \( i \) seeks a coefficient vector

\[
\Gamma_i = [\gamma_i, \delta, \gamma_i, \delta_{\gamma}, \ldots, \gamma_i, 0]
\]

such that

\[
[z_i(\delta), z_i(\delta - 1), \ldots, z_i(0)] \Gamma_i = \frac{1}{n} 1_n z(0).
\]

Let \( g_i(g) \) be the \( i \)-th minimal polynomial, i.e., the monic polynomial of smallest degree such that \( e_i g_i(W) = 0 \), where \( e_i \) is the \( i \)-th vector of the canonical basis. If the
conditions 1) and 2) from [33] are verified, then \( q_i(g) \) has a root in \( g = 1 \) with multiplicity 1 [39].

The minimal polynomial \( q_i(g) \) can be expressed as

\[
q_i(g) = g^{\delta_i + 1} + \alpha_{i,\delta_i} g^{\delta_i} + \alpha_{i,\delta_i - 1} g^{\delta_i - 1} + \ldots + \alpha_{i,0},
\]

Let the polynomial

\[
p_i(g) = \frac{q_i(g)}{g - 1} = g^{\delta_i} + \beta_{i,\delta_i} g^{\delta_i - 1} + \ldots + \beta_{i,0}.
\]

The terms \( \gamma_{i,j} \) for \( i = 1, \ldots, n, j = 0, \ldots, \delta_i \) are calculated as:

\[
\gamma_{i,j} = \frac{\beta_{i,j}}{n p_i(g)}
\]

where \( p_i(g) \) is the value of \( p_i(g) \) evaluated at \( g = 1 \) and we take \( \beta_{i,\delta_i} = 1 \).

As for the calculation of the coefficients \( \beta_{i,j} \) of \( q_i(g) \), notice that, since \( e_i q_i(W) = 0 \) and \( z(t) = W^t z(0) \) for all integers \( t \geq 0 \), it holds

\[
e_i^t \left( W^{\delta_i + 1} + \alpha_{i,\delta_i} W^{\delta_i} + \alpha_{i,\delta_i - 1} W^{\delta_i - 1} + \ldots + \alpha_{i,0} I_N \right) = 0
\]

or equivalently

\[
\left[ \alpha_{i,0} \ldots \alpha_{i,\delta_i - 1} \alpha_{i,\delta_i} 1 \right] \Theta_i = 0
\]

where

\[
\Theta_i = \frac{1}{n} \begin{bmatrix}
1_n^t \\
1_n W \\
\vdots \\
1_n^t W^{\delta_i + 1}
\end{bmatrix}
\]

The polynomial \( q_i(g) \) can be easily constructed in a centralized fashion noting that \( \Theta_i \) is the observability matrix of the pair \((W,e_i)\), hence \( q_i(g) \) can be found by forming the matrix \( \Theta_i \) and increasing \( \delta_i \) until \( \Theta_i \) loses rank. The coefficients of \( q_i(g) \) can be obtained from the left nullspace of \( \Theta_i \).

Assuming that the agents know an upper bound \( \tilde{n} \) of \( n \), a distributed procedure to let each agent construct \( q_i(\cdot) \) is as follows. Let \( z_{*,0}(0), \ldots, z_{*,\tilde{n}}(0) \) denote a set of \( \tilde{n} \) different initial condition vectors for all the agents. For each of such initial conditions, each agent executes the update rule of Eq. \( 8 \) for \( \tilde{n} + 1 \) iterations (i.e., they calculate \( z_{*,j}(t + 1) = W z_{*,j}(t) \) for \( j = 1, \ldots, \tilde{n} \) and \( 0 \leq t \leq \tilde{n} - 1 \)). Each agent \( i \) has then access to the \( \tilde{n} \times (\delta_i + 2) \) matrix

\[
Z_{i,\delta_i} = \begin{bmatrix}
z_{i,1}(\delta_i + 1) & z_{i,1}(\delta_i) & \cdots & z_{i,1}(0)
z_{i,2}(\delta_i + 1) & z_{i,2}(\delta_i) & \cdots & z_{i,2}(0)
\vdots & \vdots & \ddots & \vdots 
z_{i,\tilde{n}}(\delta_i + 1) & z_{i,\tilde{n}}(\delta_i) & \cdots & z_{i,\tilde{n}}(0)
\end{bmatrix}
\]

for any \( 0 \leq \delta_i \leq \tilde{n} - 1 \).

Suppose \( \delta_i \) is the smallest integer such that \( Z_{i,\delta_i} \) is not full column rank. Then there is a vector

\[
\alpha_i = [1 \alpha_{i,\delta_i} \alpha_{i,\delta_i - 1} \ldots \alpha_{i,0}]^T
\]

such that \( Z_{i,\delta_i} \alpha_i = 0 \), implying that

\[
z_{i,j}(\delta_i + 1) + \alpha_{i,\delta_i} z_{i,j}(\delta_i) + \ldots + \alpha_{i,0} z_{i,j}(0) = 0.
\]

Note that \( z_{i,j}(t) = e_i^t z_{*,j}(0) = e_i^t W^t z_{*,j}(0) \), hence it holds

\[
e_i^t \left( W^{\delta_i + 1} + \alpha_{i,\delta_i} W^{\delta_i} + \ldots + \alpha_{i,0} I \right) z_{*,j}(0) = 0
\]

for \( j = 1, 2, \ldots, \tilde{n} \). If \( z_{*,1}(0), \ldots, z_{*,\tilde{n}}(0) \) are linearly independent, the above expression is equivalent to that of Eq. \( 9 \). This means that the degree of \( q_i(g) \) can be determined from the smallest integer \( \delta_i \) for which matrix \( (10) \) loses rank, and the nullspace of the corresponding matrix provides the coefficients of \( q_i(g) \).

