Optimization in Open Networks via Dual Averaging

Yu-Guan Hsieh[1], Franck Iutzeler[1], Jérôme Malick[2], and Panayotis Mertikopoulos[2,3]

Abstract— **In networks of autonomous agents (e.g., fleets of vehicles, scattered sensors), the problem of minimizing the sum of the agents' local functions has received a lot of interest. We tackle here this distributed optimization problem in the case of open networks when agents can join and leave the network at any time. Leveraging recent online optimization techniques, we propose and analyze the convergence of a decentralized asynchronous optimization method for open networks.**

I. Introduction

Multi-agent systems are a powerful modeling framework for the analysis of signal processing or machine learning over sensor networks, fleets of autonomous vehicles, opinion dynamics, etc. This framework calls for decentralized optimization methods where the agents seek to minimize the sum of the individual functions by exchanging information through a communication graph, without the help of a central authority. Indeed, such methods are key to perform distributed computing, signal processing, or learning from scattered sources in communication-constrained or large-scale environments; see e.g [1]–[4] and references therein.

In this regard, decentralized optimization methods have been extensively studied in the literature. Depending on the agents' computing abilities and tasks at hand, several types of algorithms were considered. On the one hand, gradient-based methods such as decentralized gradient descent [5]–[7], and decentralized dual averaging [8]–[11] update the local variables using gradients of the agents' objective functions (see also [12] for a recent review). On the other hand, splitting methods such as the alternating direction method of multipliers (ADMM) [13]–[15] imply that each agents has to minimize (a regularized version of) its local function, which can be too demanding depending on the application.

Despite the abundance of literature on this topic, most of them assume a network of fixed composition, that is, the agents that participate in the process always remain the same. On the contrary, this work focuses on the case of an *open network* where the agents can join and leave the system at any moment. This can happen in numerous situations, e.g.,

- When a cluster of servers is used to train a machine learning task, a node may leave the network due to a system failure or simply because the resource is acquired by another job. New nodes can also be deployed to accelerate the training process or process additional data.
- We can also think of the case of volunteer computing where volunteers provide computing resources for a distributed task. The network is naturally open since a device is only involved when the volunteer desires to participate.
- In multi-vehicle coordination, the set of vehicles that are considered by the algorithm can evolve with time.

Due to the dynamic nature of these open multi-agent systems, several works have studied the stability of consensus algorithm over the mean, maximum, or median of the agents values [16]–[20]. Among the very few works that tackle the problem of decentralized optimization over open networks, [21] showed that decentralized gradient descent is stable when agents/functions change over time if their objectives are sufficiently smooth.

Distributed algorithms also need to cope with asynchronous communications (i.e., the agents do not synchronize to communicate between optimization steps) with delays (i.e., there may be some gap between sending and receiving times). Such capacity is an important feature for scalability and flexibility. The study of this aspect was thus concomitant to the development of different decentralized optimization methods; see e.g., [22]–[24].

In this paper, we focus on the particular case of asynchronous open networks where i) the agents can communicate with each other asynchronously following a time-varying communication graph; ii) the exchanges and local processing incur delays; and iii) the agents may join and leave the system for arbitrary periods of time. While the first two points are relatively well studied in the literature as mentioned above, the last point tremendously complicates the analysis since the system may completely change from one step to another, see e.g., [25] and references therein.

To address this challenge, we develop the idea that (offline) optimization problems over open networks can be efficiently handled by tools from *online optimization*. Using this viewpoint and building on recent results on dual averaging for online learning with delays [26], we introduce DAERON (Dual AvERaging for Open Network), a method for optimization over open networks that allows for asynchronous communications. On the theoretical side, we study the algorithm's performance with respect to the average (over both time and agents) of the agents' functions. We then provide numerical simulations on a decentralized regression problem to illustrate the potential of our method.

The rest of the paper is organized as follows. In Section II, we formulate the open multi-agent optimization problem mathematically and define the corresponding performance measures. The main algorithm is described in Section III and

This work has been partially supported by MIAI@Grenoble Alpes (ANR-19-P3IA-0003)

[1] Univ. Grenoble Alpes, LJK, 38000 Grenoble, France `firstname.lastname@univ-grenoble-alpes.fr`

[2] CNRS, Univ. Grenoble Alpes, 38000 Grenoble, France `firstname.lastname@univ-grenoble-alpes.fr`

[3] Criteo AI Lab

analyzed in Section IV. Section V is dedicated to numerical experiments. Finally, Section VI concludes the paper and provides several clues for future research.

II. An Open Network of Computing Agents

A. Model

We consider a (possibly infinite) set of agents $\mathcal{V}$; each of them associated with an individual convex cost function $f_i\colon \mathbb{R}^d \to \mathbb{R}$. At each time $t = 1, 2, \ldots$, only a (finite) subset of agents $\mathcal{V}_t$ is active and may communicate using undirected communications links $\mathcal{E}_t := \{\{i,j\} \in \mathcal{V}_t^2 : i \text{ and } j \text{ can exchange at } t\}$. Let $m_t := \operatorname{card}(\mathcal{V}_t)$ denote the number of active agents at time t and we will also write $M_t := \sum_{s=1}^{t} m_s$.

