Network Support for High-performance Distributed Machine Learning

Network Support for High-performance Distributed Machine Learning

Francesco Malandrino, *Senior Member, IEEE,* Carla Fabiana Chiasserini, *Fellow, IEEE,*
Nuria Molner, *Student Member, IEEE,* and Antonio de la Oliva, *Member, IEEE*

***Abstract*—The traditional approach to distributed machine learning is to adapt learning algorithms to the network, e.g., reducing updates to curb overhead. Networks based on intelligent edge, instead, make it possible to follow the opposite approach, i.e., to define the logical network topology *around* the learning task to perform, so as to meet the desired learning performance. In this paper, we propose a system model that captures such aspects in the context of supervised machine learning, accounting for both learning nodes (that perform computations) and information nodes (that provide data). We then formulate the problem of selecting (i) which learning and information nodes should cooperate to complete the learning task, and (ii) the number of epochs to run, in order to minimize the learning cost while meeting the target prediction error and execution time. After proving important properties of the above problem, we devise an algorithm, named DoubleClimb, that can find a $1 + 1/|\mathcal{I}|$-competitive solution (with $\mathcal{I}$ being the set of information nodes), with cubic *worst-case* complexity. Our performance evaluation, leveraging a real-world network topology and considering both classification and regression tasks, also shows that DoubleClimb closely matches the optimum, outperforming state-of-the-art alternatives.**

***Index Terms*—Network orchestration, machine learning, edge computing.**

I. Introduction

Owing to the ever-increasing scale and complexity of the learning tasks to perform, machine learning (ML) algorithms have swiftly been extended to work in a distributed fashion, with the purpose of leveraging the computational capability of multiple nodes, possibly across multiple datacenters [1]–[4] and/or allowing nodes belonging to different parties to cooperate in a learning task without sharing sensitive data [5]–[7].

More recently, distributed ML has emerged also as an excellent match for new generation (5G-and-beyond) networks. It can be used for the management of the network (as envisioned by such initiatives as ETSI ZSM [8], ENI [9], and O-RAN [10]), as well as to enable user services within the so-called *intelligent edge* [11]. In general, new generation networks can (a) integrate a wide number of *heterogeneous* nodes, including those that can provide the data used for ML tasks, (b) provide a distributed computational infrastructure needed to run the ML algorithms (see e.g., [12]), and (c) be dynamically reconfigured so as to perform the ML task at hand with the required performance.

However, implementing an ML task in a 5G-and-beyond network also poses important challenges. Specifically, it requires to define the *logical topology* of the nodes that cooperate towards the ML task, i.e., making decisions on:

- which computing nodes in the different locations of the network edge should interact during the learning process;
- how many (and which) data sources to exploit, and which computing nodes should receive their data.

The above decisions influence each other, often in counterintuitive ways: as an example, seeking information from too many nodes may result in longer learning times, due to the additional waiting. Furthermore, a given target learning error (e.g., classification accuracy) may be reached through alternative, completely different approaches, e.g., collecting a significant quantity of information *or* performing more epochs to process a smaller set of data.

In spite of the wide usage of ML in mobile networks and the considerable attention devoted to it, most of the works aim at exploiting the network more efficiently, e.g., reducing the overhead [1], [13] or dealing with straggling nodes [14]. Just a small number of recent works [5], [15] have characterized the impact of the network topology on the performance of distributed ML, providing interesting insights on, e.g., the optimal network connectivity. However, *none* of these works tackle the problem of defining the logical network topology *around* the ML task to perform.

In this work, we focus on distributed, supervised learning, and aim at filling this gap by making the following main contributions:

- we develop a system model that can represent several relevant supervised ML tasks and account for the specific features of a 5G-and-beyond environment, most notably, the interaction between learning nodes and information nodes;
- we formulate the problem of choosing the computing nodes and data sources, as well as the links connecting them, with the goal of minimizing the (monetary or energy) cost of the learning process, subject to prediction quality and learning time requirements;
- we prove that the problem is NP hard, but also, and most importantly, that it is submodular. In particular, although its constraints are not monotonically increasing, we show that it can be solved via an iterative algorithm with excellent competitive ratio guarantees;

F. Malandrino and C. F. Chiasserini are with CNR-IEIIT and CNIT, Italy. C. F. Chiasserini is with Politecnico di Torino, Italy. N. Molner is with IMDEA Networks, Spain. N. Molner and A. de la Oliva are with Universidad Carlos III de Madrid, Spain.

This work was supported through the EU 5Growth project (Grant No. 856709).

- we propose an iterative algorithm, called DoubleClimb, which has cubic *worst-case* time complexity and attains a $1 + 1/|\mathcal{I}|$ *competitive ratio*, with $\mathcal{I}$ being the set of information nodes. We evaluate DoubleClimb over a real-world topology, showing that it closely matches optimal decisions and substantially outperforms state-of-the-art alternatives.

The rest of the paper is organized as follows. After reviewing related work in Sec. II, we describe our system model and how it can represent different supervised ML tasks in Sec. III. In Sec. IV, we formulate the problem we tackle and discuss its complexity. Sec. V characterizes the learning performance, while important properties of our problem are proven in Sec. VI. We then present the DoubleClimb algorithm and analyze its complexity in Sec. VII, before evaluating its performance in Sec. VIII. We conclude the paper in Sec. IX.

II. Related Work

Our work is related to the body of research works on distributed learning. In this context, in the simplest scenarios [16], all training data is known before the training itself starts, and the purpose of performing distributed learning is simply to leverage more computational power. A more complex variation is represented by *active learning* where new information arrives during the learning process, and is combined with the offline training set [17], [18]. Applications include drone planning [2] and network management [19], [20].

Federated learning is a more recent trend, tackling scenarios where participating devices are not required to share potentially sensitive data [7], [21]. Depending upon the specific scenario, new data may or may not arrive during the training process.

Several works propose generic methodologies to mitigate common hurdles of distributed ML, including scaling the parameter servers [1], dealing with slower nodes [14], and trading learning efficiency for convergence speed [13]. All these works propose novel algorithms and/or approaches to *adapt to* the existing network structure, e.g., by limiting the overhead, to perform the learning task at hand as efficiently as possible. Importantly, *none* of them envision to do the opposite, i.e., adapting the nodes' interaction to the learning task.

Some works seek to theoretically characterize the convergence of supervised ML and how it is influenced by the cooperation among learning nodes. The study in [4] characterizes the convergence of a wide class of multi-agent algorithms. Using tools from spectral graph analysis, it establishes a relation between the topology formed by pairs of cooperating nodes and the convergence of the algorithm they run. [15] focuses on distributed ML over regular topologies, and seeks to establish the graph degree associated with the shortest convergence *time* – as opposed to the lowest number of epochs –, finding that such a degree depends on the distribution of the nodes' computing time. Through similar steps and targeting a resource-constrained edge-computing scenario, [5] searches for the optimal trade-off between local computation and global parameter exchange in federated learning scenarios.

TABLE I
Main notation

Symbol	Meaning
$\mathcal{L}$, $\mathcal{I}$	L-nodes and I-nodes set (resp.)
$\rho_i(t)$	pdf of sample generation time at I-node $i \in \mathcal{I}$
r_i	ave. no. of samples per epoch by I-node i
X_l^k	amount of samples at the beginning of epoch k at L-node l
c_l, c_i	operational cost of L-node l and I-node i (resp.)
$c_{l,l'}$	communication cost between L-nodes l, l'
$c_{i,l}$	communication cost between I-node i and L-node l
$\epsilon^{\max}$	maximum learning error
$T^{\max}$	maximum duration of the learning process
$p(l,l')$	binary variable determining if L-nodes l and l' cooperate (matrix $\mathbf{P}$)
$q(i,l)$	binary variable determining if L-node node l obtains samples from I-node i (matrix $\mathbf{Q}$)
K	number of epochs to run
$\tau_l^k(t)$	pdf of the computation time at L-node l and epoch k
$\epsilon^K(\mathbf{P},\mathbf{Q})$	global error at the end of the whole learning process
$T^K(\mathbf{P},\mathbf{Q})$	expected time to complete the whole learning process
$C^K(\mathbf{P},\mathbf{Q})$	global cost for running the whole learning process

With respect to [4], [5], [15], we (i) seek to adapt the logical network topology to the learning task, and (ii) consider not only learning nodes (in charge of processing information), but also information nodes, where data comes from. The latter is especially critical, as it allows us to characterize and study the trade-off between gathering information and extracting knowledge from it.

III. System Model

Our system model addresses a generic distributed, supervised ML task where multiple nodes cooperatively seek to minimize a *loss function*, via gradient descent approaches such as the *stochastic gradient descent* (SGD) algorithm [3], [5], [15], [22]. In the following, we discuss how the behavior of individual nodes and their interactions are described by our system model, with reference to different real-world ML approaches.

Nodes' interactions. A *unique* feature of our model is its ability to capture the presence of two different types of nodes:

- *learning nodes*, or L-nodes for short, that, having computational capabilities, run the ML algorithm and can exchange gradient data during learning; we denote their set by $\mathcal{L}$;
- *information nodes*, or I-nodes for short, which can provide information to the L-nodes; we denote their set by $\mathcal{I}$.

Real-world counterparts of L-nodes include physical servers and virtual machines running at the intelligent network edge [11] or in the cloud. I-nodes, on the other hand, represent such entities as monitoring platforms, network nodes, and sensors.

In our system model, L-nodes behave in a similar way to their equivalents in [5], [15]. Their high-level goal is to cooperatively train a ML model network, and do so by minimizing a loss function via distributed optimization. The computation time at each epoch of the learning process at

a generic node $l \in \mathcal{L}$ follows an arbitrary distribution with probability density function (pdf) $\tau_l^k(t)$, which also accounts for the node capability and the performance of the algorithm it runs. Note that, in the most general case, such a pdf depends on the current epoch (k) of the learning process, since the amount of samples used for learning may vary from an epoch to the next one. This reflects the need to exploit all the available data as soon as it becomes available [18], [23], as opposed to training on a fixed number of samples as in more static scenarios. L-nodes are logically connected to form an arbitrary *logical topology*, i.e., a graph where vertices represent L-nodes and edges, hereinafter referred to as L-L edges, represent the logical links connecting them. As exemplified in Fig. 1 (steps 3–4), after every epoch, each L-node sends its gradient data to its neighboring L-nodes on the logical topology, and waits for them to do the same before moving on. The logical topology, i.e., which pairs of L-nodes are neighbors and exchange gradient data, is one of our main decision variables.

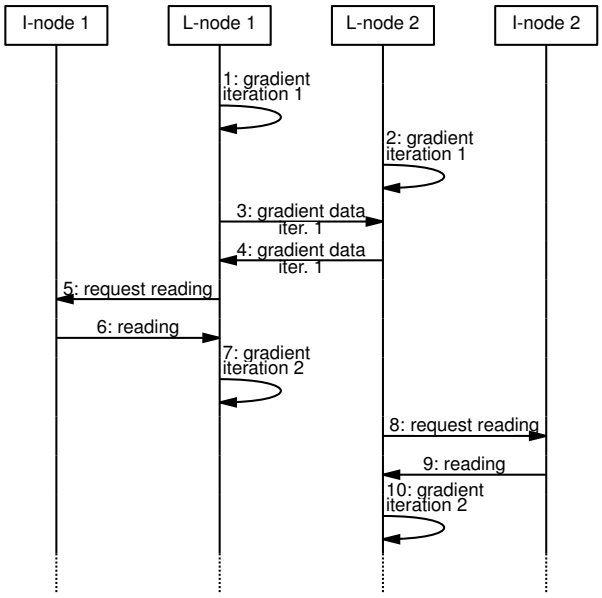

Fig. 1. Scheme of the interactions between L- and I-nodes in a general case.