**Remark 4:** There are various ways to ensure the linear independence of the \( \tilde{n} \) different initial conditions. For example, if all initial conditions are taken to be independent, identically distributed random variables with a Gaussian distribution, then the vectors \( z_{*,1}(0), \ldots, z_{*,\tilde{n}}(0) \) will almost surely be linearly independent. Alternatively, if the agents have unique identifiers, one can artificially choose the initial conditions to impose linear independence. For instance, during the \( j \)-th run, the agent with the \( j \)-th greatest identifier can choose its initial condition to be \( 1 \), whereas all other agents set their initial conditions to be zero. Notice that several max-consensus algorithms are required to choose the agents with greatest identifiers.

Regarding the computational complexity, for each agent the update rule is executed exactly \((\tilde{n} + 1)^2\) times, hence for vectorial initial conditions \( z_i(0) \in \mathbb{R}^d \), the complexity is \( O(dn^2) \) for each agent. Such a complexity is comparable with that of max-consensus algorithm, and it is in many cases smaller than the complexity of standard average-consensus algorithm, where \( t_{\max} \) is typically big.

In the following, with a slight abuse of notation, we will denote by

\[
\tau_i = \text{FTA-consensus}_i(z_i(0), z_j(0)) \quad j \neq i, G, \tilde{n}
\]

the execution of the finite time average-consensus procedure by the \( i \)-th agent in a network \( G \), starting from the initial opinion \( z_i(0) \), while \( \tau_i \) is the state of the \( i \)-th agent at the end of the algorithm and \( \tilde{n} \) is the upper bound of \( n \) used within the algorithm. Again, we assume that all other agents are executing the same algorithm in a synchronous manner, each with its own initial condition.

We will also denote by

\[
n_i = \frac{1}{\text{FTA-consensus}_i(n_i^0, z_j(0)) \quad j \neq i, G, \tilde{n}}
\]

the result of the distributed calculation, performed by the \( i \)-th agent, of the number of agents \( n \) belonging to the network, using the approach in [33] but employing the finite time average-consensus algorithm in [39], where \( n_i^0 = 1 \) if \( i \) is the leader elected via max-consensus and
5 Distributed \( k \)-means Algorithm

Let a network of \( n \) agents be represented by a fixed and undirected graph \( G \) and suppose that each agent is able to communicate with all its one-hop neighbors in a synchronous way. Each agent is endowed with a real vector \( x_i \in \mathbb{R}^d \) representing a piece of information or a set of measures. Moreover, each agent has a unique identifier and knows an upper bound \( \tilde{n} \) of the number of agents \( n \) (we assume each agent has the same value for \( \tilde{n} \)).

The objective of the distributed \( k \)-means algorithm is to partition the agents in \( k \) clusters minimizing the functional \( D \) specified in eq. (1) (or eq. (5)) via a fully distributed approach. In the following we will neglect, without loss of generality, the procedure to mitigate the risk of obtaining a local minimum by iterating the algorithm several times and selecting the best result. A flow chart of the proposed \( k \)-means distributed algorithm is provided in Figure 3 while the pseudocode is given in Algorithm 1. We assume that Algorithm 1 is executed synchronously by each agent and that the agents exchange the necessary information with their neighbors or with (a subset of) their one-hop neighborhood.

5.1 Initialization

The initialization phase of the Algorithm 1 is as follows.

As first step, a leader is elected. Supposing that each agent has a unique identifier, the agents challenge on their identifiers via max-consensus. The agent \( i^* \) whose identifier is the greatest is elected as leader after no more than \( \tilde{n} \) steps.

Based on the information on leader and non-leader agents, the size \( n \) of the network is calculated resorting to the approach in [35], as discussed in Remark 3. Notice that this operation is done by means of the finite-time average-consensus algorithm introduced in [39] and discussed in Section 4.3 using the upper bound \( \tilde{n} \). As a result of this procedure, the value of \( n \) is obtained, hence for the rest of the algorithm the agents use the actual value of \( n \) instead of the upper bound \( \tilde{n} \).

To conclude the initialization phase, the leader chooses the initial centroids. Specifically, leader \( i^* \) selects \( c_{i^*,j}(0) \in \mathbb{R}^d \) at random (within the range of values of interest for each component) for all \( j = 1, \ldots, k \).

The non-leader agents, conversely, choose for all \( j = 1, \ldots, k \)

\[ c_{ij}(0) = -[\infty, \ldots, \infty]^t, \quad \forall i \neq i^*. \]

Remark 5: The initial centroid choice can be substituted by other approaches, such as choosing random observations or using the \( k \)-means++ algorithm, as discussed in Remark 1. The first approach is easy to implement in a distributed way by letting each agent select a random value and then elect \( k \) leaders by iterating max-consensus procedures. The second approach requires...
Algorithm 1: Distributed $k$-means algorithm (executed synchronously by all agents)

Data: Identifier $i$, upper bound $\hat{n}$, observation $x_i$, Neighborhood $\mathcal{N}_i$, Exit method $C_1$ or $C_2$, Data from neighbors

Result: Centroids $c_1, \ldots, c_k$, centroid choice $\overline{j}_i$, final neighborhood $\mathcal{N}_i^c$

