Scheduling Algorithms for Federated Learning with Minimal Energy Consumption

Laércio Lima Pilla

▶ **To cite this version:**

Laércio Lima Pilla. Scheduling Algorithms for Federated Learning with Minimal Energy Consumption. IEEE Transactions on Parallel and Distributed Systems, In press, 10.1109/TPDS.2023.3240833 . hal-03775491v2

HAL Id: hal-03775491
https://hal.science/hal-03775491v2

Submitted on 3 Feb 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Scheduling Algorithms for Federated Learning with Minimal Energy Consumption *

Laércio Lima Pilla

Abstract—Federated Learning (FL) has opened the opportunity for collaboratively training machine learning models on heterogeneous mobile or Edge devices while keeping local data private. With an increase in its adoption, a growing concern is related to its economic and environmental cost (as is also the case for other machine learning techniques). Unfortunately, little work has been done to optimize its energy consumption or emissions of carbon dioxide or equivalents, as energy minimization is usually left as a secondary objective. In this paper, we investigate the problem of minimizing the energy consumption of FL training on heterogeneous devices by controlling the workload distribution. We model this as the Minimal Cost FL Schedule problem, a total cost minimization problem with identical, independent, and atomic tasks that have to be assigned to heterogeneous resources with arbitrary cost functions. We propose a pseudo-polynomial optimal solution to the problem based on the previously unexplored Multiple-Choice Minimum-Cost Maximal Knapsack Packing Problem. We also provide four algorithms for scenarios where cost functions are monotonically increasing and follow the same behavior. These solutions are likewise applicable on the minimization of other kinds of costs, and in other one-dimensional data partition problems.

Index Terms—Scheduling, optimization, machine learning, federated learning, energy conservation, dynamic programming, parallel processing, knapsack problems.

1 Introduction

Federated Learning (FL) is a distributed machine learning technique used for training a shared model collaboratively while not sharing local data [1], [2], [3], [4]. This technique reduces privacy and security risks while also improving communication efficiency [1]. These features make FL attractive for applications from next-word prediction [5] and on-device item ranking [2], to cyberattack detection [3] and graph classification [6]. Due to its data privacy design, FL has also received significant attention in medical applications. It has been used in brain tumor segmentation [7], tumor classification [8], and chest X-ray diagnosis for COVID-19 [9].

In its most standard form, FL is based on the idea of a central server that coordinates the work of participating devices (mostly heterogeneous mobile or edge devices, but also local computers or Cloud instances in some application cases). The server starts a training round by sending an initial model to some of the devices. The devices train the model with their own local data, and send the updated model weights back to the server. The latter then aggregates all updates and combines their model weights (e.g., by averaging their values) in order to start the next training round. This process is repeated until a given deadline is met, a fixed number of training rounds is achieved, or until the model converges to a target accuracy.

The accuracy of **machine learning models (FL included)** has seen improvements at the cost of larger models and **exponentially-growing computational demands** [10], [11], [12]. When combined with an increase in usage, this is leading to an increased attention to machine learning's economic and environmental costs [11], [12]. An initial study [13] indicates that **FL's energy consumption can be one order of magnitude greater than an equivalent centralized model, while its carbon footprint may even be two orders of magnitude larger** (mostly due to the energy sources available for people participating in training in different locations around the globe). Energy is also of concern for FL due to the limited batteries of mobile devices.

All of the aforementioned concerns make FL a prime target for energy consumption optimizations. Yet, little work has been done on the subject so far [14]. Energy consumption is often seen as a secondary or tertiary objective after accuracy and execution time [15], [16], [17], [18]. Additionally, the energy consumed during training is usually modeled in a simple manner to reduce the complexity of the optimization problems and ease the use of heuristics [17], [18], [19], [20], [21], [22], [23].

In this context, **we present optimal scheduling algorithms to minimize the energy consumption of Federated Learning**. Our algorithms define, for a target volume of training data (number of mini-batches), how much local data each device should use. We show that this scheduling problem, in its general definition (without any specific assumptions of the devices' energy behavior), is equivalent to a **new generalization of a 0-1 knapsack problem that can be solved in pseudo-polynomial time**. We also present **four algorithms with lower complexity for specific energy behavior scenarios** where costs increase monotonically. All of these algorithms have been designed for energy conservation, but they can be directly applied to minimize the carbon footprint, monetary cost, or any other cost function, weighted or not. We can summarize our main contributions

*Author's accepted version. Definitive version available at https://doi.org/10.1109/TPDS.2023.3240833

• *L. Lima Pilla is with Univ. Bordeaux, CNRS, Bordeaux INP, Inria, LaBRI, UMR 5800, F-33400 Talence, France.*
E-mail: laercio.lima-pilla@labri.fr

as follows:

- ⋆ We provide a formulation of the minimal cost FL schedule problem with lower limits, upper limits, and arbitrary cost functions per device;
- ⋆ We propose an optimal pseudo-polynomial solution based on the new Multiple-Choice Minimum-Cost Maximal Knapsack Packing Problem; and
- ⋆ We present four optimal scheduling algorithms for scenarios with monotonically increasing cost functions with specific marginal cost behaviors, with or without upper limits.

The remaining of this paper is organized as follows: Section 2 introduces related work on FL, energy, and scheduling. Section 3 provides a formulation of our scheduling problem. Section 4 presents an optimal solution to this problem based on a new knapsack problem and solution. Section 5 shows algorithms optimized for specific scenarios. Section 6 summarizes our experimental results. They are further detailed in the appendices of the Supplemental Material. Section 7 presents concluding remarks.

2 Related Work

Federated Learning is the subject of a large body of work in both research and development. We point our readers to the surveys done by Lim et al. [3], Zhang et al. [24], and Gu et al. [4] for a comprehensive view on the subject and associated topics. Our focus here is dedicated to works that deal with topics related to workload distribution, energy optimization for FL, and energy profiling and modeling.

2.1 Workload Distribution

When FL was originally proposed by McMahan et al. [1], no special attention was paid to the training time or to the energy consumption of the mobile devices. It is only natural then that no mechanism to control the workload distribution (i.e., how many mini-batches each device should use for training) was proposed at the time.

Wang, Wei, and Zhou [25] propose Fed-LBAP and Fed-MinAvg as mechanisms to control the workload distribution in order to minimize the computation time and accuracy loss when training FL models. They identified the computation time as the main bottleneck for FL. This argument has also been emphasized by others that postulate a decrease on the impact of communication during training with the rise of 5G technologies [4], [21]. Wang, Yang, and Zhou [26] have later presented an algorithm named MinCost for this same optimization problem. We have previously proposed OLAR [27] as an optimal greedy solution for minimizing the computation time of FL. Nonetheless, all of these algorithms help minimize the maximum cost (execution time, in this case), while our energy consumption problem requires the minimization of the total cost (its sum).

Outside FL, Khaleghzadeh, Manumachu, and Lastovetsky [28] introduce a branch-and-bound solution to minimize the computation time when the cost function of each resource (how much time it takes to process a given amount of data) is arbitrary — in other words, they do not follow any specific behavior like being monotonically increasing. Their algorithm has its worst-case complexity in $O(n^3T^3)$ for n heterogeneous resources and a workload of size T. Later, Khaleghzadeh et al. [29] have adapted this algorithm to optimize both execution time and energy consumption (i.e., find the Pareto front). Their new solution has its time complexity in $O(n^3T^3 \log(nT))$. Meanwhile, our focus lies only on energy consumption. As we will show later, our solution for arbitrary cost functions has a time complexity in $O(T^2n)$, while the solutions for specific scenarios vary between $O(Tn^2)$ and $\Theta(n)$.

A constraint in our workload distribution comes in the form of lower and upper limits on the amount of work to be given to each device. These limits play an important role in our problem. Lower limits can be used to enforce device participation at minimum levels (providing different data sources), which could also help with fairness [30]. They also help reduce the time a scheduling algorithm takes to achieve its decisions [27]. Meanwhile, upper limits avoid data over-representation from better-performing or more energy-efficient devices [3]. They can be naturally found by considering the amount of data available in a device [19]. Finally, they can also be used to set contracts with participants and help incentivize their participation in training [20].