Each L-node can be logically connected to one or more I-nodes, through the so-called I-L edges. Only I-nodes that are connected to at least one L-node are added to the logical topology. After each epoch of the learning process, an L-node requests data from the I-nodes it is connected to (steps 5 and 8 in Fig. 1), receiving such data (steps 6 and 9 in Fig. 1) after a *sample sending time* modeled by pdf $\rho_i(t)$. In the following, we denote with r_i the expected number of samples provided by I-node i at each epoch. The received samples are used by an L-node l to perform the next epoch, in addition to the data it received in the previous epochs and the number X_l^0 of (offline) samples initially available at l. Note that this behavior is compatible with current, widely deployed applications (e.g., IoT) using publish/subscribe mechanisms, such as MQTT [24], or Zenoh [25], or even the notification mechanisms included in the 3GPP Service Based Architecture [26] of Release 15 and above.

Both L-nodes and I-nodes have per-epoch operational costs, denoted by c_l and c_i, respectively. Moreover, communication between nodes that are neighbors in the logical topology involve additional costs, denoted by $c_{l,l'}$ or $c_{i,l}$ depending on the type of nodes. In general, such costs vary across different pairs of nodes, which also account for the fact that a logical link may correspond to multiple physical ones, hence, entail a higher cost due to energy and/or infrastructure payments.

Modeling real-world supervised ML tasks. As mentioned, our model can describe a wide range of real-world ML tasks, falling in the category of *supervised learning*, for which a ground truth is available. The most prominent examples of supervised learning tasks are classification (where the quantity to predict is discrete, e.g., whether or not a given transaction is fraudulent) and regression (where the quantity to predict is continuous).

In a distributed setting, supervised learning can be performed in two main modes:

- *distributed learning with static data*, where no new data arrive during the learning process. In this case, there are no I-nodes, and each L-node learns from its X_l^0 initial samples, as well as the gradient data from the other L-nodes;
- *active learning* [17], where new samples can be collected from data sources (e.g., sensors) during the learning process so as to improve the learning quality. In this case, the network topology includes both L- and I-nodes.

Importantly, our model can also capture *federated learning* [5], [6], [27], an emerging paradigm whereby different devices (e.g., smartphones) cooperatively train a model without sharing (potentially sensitive) data. In this case, each device is modeled as an L-node; if, in the specific scenario at hand, devices collect or generate additional information while learning, an I-node per device is added, only connected to the corresponding L-node.

For all tasks and approaches, our model can capture the cases where the communication between nodes happens in a peer-to-peer fashion [4], [15], as well as those when it is mediated by a *parameter server*, also known as *broker* [5], [13], [27]. In the latter case, the logical topology created by the L-nodes is fully connected.

IV. Problem Formulation and Approach

Our decisions concern which nodes' interactions should be enabled, and the number of epochs to execute during the learning process. We thus define the following decision variables:

- the set of binary variables $p(l,l') \in \{0,1\}$, expressing whether L-nodes l and l' cooperate during learning;
- the set of binary variables $q(i,l) \in \{0,1\}$, expressing whether L-node $l \in \mathcal{L}$ obtains samples from I-node $i \in \mathcal{I}$;
- the total number of epochs, K, to perform so that the learning task meets the desired learning quality and execution time.

For compactness of notation, we will collect the p- and q-variables in matrices $\mathbf{P} = \{p(l,l')\}$ and $\mathbf{Q} = \{q(i,l)\}$,

respectively. Given the decisions $\mathbf{P}$, $\mathbf{Q}$, and K, we can compute the following system performance metrics:

- the expected time required to the system to complete the learning process, denoted by $T^K(\mathbf{P},\mathbf{Q})$;
- the total cost $C^K(\mathbf{P},\mathbf{Q})$, incurred by the system to complete the learning process, which accounts for (i) the cost of the infrastructure required to run the distributed learning, and (ii) the cost of the communication between the involved nodes;
- the (system-wide) learning error $\epsilon^K(\mathbf{P},\mathbf{Q})$ at the end of the learning process (i.e., after K epochs).

Notice that the above error, cost, and learning time may depend upon other quantities, e.g., the number of samples available for training; however, to simplify the notation, we will write explicitly only the dependences on our decision variables K, $\mathbf{P}$ and $\mathbf{Q}$ as their indices. Also, it is important to point out that in general the concrete definition of error ϵ depends on the type of learning task being performed, e.g.,

- for classification tasks, $\epsilon \triangleq 1-\alpha$, where α is the classification accuracy (i.e., the rate of correctly labeled items);
- for regression tasks, $\epsilon \triangleq 1-R^2$, where R^2 is the coefficient of determination [28].

In both cases, $\epsilon = 0$ corresponds to perfect learning, while larger ϵ values identify worse learning quality, i.e., higher error. In the remainder of the paper, we use *learning error* or *learning quality* when referring to generic machine learning, and more precise terms (e.g., *accuracy* for classification) when discussing specific learning tasks.

Our objective is to minimize the total cost, while ensuring that the final learning error does not exceed the limit $\epsilon^{\max}$, i.e., $\epsilon^K(\mathbf{P},\mathbf{Q}) \leq \epsilon^{\max}$, and the learning is completed within the target time, i.e., $T^K(\mathbf{P},\mathbf{Q}) \leq T^{\max}$. The problem can then be synthetically formulated as:

$$\min_{\mathbf{P},\mathbf{Q},K} C^K(\mathbf{P},\mathbf{Q}), \tag{1}$$

$$\text{s.t.} \min\left\{\frac{\epsilon^{\max}}{\epsilon^K(\mathbf{P},\mathbf{Q})}, \frac{T^{\max}}{T^K(\mathbf{P},\mathbf{Q})}\right\} \geq 1. \tag{2}$$

The problem is combinatorial in nature and includes a large number of binary variables (the elements of matrices $\mathbf{P}$ and $\mathbf{Q}$). This makes it very hard to solve, even without considering the complexity of computing the quantities $C^K(\mathbf{P},\mathbf{Q})$, $\epsilon^K(\mathbf{P},\mathbf{Q})$, and $T^K(\mathbf{P},\mathbf{Q})$. Specifically, we prove in Sec. VI that the problem is NP hard.

Remarkably, in spite of the problem complexity, we can design an efficient and provably effective solution strategy. We do so by first characterizing the system performance as functions of the problem decision variables (Sec. V), and then showing that the problem in (1) and (2) is *submodular* (Sec. VI). Leveraging this result, we can devise the Double-Climb algorithm (Sec. VII), which has cubic worst-case time complexity and proves to be $1+1/|\mathcal{I}|$ competitive.

V. Characterizing the Behavior of the Learning Process

In order to make our decisions, i.e., to choose the best values for the $\mathbf{P}$ and $\mathbf{Q}$ matrices, we need to understand their impact on the learning behavior, e.g., how the learning quality evolves across epochs. In spite of its importance, and the vast quantity of research devoted to it, the goal of fully characterizing a learning process has not yet been achieved. Indeed, as reported in [29], the learning process can best be described as *empirically predictable*. In other words, (i) learning tasks consistently behave according to the same laws, but (ii) the parameters of such laws depend upon the concrete learning task at hand (e.g., the selected neural network architecture and the data used for training). In this section, we describe how to characterize the learning accuracy (Sec. V-A), the time it takes (Sec. V-B), and the associated cost (Sec. V-C).

A. Learning accuracy

One of the main metrics in our problem is the learning quality, or, equivalently, error ϵ, and how it changes according to (i) the number of epochs being performed, (ii) the connectivity among L-nodes, and (iii) the connectivity between I- and L-nodes. Concerning the first two aspects, [4], [15] have derived a square-root behavior, which can be expressed as:

$$\epsilon^K = a_1 + \frac{a_2}{\sqrt{K\gamma}}$$

where K is the number of epochs performed, and γ is the spectral gap[1] of the graph describing the cooperation among L-nodes. Notice that such a result has been proven without reference to a specific dataset or neural network architecture; these elements are accounted for through the a_1 and a_2 coefficients.

Then let us define X as the number of available samples, averaged over epochs and learning nodes. The relationship between the average size X of local datasets and the learning quality is a case of "empirical predictability": in spite of the lack of theoretical results explaining such a behavior, all measurement works we have surveyed [3], [17], [30]–[32], as well as our own experiments, have invariably found a logarithmic law, i.e.,

$$\epsilon^K \propto \log\left(a_3 + X\right).$$

Combining the two above expressions, we can write:

$$\epsilon^K = c_1 + \frac{c_2 \log(c_3 + X)}{\sqrt{K\gamma}}. \tag{3}$$

In terms of our decision variables $\mathbf{P}$ and $\mathbf{Q}$, γ is the difference between the first and second eigenvalues of matrix $\mathbf{P}$, i.e.,

$$\gamma = |\text{eig}_1(\mathbf{P})| - |\text{eig}_2(\mathbf{P})|,$$

while the size X of local datasets can be written as:

$$X = \frac{1}{K|\mathcal{L}|}\sum_{l\in\mathcal{L}}\sum_{k=1}^{K}\left(X_l^0 + \sum_{i\in\mathcal{I}} kr_i q(i,l)\right),$$

where $q(i,l) \in \{0,1\}$ is the element of $\mathbf{Q}$ describing whether I-node i is connected with L-node l. Notice that, by using expected values, we are able to write (3) using deterministic,

[1]The spectral gap of a graph is the difference between the moduli of the two largest eigenvalues of its adjacency matrix.

known quantities, in spite of the fact that the underlying process is stochastic in nature.

The generic law in (3) describes, as confirmed by overwhelming evidence [3]–[5], [15], [17], [29]–[32] a very wide set of ML tasks in a very large set of applications. However, the concrete values of coefficients c_1–c_3 depend upon the concrete learning task at hand, including the DNN architecture and dataset being used. As a consequence, a small-scale *profiling* of the selected DNN and dataset is necessary, in order to establish the c_1–c_3 coefficients; afterwards, our system model and the solution strategy described in Sec. VII can be leveraged to optimize the actual, large-scale learning. Such an approach has been used in [29], and successfully validated over multiple learning tasks, models, datasets, and applications, including speech recognition using LSTM networks, image classification with convolutional networks, and human attention model with recurrent networks.

Importantly, once the concrete learning task to perform is known and the profiling phase has been completed, the exact values of all the quantities needed to compute (3) are known, i.e., such values are known parameters of our problem (as opposed to random variables).

B. Learning time

We now consider that the total number of epochs K, the pdfs $\rho_i(t)$ of the sample generation time at each I-node i, and the pdfs $\tau_l^k(t)$ of the computation time of each L-node l at epoch k are given. Recall that $\tau_l^k(t)$ depends on k, as the presence of I-nodes in our system model implies that the computation time distribution must account for the quantity X_l^k of available data at L-node l and epoch k. In view of the fact that the computation time of DNNs grows linearly [33] with the quantity of data, we can write:

$$\tau_l^k(t) = \frac{X_l^k}{X^0}\tau_l^0(t)\,. \tag{4}$$

Notice how the linear relationship in (4) is consistent with real-world measurements [29], theoretical studies [5], [15], and the intuition that, especially when data is processed in (mini-) batches, processing twice the data requires twice the effort.