/* Initialization */
$c_{ij}(0) = -[\infty, \ldots, \infty]$ for all $j = 1, \ldots, k$;
$\mathcal{N}_i^c(0) = \mathcal{N}_i$;
$\overline{j}_i(0) = 0$;
/* Leader Election */
i* = max-consensus$_i(i, j \neq i, G, \hat{n})$;
/* Network Size Calculation */
$n_i^0 = \begin{cases} 
1 & \text{if } i = i^* \\
0 & \text{else}
\end{cases}$;
$n_i = 1/$FTA-consensus$_i(n_i^0, n_j^0 | j \neq i, G, \hat{n})$;
/* Random Center Choice */
if $i = i^*$ then
agent $i$ is the leader;
c$_i(0) = \text{random}(d \times 1)$ for all $j = 1, \ldots, k$;
/* Main Cycle */
$T = 1$;
exit $=$false;
while $T \leq M$ and exit $=$false do
/* Centroid Propagation */
$c_i^0(T) = \begin{cases} 
c_i(0) & \text{if } T = 1 \\
\mu_i(T - 1) \odot c_{\overline{j}_i}(T - 1) + \hat{\mu}_i(T - 1) \odot 1_d & \text{if } T > 1
\end{cases}$;
$c_i(T) = \text{max-consensus}_i(c_i^0(T), c_j^0(T) | j \neq i, G, n_i)$;
/* Nearest Centroid Choice */
$\overline{j}_i = \arg \min_{j=1}^k c_{ij}(T) - x_i$;
$\mu_{ij}(T) = \begin{cases} 
1 & \text{if } j = \overline{j}_i \\
0 & \text{else}
\end{cases}$;
$\hat{\mu}_{ij}(T) = \begin{cases} 
0 & \text{if } j = \overline{j}_i \\
-\infty & \text{else}
\end{cases}$;
/* Nearest Centroid Choice Broadcast */
each agent provides $\overline{j}_i$ to the neighbors in $\mathcal{N}_i$;
/* Cluster Neighborhood Choice */
each agent selects $\mathcal{N}_i^c(T) \subseteq \mathcal{N}_i$ based on $\overline{j}_i$ for each $j \in \mathcal{N}_i$;
/* Sub-cluster Centroid Identification */
$\text{SCC}_{\overline{j}_i}(\nu)(T) =$ FTA-consensus$_i(x_i, x_j) \; j \neq i, G^c(T), n$
/* Sub-cluster Leader Election */
if $\overline{j}_i = i^*$ then
agent $i$ is a sub-cluster leader;
/* Sub-cluster Size Calculation */
$\text{SCS}_{\overline{j}_i}(\nu)(T) =$
FTA-consensus$_i(SCL_i(T), SCL_j(T) | j \neq i, G^c(T), n)$;
/* Sub-Cluster Meta Information Gathering */
$i_i(T) = \begin{cases} 
\text{SCC}_{\overline{j}_i}(\nu)(T) & \text{if } i = i^* \\
0_d & \text{else}
\end{cases}$;
$\epsilon_i(T) = \begin{cases} 
\text{SCS}_{\overline{j}_i}(\nu)(T) & \text{if } i = i^* \\
0 & \text{else}
\end{cases}$;
$\eta_i(T) = c_{\overline{j}_i}(T)$;
$\pi(T) =$ FTA-consensus$_i(\eta_i^0(T), \eta_j^0(T) | j \neq i, G, n)$;
/* Centroid Calculation */
$c_i(T) = \frac{\pi_j(T)}{c_j(T)}, \forall j = 1, \ldots, k$;
/* Exit Condition Evaluation */
if $C_1$ then
$\pi(T) =$ max-consensus$_i(\nu_i^0(T), \nu_j^0(T) | j \neq i, G, n)$;
if $\pi(T) = 0$ then
exit $=$true;
else
$\pi(T) =$ FTA-consensus$_i(\nu_i^0(T), \nu_j^0(T) | j \neq i, G, n)$;
if $\pi(T) - \nu(T - 1) < \Delta_{\text{max}}$ then
exit $=$true;
return $c_i(T), \ldots, c_k(T), \overline{j}_i(T)$ and $\mathcal{N}_i^c(T)$;
each agent to know the current centroids and to calculate the distance between the nearest centroid and its observation, which can be implemented in a distributed fashion in a way similar to the centroid propagation procedure described in Section 5.2.1. Then, to chose the next centroid, the leader selects a random number, which is propagated via max-consensus; eventually, the agents challenge on the absolute value of the difference between the random number and the distance, again via max-consensus. In conclusion, both of the above variations of the standard k-means algorithm can be implemented in a distributed fashion; in the following, for the sake of simplicity, we will consider the standard k-means algorithm.

5.2 Main Cycle

After the initialization phase, the main cycle is executed \( M \) times, or until an exit condition is verified, as better explained in Section 5.2.7.

At every step \( T \), the main cycle is composed of the following phases:

5.2.1 Centroid Propagation

At each step \( T \), the information on the current centroids is diffused to all the agents.

Specifically, each agent selects a vector \( c_i^0(T) \in \mathbb{R}^{kd} \) as follows:

\[
c_i^0(T) = \begin{cases} 
    [c_{i1}(0), \ldots, c_{ik}(0)]' & \text{if } T = 1 \\
    \mu_i(T-1) \otimes c_{ij}(T-1) + \mu_i(T-1) \otimes 1_d & \text{if } T > 1
\end{cases}
\]  

(11)

where \( c_{ij}(T-1) \in \mathbb{R}^d \) is the centroid chosen by agent \( i \) at step \( T-1 \) and \( \mu_i(T-1), \hat{\mu}_i(T-1) \in \mathbb{R}^k \) are vectors representing the choice of a centroid at step \( T-1 \). The expressions for \( \mu_i(T) \), \( \hat{\mu}_i(T) \) are given in Eq. (13) and Eq. (14), respectively.

Vectors \( c_i^0(T) \), calculated according to Eq. (11), are structured so that, if used as initial conditions for a vectorial max-consensus procedure, the result is the stack vector \( c(T) \) containing all the \( k \) centroids of step \( T \).

Lemma 1: Let us assume graph \( G \) is connected, and let \( c(T) \) be the stack vector of all the centroids at step \( T \) of the distributed k-means algorithm. For each time step \( T \geq 1 \) and for each agent \( i = 1, \ldots, n \) it holds

\[c(T) = \text{max-consensus}_i(c_i^0(T), c_{j}^0(T) \mid j \neq i, G, n) \]

where \( c_i^0(T) \) is calculated according to Eq. (11).