2.2 Energy Optimization in FL

Instead of controlling the workload distribution, most works that consider the energy consumption of FL devices act on the clock frequency of their processors [15], [16], [17], [18], [23], [31], among other options. For instance, Xu, Li, and Zou [15] reduce the clock frequency of participating devices while respecting a set training round deadline in order to conserve energy. The same kind of strategy is employed by SmartPC [16]. Tran et al. [17] optimize for both time and energy by controlling the clock frequency and the fraction of communication time allocated to the devices. Meanwhile, Khan et al. [31] let Edge devices choose their own clock frequencies by using rewards as an incentive mechanism in a Stackelberg game.

The exploration of multiple optimization options concurrently is not uncommon in the literature. Nguyen et al. [18] optimize both time and energy consumption by setting the transmission time, bandwidth allocation, and clock frequency of devices. Yang et al. [23] consider in addition the transmission power and learning accuracy when minimizing the total energy consumption of FL over wireless networks. In contrast, we prefer to act only on the workload distribution as this keeps the optimization at the software level (i.e., no changes take effect in the devices' hardware). Still, acting on additional control points could be explored in future work.

There are also some works that do not use the clock frequency of the devices in their decisions, indicating other venues for optimization. Luo et al. [32] optimize time and energy together by choosing which devices should participate in a training round and by setting their local number of epochs to compute. Li et al. [21] control the communication compression in order to reduce the energy consumption when computing and communicating the FL model. Zaw et al. [22] set total energy consumption as a constraint when minimizing the training time of FL.

Finally, Anh et al. [19] are the only ones to act on something similar to the workload distribution when op-

timizing the energy consumption of FL devices. They use Deep Reinforcement Learning to decide, under uncertainty, how many data units and energy units a device should use for training in order to minimize the training time and energy consumption. Still, instead of having a set workload to distribute, this scheme tries to balance using as much data in each device with using as little energy as possible.

2.3 Energy Modeling and Profiling

Many works in the FL community model the energy consumption or execution time of training with each data unit as constants [17], [18], [19], [20], [21], [22], [23]. This has the benefit of simplifying the problem at the cost of generality. On the other hand, Khaleghzadeh, Manumachu, and Lastovetsky [28] have shown that the actual performance of parallel applications running on heterogeneous resources may vary with workload size (i.e., costs are not constant). Khaleghzadeh et al. [29] have later shown that this is also the case for energy. We employ a general model with an optimal solution in Sections 3 and 4, and we move to more constrained scenarios in Section 5.

The energy consumed by a device during training is strongly dependent on its resources and the machine learning model being used. For instance, Lane et al. [33] have shown that the energy consumption for a single inference may vary between one and three orders of magnitude depending on the device and ML model. Qiu et al. [13] illustrate differences in energy consumption during training depending on the model, device, and FL strategy. Additionally, besides hardware heterogeneity, the models being trained together may differ among devices, as is the case for personalized FL [34].

Obtaining accurate energy consumption information for scheduling on mobile devices can be done by monitoring the devices' utilization. For instance, Walker et al. [35] are able to model power consumption for multiple applications with low error ($< 3.5\%$ on average) using performance monitoring counters available on mobile devices. Kim and Wu [14] use external power meters to gather power information from different devices at different clock frequencies and states (busy or idle). They then organize devices at different power and performance levels, and use this information in combination with processor utilization information (from Unix commands) to model energy.

In the specific case of FL, I-Prof [36] has been proposed for profiling FL devices in order to predict the largest mini-batch size they can handle while respecting energy consumption and execution time limits. We believe I-Prof could be adapted to gather the energy consumption information required for our scheduling needs. Another option lies in Flower [37], a FL framework that already has the capacity to measure energy consumption on different devices, and that has been shown to be extensible [38].

Finally, for a more general review of techniques to estimate the energy consumption of ML models, we point our readers to the survey by Garcia-Martin et al. [39].

3 Definition of the Scheduling Problem

We can combine the ideas related to workload distribution (Section 2.1) and energy optimization (Section 2.2) into our scheduling problem. The problem of minimizing the energy consumption of heterogeneous devices during one round of Federated Learning resembles the problem of scheduling identical, independent, and atomic tasks on heterogeneous resources, which have been previously seen in the context of Collaborative and Grid Computing [40, Chapter 6.1]. Our model is similar to the one employed for minimizing the duration of a training round [25], [26], [27], and for partitioning data for matrix multiplication, FFT, and gene sequencing applications [28], [29]. From now on, we will refer to our mobile or Edge devices as **resources**, to the data units or mini-batches to schedule as **tasks**, and to the energy consumed by a device as its **cost**. Table 1 summarizes the main notation used throughout this text.

Our problem definition makes three main assumptions, namely that: (I) costs can be obtained, (II) costs are mainly affected by the number of tasks in a resource, and (III) costs are not affected by the actual contents of a task. While the first and second points are motivated by related work (Sections 2.3 and 2.1, respectively), the third point remains to be confirmed. So far, reports have provided results for the energy consumed for the whole training of a FL system [13] or mostly for inference tasks [33], [39]. In other words, energy consumption at the mini-batch level is a topic that still requires more study. Nevertheless, even if future research shows us energy consumption differences, we may still be required to obfuscate these variations in order to protect data privacy (as it is done for model parameters or gradients [24]).

Consider a situation with n heterogeneous resources organized in a set $\mathcal{R} = \{1, 2, \ldots, n\}$. Together, these resources must train their machine learning models with a workload of total size $T \in \mathbb{N}$. This workload is composed of identical, independent, and atomic tasks[1].

All resources have their own upper and lower limits on the number of tasks that they can use for training during the round. We represent these upper and lower limits as $\mathcal{U} = \{U_1, \ldots, U_n\}$ and $\mathcal{L} = \{L_1, \ldots, L_n\}$, respectively ($U_i$ and $L_i \in \mathbb{N}$, $\forall i \in \mathcal{R}$). A resource $i \in \mathcal{R}$ has its own local cost function $C_i{:}[L_i, U_i] \to \mathbb{R}_{\geq 0}$ that represents the energy consumed for a given number of tasks. This cost includes training the model, organizing and handling the mini-batches, communicating with the central server, etc. We use $\mathcal{C} = \{C_1, \ldots, C_n\}$ to represent the set of cost functions. Finally, consider the schedule $X = \{x_1, \ldots, x_n\}$ that assigns $x_i \in [L_i, U_i]$ tasks to each resource $i \in \mathcal{R}$.

Definition 1 (Minimal Cost FL Schedule)**.** *Given an instance $(\mathcal{R}, T, \mathcal{U}, \mathcal{L}, \mathcal{C})$, the goal is to find an optimal schedule X^* that minimizes the total cost ΣC, i.e.:*

$$\textit{minimize}_X \ \Sigma C \coloneqq \sum_{i \in \mathcal{R}} C_i(x_i) \tag{1a}$$

$$\textit{subject to} \ \sum_{i \in \mathcal{R}} x_i = T, \tag{1b}$$

$$L_i \leq x_i \leq U_i, \ \forall i \in \mathcal{R} \tag{1c}$$

1. In other words, the mini-batches cannot be distinguished, cannot be fractioned, and their use for training does not depend on the previous use of other mini-batches.

TABLE 1
Summary of main notation and symbols (by order of appearance).