Also, we define the sets $\mathcal{I}_l = \{i \in \mathcal{I}\colon q(i,l) = 1\}$ and $\mathcal{L}_l = \{l' \in \mathcal{L}\colon p(l,l') = 1\}$ of I-nodes and L-nodes (resp.) each L-node is connected with. Our goal is to compute $T^K(\mathbf{P},\mathbf{Q})$, i.e., the total time required to complete the whole learning process.

As highlighted in Fig. 1, at every epoch each L-node must perform the following steps:

- wait for the information coming from the I-nodes $i \in \mathcal{I}_l$;
- perform its own gradient computation;
- wait for the gradient data coming from the other L-nodes $l' \in \mathcal{L}_l$ it is cooperating with.

The first step is complete when *all* nodes in $\mathcal{I}_l$ send their samples. Recalling that each I-node has a sample generation time distributed with pdf $\rho_i(t)$, we can derive the cumulative distribution function (CDF) of the maximum of a set of independent random variables as the product of individual CDFs $R_i(t)$, i.e., $\prod_{i\in\mathcal{I}_l} R_i(t)$. Once all data arrive, l can perform its own gradient computation, whose duration is distributed according to pdf $\tau_l^k(t)$. Recalling that the pdf of the sum of two independent random variables is the convolution of individual pdfs, we can write: $h_l^k(t) = \tau_l^k(t) * \frac{\mathrm{d}(\prod_{i\in\mathcal{I}_l} R_i(t))}{\mathrm{d}t}$.

For the system as a whole to move to the next epoch, all L-nodes must have received the gradient data they need. This, in turn, requires the slowest L-node to have obtained its information and have performed the computation. Working again with CDFs, the time taken by such a node is distributed according to: $H^k(t) = \prod_{l\in\mathcal{L}} H_l^k(t)$, where $H_l^k(t)$ denotes the CDF of the time to complete epoch k at L-node l. By letting $h^k(t) = \frac{\mathrm{d}H^k(t)}{\mathrm{d}t}$, the expected duration of the learning process is then given by:

$$T^K(\mathbf{P},\mathbf{Q}) = \sum_{k=1}^{K}\int_0^\infty x h^k(t)dt.$$

A numerical example. Fig. 2 exemplifies our methodology in a case where both the I-node sample generation times and the L-node computation times are uniformly distributed; specifically, $\rho_i(t) \sim \mathcal{U}(0.1, 1.9)$ and $\tau_l^k(t) \sim \mathcal{U}(1.35, 1.65)$. Furthermore, there are $|\mathcal{L}| = 10$ L-nodes, each connected to $|\mathcal{I}| = 5$ I-nodes.

We begin from the blue line in the plot, representing $\rho_i(t)$. To obtain the pdf of the sample generation time of the slowest I-node, we have to integrate $\rho_i(t)$ (obtaining $R_i(t)$, a ramp-like function), then raise it to the $|\mathcal{I}|$-th power (obtaining a 5th-degree polynomial), and finally derive it, obtaining the 4th-degree polynomial shown by the red line in Fig. 2.

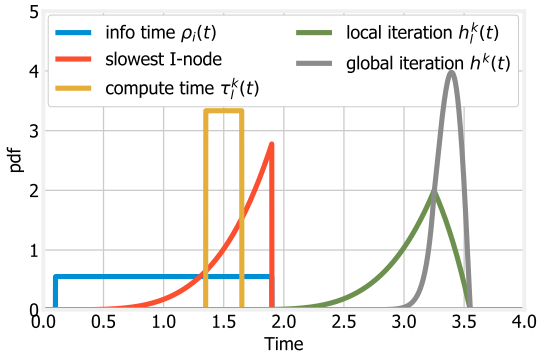

Fig. 2. Toy scenario with $|\mathcal{L}| = 10$ and $|\mathcal{I}| = 5$ where both I-node sample generation times and L-node computation times are uniformly distributed. Left: pdfs of the I-node generation time $\rho_i(t)$ (blue), of the time required by the slowest I-node (red) and of the compute time $\tau_l^k(t)$ (yellow). Right: pdfs of the time taken by local (green) and global (gray) epochs.

We next perform the convolution between the latter pdf and $\tau_l^k(t)$, represented by the yellow line in the plot. The result is $h_l^k(t)$, represented by the green line in Fig. 2. The last step consists in computing the distribution of the time taken by the whole learning epoch, hence, by the slowest L-node. Integrating $h_l^k(t)$, we obtain $H_l^k(t)$, which we raise to the $|\mathcal{L}| = 10$-th power, and then derive it, obtaining the pdf $h^k(t)$ shown by the gray curve in Fig. 2.

Fig. 3 presents two Gantt charts showing the activity of I- and L-nodes (blue and yellow bars, respectively) over three epochs. The top plot refers to the case where all L-nodes use

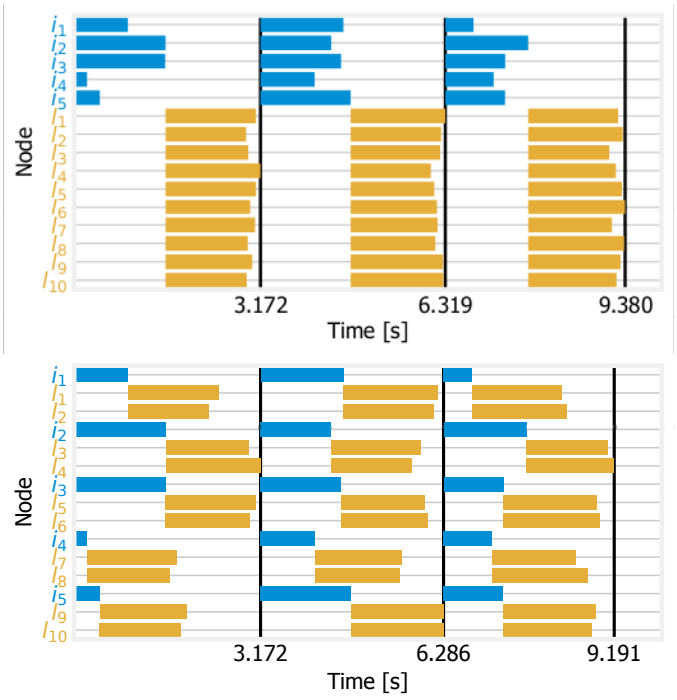

Fig. 3. Toy scenario with $|\mathcal{L}| = 10$ and $|\mathcal{I}| = 5$ where both I-node sample generation times and L-node computation times are uniformly distributed. Gantt charts show the activity of I- and L-nodes (blue and yellow bars, respectively), when all L-nodes use all I-nodes (top) and when each L-node only uses one I-node (bottom). Black vertical lines mark the end time of each epoch.

all I-nodes, as in Fig. 2: accordingly, it is possible to observe how all yellow bars start at the same time, after all blue bars finish. In the bottom plot, we move to a scenario where each L-node uses only one I-node (specifically, l_1 and l_2 use i_1, l_3 and l_4 use i_2, etc.). This allows many L-nodes to begin their work early, however, as we can see from the vertical bars, the overall decrease in epoch duration is modest – even absent in the first epoch, where the slowest L-node had to wait for the slowest I-node. This is due to the fact that, in this toy example, both the I-node sample generation times and the L-node computation times are uniformly distributed: indeed, as also [15] reports, pruning I-L links is most beneficial when computation and generation times follow more skewed distributions. At last, we note that the limited reduction in the learning time shown in the figure is due to the small size of the scenario; nonetheless, such a gain validates our approach.

Closed-form expression for special cases. The methodology outlined above does not require any assumption on the $\tau_l^k(t)$ and $\rho_i(t)$ distributions, nor on the logical links between nodes, and the computations it requires can always be performed numerically. However, closed-form expressions are available in relevant special cases. Let us focus on a scenario where (i) all nodes are connected to each other, and (ii) the computation and the sample generation times are i.i.d. and exponentially distributed with parameter λ_L^k and λ_I, respectively. Such a scenario is sufficiently simple to result in manageable expressions, but also sufficiently complex to allow us to properly illustrate the power of our methodology.

The computation time T^K can be written as:

$$T^K = -\sum_{k=1}^{K} \sum_{\substack{\mathcal{A}\subset\mathbb{N}:\\ |\mathcal{A}|=|\mathcal{I}|+2\\ \sum_{a\in\mathcal{A}} a=|\mathcal{L}|}} \binom{|\mathcal{L}|}{\mathcal{A}} \frac{\prod_{w=1}^{|\mathcal{I}|+2} (A^k(\mathcal{A},w))^{a_w}}{\lambda_I \sum_{w=1}^{|\mathcal{I}|} w a_w + \lambda_L^k a_{|\mathcal{I}|+2}}.$$

In the above expression, the sum over k accounts for all epoch, $k = 1, \ldots, K$. The inner sum comes from the multinomial expansion [34] of a sum of $|\mathcal{I}|+2$ terms (one for each I-node, one for the L-node connected to them, and one representing the coefficient) raised to the $|\mathcal{L}|$-th power, where each term is a polynomial (see also the expression of $h_l^k(t)$). Therefore, the inner summation is over all sets $\mathcal{A}$ of natural numbers such that their size is $|\mathcal{I}|+2$ and their sum is $|\mathcal{L}|$, and $\binom{|\mathcal{L}|}{\mathcal{A}} = \frac{|\mathcal{L}|!}{\prod_{a\in\mathcal{A}} a!}$ is the multinomial coefficient. The term $A^k(\mathcal{A}, w)$ associated with the w-th element of each set $\mathcal{A}$ is:

$$A^k(\mathcal{A},w) = \begin{cases} \sum_{z=1}^{|\mathcal{I}|} \binom{|\mathcal{I}|}{z} (-1)^{z+1}, & \text{if } w{=}|\mathcal{I}|+1 \\ \sum_{z=1}^{|\mathcal{I}|} \binom{|\mathcal{I}|}{z} (-1)^{z+1} \frac{z\lambda_I}{\lambda_L^k - w\lambda_I}, & \text{if } w{=}|\mathcal{A}| \\ \binom{|\mathcal{I}|}{w} (-1)^{w+1} \frac{\lambda_L^k}{w\lambda_I - \lambda_L^k}, & \text{otherwise.} \end{cases}$$