Proof: At step \( T = 1 \) the only vector \( c_i^0(T) \) with components greater than \( -\infty \) is \( c_i^0(T) \) and the result of \( n \) iterations of a max-consensus algorithm over a connected graph \( G \) is \( c(T) = c_i^0(T) \) for all agents. For \( T > 1 \) it can be noted from Eq. (11), Eq. (13) and Eq. (14) that each agent vector \( c_i^0(T) \) has finite-valued entries only in the correspondence of the chosen centroid, hence a max-consensus results in a stack vector containing all the centroids.

Remark 6: Similarly to the centralized k-means algorithm, there is the risk that some centroid is not chosen by the agents. In this case the proposed distributed k-means algorithm overwrites each of these centroids with a vector whose components are all equal to \( -\infty \).

This is due to the particular choice of \( c_i^0(T) \); in fact, except for the leader at the first time step, each agent chooses just a centroid and the vector \( c_i^0(T) \) is filled with \( -\infty \) for the unchosen centers. If a center is not selected by any agent, then the corresponding components of \( c_i^0(T) \) will be \( -\infty \) for each agent.

Such an issue is however easy to fix: if a center \( c_j \) is not chosen, then each agent will have a vector \( c_i^0(T) \) such that \( c_j = -\infty \). In this case the leader may select another random centroid and a propagation can be implemented via max consensus.

5.2.2 Nearest Centroid Choice and Broadcast

Each agent chooses the index \( \tilde{j}_i \) of the nearest among the current centroids, i.e.,

\[\tilde{j}_i = \arg \min_{j=1}^{k} |c_{ij}(T) - x_i| \]  

(12)

and updates \( \mu_i(T) \in \mathbb{R}^d \) and \( \hat{\mu}_i(T) \in \mathbb{R}^d \) as follows:

\[\mu_{ij}(T) = \begin{cases} 
    1 & \text{if } j = \tilde{j}_i \\
    0 & \text{else}
\end{cases} \]  

(13)

\[\hat{\mu}_{ij}(T) = \begin{cases} 
    0 & \text{if } j = \tilde{j}_i \\
    -\infty & \text{else}
\end{cases} \]  

(14)

Then, each agent provides its choice of \( \tilde{j}_i \) to the agents in its neighborhood \( N_i \).

5.2.3 Cluster Neighborhood Choice

Each agent selects, among its neighbors in \( N_i \), those agents that share the same choice of \( \tilde{j}_i \). As a result a new neighborhood \( N_i^c(T) \subseteq N_i \) is obtained. The new neighborhood \( N_i^c(T) \) is then used to update the centroids.

5.2.4 Sub-cluster Centroid Identification

The new neighborhood induces a subgraph \( G_i^c(T) \) of \( G_i \) where each agent is connected only to agents with the same centroid choice.

Notice that, as shown in the example of Figure 4, this does not imply that the agents with the same center choice are connected in \( G_i^c(T) \) as in general there is no strong relation between the observation associated to each agent and the topology.

Specifically, the purple crosses in Figure 4(a) represent the \( k = 4 \) initial random centroids selected by the leader. Figure 4(b) reports in blue the agents belonging to one of the clusters. The red x represents the correct centroid of the cluster composed of the agents in blue; however note that the blue agents are decomposed in 2 sub-clusters.

Using the finite time average-consensus algorithm over \( G_i^c(T) \), therefore, each agent \( i \) obtains the centroid
Let us now discuss how to cope with such an issue.

5.2.5 Sub-cluster Leader Election and Meta Information Gathering

In order to handle the situation depicted in Figure 4, the proposed algorithm features a mechanism to calculate the cluster centroids by electing a sub-cluster leader for each sub-cluster and then by calculating the size of each sub-cluster.

The sub-cluster leaders $i^*_j(T)$ are elected for each cluster $j$ and each sub-cluster $h$ by means of a max-consensus algorithm over the graph $G^c(T)$.

The sub-cluster centroids $SCC^c(T)$ are obtained by means of a finite-time average-consensus algorithm over $G^c(T)$, where each agent selects its observation $x_i$ as initial condition.

The sub-cluster size $SCS^c_j(T)$ of each sub-cluster $h$ of a cluster $j$ is eventually calculated by each agent in the sub-cluster by resorting to the approach in [35] over the graph $G^c(T)$, as discussed in Remark 3 the sub-cluster leader $i^*_j(T)$ of each sub-cluster selects an initial condition equal to one, while other agents in the sub-cluster select zero.

Lemma 2: Let us assume graph $G$ is connected, and let $G^c(T)$ be the graph representing the segmentation in communities at step $T$ of the distributed $k$-means algorithm. For each time step $T \geq 1$ and for each agent $i = 1, \ldots, n$ it holds

$$SCC^c_{j,h}(T) = \text{FTA-consensus}_i(x_i, x_j | j \neq i, G^c(T), n),$$

and

$$i^*_j(T) = \max\text{-consensus}_i(i, j | j \neq i, G^c(T), n)$$

$SCS^c_j(T) = \frac{1}{\text{FTA-consensus}_i(SCL_i(T), SCL_j(T)) \neq i, G^c(T), n)}$, where $SCC^c_{j,h}(T)$, $i^*_j(T)$, and $SCS^c_j(T)$ are the centroid, the index of the leader and the size of the sub-cluster $h$ of the cluster $j_i$, agent $i$ belongs to, respectively, and

$$SCL_i(T) = \begin{cases} 
1 & \text{if } i = i^*_j, \\
0 & \text{else}.
\end{cases}$$

Proof: Graph $G^c(T)$ contains several disconnected sub-clusters, and since $n$ is an upper bound of the size of each sub-cluster the finite time average-consensus algorithm [39] provides each agent in a sub-cluster with the component-wise average of the observations belonging to the sub-cluster, i.e., with the sub-cluster centroid $SCC^c_{j,h}(T)$. Similarly using the max-consensus algorithm over $G^c(T)$ for $n$ steps, where the initial state of each agent is its identifier, provides each agent with the identifier $i^*_j$ of the leader of the sub-cluster it belongs to. The finite time average consensus over $G^c(T)$ with initial state equal to $SCL_i(T)$ for all agents, eventually, provides each agent with $1/SCS^c_j(T)$.