Symbol	Meaning
n	Number of resources or disjoint classes of items.
$\mathcal{R}$	Set of resources.
T	Size of the workload or knapsack capacity.
U_i	Upper limit of tasks of resource i.
L_i	Lower limit of tasks of resource i.
$C_i(j)$	Cost of assigning j tasks to i.
x_i	Number of tasks assigned to resource i.
X	Schedule assigning tasks to all resources.
X^*	Optimal schedule.
ΣC	Total cost of a schedule.
N_i	Disjoint class of items.
c_{ij}	Cost of item j from class N_i.
w_{ij}	Weight of item j from class N_i.
x_{ij}	Binary variable for choosing item j from class N_i.
y	Actual capacity of the knapsack occupied by the schedule.
$\mathcal{Z}_r$	Partial solution value with the first r item classes.
τ	Restricted knapsack capacity.
$\mathcal{X}(T)$	Optimal solution for (MC)^{2}MKP with capacity T.
K	Minimal costs table for dynamic programming.
I	Items table for dynamic programming.
T^*	Capacity used in the optimal solution of (MC)^{2}MKP.
$M_i(j)$	Marginal cost of assigning the j^{th} task to resource i.
x_i^t	Partial assignment of tasks to resource i.
X^t	Schedule assigning t tasks among the resources.
ΣC^t	Total cost of a schedule with t tasks.
$\mathcal{R}^{lim}$	Subset of resources with upper limits.
$\mathcal{R}^{unl}$	Subset of resources without upper limits.
n^{lim}	Number of resources with upper limits.
γ	Translation from disjoint classes to resources.

Throughout this text, we focus on non-trivial, valid problem instances. In general, this means that no resource has an upper limit that is smaller than its lower limit, and that the number of tasks to assign is greater than the sum of lower limits and smaller than the sum of upper limits. Without loss of generality, we also assume that $T > n$.

3.1 Problem Example

Consider the situation where $\mathcal{R} = \{1, 2, 3\}$, $\mathcal{U} = \{6, 6, 5\}$, $\mathcal{L} = \{1, 0, 0\}$, and $\mathcal{C} = \{\{1{:}2, 2{:}3.5, 3{:}5.5, 4{:}8, 5{:}10, 6{:}12\}, \{0{:}0, 1{:}1.5, 2{:}2.5, 3{:}4, 4{:}7, 5{:}9, 6{:}11\}, \{0{:}0, 1{:}3, 2{:}4, 3{:}5, 4{:}6, 5{:}7\}\}$. Figs. 1 and 2 illustrate this as Gantt charts, where each line represents a resource and the numbers in blue provide the local cost of assigning a given number of tasks to each resource. The representation as Gantt charts is possible here because the costs are monotonically increasing, but this is not a constraint to our scheduling problem.

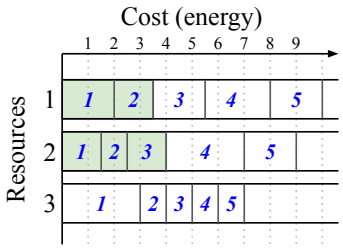


Fig. 1. Gantt chart with the optimal schedule for $T = 5$ shaded in green.

Fig. 1 shows the optimal schedule when $T = 5$, i.e., $X^* = \{2, 3, 0\}$. It provides a total cost $\Sigma C = 7.5$. Although assigning all tasks to resource 3 would provide a smaller total cost, it would not respect L_1. Meanwhile, Fig. 2 illustrates the optimal schedule for $T = 8$, i.e., $X^* = \{1, 2, 5\}$ with $\Sigma C = 11.5$. This assignment reaches both L_1 and U_3. We can also notice that the solution for the second problem does not contain the solution of the first smaller problem. This provides us with the important insight that simple greedy algorithms will not find optimal schedules.

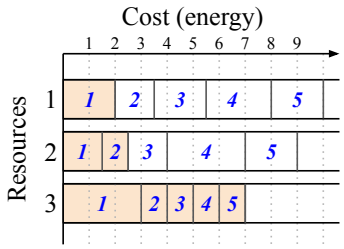


Fig. 2. Gantt chart with the optimal schedule for $T = 8$ shaded in orange.

4 Optimal Solution as a Knapsack Problem

The Minimal Cost FL Schedule problem, as shown in Definition 1, is structured as a minimization problem where the solution has to include a valid item (schedule) from each class of items (possible schedules for each resource). These properties can be directly mapped to a previously unexplored knapsack problem, which we will call the **Multiple-Choice Minimum-Cost Maximal Knapsack Packing Problem** (**(MC)^{2}MKP**). Throughout this section, we will explain this knapsack problem and show how it generalizes our scheduling problem. We follow this discussion with the presentation of its recurrence function and an optimal dynamic programming (DP) solution.

4.1 The Multiple-Choice Minimum-Cost Maximal Knapsack Packing Problem

(MC)^{2}MKP inherits characteristics from other knapsack problems. It shows some similarities to the Multiple-Choice Knapsack Problem (MCKP) [41], a maximization knapsack problem where the set of items is partitioned into classes. Although MCKP in minimization form can be easily turned into an equivalent maximization problem, its optimal solution may not use the full capacity of the knapsack. Meanwhile, the Minimum-Cost Maximal Knapsack Packing Problem (MCMKP) [42] is a less studied 0-1 knapsack problem variation [43] where the objective is to fully occupy the knapsack while also trying to minimize its total cost. Nonetheless, it lacks the classes present in MCKP. In a sense, (MC)^{2}MKP acts as a generalization of MCMKP, just as MCKP generalizes the ordinary 0-1 Knapsack Problem [41].

For the construction of (MC)^{2}MKP, consider a set of n disjoint classes $\mathcal{N} = \{N_1, \ldots, N_n\}$ of items to be packed into a knapsack of capacity T. Each item $j \in N_i$ has a cost $c_{ij} \in \mathbb{R}_{\geq 0}$ and a weight $w_{ij} \in \mathbb{N}$. The problem is to choose exactly one item from each class such that the cost sum is minimized while using the knapsack's capacity

to its maximum[2]. We use binary variables x_{ij} that take on value 1 if and only if j is chosen in class N_i, and an integer variable y that represents how much of the knapsack's capacity is being occupied by the items in a solution. With this components in mind, we can define the (MC)^{2}MKP as follows:

Definition 2 (Multiple-Choice Minimum-Cost Maximum Knapsack Packing Problem)**.** *Given a knapsack instance $(\mathcal{N}, c, w, T)$, the goal of (MC)^{2}MKP is to find a maximal knapsack packing that minimizes the cost of selected items, i.e.:*

$$\textit{minimize}_{x,y} \quad \sum_{i=1}^{n} \sum_{j \in N_i} c_{ij} x_{ij} - y \sum_{i=1}^{n} \sum_{j \in N_i} c_{ij}, \tag{2a}$$

$$\textit{subject to} \quad \sum_{i=1}^{n} \sum_{j \in N_i} w_{ij} x_{ij} \leq T, \tag{2b}$$

$$\sum_{i=1}^{n} \sum_{j \in N_i} w_{ij} x_{ij} \geq y, \tag{2c}$$

$$\sum_{j \in N_i} x_{ij} = 1, \;\; i = 1, \ldots, n, \tag{2d}$$

$$x_{ij} \in \{0, 1\}, \;\; i = 1, \ldots, n, \;\; j \in N_i, \tag{2e}$$

$$y \in [0, T] \tag{2f}$$

The formulation in Definition 2 is similar to the formulation of MCKP [41]. For instance, rule (2b) constrains the solutions to the ones that fit into the knapsack, and rule (2d) says that a single item from each set must be included. The outliers here are rules (2a) and (2c) that adapt the idea of a maximal knapsack packing [42]. Given that maximizing the occupancy of the knapsack has precedence over the minimal cost, rule (2a) unifies both objectives by multiplying y by a negative constant of absolute value larger than any possible minimal cost. Despite this, the actual cost of the solution relates only to the $\sum \sum c_{ij} x_{ij}$ part of rule (2a). Meanwhile, rule (2c) enforces that y can only be as high as the weight of the items in the knapsack, forcing the solution to maximize the knapsack's occupancy.

4.1.1 Transformation from Scheduling to Knapsack Problem

Before going through the solution to this problem, we would like to emphasize how (MC)^{2}MKP generalizes the Minimal Cost FL Schedule problem presented in Section 3.