A closed-form expression for the expected duration of the learning process can also be obtained when each L-node receives information from all I-nodes, and the I-nodes' sample generation times and the L-nodes's computation times are i.i.d. and uniformly distributed over (a_I, b_I) and (a_L^k, b_L^k), respectively. For simplicity and without loss of generality, let us assume $a_L^k \le a_I \le b_I \le b_L^k, \forall k$; then, we have:

$$\begin{aligned} T^K = &\sum_{k=1}^{K} \sum_{\substack{\mathcal{A}\subset\mathbb{N}:\\ |\mathcal{A}|=|\mathcal{I}|+2\\ \sum_{a\in\mathcal{A}} a=|\mathcal{L}|}} \binom{|\mathcal{L}|}{\mathcal{A}} \frac{\sum_{w=1}^{|\mathcal{I}|+1} w a_w}{\sum_{w=1}^{|\mathcal{I}|+1} w a_w + 1} \times \\ &\times \Bigg[\prod_{w=1}^{|\mathcal{I}|+2} (A_1^k(\mathcal{A},w))^{a_w} \left(Z_1^{\sum_{w=1}^{|\mathcal{I}|+1} w a_w + 1} - Z_2^{\sum_{w=1}^{|\mathcal{I}|+1} w a_w + 1} \right) \\ &+ \prod_{w=1}^{|\mathcal{I}|+2} (A_2^k(\mathcal{A},w))^{a_w} \left(Z_3^{\sum_{w=1}^{|\mathcal{I}|+1} w a_w + 1} - Z_4^{\sum_{w=1}^{|\mathcal{I}|+1} w a_w + 1} \right) \\ &+ \prod_{w=1}^{|\mathcal{I}|+2} (A_3^k(\mathcal{A},w))^{a_w} \left(Z_5^{\sum_{w=1}^{|\mathcal{I}|+1} w a_w + 1} - Z_6^{\sum_{w=1}^{|\mathcal{I}|+1} w a_w + 1} \right) \Bigg] \end{aligned}$$

where $Z_1 = a_L^k + b_I$, $Z_2 = a_L^k + a_I$, $Z_3 = b_L^k + a_I$, $Z_4 = a_L^k + b_I$, $Z_5 = b_L^k + b_I$, $Z_6 = b_L^k + a_I$. As in the previous case, the above expression comes from the multinomial expansion [34], and, after some algebra, one can obtain the terms $A_1^k(\mathcal{A}, w)$, $A_2^k(\mathcal{A}, w)$, and $A_3^k(\mathcal{A}, w)$ associated with the w-th element of each set $\mathcal{A}$, as:

$$A_1^k(\mathcal{A},w) = \begin{cases} \frac{(-2a_I)^{|\mathcal{I}|} - (a_L^k - a_I)^{|\mathcal{I}|}}{(b_I - a_I)^{|\mathcal{I}|}(b_L^k - a_L^k)}, & \text{if } w{=}1 \\ \frac{(a_L^k - a_I)^{|\mathcal{I}|}(|\mathcal{I}|(a_L^k + a_I) + 2a_I) + (-2a_I)^{|\mathcal{I}|+1}}{(|\mathcal{I}|+1)(b_I - a_I)^{|\mathcal{I}|}(b_L^k - a_L^k)}, & \text{if } w{=}|\mathcal{A}| \\ \frac{\binom{|\mathcal{I}|+1}{w}(-2a_I)^{|\mathcal{I}|+1-w}}{(|\mathcal{I}|+1)(b_I - a_I)^{|\mathcal{I}|}(b_L^k - a_L^k)}, & \text{else.} \end{cases}$$

$$A_2^k(\mathcal{A},w)=\begin{cases} A_1^k(\mathcal{A},|\mathcal{A}|)+\sum\limits_{z=1}^{|\mathcal{I}|+1} A_1^k(\mathcal{A},z)(a_L^k+b_I)^z+ \\ +\frac{\left(a_L^k-a_I\right)^{|\mathcal{I}|+1}-\left(a_L^k+b_I-2a_I\right)^{|\mathcal{I}|+1}}{(|\mathcal{I}|+1)(b_I-a_I)^{|\mathcal{I}|}(b_L^k-a_L^k)}+ \\ +\frac{(b_I+a_I)^{|\mathcal{I}|+1}-(2a_I)^{|\mathcal{I}|+1}}{(|\mathcal{I}|+1)(b_I-a_I)^{|\mathcal{I}|}(b_L^k-a_L^k)}, & \text{if } w{=}|\mathcal{A}| \\ \frac{-\binom{|\mathcal{I}|+1}{w}((-b_I-a_I)^{|\mathcal{I}|+1-w}-(-2a_I)^{|\mathcal{I}|+1-w})}{(|\mathcal{I}|+1)(b_I-a_I)^{|\mathcal{I}|}(b_L^k-a_L^k)}, & \text{else.} \end{cases}$$

$$A_3^k(\mathcal{A},w)=\begin{cases} \frac{(b_L^k-a_I)^{|\mathcal{I}|}-(b_I+a_I)^{|\mathcal{I}|}(-1)^{|\mathcal{I}|}}{(b_I-a_I)^{|\mathcal{I}|}(b_L^k-a_L^k)}, & \text{if } w{=}1 \\ A_2^k(\mathcal{A},|\mathcal{A}|)+\sum\limits_{z=1}^{|\mathcal{I}|+1} A_2^k(\mathcal{A},z)(b_L^k+a_I)^z- \\ -\frac{(|\mathcal{I}|+1)\left(b_L^k-a_I\right)^{|\mathcal{I}|}(b_L^k+a_I)}{(|\mathcal{I}|+1)(b_I-a_I)^{|\mathcal{I}|}(b_L^k-a_L^k)}+ \\ +\frac{\left(-b_L^k-a_I\right)^{|\mathcal{I}|+1}-\left(b_L^k-b_I\right)^{|\mathcal{I}|+1}}{(|\mathcal{I}|+1)(b_I-a_I)^{|\mathcal{I}|}(b_L^k-a_L^k)}, & \text{if } w{=}|\mathcal{A}| \\ \frac{\binom{|\mathcal{I}|+1}{w}(-b_I-a_I)^{|\mathcal{I}|+1-w}}{(|\mathcal{I}|+1)(b_I-a_I)^{|\mathcal{I}|}(b_L^k-a_L^k)}, & \text{else.} \end{cases}$$

Intuitively, the three different terms $A_*^k(\mathcal{A},w)$ are due to the convolution of the pdfs, which results in a piece-wise function (see also the expression of $h_l^k(t)$). The support of the different pieces of the function are as follows: $\left[a_L^k+a_I, a_L^k+b_I\right)$ for the first piece where only one pdf is active, $\left[a_L^k+b_I, b_L^k+a_I\right]$ for the second piece where both pdfs are active and overlap, and $\left(b_L^k+a_I, b_L^k+b_I\right]$ for the third piece where only the other pdf is active.

C. Learning cost

We define the per-epoch cost as the sum of operational and communication costs of the L- and I- nodes contributing to each epoch, i.e.,

$$C(\mathbf{P},\mathbf{Q}) = \sum_{l\in\mathcal{L}}\left(c_l+\sum_{l'\in\mathcal{L}} c_{l,l'}p(l,l')+\sum_{i\in\mathcal{I}} c_{i,l}q(i,l)\right) + \sum_{i\in\mathcal{I}} c_i \mathbb{1}_{\exists q(i,l)>0}\,. \quad (5)$$

Then, we can write the total learning cost over the K epochs as $C^K(\mathbf{P},\mathbf{Q}) = K\cdot C(\mathbf{P},\mathbf{Q})$.

D. Number of epochs

The number K of epochs needed to reach the target error $\epsilon^{\max}$ depends on two factors. The first is the quantity of available training data: the more data is available, the more the learning quality improves at each epoch. The second is the level of cooperation between L-nodes: the more nodes cooperate, the higher the quality achieved at each epoch. From (3), we get:

$$K \propto \frac{\log^2 X}{\gamma\left(\epsilon^{\max}\right)^2}.$$

On the one hand, a high degree of L-nodes makes the learning process faster, as convergence requires fewer epochs; on the other hand, each epoch takes longer to complete as there are more nodes to wait for.

VI. Problem Analysis

We first prove that the problem at hand, formulated in Sec. IV, is NP hard. On the positive side, we also show that the problem objective function is submodular and non-decreasing, while the constraint is submodular and exhibits only one maximum (we prove the latter part separately for I-L and L-L edges).

Lemma 1. *The problem of optimally configuring the system for an ML task, expressed in (1) and (2), is NP hard.*

Proof: The proof can be obtained via a reduction from the knapsack problem [35], a combinatorial optimization problem where a set of $\mathcal{S}$ items (with cardinality S) is given, each of them associated with a weight ω_s and a value ν_s. The goal is to select a subset of items with maximum total value and total weight less or equal to a maximum given capacity, Ω. Our reduction maps any given instance of the knapsack problem to a simpler, *special-case* instance of our own, as set forth next.

The sets of L-nodes and I-nodes are, respectively, $\mathcal{L}=\{l_1\}$ and $\mathcal{I}=\{i_1\ldots i_S\}$, i.e., there is only one L-node and as many I-nodes as the number of items in the knapsack problem. Further, the L-node is connected with all the I-nodes. We also set the number of epochs to an arbitrary number $\hat{K}>0$, and the number of samples generated by each I-node to an arbitrary number $r>0$.

Given the above, $\mathbf{P}$ is fixed and the decisions concern only $\mathbf{Q}$, which is now a vector with elements $q(i_s,l_1)$, mapping into the x_s variables in the knapsack problem. Specifically, we activate edge (i_s,l_1) in our problem if and only if $x_s=1$, i.e., $q(i_s,l_1)\leftarrow x_s$. Furthermore, we map edge costs in our problem into item weights in the knapsack problem. In particular, let ν_s correspond to the opposite of the link cost c_{i_s,l_1}, then we have a perfect correspondence between the objective of the knapsack problem and that in (1).

Next, we need to map the capacity constraint in the knapsack problem to constraint (2). To this end, we first set $T^{\max}\leftarrow\infty$. Then, given that $\mathbf{P}$ is fixed, $\gamma=1$, and the L-node can only receive data from any number of I-nodes, the amount of data received by L-node l_1 in each epoch is r or 0, depending on the value of x_s. A correspondence between the constraint in the knapsack problem and that in our problem is then established by fixing $\epsilon^{\max}\leftarrow\Omega$, and setting the c_1–c_3 coefficients in (3) in such a way that setting x_s to 1 results in an increase of learning quality of ω_s, i.e.,

$$\frac{\log(c_3+X+X_s)-\log(c_3+X)}{\sqrt{K}}c_2=\omega_s,$$

where $X_s=\frac{r(K+1)}{2}$ is the increase in the expected number of samples obtained by using I-node i_s. In other words, the equation above represents the increase in the value of learning quality (see (3)) obtained by activating i_s; we thus set the parameters of our problem so that the above increase results to be equal to the weight ω_s assigned to s in the knapsack problem.

Last, we need the reduction to take (at most) polynomial time. In our case, it is straightforward to see that the mapping

takes linear time, namely $O(|\mathcal{L}| + |\mathcal{I}|)$, hence, the condition is fulfilled.

In summary, any instance of an NP-hard problem can be transformed into a special-case instance of our own, which proves the thesis. ∎

In spite of its complexity, the problem of minimizing (1) subject to constraint (2) presents several features that can be exploited to solve it efficiently and effectively. Specifically, both the objective in (1) and the constraint in (2) are submodular (intuitively, the set-wise equivalent of convex [36]). Submodular optimization problems can often be solved with polynomial- or even linear-time greedy algorithms, with very good, even constant, competitive ratios [37]. Notice how both the results in [36] and [37] are presented with reference to abstract, generic problems where the goal is to select some elements from set $\mathcal{X}$, with no reference (hence, no reliance) on specific scenarios.

Let us indicate with $f(\mathcal{Y})$ the *objective function* in (1), and with $g(\mathcal{Y})$ the *constraint* in (2). In our case, the set $\mathcal{X}$ of elements to choose from is given by $\mathcal{X} = \mathcal{L} \times \mathcal{L} \cup \mathcal{L} \times \mathcal{I}$, i.e., the set of possible I-L and L-L edges we can create, and $\mathcal{Y}$ is the subset of actually selected edges.

The objective $f(\mathcal{Y})$ and constraint $g(\mathcal{Y})$ of our problem have several interesting and useful properties. Concerning the former, it is possible to prove the following result.