In open multi-agent systems, it is in general impossible to define a temporally invariant global objective to minimize over time. Since those who are present in the network are usually the entities that we really care about, an alternative is to focus exclusively on the active agents and define the instantaneous loss at time t as

$$f_t^{\text{inst}}(x) = \frac{1}{m_t} \sum_{i \in \mathcal{V}_t} f_i(x).$$

The problem of interest is then the minimization of f_t^{inst}. However, providing a proper analysis for this time-varying problem is still challenging because the set of active agents $\mathcal{V}_t$ may change drastically between two consecutive time instants, which also leads to an abrupt change in the objective. In addition, agreeing on a consensus value in an open network is already a difficult problem [18], [20].

B. Quantity of Interest

Instead of tackling the minimization of f_t^{inst} directly, we draw inspiration from online learning and analyze the *running loss* defined for a time-horizon T as

$$\overline{\mathbf{Loss}}(T) = \frac{1}{M_T}\left(\sum_{t=1}^{T}\sum_{i\in\mathcal{V}_t} f_i(x_t^{\text{ref}}) - \min_{u\in\mathcal{X}}\sum_{t=1}^{T}\sum_{i\in\mathcal{V}_t} f_i(u)\right) \quad (1)$$

where $\mathcal{X} \subset \mathbb{R}^d$ is the shared constrained set and x_t^{ref} is a reference point for time t. In the sequel, we will consider algorithms in which each agent $i \in \mathcal{V}_t$ produces a local variable $x_{i,t}$ at time t. It is thus reasonable to set $x_t^{\text{ref}} = x_{i_t,t}$ for a reference agent $i_t \in \mathcal{V}_t$ that is chosen arbitrarily at each time, and $\overline{\mathbf{Loss}}$ would then represent the average network suboptimality over time for these reference agents. Note that the suboptimality is compared with the best fixed *a posteriori* action which is the solution $u^\star \in \mathcal{X}$ of

$$\min_{u\in\mathcal{X}} \left\{ f_T^{\text{run}}(u) := \frac{1}{M_T} \sum_{t=1}^{T} \sum_{i\in\mathcal{V}_t} f_i(u) \right\}.$$

This quantity mimics the collective regret in [26] with an additional $1/M_T$ which takes into account the total number of agents that have participated until time T. The notable difference here with the distributed online optimization literature [27]–[29] is that we consider an open network of agents so that the composition of the system can change over time.

Algorithm 1 DAERON at node i for active period t^{join}-t^{leave}

```
1: Parameters: $t^{\text{join}}$ time when the agent joins the network;
   $t^{\text{leave}}$ time when the agent leaves the network
2: Initialize: $\mathcal{S}_{i,t^{\text{join}}} \leftarrow \mathcal{S}_{j,t^{\text{join}}}$ for $j \in \mathcal{V}_{t^{\text{join}}-1} \cap \mathcal{V}_{t^{\text{join}}}$
3: for $t = t^{\text{join}}, \ldots, t^{\text{leave}}$ do
4:     Generate the prediction $x_{i,t}$ using (2)
5:     Compute the local subgradient $g_{i,t} \in \partial f_i(x_{i,t})$
6:     Send subgradients to other active agents
7:     Receive subgradients identified by the index set $\mathcal{G}_{i,t}$
8:     Update $\mathcal{S}_{i,t+1} \leftarrow \mathcal{S}_{i,t} \cup \mathcal{G}_{i,t} \cup \{g_{i,t}\}$
9: end for
```

III. DAERON: Dual AvERaging for Open Network

A. Algorithm

DAERON, as described in Algorithm 1, is a (sub)gradient-based method that applies dual averaging [30] at the level of the whole network. For this, we assume that given any point $x_{i,t} \in \mathcal{X}$, the agent i is able to compute a subgradient $g_{i,t} \in \partial f_i(x_{i,t})$. This information is transmitted to the whole network and used by all the agents for computing their own variable. Formally, let us define $\mathcal{S}_{i,t}$ as the index set of the subgradients that the agent $i \in \mathcal{V}_t$ has received/computed by time t. Then, the agent i generates the variable $x_{i,t}$ as

$$x_{i,t} = \Pi_{\mathcal{X}}\left(x_1 - \eta_{i,t} \sum_{(j,s)\in\mathcal{S}_{i,t}} g_{j,s} \right). \quad (2)$$

where $\Pi_{\mathcal{X}} \colon y \mapsto \arg\min_{x\in\mathcal{X}} \|x - y\|$ denotes the projection onto $\mathcal{X}$, x_1 is a common starting point, and $\eta_{i,t} > 0$ is a learning rate that can be both time and agent dependent. Note also that the communication of the subgradients (lines 6-7) can be done asynchronously in parallel with the other steps (lines 4-5) of the algorithm.

Remark 1 (Arriving agents): In Algorithm 1, we initialize an agent arriving at time t with the subgradient set of an arbitrary agent in $\mathcal{V}_{t-1} \cap \mathcal{V}_t$, which implicitly assumes that $\mathcal{V}_{t-1} \cap \mathcal{V}_t \neq \emptyset$. This is not crucial, as what really counts is that the agent is initialized with sufficient knowledge about what the network has computed, but we will stick to this model in the remainder of the paper for simplicity.