As a first step, we can focus on the transformation between problem instances $(\mathcal{N}, c, w, T)$ and $(\mathcal{R}, T, \mathcal{U}, \mathcal{L}, \mathcal{C})$. This is based on the idea that a disjoint class N_i can be composed by all possible schedules for resource $i \in \mathcal{R}$, i.e., $N_i = \{L_i, L_i + 1, \ldots, U_i\}$. In this situation, the cost and weight of an item $j \in N_i$ are set as $c_{ij} = C_i(j)$ and $w_{ij} = j$. The solution of (MC)^{2}MKP ($x_{ij} = 1$) can be transformed to its scheduling equivalent (x_i in Definition 1) by setting $x_i = j, \; \forall i \in \mathcal{R}, x_{ij} = 1$.

Concerning the generalization, (MC)^{2}MKP relaxes two different constraints from the Minimal Cost FL Schedule problem. The first difference is that the classes in (MC)^{2}MKP can contain items with any arbitrary weights, while our scheduling problem considers all solutions in a given interval with upper and lower limits. The second difference relates to T: (MC)^{2}MKP accepts packings that use less than the total capacity of the knapsack, while our scheduling problem requires assigning all tasks to resources. These two distinctions are related: as our scheduling problem considers solution intervals for each resource, there are always solutions that assign all tasks. Meanwhile, given the arbitrary weights of items in (MC)^{2}MKP, there may be no solution that fully occupies the knapsack, leading to the second relaxation. Nonetheless, we can count on an optimal solution to (MC)^{2}MKP to also optimize the Minimal Cost FL Schedule problem.

4.2 Dynamic Programming Solution

Optimal solutions to (MC)^{2}MKP can be found using a dynamic programming technique similar to the one used for MCKP [41], [44]. Let $\mathcal{Z}_r(\tau)$ be an optimal solution value for a partial problem with the first r item classes that fully occupies a knapsack with restricted capacity τ. Assume that $\mathcal{Z}_r(\tau) \coloneqq \infty$ if no solution exists and that $\mathcal{Z}_0(0) \coloneqq 0$. $\mathcal{Z}_r(\tau)$ is defined in (3) and its value can be recursively computed following (4).

$$\mathcal{Z}_r(\tau) \coloneqq \min \left\{ \sum_{i=1}^{r} \sum_{j \in N_i} c_{ij} x_{ij} \;\middle|\; \begin{array}{l} \sum\limits_{i=1}^{r} \sum\limits_{j \in N_i} w_{ij} x_{ij} = \tau, \\ \sum\limits_{j \in N_i} x_{ij} = 1, i = 1, \ldots, r, \\ x_{ij} \in \{0, 1\}, i = 1, \ldots, r, \\ \qquad\qquad j \in N_i \end{array} \right\} \tag{3}$$

$$\mathcal{Z}_r(\tau) = \min_{j \in N_r, w_{rj} \leq \tau} \left(\mathcal{Z}_{r-1}(\tau - w_{rj}) + c_{rj} \right) \tag{4}$$

Using the optimal solutions for partial problems found by $\mathcal{Z}$, we define $\mathcal{X}(T)$ as the optimal solution for (MC)^{2}MKP with capacity T as shown in (5). Simply put, it takes the optimal solution for a capacity T if it exists, or it takes the optimal solution found with the highest occupancy. The combined use of $\mathcal{X}$ and $\mathcal{Z}$ lets us find a maximum knapsack packing that also provides the minimum cost.

$$\mathcal{X}(T) = \begin{cases} \mathcal{Z}_n(T) & \text{if } \mathcal{Z}_n(T) \neq \infty, \\ \mathcal{X}(T-1) & \text{otherwise} \end{cases} \tag{5}$$

We use the ideas behind (4) and (5) to propose Algorithm 1, which presents a dynamic programming implementation for the optimal solution of (MC)^{2}MKP. Algorithm 1 uses two matrices, K and I, to store the minimal costs that are progressively computed and the items that are part of these solutions, respectively. The algorithm first stores all possible solutions for $\mathcal{Z}_1$ (lines 7–9) and then iteratively computes the optimal solutions for increasing numbers of item classes and all knapsack capacities (lines 10–19). By its end, it finds the highest knapsack occupancy possible and its minimum cost (lines 21–24), and it goes in the reverse order through the item classes to extract the items that belong in the optimal solution (lines 25–28).

2. Knapsack problem notations usually employ C to represent the capacity of the knapsack and p_{ij} to represent profits (instead of costs). We chose to use the notation T and c_{ij} in order to preserve some similarity with our scheduling problem.

Algorithm 1 shows a space bound in $O(Tn)$ and it requires $O(T\sum_{i=1}^{n}|N_i|)$ operations. Both are equivalent to the complexity found in the DP solution for MCKP [41]. In the context of our scheduling problem, we can have at most T assignments possible for each resource. This gives us a worst-case complexity in $O(T^2n)$.

Algorithm 1 A DP solution to (MC)^{2}MKP.

Input: Set of disjoint classes of items $\mathcal{N} = \{N_1, \ldots, N_n\}$ with costs c_{ij} and weights w_{ij}, $i = 1, \ldots, n$, $j \in N_i$. Knapsack capacity T.
Output: Total cost ΣC, required capacity T^*, and list of items in the solution $X = \{x_1, \ldots, x_n\}$.

for $i = 1, \ldots, n$ **do** ▷ *Initialization of minimal costs*
  for $t = 0, \ldots, T$ **do** ▷ *and partial solutions matrices.*
    $K[i][t] \leftarrow \infty$; $I[i][t] \leftarrow \varnothing$
  end for
  $x_i \leftarrow \varnothing$
end for
for $j \in N_1$ **do** ▷ *Only solutions for* $\mathcal{Z}_1$.
  $K[1][w_{1j}] \leftarrow c_{1j}$; $I[1][w_{ij}] \leftarrow j$
end for
for $i = 2, \ldots, n$ **do** ▷ *Computes* $\mathcal{Z}_i$ *for all capacities.*
  for $j \in N_i$ **do** ▷ *Using all items in* N_i.
    for $t = w_{ij}, \ldots, T$ **do**
      if $K[i-1][t-w_{ij}] + c_{ij} < K[i][t]$ **then**
        ▷ *Best solution for* $\mathcal{Z}_i(t)$ *so far.*
        $K[i][t] \leftarrow K[i-1][t-w_{ij}] + c_{ij}$
        $I[i][t] \leftarrow j$
      end if
    end for
  end for
end for
$T^* \leftarrow T$
while $K[n][T^*] = \infty$ **do** ▷ *Finds* T^*.
  $T^* \leftarrow T^* - 1$
end while
$\Sigma C \leftarrow K[n][T^*]$ ▷ *Finds* ΣC.
$t \leftarrow T^*$
for $i = n, \ldots, 1$ **do** ▷ *Finds* X.
  $j \leftarrow I[i][t]$; $x_i \leftarrow j$; $t \leftarrow t - w_{ij}$
end for
return ΣC, T^*, X

4.2.1 Proof of optimality

The optimality of Algorithm 1 can be easily demonstrated by induction and is kept here for the sake of completeness. We first highlight the optimality of the base case for $\mathcal{Z}_1$ in Lemma 1. We then prove the induction step in Lemma 2. Finally, we combine these ideas with the selection of the minimum cost solution with the highest knapsack occupancy in Theorem 1.