The centroids are calculated as follows.

5.2.6 Centroid Calculation

The agents execute a finite-time average consensus algorithm over $G$ choosing for each agent $i$ an initial condition

$$\eta^0_i(T) = c_j^i \otimes \sigma_i(T): c_j^i \in \mathbb{R}^{k(d+1)} ,$$

where $c_j^i$ is the $j_i$-th vector in the canonical basis and $\sigma_i(T) \in \mathbb{R}^d$ is a vector containing the sub-cluster centroid $SCC_i(T)$ weighted by the sub-cluster size $SCS_i(T)$ if agent $i$ is a sub-cluster leader and is zero otherwise, i.e.,

$$\sigma_i(T) = \begin{cases} 
SCC_i(T) \cdot SCS_i(T), & \text{if } i \text{ is a sub-cluster leader} \\
0_d, & \text{else}.
\end{cases}$$

The scalar $\epsilon_i(T)$ is equal to the sub-cluster size $SCS_i(T)$ if agent $i$ is a sub-cluster leader and is zero otherwise, i.e.,

$$\epsilon_i(T) = \begin{cases} 
SCS_i(T), & \text{if } i \text{ is a sub-cluster leader} \\
0, & \text{else}.
\end{cases}$$

Lemma 3: Let us assume graph $G$ is connected. For each time step $T \geq 1$ of the distributed $k$-means algorithm and for each agent $i = 1, \ldots, n$ it holds

$$\eta_i(T) = \text{FTA-consensus}_i(\eta_i^0(T), \eta_j^0(T)) \neq i, G, n)$$
where
\[
\eta(T) = \begin{bmatrix} \eta_1(T) \\ \vdots \\ \eta_k(T) \end{bmatrix}
\]
and each term \( \eta_j(T) = [\sigma_j(T) \eta_j(T)]' \) is such that
\[
c_j(T) = \frac{\sigma_j(T)}{\eta_j(T)}
\]
(15)

where \( c_j(T) \) is the \( j \)-th centroid at step \( T \).

Proof: Under the hypothesis that the graph is undirected and connected, the result \( \eta(T) \) of the finite time average-consensus for each agent is such that, for all \( j = 1, \ldots, k \)
\[
\eta_j = \begin{bmatrix} \sigma_j(T) \\ \tau_j(T) \end{bmatrix} = \frac{1}{n} \left[ \sum_{h=1}^{h_{\max}(T)} SCS_{jh}(T) \cdot SCC_{jh}(T) \right]
\]
where \( h_{\max}(T) \) is the number of sub-clusters the \( j \)-th cluster is composed of. Hence Eq. (15) holds true and the proof is complete. \( \Box \)

5.2.7 Exit Condition Evaluation

As discussed at the end of Section 2 there is the need to define an exit criterion, because the maximum number of iterations \( M \) is typically a big quantity and it is preferable to identify an alternative way to terminate the algorithm, in order to save resources.

The stopping criterion 2) discussed at the end of Section 2 (stop if there is no variation in the association variables \( r_{ij}(T) \) with respect to step \( T - 1 \) can be easily implemented in a distributed fashion. Specifically, each agent \( i \) executes a max-consensus algorithm with the following initial condition:
\[
\nu_i(0) = \begin{cases} 0 & \text{if } \overline{r}_i(T) = \overline{r}_i(T-1) \\ 1 & \text{else} \end{cases}
\]

If the result of the max-consensus is \( \nu_i = 0 \), each agent concludes that there has been no variation in the assignment of the observations to the centroids from step \( T - 1 \) to step \( T \), hence the algorithm can be terminated.

The stopping criterion 3) is
\[
|D(T) - D(T-1)| < \Delta_{\max} \Rightarrow \text{stop}
\]

and can be implemented in a distributed way by means of a finite-time average-consensus where the initial condition of each agent is
\[
\nu_i(0) = n |c_{\mathcal{G}_i}(T) - x_i|^2.
\]

The result of such a procedure is \( \nu_i = D(T) \), hence keeping track of \( D(T-1) \) each agent is able to verify if the criterion is satisfied.

Notice that, since the exit conditions may not be met, each agent needs a counter to keep trace of the number of iterations in order to eventually stop when \( M \) iterations are executed, where \( M \) is the same for all the agents.

6 Correctness and Time/Memory Complexity

In this Section we discuss the formal correctness of the distributed \( k \)-means, i.e., the fact that it responds to the same specifications as the centralized \( k \)-means algorithm and that the two algorithms provide the same results if the initial centroids are the same. We also inspect the time and memory complexity of the distributed \( k \)-means, and provide a comparison with the centralized one.

6.1 Correctness

Theorem 1: Let us assume that \( G \) is connected and undirected, and that each agent knows an upper bound \( \hat{n} \) for \( n \). Suppose further that the distributed \( k \)-means algorithm selects the initial centroids \( c_{i\hat{r}_j}(0) \) for \( j = 1, \ldots, k \). The distributed \( k \)-means is correct, i.e., it provides the same results as the standard \( k \)-means algorithm when the same initial centroids \( c_{i\hat{r}_j}(0) \) are chosen.