B. Practical Implementation

Storing all the available subgradients at a node can be prohibitively expensive, or even infeasible since this would require infinite storage capacity when t goes to infinity. It is thus important to note that DAERON is just a conceptual algorithm that can be implemented in different ways to circumvent this issue. We provide below two possible workarounds to demonstrate the flexibility of our method:

- We can maintain $y_{i,t} = \sum_{(j,s)\in\mathcal{S}_{i,t}} g_{j,s}$ while keeping track of the most recent subgradients in order to communicate them with other agents. Formally, suppose that each node has a number of potential neighbors they may be connected to; then a subgradient $g_{j,s}$ only needs to be stored until the time that it has been sent to or received from these potential neighbors.

- If the number of involved agents m is small, we may define the sum of the subgradients computed by agent i as $\check{y}_{i,t} = \sum_{s=1}^{t-1} g_{i,s}$ with the convention $g_{i,s} = 0$ when $i \notin \mathcal{V}_s$. Each node $i \in \mathcal{V}_t$ then stores a table of size $m \times d$ containing the vectors $\check{y}_{1,t-\tau_{1,i}(t)}, \dots, \check{y}_{m,t-\tau_{m,i}(t)}$ corresponding to delayed versions of the computed subgradient sums over the network, with $\tau_{j,i}(t)$ measuring this delay. These vectors are updated through communication. This strategy can also be adopted in a semi-centralized network with m central nodes and an arbitrary number of edge computing devices that retrieve information from these central nodes.

C. Quantities at play and assumptions

For our analysis, we make the following technical assumptions on the local cost functions and the constraint set.

Assumption 1: Each f_i is convex and G-Lipschitz. The common constraint set $\mathcal{X}$ is closed and convex.

In order for the problem to make sense, we will assume that the communication delays are upper bounded.

Assumption 2: The delays of the algorithm are upper bounded by τ, i.e., for any $t, s \in \mathbb{N}$ such that $t > s + \tau$ and any $i \in \mathcal{V}_t$, $j \in \mathcal{V}_s$, we have $(j, s) \in \mathcal{S}_{i,t}$.

In particular, Assumption 2 supposes that every piece of information is spread to the whole network in a finite amount of time. While this is quite evident in a static network, in the case of an open network this requires that the evolution of the network is slow enough with respect to the communication between the agents.

For concreteness, let us consider a model where every node communicates all its available gradients to its neighbors at each iteration. Then for a static network with a fixed topology $\mathfrak{G} = (\mathcal{V}, \mathcal{E})$, we have clearly $\tau = \mathrm{diam}(\mathfrak{G})$ where $\mathrm{diam}(\mathfrak{G})$ stands for the diameter of the graph. The case of an open network with changing topology is considerably more complicated. For instance, in the case of a line graph where at each time a new agent joins at the end of the graph, the diameter grows linearly with t and it becomes impossible to propagate information to the whole network. This is one kind of situation we want to avoid here. Formally, we define $\mathcal{V}_t^{j,s} = \{i \in \mathcal{V}_t : (j, s) \in \mathcal{S}_{i,t}\}$ as the set of active agents that possess $g_{j,s}$ at time t. The following example provides a sufficient stability assumption which allows information to spread across the network.

Example 1 (Bounded delays): Come back to the same situation as above where every node communicates all its available gradients to its neighbors during each iteration. Suppose further that the set of the active agents remain unchanged for periods of B iterations (i.e., for $t = lB, \dots, (l + 1)B - 1$), and that the graph $\mathfrak{G}_l = (\mathcal{V}_t, \bigcup_{t=lB}^{l(B+1)-1} \mathcal{E}_s)$ is k-vertex-connected (i.e., removing at least k nodes is necessary to disconnect it). If the number of arriving plus leaving agents at the end of iteration $(l + 1)B - 1$ is $q < k$, then for any $s \leq lB - 1$ and $j \in \mathcal{V}_s$, the number of nodes that do not possess $g_{j,s}$ is reduced by at least $k - q > 0$ after the B iterations, i.e., $\mathrm{card}(\mathcal{V}_{(l+1)B}^{j,s}) \leq \max(0, \mathrm{card}(\mathcal{V}_{lB}^{j,s}) - (k - q))$. In that case, the maximal delay is thus bounded by $\max_t m_t B/(k - q) + B$.

The above example means that provided that the number of exiting/incoming agents is not too large compared to the connectivity of the communication graph, the delays are naturally bounded.

Remark 2 (Loss of information due to agent departure): When an agent leaves the network, some of its computed subgradients may be lost for the network. This can happen either because it never communicated them or because the agents to which it communicated them also left the network. In Example 1, the change of composition at the end of iteration $(l + 1)B - 1$ could notably cause the loss of the subgradients computed at time $t = lB, \dots, (l + 1)B - 1$. This does not affect the proposed algorithm. The only difference is in the analysis: we will consider that the agent j is not in $\mathcal{V}_s$ if $\mathcal{V}_t^{j,s} = \emptyset$ starting from some t.[1]

IV. Analysis

A. Performance in the general case

We provide the convergence result for DAERON in terms of the running loss defined in (1). To do so, we define the quadratic mean and the average number of active agents over time:

$$m_{\mathrm{QM}} = \sqrt{\frac{1}{T}\sum_{t=1}^{T} m_t^2} \quad \text{and} \quad \overline{m} = \frac{1}{T}\sum_{t=1}^{T} m_t.$$

Note that we always have $m_{\mathrm{QM}} \geq \overline{m}$.