Lemma 1. *The solutions in* $\mathcal{Z}_1$ *are optimal.*

Proof. The only possible solutions for $\mathcal{Z}_1$ are $\mathcal{Z}_1(w_{1j}) = c_{1j}$ for all $j \in N_1$, therefore they are optimal. □

Lemma 2. *If the solutions in* $\mathcal{Z}_i$ *are optimal, then the solutions in* $\mathcal{Z}_{i+1}$ *are also optimal.*

Proof. By the definition in (4), we know that the value of $\mathcal{Z}_{i+1}(\tau)$ for $\tau \in [0, T]$ will be equal to the minimal cost among all possible pairs of solutions in $\mathcal{Z}_i$ (with weight $\tau - w_{(i+1)j}$) and items from N_{i+1} (with weight $w_{(i+1)j} \leq \tau$) that, together, fill a knapsack with capacity τ. In order to have a better solution for $\mathcal{Z}_{i+1}(\tau)$, one of two conditions are necessary: another pair of solution from $\mathcal{Z}_i$ and item from N_{i+1} would need to have a smaller cost (a contradiction, as the minimal value for all combinations is already taken), or another solution with a smaller cost from the previous i item classes would be required (another contradiction, as $\mathcal{Z}_i$ is optimal by definition). Therefore, the solutions for $\mathcal{Z}_{i+1}$ are optimal.

□

Theorem 1. $\mathcal{X}(T)$ *provides the optimal solution for (MC)^{2}MKP.*

Proof. Lemmas 1 and 2 prove the optimality of the base case and the induction step for $\mathcal{Z}$, therefore the solutions provided by $\mathcal{Z}$ are optimal (i.e., minimal). $\mathcal{X}(T)$ returns the solution of $\mathcal{Z}(\tau)$ for the highest value of $\tau \in [0, T]$, thus it provides the minimum cost for the maximal knapsack packing possible, making its solution for (MC)^{2}MKP optimal. □

5 Optimizations for Scenarios with Monotonically Increasing Cost Functions

As we have seen previously, an optimal solution for the Minimal Cost FL Schedule can be found in pseudo-polynomial time in $O(T^2n)$ for any arbitrary, valid problem instance. Algorithm 1 makes no special assumptions regarding the behavior of cost functions in relation to the number of tasks assigned to a given resource. Meanwhile, other works in the state of the art model the execution time or energy consumption of FL tasks as linearly-proportional to the number of tasks (mini-batches) [17], [18], [19], [20], [21], [22], [23]. This kind of assumption has a direct impact on an algorithm's design and it can lead to simpler, faster solutions.

In this section, we discuss more optimized solutions for variants of our scheduling problem where the cost functions of all resources increase monotonically and follow the same behavior. Section 5.1 provides additional definitions related to marginal costs that affect the choice of algorithm. Section 5.2 includes a simple set of rules to remove the lower limits on our scheduling problem. These rules serve only to simplify the presentation of our algorithms, having no impact on the actual quality of solutions or constraints of the problem. Sections 5.3 and 5.4 present optimal algorithms for increasing and constant marginal costs, respectively. Finally, Sections 5.5 and 5.6 focus on the optimal solutions for problems with decreasing marginal costs without and with upper limits. Based on all the optimal algorithms being proposed, Table 2 presents the algorithms that provide the lowest time complexity for each scenario.

5.1 Additional Definitions and Notation

Given a resource $i \in \mathcal{R}$ and its respective cost function C_i, we can define its marginal cost function $M_i{:}[L_i, U_i] \rightarrow \mathbb{R}_{\geq 0}$ to represent the cost of assigning each additional task to i based on (6). By considering only non-negative marginal

TABLE 2
Solutions with the smallest complexity for the variations of our scheduling problem. We assume $T > n$.

	Arbitrary Costs	Marginal Costs		
		Increasing	Constant	Decreasing
Without upper limits	(MC)²MKP – $O(T^2n)$	MarIn – $\Theta(T \log n)$	MarDecUn – $\Theta(n)$	MarDecUn – $\Theta(n)$
With upper limits	(MC)²MKP – $O(T^2n)$	MarIn – $\Theta(T \log n)$	MarCo – $\Theta(n \log n)$	MarDec – $O(Tn^2)$

costs, we are also enforcing situations where all cost functions are monotonically increasing.

$$M_i(j) = \begin{cases} 0 & \text{if } j = L_i, \\ C_i(j) - C_i(j-1) & \text{otherwise} \end{cases} \tag{6}$$

Marginal costs are useful to synthesize and differentiate cost behaviors into three classes of interest: increasing (convex), constant, and decreasing (concave). These problems are characterized in Definition 3. In other words, they represent situations where — due to the specific hardware, software, and machine learning methods employed — the energy consumption of each resource grows in a superlinear, linear, or sublinear fashion with the increase in the number of tasks used for training.

Definition 3 (Increasing, Constant, and Decreasing Marginal Costs Problems)**.** *Given a problem instance $(\mathcal{R}, \mathcal{U}, \mathcal{L}, \mathcal{C})$, it is said to have increasing, constant, or decreasing marginal costs if and only if it follows* (7a)*,* (7b)*, or* (7c)*, respectively.*

$$\textit{increasing: } M_i(j) \leq M_i(j+1), \quad \forall i \in \mathcal{R}, j \in \,]L_i, U_i[\tag{7a}$$
$$\textit{constant: } M_i(j) = M_i(j+1), \quad \forall i \in \mathcal{R}, j \in \,]L_i, U_i[\tag{7b}$$
$$\textit{decreasing: } M_i(j) \geq M_i(j+1), \quad \forall i \in \mathcal{R}, j \in \,]L_i, U_i[\tag{7c}$$

Finally, we represent the partial schedule of $t < T$ tasks as $X^t = \{x_1^t, \ldots, x_n^t\}$. In this situation, x_i^t represents the number of tasks specifically assigned to resource i, and ΣC^t represents the total cost of this schedule.

5.2 Simplification by Lower Limit Removal

Independently of the specific scheduling scenario being treated, any valid solution is required to give each resource a number of tasks that respects its lower limit, as given by (1b). In this sense, we can transform any problem instance $(\mathcal{R}, T, \mathcal{U}, \mathcal{L}, \mathcal{C})$ into an equivalent instance $(\mathcal{R}, T', \{U_1', \ldots, U_n'\}, \{0\}^n, \{C_1', \ldots, C_n'\})$ that simplifies the problem by shifting all lower limits to zero and adapting all other related values. These operations are resumed in (8), (9), and (10). The resulting schedule for the equivalent instance, $X' = \{x_1', \ldots, x_n'\}$ can be transformed back to the original instance following (11). This simple set of rules represents a number of operations in $O(n)$ for any optimal solution, as the cost functions in (10) can be computed only when necessary.

$$T' \coloneqq T - \sum_{i \in \mathcal{R}} L_i \tag{8}$$

$$U_i' \coloneqq U_i - L_i, \ \forall i \in \mathcal{R} \tag{9}$$

$$C_i'(j) \coloneqq C_i(j + L_i) - C_i(L_i), \ \forall i \in \mathcal{R}, \ j \in [0, U_i'] \tag{10}$$

$$x_i \coloneqq x_i' + L_i \tag{11}$$

This simplification ensures that all scheduling algorithms start with an initial schedule X^l for $l = \sum_{i \in \mathcal{R}} L_i$. This schedule is trivially shown to be optimal in Lemma 3 and used for other optimality proofs later in this section.

Lemma 3. *The partial schedule X^l is optimal.*

Proof. X^l is the only valid schedule that respects the lower limits of all resources, therefore it is optimal. □

5.3 Increasing Marginal Costs (MarIn)

Our first scenario of interest considers the situation where the marginal costs in all resources are monotonically increasing. In this scenario, we can minimize the total cost ΣC by adapting a solution previously employed to minimize the maximum resource cost (in its context, the makespan). The solution, called MarIn and adapted from OLAR [27], is described in Algorithm 2. Its main idea is to assign the next task $t+1$ to a resource i that has the minimal marginal cost $M_i(x_i^t + 1)$ (instead of the minimal cost, as originally done by OLAR) and that has not yet reached its upper limit (lines 5–6).

Algorithm 2 MarIn — adapted from OLAR [27].