Property 1. *The objective function in (1) is submodular and non-decreasing.*

Proof: Let $j = (a, b)$ be an edge in our logical topology graph, with $a \in \mathcal{L}$ and $b \in \mathcal{L} \cup \mathcal{I}$; let $\mathcal{S} \subset \mathcal{X}$ be the set of currently selected edges. By adding j, we incur the per-edge communication cost $c_{a,b}$; also, we may incur per-node operational costs c_a or c_b, depending on whether or not there are already edges in $\mathcal{S}$ with a or b as endpoints. Similar arguments hold for the cost of adding j to $\mathcal{T} \supset \mathcal{S}$. Thus,

$$\begin{aligned} f(\mathcal{S} \cup \{j\}) - f(\mathcal{S}) &= c_{a,b} + c_a \mathbb{1}_{a \notin \mathcal{S}} + c_b \mathbb{1}_{b \notin \mathcal{S}} \\ f(\mathcal{T} \cup \{j\}) - f(\mathcal{T}) &= c_{a,b} + c_a \mathbb{1}_{a \notin \mathcal{T}} + c_b \mathbb{1}_{b \notin \mathcal{T}}. \end{aligned}$$

Since $\mathcal{S}$ is a subset of $\mathcal{T}$, it also holds that $\mathbb{1}_{a \notin \mathcal{S}} \geq \mathbb{1}_{a \notin \mathcal{T}}$ and $\mathbb{1}_{b \notin \mathcal{S}} \geq \mathbb{1}_{b \notin \mathcal{T}}$, from which it follows that $f(\mathcal{S} \cup \{j\}) - f(\mathcal{S}) \geq f(\mathcal{T} \cup \{j\}) - f(\mathcal{T})$, i.e., the very definition of submodularity [36]. The fact that (1) is non-decreasing trivially comes from the observation that, as more I-L or L-L edges are added, the cost always increases. ∎

As for the constraint, the analysis is a little more complex, and we perform it separately for I-L and L-L edges, proving first Property 2 concerning the former, and then Proposition 1 concerning the latter.

For simplicity of notation, we drop the dependency on $\mathbf{P}$ and $\mathbf{Q}$ while presenting our derivations.

Property 2. *When the choices are limited to I-L edges, i.e., $\mathcal{X} = \mathcal{L} \times \mathcal{I}$, then the constraint in (2) is submodular and has exactly one maximum.*

Proof: Let us study the two parts of the constraint (2) separately, writing $g_1 = \frac{\epsilon^{\max}}{\epsilon^K}$, $g_2 = \frac{T^{\max}}{T^K}$, and $g(\mathcal{Y}) = \min\{g_1, g_2\}$, as exemplified in Fig. 4. From (3), $g_1 = \frac{\epsilon^{\max}\sqrt{\gamma K}}{c_1\sqrt{\gamma K} + c_2 \log(c_3 + X)}$, and its second derivative is $\frac{c_2 \epsilon^{\max}\sqrt{\gamma K}}{(c_3+X)^2\left(c_1\sqrt{\gamma K} + c_2 \log(c_3+X)\right)^2} + \frac{2{c_2}^2 \epsilon^{\max}\sqrt{\gamma K}}{(c_3+X)^2\left(c_1\sqrt{\gamma k} + c_2 \log(c_3+X)\right)^3}$. Such a derivative is always positive, hence, g_1 is submodular.

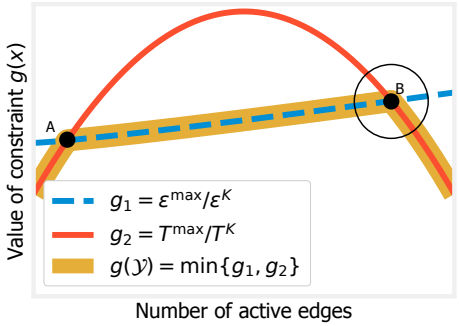

Fig. 4. Qualitative example of the constraint in (2) and its components.

The behavior of g_2 is more complex: we know from (3) that the number of epochs decreases as X increases, according to an inverse-log law. Also, as shown in Sec. V, $\tau_l^k(t)$ and $\frac{\mathrm{d}H^k(t)}{\mathrm{d}t}$ are proportional to X_l^k and $\prod_{l \in \mathcal{I}} X_l^k$, respectively. Thus, T^K is proportional to K and $\prod_{l \in \mathcal{I}} X_l^k$. Replacing K with (3), we get that T^K behaves like $\frac{\prod_{l \in \mathcal{I}} X_l^k}{\log X}$, i.e., it can be shown that it decreases until it reaches a minimum, and then increases. It follows that $g_2 = \frac{T^{\max}}{T^K}$ is concave, hence, submodular.

Looking now at $g(\mathcal{Y})$, the minimum of two submodular functions is not guaranteed to be submodular in general; however, since g_1 is not only submodular but also monotonically increasing, the submodularity of g_2 also implies that $g(\mathcal{Y})$ as a whole is submodular [36]. Next, consider the maximum of $g(\mathcal{Y})$, with the latter being equal to $\min\{g_1, g_2\}$. As exemplified in Fig. 4, we know that g_1 starts from a value close to $\epsilon^{\max}$ and then monotonically increases towards infinity, while g_2 starts with a small value, increases until it has a global maximum, and then decreases again. If g_2 is always smaller than g_1, then $g(\mathcal{Y}) = g_2$ has exactly one global maximum, consistently with the hypothesis. If they cross (as in Fig. 4)), they do so in exactly two points, say A and B, such that the maximum of g_2 is between A and B. Then, the following holds: (i) before A, $g(\mathcal{Y}) = g_2$, which is increasing before its maximum; (ii) between A and B, $g(\mathcal{Y}) = g_1$, which is always increasing; (iii) after B, $g(\mathcal{Y}) = g_2$ and, since we are after its maximum, $g(\mathcal{Y})$ is decreasing – hence, B is $g(\mathcal{Y})$'s only maximum. Therefore, in all cases $g(\mathcal{Y})$ is submodular and has exactly one maximum, and, until such a maximum is reached, $g(\mathcal{Y})$ is also monotonically non-decreasing. ∎

As for L-L edges, their influence on the learning process can be quantified by studying the graph they form. Specifically, we are able to state the following result concerning regular graphs:

Proposition 1. *When the choices are limited to sets of L-L edges such that the graph created by L-nodes is uniform, then the constraint (2) is submodular and has exactly one maximum.*

Proof: The arguments in support of Proposition 1 can be summarized as follows: 1) the error reached after a given number K of epoch is proportional to $1/\gamma$ [15, Eq. (7)]; 2)

the learning time is proportional to $1/\gamma$ [15, Eq. (18)]; 3) for random regular graphs, the relationship between the spectral gap and the graph degree has been shown [38], [39] to follow a square-root law, which is concave. Recalling that concavity is the continuous equivalent of submodularity, the first part of the proposition follows. The second part follows from the fact that (2) is the minimum between a monotonic function (as we add more L-L edges, the error decreases) and a function with at most one maximum (the inverse of the learning time, which decreases until an optimal degree is reached and then increases, as shown in [15]). ■

Combining Property 2 and Proposition 1, we can now prove the following result:

Corollary 1. *When the graph created by L-nodes is uniform, constraint (2) is submodular and has exactly one maximum.*

Proof: The possible actions are either adding an I-L edge, or an L-L one. As per (respectively) Property 2 and Proposition 1, both actions preserve the submodularity property, and result in a function with exactly one maximum. ■

VII. The DoubleClimb Algorithm

We now seek to solve the problem stated in Sec. IV, i.e., determining the $\mathbf{P}$, $\mathbf{Q}$ and K resulting in the lowest cost (1) subject to the constraint in (2), in a practical and efficient way. To this end, we first extend existing results on the performance of greedy algorithms when optimizing submodular problems, in Sec. VII-A. Based on such results, we present our own DoubleClimb algorithm in Sec. VII-B, and analyze its properties in Sec. VII-C.

A. Greedy solutions to submodular problems

Let us consider Alg. 1, which solves submodular problems with non-decreasing objective and constraints. More formally, it selects a subset $\mathcal{S} \subseteq \mathcal{X}$ of elements subject to a submodular non-decreasing constraint $g(\mathcal{S}) \geq 1$, while minimizing a submodular non-decreasing cost function $f(\mathcal{S})$. At every iteration, Line 3 selects the element minimizing the cost to benefit ratio $\frac{f(\mathcal{S}\cup\{j\})-f(\mathcal{S})}{g(\mathcal{S}\cup\{j\})-g(\mathcal{S})}$; such an element is then added to $\mathcal{S}$ (Line 4). As shown in [40, Thm. 4.7], Alg. 1 is $1+\frac{1}{|\mathcal{X}|}$-competitive. However, the original proof requires both the objective and the constraint to be submodular and non-decreasing. In our case, Property 2 and Proposition 1 prove *weaker* properties, in that our constraint is not guaranteed to be non decreasing, as in Fig. 4; therefore, the result in [40] cannot immediately be applied to our problem.

None the less, it is possible to prove that a less restrictive condition than being non-decreasing, namely, having only one maximum, is sufficient for the result to hold:

Algorithm 1 Greedy algorithm for submodular problems

1: $\mathcal{S} \leftarrow \emptyset$
2: **while** $g(\mathcal{S}) \geq c$ **do**
3: $\quad j^* \leftarrow \arg\min_{\mathcal{X}\setminus\mathcal{S}} \frac{c_j}{g(\mathcal{S}\cup\{j\})-g(\mathcal{S})}$
4: $\quad \mathcal{S} \leftarrow \mathcal{S} \cup \{j\}$
5: **return** $\mathcal{S}$

Property 3. *If $f(\mathcal{Y})$ is submodular non-decreasing and $g(\mathcal{Y})$ is submodular and has only one maximum, then the above algorithm minimizes $f(\mathcal{Y})$ s.t. $g(\mathcal{Y}) > 0$, with a competitive ratio of $1+\frac{1}{|\mathcal{X}|}$.*

Proof: The property generalizes the results in [40, Thm. 4.7]. The proof therein follows from analyzing the steps of the above algorithm until its convergence, and leveraging the fact that the sequences of marginal cost increases and constraint improvements are (resp.) monotonically non-decreasing and monotonically non-increasing. This is of course true if, as in the original hypotheses, $g(\mathcal{Y})$ is monotonically non-decreasing. However, this also holds if $g(\mathcal{Y})$ has only one maximum, as per the hypothesis of our property. This is because, if the algorithm cannot find a feasible solution before the maximum of $g(\mathcal{Y})$, i.e., as constraints become *closer* to being satisfied, it will also be impossible to find a feasible solution after the maximum, i.e., when constraints will get *farther* from being met. Thus, the sequences of marginal costs and improvements *of the selected elements* of $\mathcal{X}$ have the required behavior. Indeed, the behavior of $g(\mathcal{Y})$ for the non-selected items of $\mathcal{X}$ *has no impact* on the validity of [40, Thm. 4.7], nor of this property. ■

B. Algorithm description

Property 3 implies that the algorithm in Sec. VII-A could efficiently select the L-L links $\mathbf{P}$ and the I-L links $\mathbf{Q}$, i.e., which L-L nodes cooperate with one another and which information they can leverage *if such decisions could be made independently*, without one impacting the other. However, they are clearly interlinked; thus, we propose a more complex solution strategy, called DoubleClimb, which operates as follows.