Theorem 1: Let Assumptions 1 and 2 hold. Running DAERON with constant learning rate

$$\eta_{i,t} \equiv \eta = \frac{r_0}{m_{\mathrm{QM}} G \sqrt{(3\tau + 1)T}}$$

for some $r_0 > 0$ guarantees that

$$\overline{\mathbf{Loss}}(T) \leq \frac{m_{\mathrm{QM}}}{\overline{m}} \frac{2\alpha r G \sqrt{3\tau + 1}}{\sqrt{T}}, \tag{3}$$

where $r = \|u^\star - x_1\|^2$ and $\alpha = \max(r/(2r_0), r_0/r)$.

Proof: An important feature of DAERON is that the subgradients are not always applied to the points where they are evaluated. To accommodate this in our analysis, we consider the virtual iterates $(\widetilde{x}_t)_{t\in\mathbb{N}}$ defined by

$$\widetilde{x}_t = \Pi_{\mathcal{X}}\left(x_1 - \eta \sum_{t=1}^{t-1} \sum_{i\in\mathcal{V}_t} g_{i,t}\right).$$

[1]Note that in Example 1, we have either $\mathcal{V}_t^{j,s} = \emptyset$ or $\mathcal{V}_t^{j,s} = \mathcal{V}_t$ for t sufficiently large (assuming that s is fixed).

From the regret analysis of dual averaging (see, e.g., [31, Prop. 2]), the following holds for all $u \in \mathcal{X}$,

$$\begin{aligned}\sum_{t=1}\sum_{i\in\mathcal{V}_t}\langle g_{i,t}, \widetilde{x}_t - u\rangle &\le \frac{\|u-x_1\|^2}{2\eta} + \frac{\eta}{2}\sum_{t=1}^{T}\left\|\sum_{i\in\mathcal{V}_t} g_{i,t}\right\|^2 \\ &\le \frac{\|u-x_1\|^2}{2\eta} + \frac{\eta}{2}\sum_{t=1}^{T} m_t^2 G^2.\end{aligned}$$

In the second line we have used the Lipschitz continuity of the functions which implies that $\|g_{i,t}\| \le G$. Next, by using the convexity and the Lipschitz continuity of the functions, each term of (1) can be bounded for any $u \in \mathcal{X}$ by

$$\begin{aligned}f_i(x_t^{\text{ref}}) - f_i(u) &= f_i(x_{i_t,t}) - f_i(x_{i,t}) + f_i(x_{i,t}) - f_i(u) \\ &\le G\|x_{i_t,t} - x_{i,t}\| + \langle g_{i,t}, x_{i,t} - u\rangle.\end{aligned}$$

We proceed to bound the second term in the above inequality

$$\begin{aligned}\langle g_{i,t}, x_{i,t} - u\rangle &= \langle g_{i,t}, x_{i,t} - \widetilde{x}_t\rangle + \langle g_{i,t}, \widetilde{x}_t - u\rangle \\ &\le G\|x_{i,t} - \widetilde{x}_t\| + \langle g_{i,t}, \widetilde{x}_t - u\rangle.\end{aligned}$$

Using the triangle inequality, we get $\|x_{i_t,t} - x_{i,t}\| \le \|x_{i_t,t} - \widetilde{x}_t\| + \|x_{i,t} - \widetilde{x}_t\|$. Now, let $\Gamma_t = \max_{i\in\mathcal{V}_t}\|x_{i,t} - \widetilde{x}_t\|$ and set $u \leftarrow u^\star$. We have from the above

$$\overline{\mathbf{Loss}}(T) \le \frac{1}{M_t}\left(\frac{\|u^\star - x_1\|^2}{2\eta} + \sum_{t=1}^{T}(\frac{\eta}{2}m_t^2G^2 + 3m_tG\Gamma_t)\right).$$

To conclude, we may bound Γ_t thanks to the bounded delay assumption. In fact, since the projection operator is non-expansive, it holds for all $i \in \mathcal{V}_t$ that

$$\begin{aligned}\|x_{i,t} - \widetilde{x}_t\| &\le \left\|\eta\sum_{t=1}^{t-1}\sum_{j\in\mathcal{V}_t} g_{i,t} - \eta\sum_{(j,s)\in\mathcal{S}_{i,t}} g_{j,s}\right\| \\ &\le \left\|\eta\sum_{s=t-\tau}^{t-1}\sum_{j\in\mathcal{V}_s} g_{j,s}\right\| \le \eta\sum_{s=t-\tau}^{t-1} m_s G.^{2}\end{aligned}$$

Then, with $m_t m_s \le (m_t^2 + m_s^2)/2$, we obtain the following

$$\overline{\mathbf{Loss}}(T) \le \frac{1}{M_t}\left(\frac{\|u^\star - x_1\|^2}{2\eta} + \eta G^2\sum_{t=1}^{T}(3\tau+1)m_t^2\right).$$

Inequality (3) follows immediately by the definition of m_{QM}, $\overline{m}$, and α. ■

Theorem 1 shows that when the ratio $m_{\text{QM}}/\overline{m}$ is upper bounded, the running loss of the algorithm has a convergence rate in $\mathcal{O}(1/\sqrt{T})$. The factor $m_{\text{QM}}/\overline{m}$ also indicates that the algorithm converges slower when the number of active agents varies greatly across iterations, which is expected because the algorithm would need more time to accommodate the change in this situation.