Input: Set of resources $\mathcal{R}$, number of tasks to schedule T, set of upper limits $\mathcal{U}$, set of cost functions $\mathcal{C}$.
Output: Optimal schedule X.

```
1: for all i ∈ R do
2:     x_i ← 0                    ▷ All resources start without any tasks.
3: end for
4: for t = 1, ..., T do
5:     k ← arg min_{i∈R, x_i<U_i} M_i(x_i + 1)
6:     x_k ← x_k + 1
7: end for
8: return X
```

Algorithm 2 has a space bound in $O(n)$ and it is computed using $\Theta(n + T \log n)$ operations. This complexity is achieved by employing a minimum binomial heap for storing the next task assignments (line 5). This heap's insertion and removal of the minimal item operations are in $\Theta(1)$ and $\Theta(\log n)$, respectively.

5.3.1 Proof of Optimality

MarIn's optimality can be proved by induction by combining Lemmas 3 and 4 in Theorem 2.

Lemma 4. *If X^t is optimal, then X^{t+1} computed by MarIn is optimal.*

Proof. By definition, MarIn assigns task $t+1$ to a resource with minimum marginal cost $M_i(x_i^t + 1)$ for $x_i^t \in X^t$ and

$x_i^t < U_i$. As all marginal cost functions are monotonically increasing (7a), all previous assignments in X^t had equal or smaller marginal costs. This means that X^{t+1} schedules $t+1$ tasks to the resources with the smallest marginal costs. This makes its ΣC^{t+1} minimal and, therefore, optimal. □

Theorem 2. *The schedule X computed by MarIn is optimal.*

Proof. Lemmas 3 and 4 prove the optimality of the base case and the induction step, thus the solution provided by MarIn is optimal. □

5.4 Constant Marginal Costs (MarCo)

As constant marginal costs are also monotonically increasing, this scenario could be optimally solved by MarIn. Nonetheless, having constant costs facilitates the scheduling decisions, as we can decide to assign more than one task at each step. This optimization is present in MarCo and illustrated in Algorithm 3.

Algorithm 3 MarCo.

Input: Set of resources $\mathcal{R}$, number of tasks to schedule T, set of upper limits $\mathcal{U}$, set of cost functions $\mathcal{C}$.
Output: Optimal schedule X.

```
1: for all i ∈ R do
2:     x_i ← 0                ▷ All resources start without any tasks.
3: end for
4: t ← 0
5: while t < T do
6:     k ← arg min_{i∈R, x_i≠U_i} M_i(1)
7:     x_k ← min(U_k, T − t)  ▷ Assigns the most tasks possible.
8:     t ← t + x_k
9: end while
10: return X
```

Algorithm 3's main loop (lines 5–9) uses the knowledge of constant marginal costs to find, on each of its iterations, an available resource with minimal marginal costs (line 6). This resource is assigned the maximum number of remaining tasks possible (line 7) (i.e., its upper limit or all the remaining tasks), making this resource unavailable further on. An iteration finishes with an update to the number of assigned tasks (line 8), and the loop finishes when no tasks remain to be scheduled.

MarCo displays the same space bound of MarIn but it only requires $\Theta(n \log n)$ operations. This complexity is achieved by organizing the marginal costs of all resources (used on line 6) in a sorted list, so any searches in the main loop require only a constant number of operations.

5.4.1 Proof of optimality

MarCo's optimality can be proved similarly to MarIn before it. Lemma 5 proves that each step of the algorithm is optimal, and Theorem 3 uses this information to convey that the algorithm is optimal.

Lemma 5. *If X^t is optimal, then X^{t+a} computed by MarCo is optimal.*

Proof. By definition, MarCo assigns the next a tasks to a resource with minimum marginal cost $M_i(1)$ for $i \in \mathcal{R}$ and $x_i^t \neq U_i$. By (7b), the marginal costs are constant, so any new assignments to other available resources would have equal or greater marginal costs. Additionally, all previous assignments in X^t had smaller or equal marginal costs. This means that X^{t+a} schedules $t+a$ tasks to the resources with the smallest marginal costs. This makes its ΣC^{t+a} minimal and, therefore, optimal. □

Theorem 3. *The schedule X computed by MarCo is optimal.*

Proof. Lemmas 3 and 5 prove the optimality of the base case and the induction step, thus the solution provided by MarCo is optimal. □

5.5 Decreasing Marginal Costs without Upper Limits (MarDecUn)

The presence of decreasing marginal costs requires an approach that is different from previous scenarios. While previous scenarios made it possible to incrementally assign the tasks with the smallest marginal costs, here the smallest marginal costs come from the last tasks assigned to a resource. In this sense, optimal assignments can include tasks with high marginal costs if enough tasks are assigned to the same resource, reducing the average cost per task.

Algorithm 4, named MarDecUn, focuses on the situation where the upper limits of all resources are equal or superior to the actual number of tasks to schedule. In this situation, the optimal solution is found by simply scheduling all tasks to a resource with minimal average cost per task or, in other words, a resource with minimum cost for T tasks (lines 4–5).

Algorithm 4 MarDecUn.

Input: Set of resources $\mathcal{R}$, number of tasks to schedule T, set of upper limits $\mathcal{U}$, set of cost functions $\mathcal{C}$.
Output: Optimal schedule X.

```
1: for all i ∈ R do
2:     x_i ← 0                ▷ All resources start without any tasks.
3: end for
4: k ← arg min_{i∈R} C_i(T)
5: x_k ← T                    ▷ Assigns all tasks to the same resource.
6: return X
```

Due to its simplicity, MarDecUn requires only $\Theta(n)$ operations to find a resource with minimal cost. Its space bound is still the same $O(n)$ of previous algorithms because its solution includes a schedule for all resources. If its output were to be changed to inform only the resource receiving all tasks, this bound could be reduced to a constant.

5.5.1 Proof of optimality

MarDecUn's optimality can be proved based on two ideas. The first idea is that MarDecUn assigns all possible tasks to a resource with minimal cost. The second idea is related to the behavior of decreasing functions, which is presented in Lemma 6. These ideas are combined in Theorem 4.

Lemma 6 (Sum of contiguous intervals of decreasing functions)**.** *If f, g: $\mathbb{N} \to \mathbb{R}$ are monotonically decreasing functions and $f(s_f + 1) \leq g(s_g + 1)$, then* (12) *is true for any intervals $[i_f, s_f + s_g - i_g + 1]$ and $[i_g, s_g + s_f - i_f + 1]$.*

$$\sum_{i=i_f}^{s_f} f(i) + \sum_{i=i_g}^{s_g} g(i) \geq \sum_{i=i_f}^{s_f+s_g-i_g+1} f(i) \tag{12}$$

Proof. By definition, the decreasing functions f and g follow the behavior illustrated in (13).

$$\begin{aligned} f(i_f) \geq \cdots \geq f(s_f+1) \geq \cdots \geq f(s_f+s_g-i_g+1) \\ g(i_g) \geq \cdots \geq g(s_g+1) \geq \cdots \geq g(s_g+s_f-i_f+1) \end{aligned} \tag{13}$$

If $g(s_g+1) \geq f(s_f+1)$, then this should also hold for all values larger than s_f+1 and smaller than s_g+1 (14).

$$g(i_g) \geq \cdots \geq g(s_g{+}1) \geq f(s_f{+}1) \geq \cdots \geq f(s_f{+}s_g{-}i_g{+}1) \tag{14}$$

As $g(s_g) \geq f(s_f{+}1)$, changing the intervals to $[i_f, s_f{+}1]$ and $[i_g, s_g{-}1]$ should lead to a smaller or equal sum without changing the number of elements considered. This idea can be applied iteratively, leading to (15) and proving (12).

$$\begin{aligned} \sum_{i=i_f}^{s_f} f(i) + \sum_{i=i_g}^{s_g} g(i) &\geq \sum_{i=i_f}^{s_f+1} f(i) + \sum_{i=i_g}^{s_g-1} g(i) \\ &\geq \cdots \geq \sum_{i=i_f}^{s_f+s_g-i_g} f(i) + \sum_{i=i_g}^{i_g} g(i) \\ &\geq \sum_{i=i_f}^{s_f+s_g-i_g+1} f(i) \end{aligned} \tag{15}$$