- First, based on the nodes capabilities defined in Sec. III, DoubleClimb determines $\mathbf{P}$ and $\mathbf{Q}$. It does so by selecting I-L and L-L edges in two nested loops, with L-L edges resulting in a uniform graph [15]. It also selects the most appropriate value of K for each set of selected edges.
- Given such decisions, it computes the system performance characterized in Sec. V, thus yielding the error $\epsilon^K(\mathbf{P},\mathbf{Q})$, the learning time $T^K(\mathbf{P},\mathbf{Q})$, as well as the cost $C^K(\mathbf{P},\mathbf{Q})$.
- It then compares the obtained values for the learning time and error against the limits $\epsilon^{\max}$ and $T^{\max}$, and evaluates whether a sufficiently low cost has been achieved. If so, DoubleClimb returns the problem solution; otherwise, it tries to improve the decisions until the system constraints are met and the cost is further reduced.

Intuitively, we begin with a sparsely connected graph with no L-L and no I-L edges, and then we keep increasing the connectivity until the constraints are satisfied.

The DoubleClimb algorithm is presented in Alg. 2 and detailed below. It begins (Line 1) by setting to zero the degree d_{L} of the subgraph made of L-L edges, and to the empty set the best solution `best_sol`. Then, while $d_{\mathrm{L}} < |\mathcal{L}|$, i.e., while such a subgraph is not a clique, d_{L} is first incremented by one (Line 4), and then the cheapest L-L uniform subgraph of degree d_{L} is chosen in Line 5.

Given such a choice of L-L edges, the algorithm selects the I-L edges essentially in the same way as described in

Algorithm 2 The DoubleClimb algorithm

1: $d_{\mathrm{L}} \leftarrow 0$
2: $\texttt{best\_sol} \leftarrow \emptyset$
3: **while** $d_{\mathrm{L}} < |\mathcal{L}|$ **do**
4: $\quad d_{\mathrm{L}} \leftarrow d_{\mathrm{L}} + 1$
5: $\quad \texttt{ll} \leftarrow$ **cheapest_uniform**(d_{L})
6: $\quad \texttt{il} \leftarrow \emptyset$
7: $\quad$ **while** (2) is not verified $\wedge \texttt{il} \not\equiv \mathcal{I} \times \mathcal{L}$ **do**
8: $\quad\quad i^*, j^* \leftarrow \arg\min_{i,l} \frac{c_{i,l}}{g(\texttt{il}) - g(\texttt{il} \cup \{(i,l)\})}$
9: $\quad\quad \texttt{il} \leftarrow \texttt{il} \cup \{(i^*, j^*)\}$
10: $\quad$ **if** $C^{\text{curr}} < C^{\text{best}}$ **then**
11: $\quad\quad \texttt{best\_sol} \leftarrow \texttt{ll} \cup \texttt{il}$
12: $\quad$ **else if** $C^{\text{curr}}_{\text{LL}} > C^{\text{best}}_{\text{LL}} \wedge C^{\text{curr}}_{\text{IL}} > C^{\text{best}}_{\text{IL}}$ **then**
13: $\quad\quad$ **break**
14: **return** $\texttt{best\_sol}$

Sec. VII-A: for all possible edges, the cost/benefit ratio – i.e., the ratio between the cost of adding the edge and how closer to feasibility the problem becomes by doing so – is computed in Line 8, and the edge associated with the lowest ratio is chosen. The loop continues until either all I-L edges are exhausted, or a feasible solution, satisfying constraint (2), is found (Line 7). In the latter case, the cost of the current solution C^{curr}, computed as per (5), is compared to the one of the best solution found so far (C^{best}); note that, by convention, the cost of the empty set is equal to ∞. If warranted, the best solution is updated (Line 11), otherwise we perform the check in Line 12 to assess whether other solutions should be explored. Indeed, as proven in Proposition 2 below, the submodularity of costs implies that trying higher values of d_{L} does not lead to cheaper solutions.

If neither happens, the next value of d_{L} is tried. After all values of d_{L} are exhausted, the best solution $\texttt{best\_sol}$ is returned in Line 14. If no feasible solution has been found, the problem instance is infeasible and the algorithm returns $\emptyset$.

C. Algorithm analysis

We now prove that Alg. 2 has an excellent competitive ratio as well as low complexity. As first step, we show that the stopping condition in Line 12 is valid, i.e., no solution better than $\texttt{best\_sol}$ is ignored by halting the algorithm when the condition is met.

Proposition 2. *If the condition specified in Line 12 of Alg. 2 is met, then no solution cheaper than* $\texttt{best\_sol}$ *will be found for higher values of* d_L.

Proof: Let $d^{\text{best}}_{\text{L}}$ be the value of d_{L} for which the current best solution was found, and $C^{\text{best}}_{\text{LL}}$ and $C^{\text{best}}_{\text{IL}}$ the corresponding costs for L-L and I-L edges (resp.). At the current iteration, we have $d_{\mathrm{L}} = L^{\text{curr}} > L^{\text{best}}$, and the corresponding costs are $C^{\text{curr}}_{\text{LL}} > C^{\text{best}}_{\text{LL}}$ and $C^{\text{curr}}_{\text{IL}} > C^{\text{best}}_{\text{IL}}$. Let us now consider a future iteration where the value of d_{L} is $d^{\text{next}}_{\text{L}} > d^{\text{curr}}_{\text{L}} > d^{\text{best}}_{\text{L}}$. $C^{\text{next}}_{\text{LL}}$ depends on two effects: if we increase the number of L-L edges, the cost due to L-L edges will increase. However, more L-L edges also imply fewer epochs, thus they may lead to a reduced cost. Since similar observations hold for $C^{\text{next}}_{\text{IL}}$, which effect prevails depends on how strong the benefit of increasing d_{L} is. However, as per the submodularity property (Proposition 1), the benefit of adding L-L edges and I-L edges decreases as d_{L} increases: if moving from $d^{\text{best}}_{\text{L}}$ to $d^{\text{curr}}_{\text{L}}$ actually increased the cost of L-L and I-L edges, it is not possible that moving to $d^{\text{next}}_{\text{L}}$ will provide a better solution. ■

Thanks to Proposition 2 and Property 3, we can now prove our main result about the *competitive ratio* of Alg. 2, i.e., the ratio of the cost of the solution it yields to the one of the optimal solution.

Theorem 1. *Alg. 2 has* $1 + \frac{1}{|\mathcal{I}|}$ *competitive ratio.*

Proof: There are two possible sources of suboptimality, namely, the choice of d_{L} and that of the I-L edges to select. By Proposition 2 and considering that, if the condition in Line 12 is never triggered, all possible values of d_{L} are tried out, the choice of d_{L} is optimal. As for the I-L edges, Line 7–Line 9 of Alg. 2 reflect exactly the same algorithmic steps reported in Sec. VII-A which, as per Property 3, lead to a $1 + \frac{1}{|\mathcal{I}|}$ competitive ratio in our case. ■

Finally, we can prove that Alg. 2 has a very low, namely, cubic *worst-case* computational complexity.

Property 4. *Alg. 2 has a worst-case computational complexity of* $O(|\mathcal{L}|^2|\mathcal{I}|)$.

Proof: From inspection of the nested loops in Alg. 2, one can see that the outer one is run at most once for each value of d_{L}, i.e., at most $|\mathcal{L}|$ times. The inner one is ran at most once for each possible I-L edge, i.e., at most $|\mathcal{L}||\mathcal{I}|$ times. As for the set of edges to activate for each value of d_L (function **cheapest_uniform** in Line 5), they can be pre-computed and thus do not influence the overall complexity. ■

It is also worth stressing that Property 4 concerns the *worst-case* complexity, but the actual one is often much lower. Indeed, in Line 7–Line 9 we are likely to compute the same costs in different iterations; if such costs are cached, *à la* dynamic programming, run time can be dramatically decreased, to be slightly more than linear in $|\mathcal{L}| + |\mathcal{I}|$.

VIII. Numerical Results

In the following, we describe the reference scenario and benchmark solutions we consider (Sec. VIII-A), before studying the performance of DoubleClimb (Sec. VIII-C).

A. Reference scenario

We consider an Internet-of-things (IoT) environment similar to the one referred in [5], whereby:

- individual sensors produce samples, either periodically or as a reaction to an external event;
- *aggregators*, also known as gateways, collect and summarize the samples, before forwarding them in uplink;
- distributed ML algorithms, running at the edge of the network, leverage the samples to gather insights on the changes in external conditions.

In terms of our system model, aggregators correspond to I-nodes, and edge nodes running the ML algorithms correspond to L-nodes. New samples arrive every few seconds,

and updating the gradient computations takes a comparable time. Note that similar approaches have been proposed for such applications as smart-city monitoring [41], support of connected vehicles [42], and attack/anomaly detection [43].

With reference to the taxonomy in Sec. I, we fall in the *active learning* case, as the data arrival and gradient computation are interleaved but not synchronized, e.g., new data can arrive both before and after a gradient computation is complete.

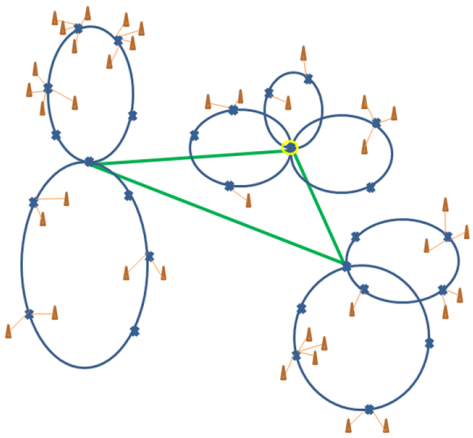

Fig. 5. Our reference topology, depicting the network of a major operator (source: [44]).

We refer to the real-world urban topology presented in [44] and shown in Fig. 5, depicting the network of a major operator. Specifically, the network nodes represented in brown act as aggregators, hence, as I-nodes, while those in blue are edge nodes acting as L-nodes. As shown in Fig. 5, all L-nodes can be connected with one another, while each I-node can only be connected to one L-node.

Normalized sample generation and gradient computation times are distributed exponentially with mean 1, while the I-L and L-L edges are randomly assigned a normalized cost between 0 and 1 units. I- and L-nodes have no operational cost, reflecting the fact that, in our reference environment, they cannot be switched off without discontinuing the service. In the *basic* version of the scenario, at every epoch each I-node generates between 10 and 100 samples; such a value is proportional to the traffic served by each node in the real-world topology [44]. In the *rich* scenario, representing applications where data is plentiful, such a value is multiplied by five.

Benchmark solutions. We compare DoubleClimb against two benchmark solutions. The first, called Opt-Unif, follows the approach used (among others) by [15], and returns the cheapest solution among the feasible ones such that both the graphs formed by L-L and I-L edges have uniform degree.

The second benchmark, labeled as "Optimum/GA" in the plots, performs the selection of the I-L edges (i.e., the inner loop in Alg. 2) leveraging a genetic algorithm (GA) approach with the following parameters:

- number of generations: 50;
- solutions per population 100;
- parents mating: 4;
- mutation probability: 15%;
- crossover type: single point;
- gene space: $\{0, 1\}$;
- number of genes: $|\mathcal{I}||\mathcal{L}|$.

Each solution corresponds to a string of binary values whose length equals the number of possible I-L edges: having a 1 in a given position means that the corresponding I-L edge is activated. The relatively large mutation probability reflects the importance of exploring multiple different solutions (i.e., exploration), given the combinatorial nature of the problem at hand and the fact that similar strings do not necessarily yield similar performance. When the size of the problem made it possible (i.e., $d_L \leq 6$), we have compared the performance of the genetic algorithm against the optimum obtained through brute force, and found that the two closely match.