Remark 3 (Extensions): One drawback of the theorem is that the expression of the learning rate involves both the sqaure mean number of the agents m_{QM} and the time horizon T. In a continuously evolving network, neither of these two quantities are known in advance. We provide below several alternatives which allow us to establish the same $\mathcal{O}(1/\sqrt{T})$ rate without these quantities. Proofs are variations of the above proof; we omit details due to space limitations.

- If the number m_t is known to every agent. We may use $\eta_{i,t} = \eta_t = \Theta(1/\sqrt{\tau(\sum_{s=1}^{t} m_s^2)})$.
- If $m_{\max} = \max_{1\le t\le T} m_t$ can be estimated and the agents have access to a global clock that indicates t, we can take $\eta_{i,t} = \eta_t = \Theta(1/(m_{\max}\sqrt{\tau t}))$.
- Note that $\sqrt{\sum_{s=1}^{t} m_s^2} \le \sqrt{m_{\max}}\sqrt{M_t}$. Therefore, provided that $m_{\max}$ is known, another alternative is to use $\eta_{i,t} = \Theta(1/\sqrt{\tau m_{\max} M_t})$. This does not require to know m_t explicitly since M_t can be estimated by $\text{card}(\mathcal{S}_{i,t})$. In particular, under Assumption 2, it holds $M_t \le \text{card}(\mathcal{S}_{i,t}) + (\tau+1)m_{\max}$.

B. *The case of fixed agents $\mathcal{V}_t = \mathcal{V}$*

For comparison with existing literature (e.g., [8]), we now turn back to the case of a "closed network" where all the agents are active at each iteration. In this situation, we can define the (fixed) global loss as

$$f(x) = \frac{1}{m}\sum_{i\in\mathcal{V}} f_i(x),$$

where $m = \text{card}(\mathcal{V})$ is the number of agents. We have $m_{\text{QM}} = \overline{m} = m$ and thus $m_{\text{QM}}/\overline{m} = 1$. As an immediate corollary of Theorem 1, if we apply DAERON with the constant stepsize

$$\eta_{i,t} \equiv \eta = \frac{r}{mG\sqrt{(3\tau+1)T}},$$

then for any agent $i \in \mathcal{V}$,

$$\begin{aligned}f\left(\frac{1}{T}\sum_{t=1}^{T} x_{i,t}\right) - \min_{u\in\mathcal{X}} f(u) &\le \frac{1}{T}\sum_{t=1}^{T} f(x_{i,t}) - \min_{u\in\mathcal{X}} f(u) \\ &\le \frac{2rG\sqrt{3\tau+1}}{\sqrt{T}}.\end{aligned}$$

This means that the running average of each agent's iterates decreases in global suboptimality at rate $\mathcal{O}(1/\sqrt{T})$, which matches the rate of [8, Th. 2] for the slightly different decentralized dual averaging algorithm.[3] Going one step further, the same result would still hold under asynchronous activation as long as the activation patterns can be described by a stationary probability distribution. On the other hand, Theorem 1 also allows us to prove the convergence of the algorithm in an open network whose composition stops changing after finite time.

V. Numerical Experiments

In this section, we demonstrate the effectiveness of DAERON with experiments on a static and an open network.

[2] If $s \le 0$, then $\mathcal{V}_s = \emptyset$ and $m_t = 0$.

[3] In [8], the subgradient are averaged by gossiping while in DAERON, they are directly exchanged.

A. Problem Description

Let us consider a decentralized least absolute deviation (LAD) regression model. Given a data set evenly distributed on m nodes $(a_{ik}, b_{ik})_{i,k\in[m]\times[n]}$ with a_{ik} in $\mathbb{R}^d$ and $b_{ik} \in \mathbb{R}$, it consists in solving

$$\min_{x\in\mathbb{R}^d} \left\{ f(x) := \frac{1}{m}\sum_{i=1}^{m}\frac{1}{n}\sum_{k=1}^{n}|a_{ik}^\top x - b_{ik}| \right\}. \quad (4)$$

Compared to least square regression, LAD is known to be more resistant to the presence of outliers. Although the use of absolute value makes the problem non-differentiable, DAERON can be run with subgradients as suggested by our analysis. For the experiments, we generate synthetic data as follows:

1) The ground truth model $x^\star \in [-5, 5]^d$ is drawn from a uniform distribution.
2) The local model $x_i^\star$ of the node i is obtained by perturbing $x^\star$ with a Gaussian noise, i.e., $x_i^\star = x^\star + \varepsilon_i$ where $\epsilon_i \sim \mathcal{N}(0, I_d)$.
3) We sample $a_{ik} \sim \mathcal{N}(0, I_d)$ and compute $b_{ik} = a_{ik}^\top x_i^\star + \varepsilon_{ik}$ with $\varepsilon_{ik} \sim \mathcal{N}(0, 1)$.
4) On each node, a random portion of samples are corrupted. For these samples, we replace b_{ik} by a random value generated from a Gaussian distribution.