Proof: If \( G \) is connected and each agent knows an upper bound \( \hat{n} \) for \( n \), then any finite time average-consensus or max-consensus procedure (possibly with vectorial state for each agent) over \( G \) will succeed.

Let us consider a generic step \( T \) of the main cycle. By Lemma 1, the Centroid Propagation phase provides each agent with a vector containing the current \( k \) centroids (for the first step the random centroids selected by the leader are obtained).

As a result of the Nearest Centroid Choice, Nearest Centroid Choice Broadcast and Cluster Neighborhood Choice phases, each agent selects a neighborhood \( \mathcal{N}_i(T) \subseteq \mathcal{N}_i \) which induces a graph \( G^c(T) \) which is possibly decomposed in sub-clusters. This is equivalent to choosing a value for the decision variables \( r_{ij}(T) \): since each agent \( i \) selects a cluster index \( \overline{r}_i \), in fact, it is easy to note that the constraints of the \( k \)-means algorithm in Eq. (1) are satisfied.

By Lemma 2 and Lemma 3 the centroids are correctly as the component-wise weighted average of the observations associated to the agents in each cluster, exactly as done by the centralized version of the \( k \)-means algorithm, hence the proof is complete. \( \Box \)

6.2 Time and Memory Complexity

The following result characterizes the time complexity of the distributed \( k \)-means algorithm.

Proposition 1: The time complexity of the distributed \( k \)-means algorithm is \( O(dkn^2M) \).

Proof: In the worst case the main cycle of the distributed \( k \)-means algorithm is executed \( M \) times, and for each iteration a combination of max-consensus and finite time average-consensus algorithms with vectorial initial conditions is done, hence the complexity of the algorithm is \( O(\phi n^2M) \), where \( \phi \) is the sum of the dimension of the initial conditions of the different consensus algorithms executed during the main cycle.
Specifically, the size of the initial conditions of the consensus algorithms used in each phase is as follows:

- \(dk\) for the centroid propagation phase;
- \(d\) for the centroid subcluster identification phase;
- 1 for the subcluster size calculation phase;
- \((d+1)k\) for the subcluster meta information gathering phase;
- 1 for the exit condition evaluation phase.

Hence it holds \(\phi = (2d + 1)k + d + 2 = O(dk)\) and the proof is complete.

Notice that the time complexity of the distributed \(k\)-means algorithm is \(O(n)\) times the time complexity of the standard \(k\)-means algorithm. This is expected since the distributedness of the proposed setup implies that the transmission of a message from an agent to another may require a number of steps that is equal to the diameter of the graph, hence proportional to the number of agents.

However, the decentralization introduces benefits in terms of robustness (i.e., a central authority is prone to failures), energy saving (i.e., all communications are local) and memory, as emphasized by the following proposition.

**Proposition 2:** The memory occupancy of the distributed \(k\)-means algorithm is \(O(kd)\), which is \(O(k/(n+k))\) times the memory occupancy of the centralized \(k\)-means algorithm.

**Proof:** In the distributed setup, a agent needs to store just the value of its piece of information \(x_i \in \mathbb{R}^d\) and \(k\) current centroids, each in \(\mathbb{R}^d\), therefore the memory complexity in this case is \(O(kd)\). In the centralized setting, conversely, the processing unit needs to store also the value of each \(x_j\), hence the memory complexity is \(O((k+n)d)\). The ratio of the above memory complexities proves the statement.

**Remark 7:** The distributed \(k\)-means algorithm has remarkably smaller memory complexity with respect to the centralized setting. In fact, since \(n \geq k\) it holds

\[
0 < \frac{k}{n+k} \leq \frac{1}{2}
\]

where values near zero are attained for \(n \gg k\), while in the worst case there is a reduction of 50\% of the memory complexity when moving from a centralized to a distributed setup.

Notice that the above ratio does not depend on \(d\), hence the reduction in memory complexity holds both in high-dimensional and low-dimensional contexts.

### 7 Simulation Results

In this section we provide some examples of application of the proposed distributed \(k\)-means algorithm.

We consider a situation where \(n = 50\) agents are embedded in the unit square in \(\mathbb{R}^2\), and the agents are connected by means of a unit disk graph topology, that is, each pair of agents with distance less than a threshold \(\rho\) is connected; we choose \(\rho = \sqrt{\frac{3}{5}}\) (i.e., 20\% of the maximum possible distance). We want to partition the agents into \(k = 4\) clusters. We choose the second stop criterion discussed at the end of Section 2.

Figure 5 shows the case where the observations coincide with the agent’s positions. Specifically the upper plots in the figure show the clustering obtained with respect to the spatial distribution of the agents (the different colors and shapes represent different clusters, while the centers are reported with purple empty stars); in this case it is evident that the agents are clustered according to their position. The lower left plot shows the evolution of the functional \(D\) during the execution of the algorithm. The result of a centralized \(k\)-means is reported with red asterisks; it is evident that at each step the distributed formulation provides the same results as the centralized one. The lower central plot shows the iterations of consensus algorithms required to implement the distributed \(k\)-means algorithm, and for each iteration the corresponding logical phase of the \(k\)-means algorithm is reported: initialization (I), centroid choice (C), centroid refinement (R) and exit condition evaluation (E). According to the discussion provided in Section 6.2 the number of iterations is \(O(dkn^2M) \approx 10^5\) which is \(n = 50\) times more the time complexity of the centralized \(k\)-means algorithm. As for the memory complexity, instead, each agent needs to store order of \(dk = 8\) values, while the centralized algorithm requires order of \((k+n)d = 108\), hence around 13.5 times more information. It can be noted that each step of the distributed \(k\)-means algorithm requires exactly the same number of consensus iterations, because finite time consensus algorithms are used.