□

Theorem 4. *The schedule X computed by MarDecUn is optimal.*

Proof. MarDecUn assigns all T tasks to a resource with minimal cost, so no other assignment to a single resource could improve the solution. By the definition of marginal costs in (6), we can rewrite the cost of a schedule for one resource as a sum of marginal costs, as shown in (16).

$$C_i(T) = C_i(0) + \sum_{t=1}^{T} M_i(t), \;\; \forall i \in \mathcal{R} \tag{16}$$

Given that all marginal cost functions are decreasing, Lemma 6 tells us that no solution splitting the T tasks among multiple resources can provide a smaller total cost, therefore the schedule computed by MarDecUn is minimal and optimal. □

5.6 Decreasing Marginal Costs with Upper Limits (MarDec)

Although the previous solution for the scenario of decreasing marginal costs and no upper limits cannot be applied in the presence of upper limits, it provides us with an important insight. Namely, Lemma 6 tells us that it is always more beneficial to put all tasks in the same resource, so an optimal solution can be found in one of two scenarios: (I) all tasks are assigned to a resource without upper limits; or (II) all tasks are assigned only to resources at maximum capacity and at most one resource at intermediary capacity. Both scenarios are covered by MarDec in Algorithm 5.

MarDec starts by splitting the resources into two subsets: one for the resources that have upper limits ($\mathcal{R}^{lim}$) and one for those who do not ($\mathcal{R}^{unl}$) (lines 1–2).

In order to compute possible solutions for scenario (II), MarDec employs a dynamic programming solution for the Minimum-Cost Maximal Knapsack Packing (MCMKP) problem [42], as it helps us find which resources have to be at maximum capacity. As MCMKP is a specialization of (MC)^{2}MKP, Algorithm 5 uses a variation of Algorithm 1 that we call (MC)^{2}MKP-matrices. It outputs the support matrices K and I to enable the reuse of its partial solutions. As previously described in Section 4.2, K stores the minimal costs that were progressively computed, while I stores the items that are part of the partial solutions. MarDec also employs Algorithm 6 to convert its variables for use in a knapsack problem, and Algorithm 7 to translate a partial MCMKP solution to a schedule.

MarDec covers scenario (II) in two steps. At first, it computes all possible minimal solutions where a resource without upper limits is set at intermediary capacity (lines 6–15). For each possible intermediary capacity (line 8), it finds the resource with minimal cost to receive the remaining tasks. It is important to emphasize here that, if no solution is found for a specific knapsack capacity, (MC)^{2}MKP provides an infinite cost, so no invalid solutions are ever considered. Additionally, when $t = T$, the solution for scenario (I) is computed (i.e., the whole workload goes to a single resource as in MarDecUn). At its second step, MarDec verifies all possible minimal solutions where one of the resources with upper limits ends up at intermediary capacity (lines 17–28). In this case, MarDec removes the resource of interest from the input of (MC)^{2}MKP (line 18) and then computes all possible optimal schedules. Throughout all these steps, the schedule X that provides the minimal cost ΣC is kept and provided at the end of the algorithm.

Algorithm 5 has a space bound in $O(Tn)$ due to its use of support matrices K and I. It requires $O(Tn^2)$ operations. This number comes from the utilization of (MC)^{2}MKP in line 19. As defined in Section 4.2, (MC)^{2}MKP requires $O(T\sum_{i=1}^{n}|N_i|)$ operations. In our case, each set of items N contains only two items (i.e., scheduling zero or U_i tasks to resource i), so its complexity is in $O(Tn)$. Given that (MC)^{2}MKP is computed at most $n+1$ times, the aforementioned complexity is achieved.

5.6.1 Proof of optimality

MarDec's optimality is demonstrated directly in Theorem 5 based on the previous proofs for MarDecUn and (MC)^{2}MKP.

Theorem 5. *The schedule X computed by MarDec is optimal.*

Proof. In the presence of decreasing marginal costs, Lemma 6 defines that an optimal schedule can be found in one of two scenarios: all remaining tasks are assigned to the same resource with minimum cost, or all tasks are assigned only to resources at maximum capacity and at most one resource at intermediary capacity (i.e., having two or more resources at intermediary capacity contradicts Lemma 6).

MarDec computes a solution to the first scenario exactly as MarDecUn does (proved optimal in Theorem 4). For the second scenario, MarDec employs (MC)^{2}MKP to compute all possible minimum-cost partial schedules using resources at maximum capacity. These partial solutions are proved optimal in Theorem 1. All possible partial schedules are combined to all possible assignments of the remaining tasks to other resources with minimum cost. By exhaustion, MarDec keeps the minimum-cost solution among every single possible solution that could provide a minimum cost in our two scenarios, therefore its schedule is optimal. □

Algorithm 5 MarDec.

Input: Set of resources $\mathcal{R}$, number of tasks to schedule T, set of upper limits $\mathcal{U}$, set of cost functions $\mathcal{C}$.
Output: Optimal schedule X.

$\mathcal{R}^{lim} \leftarrow \{i\},\ \forall i \in \mathcal{R}, U_i < T$ ▷ *Resources w/ upper limits*
$\mathcal{R}^{unl} \leftarrow \mathcal{R} \setminus \mathcal{R}^{lim}$ ▷ *Resources without upper limits*
$n^{lim} \leftarrow |\mathcal{R}^{lim}|$
$\Sigma C \leftarrow \infty$ ▷ *No valid solution so far.*
▷ *Resource from $\mathcal{R}^{unl}$ at intermediary capacity.*
if $\mathcal{R}^{unl} \neq \emptyset$ **then**
  $(\mathcal{N}, c, w, \gamma) \leftarrow \text{Prepare}(\mathcal{R}^{lim}, \mathcal{U}, \mathcal{C})$ ▷ *Algorithm 6.*
  $(K, I) \leftarrow$ (MC)^{2}MKP-matrices$(\mathcal{N}, c, w, T)$
  for $t = 0, \ldots, T$ **do** ▷ *Evaluates all partial solutions.*
    $k \leftarrow \arg\min_{i \in \mathcal{R}^{unl}} C_i(t)$
    if $C_k(t) + K[n^{lim}][T-t] < \Sigma C$ **then**
      $\Sigma C \leftarrow C_k(t) + K[n^{lim}][T-t]$ ▷ *New minimal.*
      $X \leftarrow \text{Translate}(\gamma, \mathcal{R}, \mathcal{N}, w, I, t)$ ▷ *Algorithm 7.*
      $x_k \leftarrow t$
    end if
  end for
end if
▷ *Resource from $\mathcal{R}^{lim}$ at intermediary capacity.*
for $i = 1, \ldots, n^{lim}$ **do**
  $\mathcal{N}' \leftarrow (\mathcal{N} \setminus N_i) \cup \{N_i = \{0\}\}$
  $(K, I) \leftarrow$ (MC)^{2}MKP-matrices$(\mathcal{N}', c, w, T)$
  $k \leftarrow \gamma(i)$ ▷ *Translates i to k.*
  for $t = 0, \ldots, U_k - 1$ **do** ▷ *Checks all solutions with k.*
    if $C_k(t) + K[n^{lim}][T-t] < \Sigma C$ **then**
      $\Sigma C \leftarrow C_k(t) + K[n^{lim}][T-t]$
      $X \leftarrow \text{Translate}(\gamma, \mathcal{R}, \mathcal{N}, w, I, t)$
      $x_k \leftarrow t$
    end if
  end for
end for
return X

Algorithm 6 Preparation for (MC)^{2}MKP.