B. Learning tasks

We conduct our performance evaluation with reference to the two most relevant supervised learning tasks, namely:

- a *classification* task on the famous MNIST digit database [45];
- a *regression* task on the dataset used for the ITU AI Challenge [46], with the goal of predicting the throughput of a set of Wi-Fi nodes leveraging their position and settings.

Through these two datasets, we can show how our methodology works for the two most common and relevant types of supervised learning. We tackle both learning tasks via the virtually-ubiquitous tool of deep neural networks (DNNs). Specifically, we employ a fully-connected DNN including thee hidden layers, whose sizes are 100, 50, 20 neurons, respectively. The DNN is trained via stochastic gradient descent (SGD), with a learning rate of 0.01. All experiments are implemented in Python using the `pytorch` library.

As per the methodology presented in Sec. V-A and validated, among others, in [29], we obtain the following values for the parameters in (3):

- for the classification task: $c_1 = 0.6799$, $c_2 = 0.4978$, $c_3 = 542.1$;
- for the regression task: $c_1 = 0.0956$, $c_2 = 0.5203$, $c_3 = 963.2$.

We quantify the goodness of fit through the mean square error (MSE) metric; in our experiments, the MSE for the classification and regression tasks is, respectively, 0.0027 and $9.87 \cdot 10^{-6}$. It is interesting to remark that, while the c_1–c_3 parameters are quite different in the two cases, the error is remarkably small in both instances.

In both cases, we profiled federated learning (FL) with $|\mathcal{L}| = 10$ learning nodes assisted by a central learning server, and varied the number of samples between 50% and 100% of the whole dataset. Local models are therefore averaged at every epoch, following the FedAvg [21] averaging strategy. Results have been averaged over 10 runs, changing the composition of the local datasets across different runs.

C. Performance comparison

We leverage the parameters for c_1–c_3 above to compare the performance of DoubleClimb and its alternatives, for both the tasks described earlier.

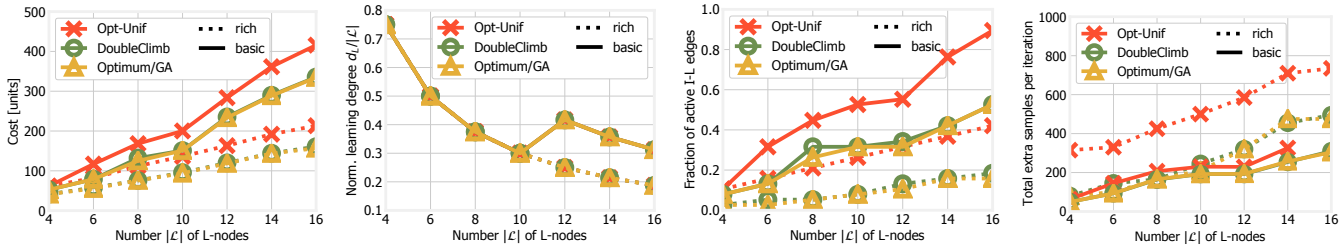

Fig. 6. **Classification task**: comparison between DoubleClimb, Opt-Unif and the optimum (obtained via brute-force) in the basic and rich scenarios, for different values of $|\mathcal{L}|$. From left to right: total cost; selected value of d_{L}, normalized (to the maximum); fraction of selected I-L edges; total number of extra samples per epoch.

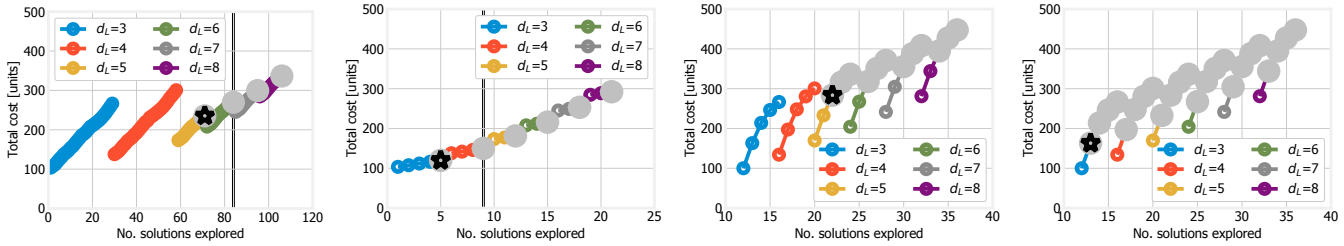

Fig. 7. **Classification task**: cost of the solutions examined at each iteration by DoubleClimb (first two plots) and Opt-Unif (last two plots), in the basic (first and third plot) and rich (second and fourth plot) scenarios.

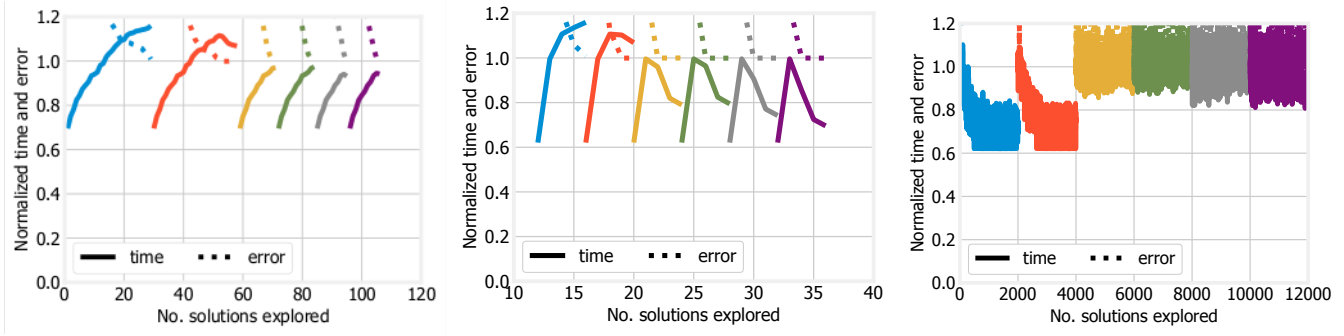

Fig. 8. **Classification task**: normalized time and error of the solutions examined at each iteration by DoubleClimb (left), Opt-Unif (center), and GA (right), in the basic scenario. Different colors correspond to different values of d_{L}, as in Fig. 7.

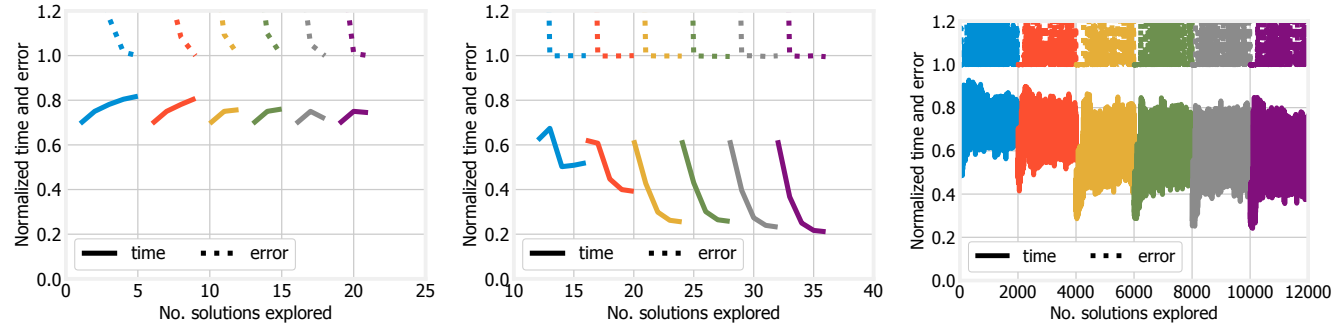

Fig. 9. **Classification task**: normalized time and error of the solutions examined at each iteration by DoubleClimb (left), Opt-Unif (center), and GA (right), in the rich scenario. Different colors correspond to different values of d_{L}, as in Fig. 7.

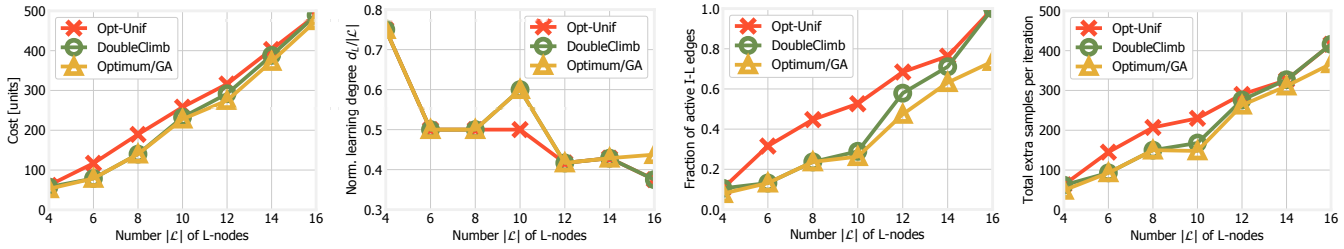

Fig. 10. **Regression task:** comparison between DoubleClimb, Opt-Unif and the optimum (obtained via brute-force) in the basic and rich scenarios, for different values of $|\mathcal{L}|$. From left to right: total cost; selected value of d_{L}, normalized (to the maximum); fraction of selected I-L edges; total number of extra samples per epoch.

We begin with the classification task and show, in the first plot in Fig. 6, the cost of DoubleClimb and its benchmarks, for different numbers of L-nodes. As expected, the cost increases with $|\mathcal{L}|$ and decreases in the rich scenario, where the higher quantity of data results in faster convergence. Also, it is clear that the cost yielded by DoubleClimb is much lower than that of Opt-Unif and matches that of Optimum/GA. GA approaches are not, in general, guaranteed to yield optimal performance; therefore, we cannot conclude that DoubleClimb makes optimal decisions other than for $d_L \leq 6$, when the comparison with brute force was possible. However, GA approaches have long been known to be remarkably good at finding optimal or near-optimal solutions for combinatorial problems such as the one at hand, at the price of long run times, as shown in Fig. 8 and Fig. 9 next. Observing that DoubleClimb matches Optimum/GA in all scenarios and for all values of d_L therefore boosts our confidence in the algorithm's effectiveness.

We now look deeper into the decisions made by each strategy. The second plot in Fig. 6 depicts the selected value of d_{L}, normalized to $|\mathcal{L}|$. Interestingly, such a value is lower in the rich scenario, confirming our intuition that a tighter cooperation between L-nodes and more data coming from I-nodes are, to an extent, alternative solutions to achieve faster learning. DoubleClimb and Opt-Unif make exactly the same decisions in all cases, which suggests that the difference in cost shown in the first plot only comes from the choice of I-L edges. Accordingly, the third plot in Fig. 6, depicting the fraction of I-L edges selected by each strategy, highlights how DoubleClimb uses substantially fewer edges than Opt-Unif. This highlights how the greater flexibility in the choice of I-L edges is an important asset of our approach, allowing us to beat state-of-the-art alternatives.

The fourth plot in Fig. 6 shows how DoubleClimb not only uses fewer I-L edges, but also chooses the *right* ones. The plot depicts the number of new samples arriving at each epoch and highlights how, in spite of the substantially smaller number of selected I-L edges, DoubleClimb obtains a similar number of samples as Opt-Unif. Such an effect is especially evident for the basic scenario, where the number of samples provided by each I-node is smaller.