Notice further that the centroid refinement phase is the most time consuming, because of the presence of sub-clusters and because of the combination of max-consensus and finite time average-consensus algorithms. This is also highlighted by the lower rightmost picture, which shows the percentage of consensus iterations used for each of the above four phases; as shown by the picture approximatively 85\% of consensus iterations are required to implement the centroid refinement phase.

Figure 6 shows an example where the agents have scalar observations in \([0,1]\), which depend on the position of the agents according to the field reported in the upper left figure, together with the graph topology. The upper right figure displays the final clustering obtained at step \(T = 4\) (again, the different colors and shapes represent different clusters). In this case the clustering pattern depends on the field, and it can be observed that very far agents belong to the same cluster.

The lower left plot shows the clustering with respect to the distribution of the observations. Specifically, the abscissa represents the value of the observations and the ordinate represents the step of the distributed \(k\)-means algorithm. Different colors and shapes represent different clusters, while the centroids are reported with purple empty stars. The plot implies that the algorithm
is indeed effective in partitioning the agents according to the observations.

The right lower leftmost plot shows the results in terms of minimization of the objective function $D$ of Eq. (1) obtained as a result of the distributed (blue dotted line) and centralized (red asterisks) $k$-means algorithms. According to Theorem 1, the distributed $k$-means algorithm correctly provides the same results of the centralized $k$-means algorithm in terms of minimization of $D$.

8 CONCLUSIONS AND FUTURE WORK

In this paper we provide a fully decentralized and distributed implementation of the $k$-means clustering algorithm considering a set of agents each of which is equipped with a possibly high-dimensional piece of information or set of measurements.

The proposed algorithm, is proven to be a correct implementation of the $k$-means algorithm in that it provides the same results in terms of centroids, associations and objective function. We also inspect the time and memory complexity of the proposed distributed implementation showing that the time complexity is $O(n)$ times more the one of the centralized version, where $n$ is the number of agents; the memory complexity, conversely, is considerably reduced from $O((k + n)d)$ to $O(kd)$, where $k$ is the number of clusters and $d$ is the dimension of the observations, hence the proposed algorithm can be implemented using computational units with less resources with respect to the centralized version.

The proposed algorithm finds a direct application in distributed data filtering/classification problems and in mobile robot coordination problems.

Further work will address the analysis of the robustness of the algorithm for noisy measurements and faults at some agents. We will also inspect dynamically changing networks.

Another interesting expected future work direction is the implementation of a distributed procedure to obtain a soft clustering, i.e., a partition where each agent belongs to different clusters with different degrees of membership. This would contribute to capture more sophisticated aspects of the relations that exist among the agents: each agent would then belong to each cluster with a different intensity.

As a result a agent, while belonging to a primary cluster, may eventually contribute to the operations of one or more other clusters, e.g., in the case of an emergency or under specific circumstances. The above extension, however, appears to be a non-trivial challenge, since the proposed algorithm largely relies on the intrinsic binary association of observations to clusters and during the evolution of the algorithm the topology is segmented in disconnected components, while a soft clustering would
require, rather than segmentation, complex weight adjusting schemes.

A final foreseen research direction is to devise more efficient finite time average-consensus algorithms, in order to lower the complexity of the distributed $k$-means algorithm.

Fig. 6. Example of execution of the distributed $k$-means algorithm with scalar states. The state of each agent is given by the field reported in the upper left figure.

REFERENCES

[1] R. Olfati-Saber and R. M. Murray, “Consensus problems in networks of agents with switching topology and time-delays,” Automatic Control, IEEE Transactions on, vol. 49, no. 9, pp. 1520–1533, 2004.
[2] R. Olfati-Saber, J. A. Fax, and R. M. Murray, “Consensus and cooperation in networked multi-agent systems,” Proceedings of the IEEE, vol. 95, no. 1, pp. 215–233, 2007.
[3] J. Cortés, “Distributed algorithms for reaching consensus on general functions,” Automatica, vol. 44, no. 3, pp. 726–737, 2008.
[4] J. C. Dunn, “A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters,” 1973.
[5] A. P. Dempster, N. M. Laird, D. B. Rubin et al., “Maximum likelihood from incomplete data via the EM algorithm,” Journal of the Royal statistical Society, vol. 39, no. 1, pp. 1–38, 1977.
[6] D. Aloise, A. Deshpande, P. Hansen, and P. Popat, “NP-hardness of Euclidean sum-of-squares clustering,” Machine Learning, vol. 75, no. 2, pp. 245–248, 2009.
[7] M. Mahajan, P. Nimbhorkar, and K. Varadarajan, “The planar $k$-means problem is NP-hard,” in WALCOM: Algorithms and Computation. Springer, 2009, pp. 274–285.
[8] J. MacQueen et al., “Some methods for classification and analysis of multivariate observations,” in Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, no. 281–297, 1967, p. 14.
[9] P. Brucker, “On the complexity of clustering problems,” in Optimization and Operations Research. Springer, 1978, pp. 45–54.
[10] A. K. Jain, “Data clustering: 50 years beyond $k$-means,” Pattern Recognition Letters, vol. 31, no. 8, pp. 651–666, 2010.
[11] D. Arthur and S. Vassilvitskii, “$k$-means++: The advantages of careful seeding,” in Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2007, pp. 1027–1035.
[12] A. Vattani, “$k$-means requires exponentially many iterations even in the plane,” Discrete & Computational Geometry, vol. 45, no. 4, pp. 596–616, 2011.
[13] D. Arthur, B. Manthey, and H. Roglin, “$k$-means has polynomial smoothed complexity,” in Foundations of Computer Science, 2009. FOCS’09. 50th Annual IEEE Symposium on, 2009, pp. 405–414.
[14] M. K. Ng, “$k$-means-type algorithms on distributed memory computer,” International Journal of High Speed Computing, vol. 11, no. 2, pp. 75–91, 2000.
[15] I. S. Dhillon and D. S. Modha, “A data-clustering algorithm on distributed memory multiprocessors,” in Large-Scale Parallel Data Mining. Springer, 2000, pp. 245–260.
[16] Y. Zhang, Z. Xiong, J. Mao, and L. Ou, “The study of parallel $k$-means algorithm,” in Sixth World Congress on Intelligent Control and Automation, vol. 2, 2006, pp. 5868–5871.
[17] L. Vendramin, R. J. G. B. Campello, L. F. Coletta, and E. R. Hruschka, “Distributed fuzzy clustering with automatic detection of the number of clusters,” in International Symposium on Distributed Computing and Artificial Intelligence. Springer, 2011, pp. 133–140.
[18] S. Patel, V. Patel, and D. Jhinwala, “Privacy preserving distributed $k$-means clustering in malicious model using zero knowledge proof,” in Distributed Computing and Internet Technology. Springer, 2013, pp. 420–431.
[19] S. Monti, P. Tamayo, J. Mesirov, and T. Golub, “Consensus clustering: a resampling-based method for class discovery and visualization of gene expression microarray data,” Machine learning, vol. 52, no. 1-2, pp. 91–118, 2003.