In the above, we introduce the second and the fourth steps mainly for two reasons. First, it makes the problem more heterogeneous, and thus more difficult. Second, it makes the communication between agents more important for finding a good approximation of $x^\star$. In the following, we will take $m = 64$ nodes, $n = 200$ samples per node, and dimension $d = 20$. On each node, the number of corrupted samples is random in $\{0, \ldots, 120\}$. We also verify that the solution $\hat{x}$ of (4) is not too far from $x^\star$.

B. Static network

We first investigate the performance of the algorithm on a static network. The nodes are arranged in a 2d grid of size 8×8. Adjacent nodes exchange gradients at each iteration. Communication-computation overlap is allowed for better efficiency— in Algorithm 1, this means that lines 4-5 and 6-7 are run in parallel. Then, with a constant stepsize η, the update writes

$$x_{i,t+1} = x_{i,t} - \eta\sum_{i=1}^{m} g_{j,t-\tau_{j,i}},$$

where $\tau_{j,i}$ is the distance between the nodes j and i. For illustration purposes, we also compare with a decentralized (sub)gradient descent (DGD) method [5] with constant stepsize γ and a mixing matrix $W = (w_{i,j})$. Its update is

$$x_{i,t+1} = \sum_{j=1}^{m} w_{i,j}x_{j,t} - \gamma g_{i,t}$$

Algorithm 2 DAERON at each node i as implemented in Section V-C

```
 1: Initialize: $\mathcal{S}_{i,1} \leftarrow \emptyset$, activation status $\zeta_i \in \{0,1\}$,
    network parameters $K \in \mathbb{N}$, $p \in [0,1]$
 2: for $t = 1, 2, \ldots$ do
 3:   Agent update
 4:   if $\zeta_i = 1$ then
 5:     Get randomly paired with another active agent $j$
 6:     Compute $x_{i,t}$ by (2) and $g_{i,t} \in \partial f_i(x_{i,t})$
 7:     Update $\mathcal{S}_{i,t+1} \leftarrow \mathcal{S}_{i,t} \cup \mathcal{S}_{j,t} \cup \{g_{i,t}\}$
 8:   end if
 9:   Network evolution
10:   if $(t+1) \equiv 0 \mod K$ then
11:     Draw a Bernoulli random variable $z_i \sim \mathcal{B}(p)$
12:     $\zeta_i \leftarrow \zeta_i + z_i \mod 2$
13:     if $z_i = 1$ and $\zeta_i = 1$ then
14:       Pick randomly $j \in \mathcal{V}_t \cap \mathcal{V}_{t+1}$
15:       Update $\mathcal{S}_{i,t+1} \leftarrow \mathcal{S}_{j,t+1}$
16:     end if
17:   end if
18: end for
```

and we take W as the Metropolis matrix of the graph in our experiments:

$$w_{i,j} = \begin{cases} 1/(\max(\deg(i), \deg(j)) + 1) & \text{if } \{i,j\} \in \mathcal{E}, \\ 1 - \sum_{k=1}^{m} w_{i,k} & \text{if } i = j, \\ 0 & \text{otherwise,} \end{cases}$$

with $\deg(i)$ denoting the degree of the node i.

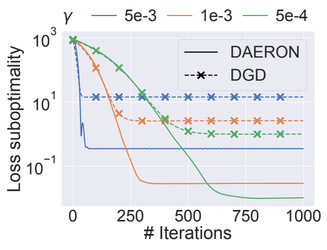
For a proper comparison of the two algorithms, it is important to notice that a subgradient is sent to all the m nodes in DAERON while it is averaged out in DGD. Therefore, we will take $\gamma = m\eta$ and refer to it as the *effective learning rate* of both methods. With this in mind, in Fig. 1a we plot the convergence of the averaged optimality gap $(1/m)\sum_{i=1}^{m} f(x_{i,t}) - \min f$ for DAERON and DGD with different choices of γ.

Interestingly, we observe that when using the same effective learning rate, the two algorithms establish similar convergence behavior until reaching their respective fixed points. However, DAERON is able to converge to a point with higher accuracy. We believe that this is because the variables $(x_{i,t})_{i\in\mathcal{V}}$ tend to be closer to each other in DAERON.[4]

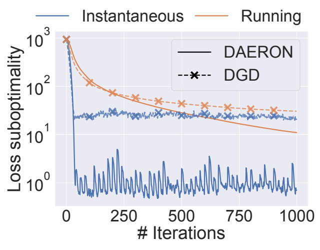
C. Open network

Now that we have shown that DAERON performs comparably to standard decentralized optimization methods in a static network, we proceed to study its behavior in an open multi-agent system (Algorithm 2). Following [32], we model the arrivals and departures of the agents by a Bernoulli

[4]Indeed, let Δ and $1-\lambda$ be respectively the diameter of the graph and the spectral gap of the mixing matrix. Then, as shown in the proof of Theorem 1, for DAERON we can roughly bound $\|x_{i,t} - x_{j,t}\|$ by $m\eta\tau G = \Delta\gamma G$. As for DGD, it is known that we have asymptotically $\|x_{i,t} - x_{j,t}\| \lesssim \gamma G/(1-\lambda)$ [12, Lem. 11]. Since it generally holds $\Delta \leq 1/(1-\lambda)$ and this is notably the case for the graph that we consider, this provides a possible explanation for the superiority of DAERON over DGD.