Comparing the DoubleClimb and Optimum/GA curves, we can observe that in some cases Optimum/GA can activate slightly fewer I-L edges than the base scenario, e.g., for $d_L = 8$. This corresponds to solutions that DoubleClimb is unable to reach due to its hill-climbing nature; however, the impact on the overall cost (see the first plot in Fig. 6) is negligible. Interestingly, DoubleClimb and Optimum/GA make the very same decisions in the rich scenario, confirming the somehow counterintuitive notion stated in Property 3, i.e., that, the solutions yielded by DoubleClimb tend to be closer to the optimum.

In Fig. 7, we seek to better understand how DoubleClimb and Opt-Unif operate. Every marker in the plots corresponds to one solution examined by the algorithms; feasible solutions are denoted by a silver circle, the cheapest of such solutions is denoted by a black star. Note that Opt-Unif explores fewer solutions than DoubleClimb, as it is restricted to creating uniform logical topologies. Also, under the rich scenario it is easier for DoubleClimb to reach a high-quality solution, hence, the algorithm ends earlier.

The first two plots, representing DoubleClimb in the basic and the rich scenario, respectively, clearly depict the behavior of Alg. 2. The algorithm begins with the lowest possible value of d_{L} and no I-L edges, hence, with a low cost. Then, new edges are added until either a feasible solution is found, or all I-L edges are exhausted (as it happens in the first plot, representing the basic scenario). The double vertical lines in the first two plots correspond to the triggering of the condition in Line 12 of Alg. 2; the plots confirm that enforcing such a condition does not result in ignoring cheaper feasible solutions.

The last two plots in Fig. 7 represent Opt-Unif (again in the basic and the rich scenario, resp.), and clearly highlight its differences from DoubleClimb. As mentioned, Opt-Unif tries fewer solutions; also, multiple feasible solutions are tried out for the same value of d_{L}, since there is no stopping criterion analogous to the one in Line 13 in Alg. 2. Importantly, the feasible solutions explored by Opt-Unif are more costly than those explored by DoubleClimb for the same value of d_{L}, a further confirmation of the importance of a flexible choice of I-L edges.

Last, in Fig. 8 and Fig. 9, we examine the error and learning time associated with each of the solutions examined by DoubleClimb and its benchmark solutions, respectively in the basic and rich scenarios. Both quantities are normalized to their respective limits, thus both lines do not exceed 1 if the corresponding solution is feasible. It is interesting to note how adding I-L edges (moving from one solution to the next) affects error and time. The former (dotted lines) steadily decreases until its limit is reached, and then stays constant – recall that the learning process is interrupted upon reaching $\epsilon^{\max}$, so the normalized error never drops substantially below 1. The time (solid lines) increases at first, owing to the need to wait for more I-nodes; then, it decreases due to the fact that learning can be completed with fewer epochs. Importantly, both behaviors exactly match those described in Property 2 for g_1 and g_2. The third plots of both Fig. 8 and Fig. 9 highlight the behavior of GA approaches, which try multiple different solutions of varying quality and, in the interest of exploration, tend not to abandon low-quality solutions, on the grounds that they may mutate into high-quality solutions at some later stage.

The x-axis of the plots in Fig. 8 and Fig. 9 highlight the number of solutions being tried out by each of the approaches we study. Comparing the first and last plots of each figure, referring to DoubleClimb and Optimal/GA, it is easy to observe how the latter examines a number of solutions that is orders of magnitude higher than the former. Recalling that, as per Fig. 6, the two approaches yield a similar performance, it is clear the major efficiency gain brought by DoubleClimb.

Finally, in Fig. 10 we examine the performance of DoubleClimb and its alternatives for the regression task. One can observe that the behavior of DoubleClimb and the other solutions shown in Fig. 10 is consistent with that presented in Fig. 6, i.e., DoubleClimb can achieve the target learning quality at a lower cost than Opt-Unif, by obtaining a similar quantity of data while activating fewer I-L edges. Further, the performance of DoubleClimb matches that of the genetic

algorithm. This confirms how our model and solution strategy can seamlessly deal with different learning tasks.

IX. Conclusion

We addressed the problem of defining an optimal level of cooperation among network nodes performing a supervised learning task. We first developed a system model accounting for the presence of both learning nodes and information nodes interacting with each other. Then we formulated the problem of choosing which learning nodes should cooperate to complete the learning task, and the information nodes that should provide them with data, as well as the number of epochs to perform. Although being NP hard, we showed some important properties of our problem, most notably its submodularity, which allowed us to define a solution algorithm that has cubic *worst-case* time complexity and is $1+1/|\mathcal{I}|$-competitive, with $\mathcal{I}$ being the set of information nodes. Numerical results also show that our approach closely matches the optimum and outperforms state-of-the-art solutions.

References

[1] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *USENIX OSDI*, 2014.

[2] H. X. Pham, H. M. La, D. Feil-Seifer, and A. Nefian, "Cooperative and distributed reinforcement learning of drones for field coverage," *arXiv preprint arXiv:1803.07250*, 2018.

[3] H. Y. Ong, K. Chavez, and A. Hong, "Distributed deep q-learning," *CoRR*, 2015.

[4] A. Nedić, A. Olshevsky, and M. G. Rabbat, "Network topology and communication-computation tradeoffs in decentralized optimization," *Proceedings of the IEEE*, 2018.

[5] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, 2019.

[6] H. H. Zhuo, W. Feng, Y. Lin, Q. Xu, and Q. Yang, "Federated deep reinforcement learning," *arXiv preprint arXiv:1901.08277*, 2019.

[7] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[8] ETSI, "Zero touch network & Service Management (ZSM)," https://www.etsi.org/committee/zsm, online; accessed July 2020.

[9] ——, "Experiential Networked Intelligence (ENI)," https://www.etsi.org/committee-activity/eni, online; accessed July 2020.

[10] Operator Defined Next Generation RAN Architecture and Interfaces, "O-RAN Working Group 2: AI/ML workflow description and requirements," Tech. Rep. O-RAN.WG2.AIML-v01.01, online; accessed July 2020.

[11] Y. Xiao, G. Shi, Y. Li, W. Saad, and H. V. Poor, "Towards self-learning edge intelligence in 6g," *IEEE Communications Magazine*, 2020.

[12] ETSI, "MEC Working Item 36, MEC in resource constrained terminals, fixed or mobile," https://portal.etsi.org/webapp/WorkProgram/, online; accessed July 2020.

[13] A. Kadav and E. Kruus, "Asap: asynchronous approximate data-parallel computation," *arXiv preprint arXiv:1612.08608*, 2016.

[14] S. Li, S. M. M. Kalan, A. S. Avestimehr, and M. Soltanolkotabi, "Near-optimal straggler mitigation for distributed gradient methods," in *IEEE IPDPSW*, 2018.

[15] G. Neglia, G. Calbi, D. Towsley, and G. Vardoyan, "The role of network topology for distributed machine learning," in *IEEE INFOCOM*, 2019.

[16] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *arXiv preprint arXiv:2005.01643*, 2020.

[17] A. A. Abdellatif, C. F. Chiasserini, and F. Malandrino, "Active learning-based classification in automated connected vehicles," in *IEEE INFOCOM PERSIST-IoT Workshop*, 2020.

[18] K. Yang, J. Ren, Y. Zhu, and W. Zhang, "Active learning for wireless iot intrusion detection," *IEEE Wireless Communications*, 2018.

[19] T. Chen, K. Zhang, G. B. Giannakis, and T. Başar, "Communication-efficient distributed reinforcement learning," *arXiv preprint arXiv:1812.03239*, 2018.

[20] Y. Li, I.-J. Liu, Y. Yuan, D. Chen, A. Schwing, and J. Huang, "Accelerating distributed reinforcement learning with in-switch computing," in *ISCA*, 2019.

[21] J. Konečný, B. McMahan, and D. Ramage, "Federated optimization: Distributed optimization beyond the datacenter," *arXiv preprint arXiv:1511.03575*, 2015.

[22] O. Shamir and T. Zhang, "Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes," in *International conference on machine learning*, 2013.

[23] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated learning for mobile keyboard prediction," *arXiv preprint arXiv:1811.03604*, 2018.

[24] OASIS Standard, "MQTT Version 5.0, Mar. 2019," https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html, online; accessed July 2020.

[25] "zenoh: Zero Overhead Pub/sub, Store/Query and Compute," http://zenoh.io, online; accessed July 2020.

[26] 3GPP, "TS23.501, System architecture for the 5G System (5GS), Rel. 15," https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3144, online; accessed July 2020.

[27] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology*, 2019.

[28] N. J. Nagelkerke *et al.*, "A note on a general definition of the coefficient of determination," *Biometrika*, 1991.

[29] J. Hestness, S. Narang, N. Ardalani, G. Diamos, H. Jun, H. Kianinejad, M. Patwary, M. Ali, Y. Yang, and Y. Zhou, "Deep learning scaling is predictable, empirically," *arXiv preprint arXiv:1712.00409*, 2017.

[30] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting unreasonable effectiveness of data in deep learning era," in *IEEE ICCV*, 2017.

[31] T. Linjordet and K. Balog, "Impact of training dataset size on neural answer selection models," in *European Conference on Information Retrieval*, 2019.

[32] C. Perlich, F. Provost, and J. S. Simonoff, "Tree induction vs. logistic regression: A learning-curve analysis," *Journal of Machine Learning Research*, 2003.

[33] G. Serpen and Z. Gao, "Complexity analysis of multilayer perceptron neural network embedded into a wireless sensor network," *Procedia Computer Science*, 2014.

[34] D. Bolton, "The multinomial theorem," *The Mathematical Gazette*, pp. 336–342, 1968.

[35] G. J. Woeginger, "Exact algorithms for NP-hard problems: A survey," in *Combinatorial optimization—eureka, you shrink!* Springer, 2003.

[36] L. Lovász, "Submodular functions and convexity," in *Mathematical Programming The State of the Art*. Springer, 1983.

[37] M. Conforti and G. Cornuéjols, "Submodular set functions, matroids and the greedy algorithm: tight worst-case bounds and some generalizations of the rado-edmonds theorem," *Discrete applied mathematics*, 1984.

[38] V. Vu, "Random discrete matrices," in *Horizons of combinatorics*. Springer, 2008.

[39] K. Tikhomirov and P. Youssef, "The spectral gap of dense random regular graphs," *The Annals of Probability*, 2019.

[40] R. K. Iyer and J. A. Bilmes, "Submodular optimization with submodular cover and submodular knapsack constraints," in *Advances in Neural Information Processing Systems*, 2013.

[41] L. Valerio, M. Conti, and A. Passarella, "Energy efficient distributed analytics at the edge of the network for iot environments," *Elsevier Pervasive and Mobile Computing*, 2018.

[42] H. Ye, L. Liang, G. Y. Li, J. Kim, L. Lu, and M. Wu, "Machine learning for vehicular networks: Recent advances and application examples," *IEEE Vehicular Technology Magazine*, 2018.

[43] A. A. Diro and N. Chilamkurti, "Distributed attack detection scheme using deep learning approach for internet of things," *Future Generation Computer Systems*, 2018.

[44] 5G-Crosshaul, "D1.2: Final 5G-Crosshaul system design and economic analysis," December 2017.

[45] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, 2012.

[46] ITU, "AI/ML in 5G Challenge 2020," https://www.itu.int/en/ITU-T/AI/challenge/2020/, online; accessed November 2020.