[20] A. P. Topchy, M. H. Law, A. K. Jain, and A. L. Fred, “Analysis of consensus partition in cluster ensemble,” in Fourth IEEE International Conference on Data Mining, 2004. ICDM’04. IEEE, 2004, pp. 225–232.

[21] A. Goder and V. Filkov, “Consensus clustering algorithms: Comparison and refinement.” in 2008 Proceedings of the Tenth Workshop on Algorithm Engineering and Experiments (ALENEX), vol. 8. SIAM, 2008, pp. 109–117.

[22] P. A. Forero, A. Cano, and G. B. Giannakis, “Consensus-based k-means algorithm for distributed learning using wireless sensor networks,” in Proc. Workshop on Sensors, Signal and Info. Process., Sedona, AZ, 2008, pp. 11–14.

[23] R. D. Nowak, “Distributed EM algorithms for density estimation and clustering in sensor networks,” Signal Processing, IEEE Transactions on, vol. 51, no. 8, pp. 2245–2253, 2003.

[24] W. Kowalczyk and N. Vlassis, “Newscast EM,” Advances in Neural Information Processing Systems 17, pp. 713–720, 2005.

[25] D. Gu, “Distributed EM algorithm for Gaussian mixtures in sensor networks,” Neural Networks, IEEE Transactions on, vol. 19, no. 7, pp. 1154–1166, 2008.

[26] J. Wolfe, A. Haghighi, and D. Klein, “Fully distributed em for very large datasets,” in Proceedings of the 25th international conference on Machine learning. ACM, 2008, pp. 1184–1191.

[27] P. A. Forero, A. Cano, and G. B. Giannakis, “Consensus-based distributed expectation-maximization algorithm for density estimation and classification using wireless sensor networks,” in Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on. IEEE, 2008, pp. 1989–1992.

[28] W.-P. Chen, J. C. Hou, and L. Sha, “Dynamic clustering for acoustic target tracking in wireless sensor networks,” Mobile Computing, IEEE Transactions on, vol. 3, no. 3, pp. 258–271, 2004.

[29] O. Younis and S. Fahmy, “Heed: a hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks,” Mobile Computing, IEEE Transactions on, vol. 3, no. 4, pp. 366–379, 2004.

[30] C. N. Hadijicostis and A. Rikos, “Distributed strategies for balancing a weighted digraph,” in 2012 20th Mediterranean Conference on Control & Automation (MED), 2012, pp. 1141–1146.

[31] B. Gharesifard and J. Cortés, “Distributed strategies for generating weight-balanced and doubly stochastic digraphs,” European Journal of Control, vol. 18, no. 6, pp. 539–557, 2012.

[32] K. Cai and H. Ishii, “Average consensus on arbitrary strongly connected digraphs with dynamic topologies,” in American Control Conference (ACC), 2012, 2012, pp. 14–19.

[33] L. Xiao and S. Boyd, “Fast linear iterations for distributed averaging,” Systems & Control Letters, vol. 53, no. 1, pp. 65–78, 2004.

[34] W. Ren, R. W. Beard, and E. M. Atkins, “A survey of consensus problems in multi-agent coordination,” in American Control Conference, 2005. Proceedings of the 2005, 2005, pp. 1859–1864.

[35] I. Shames, T. Charalambous, C. N. Hadijicostis, and M. Johansson, “Distributed network size estimation and average degree estimation and control in networks isomorphic to directed graphs,” in 50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC 2011), Dec 2011, pp. 4282–4289.

[36] L. Wang and F. Xiao, “Finite-time consensus problems for networks of dynamic agents,” Automatic Control, IEEE Transactions on, vol. 55, no. 4, pp. 950–955, April 2010.

[37] S. Sundaram and C. N. Hadijicostis, “Finite-time distributed consensus in graphs with time-invariant topologies,” in American Control Conference, 2007. ACC’07, 2007, pp. 711–716.

[38] Y. Yuan, G. B. Stan, M. Barahona, L. Shi, and J. Goncalves, “Decentralised minimal-time consensus,” in 50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC 2011), Dec 2011, pp. 4282–4289.

[39] T. Charalambous, Y. Yuan, T. Yang, W. Pan, C. N. Hadijicostis, and M. Johansson, “Decentralised minimum-time average consensus in digraphs,” in Proceedings of 52nd IEEE Conference on Decision and Control (CDC 2013), 2013, pp. 2617–2622.

[40] T.-M. D. Tran, A. Y. Kibangou et al., “Distributed design of finite-time average consensus protocols,” in Proc. IFAC Workshop on Distributed Estimation and Control (NeCSyS), 2013.