(a) Static network



(b) Open network

Fig. 1: Comparison of DAERON and DGD. For a static network we plot in (a) the averaged suboptimality. For an open network we plot in (b) the averaged instantaneous suboptimality (5) and the averaged running loss (6).

process. Initially, only half of the 64 nodes are active. Then, every $K = 20$ iterations, each agent may change its activation status (i.e., from active to inactive or vice-versa) with probability $p = 0.05$. At each iteration, the active nodes are randomly paired with each other, then each pair synchronizes their gradients.[5] Formally, if nodes i and j are paired at time t, then $\mathcal{S}_{i,t+1} \setminus \{g_{i,t}\} = \mathcal{S}_{j,t+1} \setminus \{g_{j,t}\} = \mathcal{S}_{i,t} \cup \mathcal{S}_{j,t}$. In the spirit of DGD, we also implement an algorithm which directly updates the primal variables as

$$x_{i,t+1} = \frac{x_{i,t} + x_{j,t}}{2} - \gamma g_{i,t},$$
$$x_{j,t+1} = \frac{x_{i,t} + x_{j,t}}{2} - \gamma g_{j,t}.$$

In both cases, an agent that becomes active at the end of round $t-1$ is assigned the state variable (i.e., $\mathcal{S}_{i,t}$ or $x_{i,t}$) of a random node in $\mathcal{V}_{t-1} \cap \mathcal{V}_t$. We take $\gamma = m\eta/2$ in our experiment since on average $m/2$ nodes are active.

As for the performance measure, we consider the averaged instantaneous optimality gap $\bar{\ell}^{\text{inst}}(t)$ and the averaged running loss $\bar{\ell}^{\text{run}}(t)$ defined by

$$\bar{\ell}^{\text{inst}}(t) = \frac{1}{m_t} \sum_{i \in \mathcal{V}_t} f_t^{\text{inst}}(x_{i,t}) - \min f_t^{\text{inst}} \tag{5}$$

$$\bar{\ell}^{\text{run}}(t) = \frac{1}{M_t} \sum_{s=1}^{t} \frac{1}{m_s} \sum_{i \in \mathcal{V}_s} f_s^{\text{inst}}(x_{i,s}) - \min f_t^{\text{run}}. \tag{6}$$

The evolution of these two measures for $\gamma = 0.005$ are plotted in Fig. 1b. We see that both algorithms are able to converge to an area where potential solutions are located whereas DAERON can get much closer to the optimum of f_t^{inst}. Moreover, we observe that the instantaneous loss for DAERON increases abruptly when the set of active agents changes and gets decreased again afterwards. This indicates that the algorithm actually has the ability to track the instantaneous solution.

VI. Conclusion

In this paper, we introduced a decentralized optimization method for open multi-agent systems, in which agents can freely join or leave the network during the process. This setting is particularly challenging since there is not a clear objective to minimize and even exact convergence to a consensus is generally out of reach. These difficulties led us to adopting techniques and performance measures from online optimization. We proved that our algorithm benefits from a $\mathcal{O}(1/\sqrt{T})$ convergence rate with respect to the running average of the functions of the agents present in the network. In our simulations, we showed that our method outperformed decentralized subgradient descent on a least absolute deviation problem.

The positive results in our numerical experiments also raises several important questions: How can we analyze the tracking behavior of the algorithm from a theoretical perspective? Is it possible to combine DAERON with other consensus-based methods to achieve superior performance? How do we determine the best algorithm to use given a specific open network? Each of these question leads to new research direction that is worth exploring in future works.

[5]If there is an odd number of nodes, one node is ignored in this process.

References

[1] J. N. Tsitsiklis, "Problems in decentralized decision making and computation." Massachusetts Inst of Tech Cambridge Lab for Information and Decision Systems, Tech. Rep., 1984.

[2] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE transactions on information theory*, vol. 52, no. 6, pp. 2508–2530, 2006.

[3] A. G. Dimakis, S. Kar, J. M. Moura, M. G. Rabbat, and A. Scaglione, "Gossip algorithms for distributed signal processing," *Proceedings of the IEEE*, vol. 98, no. 11, pp. 1847–1864, 2010.

[4] X. Lian, W. Zhang, C. Zhang, and J. Liu, "Asynchronous decentralized parallel stochastic gradient descent," in *International Conference on Machine Learning*. PMLR, 2018, pp. 3043–3052.

[5] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, 2009.

[6] P. Bianchi and J. Jakubowicz, "Convergence of a multi-agent projected stochastic gradient algorithm for non-convex optimization," *IEEE transactions on automatic control*, vol. 58, no. 2, pp. 391–405, 2012.

[7] D. Jakovetić, J. Xavier, and J. M. Moura, "Fast distributed gradient methods," *IEEE Transactions on Automatic Control*, vol. 59, no. 5, pp. 1131–1146, 2014.

[8] J. C. Duchi, A. Agarwal, and M. J. Wainwright, "Dual averaging for distributed optimization: Convergence analysis and network scaling," *IEEE Transactions on Automatic control*, vol. 57, no. 3, pp. 592–606, 2011.

[9] K. I. Tsianos, S. Lawlor, and M. G. Rabbat, "Push-sum distributed dual averaging for convex optimization," in *2012 ieee 51st ieee conference on decision and control (cdc)*. IEEE, 2012, pp. 5453–5458.

[10] S. Lee, A. Nedić, and M. Raginsky, "Stochastic dual averaging for decentralized online optimization on time-varying communication graphs," *IEEE Transactions on Automatic Control*, vol. 62, no. 12, pp. 6407–6414, 2017.

[11] I. Colin, A. Bellet, J. Salmon, and S. Clémençon, "Gossip dual averaging for decentralized optimization of pairwise functions," in *International Conference on Machine Learning*. PMLR, 2016, pp. 1388–1396.

[12] A. Nedić, A. Olshevsky, and M. G. Rabbat, "Network topology and communication-computation tradeoffs in decentralized optimization," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 953–976, 2018.

[13] S. Boyd, N. Parikh, and E. Chu, *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

[14] F. Iutzeler, P. Bianchi, P. Ciblat, and W. Hachem, "Asynchronous distributed optimization using a randomized alternating direction method of multipliers," in *52nd IEEE conference on decision and control*. IEEE, 2013, pp. 3671–3676.

[15] E. Wei and A. Ozdaglar, "On the o (1= k) convergence of asynchronous distributed alternating direction method of multipliers," in *2013 IEEE Global Conference on Signal and Information Processing*. IEEE, 2013, pp. 551–554.

[16] M. Franceschelli and P. Frasca, "Proportional dynamic consensus in open multi-agent systems," in *2018 IEEE Conference on Decision and Control (CDC)*, 2018, pp. 900–905.

[17] Z. A. Z. Sanai Dashti, C. Seatzu, and M. Franceschelli, "Dynamic consensus on the median value in open multi-agent systems," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019, pp. 3691–3697.

[18] M. Franceschelli and P. Frasca, "Stability of open multi-agent systems and applications to dynamic consensus," *IEEE Transactions on Automatic Control*, 2020.

[19] C. M. de Galland and J. M. Hendrickx, "Lower bound performances for average consensus in open multi-agent systems," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019, pp. 7429–7434.

[20] C. M. de Galland, S. Martin, and J. M. Hendrickx, "Open multi-agent systems with variable size: the case of gossiping," *arXiv preprint arXiv:2009.02970*, 2020.

[21] J. M. Hendrickx and M. G. Rabbat, "Stability of decentralized gradient descent in open multi-agent systems," in *2020 59th IEEE Conference on Decision and Control (CDC)*, 2020, pp. 4885–4890.

[22] K. I. Tsianos, S. Lawlor, and M. G. Rabbat, "Consensus-based distributed optimization: Practical issues and applications in large-scale machine learning," in *2012 50th annual allerton conference on communication, control, and computing (allerton)*. IEEE, 2012, pp. 1543–1550.

[23] T. Wu, K. Yuan, Q. Ling, W. Yin, and A. H. Sayed, "Decentralized consensus optimization with asynchrony and delays," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 4, no. 2, pp. 293–307, 2017.

[24] Z. Zhou, P. Mertikopoulos, N. Bambos, P. Glynn, Y. Ye, L.-J. Li, and L. Fei-Fei, "Distributed asynchronous optimization with unbounded delays: How slow can you go?" in *International Conference on Machine Learning*. PMLR, 2018, pp. 5970–5979.

[25] C. M. de Galland and J. M. Hendrickx, "Fundamental performance limitations for average consensus in open multi-agent systems," *arXiv preprint arXiv:2004.06533*, 2020.

[26] Y.-G. Hsieh, F. Iutzeler, J. Malick, and P. Mertikopoulos, "Multi-agent online optimization with delays: Asynchronicity, adaptivity, and optimism," *arXiv preprint arXiv:2012.11579*, 2020.

[27] S. Hosseini, A. Chapman, and M. Mesbahi, "Online distributed optimization via dual averaging," in *52nd IEEE Conference on Decision and Control*. IEEE, 2013, pp. 1484–1489.

[28] S. Shahrampour and A. Jadbabaie, "Distributed online optimization in dynamic environments using mirror descent," *IEEE Transactions on Automatic Control*, vol. 63, no. 3, pp. 714–725, 2017.

[29] F. Yan, S. Sundaram, S. Vishwanathan, and Y. Qi, "Distributed autonomous online learning: Regrets and intrinsic privacy-preserving properties," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 11, pp. 2483–2493, 2012.

[30] Y. Nesterov, "Primal-dual subgradient methods for convex problems," *Mathematical Programming*, vol. 120, no. 1, pp. 221–259, 2009.

[31] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *The Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[32] V. S. Varma, I.-C. Morărescu, and D. Nešić, "Open multi-agent systems with discrete states and stochastic interactions," *IEEE Control Systems Letters*, vol. 2, no. 3, pp. 375–380, 2018.