Input: Set of resources with upper limits $\mathcal{R}^{lim}$ of size n^{lim}, set of upper limits $\mathcal{U}$, set of cost functions $\mathcal{C}$.
Output: Set of disjoint classes of items $\mathcal{N} = \{N_1, \ldots, N_{n^{lim}}\}$ with costs c_{ij} and weights w_{ij}, $i = 1, \ldots, n^{lim}$, $j \in N_i$. Translation from disjoint classes to resources γ.

$i \leftarrow 1$
for all $r \in \mathcal{R}^{lim}$ **do**
  $\gamma(i) \leftarrow r$
  $N_i \leftarrow \{0, U_r\}$ ▷ *Classes with 0 or U_r tasks.*
  $c_{i0} \leftarrow 0\ ;\ c_{iU_r} \leftarrow C_r(U_r)$
  $w_{i0} \leftarrow 0\ ;\ w_{iU_r} \leftarrow U_r$
  $i \leftarrow i + 1$
end for
return $(\mathcal{N}, c, w, \gamma)$

6 Experimental Evaluation Overview

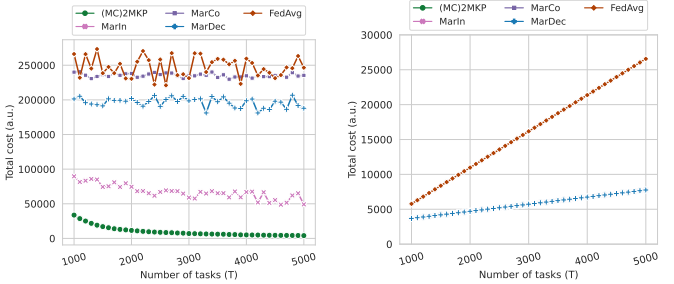
This section provides a brief overview of the experimental evaluation detailed in the appendices in the Supplemental Material. The evaluation takes into consideration the five optimal algorithms proposed in Sections 4 and 5 and FedAvg [1], resources with four different kinds of cost functions, variations in numbers of tasks and resources, and an evaluation of the total costs achieved by the algorithms and their own execution times. All code, scripts, and data of our experiments are available online [45], enabling the reproduction of the experiments and their analysis.

Algorithm 7 Translation from (MC)^{2}MKP to a schedule.

Input: Translation from disjoint classes to resources γ. Set of resources $\mathcal{R}$. Set of disjoint classes of items $\mathcal{N} = \{N_1, \ldots, N_{n^{lim}}\}$ with weights w_{ij}, $i = 1, \ldots, n^{lim}$, $j \in N_i$. Support matrix I of dimensions $n^{lim} \times T$. Knapsack capacity of interest T'.
Output: Partial schedule X.

$x_i \leftarrow 0,\ \forall i \in \mathcal{R}$
$t \leftarrow T'$
for $i = n^{lim}, \ldots, 1$ **do** ▷ *Finds X.*
  $j \leftarrow I[i][t]\ ;\ t \leftarrow t - w_{ij}$
  $x_{\gamma(i)} \leftarrow j$
end for
return X

Fig. 3 illustrates the total costs achieved by all scheduling algorithms for 100 resources, from 200 up to 2,000 tasks, and two kinds of cost functions: random and monotonically increasing with constant marginal costs. The results in Fig. 3a illustrate how (MC)^{2}MKP is unique in its capacity to find optimal schedules with minimal total costs in all situations. Meanwhile, the results in Fig. 3b show that all algorithms besides FedAvg can find optimal solutions when marginal costs are constant. This happens because constant marginal costs are also monotonically increasing and decreasing, so MarIn, MarCo, and MarDec are able to optimize the solution.

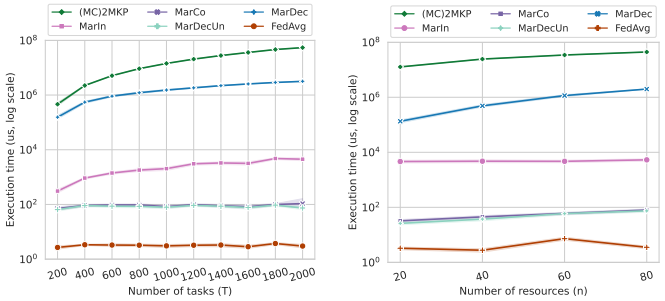


(a) Random cost function.

(b) Increasing costs with constant marginal costs.

Fig. 3. Total costs achieved by the different scheduling algorithms for varying number of tasks and 100 resources.

Fig. 4 shows the average execution times for the different algorithms for varying numbers of resources and tasks. Points are computed based on twenty samples of five repetitions each. They illustrate the importance of using the optimized algorithms proposed in Section 5 whenever possible, as execution times may differ by up to six orders of magnitude in our experiments. These differences come naturally from the time complexity of our algorithms, as previously illustrated in Table 2.



(a) $n = 100$, varying T. (b) $T = 2,000$, varying n.

Fig. 4. Average scheduling algorithm times in microseconds (logarithmic scale).

7 Concluding Remarks

In this paper, we considered the problem of minimizing the energy consumption of Federated Learning training on heterogeneous devices by controlling their workload distribution (i.e., the number of mini-batches used for training on each device). This problem is of growing interest, given the environmental costs of machine learning [11], [12] and FL systems [13]. We have modeled this as the Minimal Cost FL Schedule problem, a total cost minimization problem with identical, independent, and atomic tasks that have to be assigned to heterogeneous resources with arbitrary cost functions. In this process, we have defined a previously unexplored knapsack problem named Multiple-Choice Minimum-Cost Maximal Knapsack Packing Problem that generalizes our scheduling problem. We have proposed an optimal solution for this knapsack problem based on dynamic programming and proved its optimality, thus solving the Minimal Cost FL Schedule problem too.

We have also explored scenarios with monotonically increasing cost functions with specific behaviors, and situations with and without upper limits. In all scenarios, but especially in scenarios with constant and decreasing marginal costs, the solution that minimizes the energy consumption may require that few resources to do most of the training. In order to prevent over-representation from more energy-efficient devices [3], we recommend paying attention to the upper and lower limits set for all resources.

We would also like to emphasize that these algorithms are not only useful for energy conservation in FL systems, as they can also be used to minimize other kinds of costs (e.g., emissions of carbon dioxide or equivalents, financial costs), requiring only the cost estimates for different workload assignments. For instance, if the geographical location of devices or their energy sources are known, their energy consumption can be converted to carbon emissions [13]. Additionally, these algorithms can be applied to other problems that work with one-dimensional data partitions [28], [29].

For the foreseeable future, we envision studies related to the application and adaptation of our algorithms. First, we would like to conduct experiments in FL platforms to evaluate the impact of our algorithms compared to other solutions. This impact should be measured in energy consumption, execution time, and accuracy of the model (or convergence speed). Second, in terms of adaptation, new solutions may be required to handle dynamic changes in the system (e.g., changes in the cost behavior or loss of a device), to optimize the energy consumption of asynchronous FL systems, and to optimize FL systems that can offload parts of their computations to other Edge devices.

Acknowledgments

The author would like to thank Dr. Amina Guermouche and Dr. Mihail Popov for their feedback on earlier versions of this manuscript.

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Singh and J. Zhu, Eds., vol. 54. PMLR, 20–22 Apr 2017, pp. 1273–1282.

[2] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, H. B. McMahan *et al.*, "Towards federated learning at scale: System design," in *Proceedings of Machine Learning and Systems*, vol. 1, 2019, pp. 374–388.

[3] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2031–2063, thirdquarter 2020.

[4] R. Gu, C. Niu, F. Wu, G. Chen, C. Hu, C. Lyu, and Z. Wu, "From server-based to client-based machine learning: A comprehensive survey," *ACM Comput. Surv.*, vol. 54, no. 1, jan 2021. [Online]. Available: https://doi.org/10.1145/3424660

[5] 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.

[6] H. Xie, J. Ma, L. Xiong, and C. Yang, "Federated graph classification over non-iid graphs," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 18 839–18 852.

[7] M. J. Sheller, G. A. Reina, B. Edwards, J. Martin, and S. Bakas, "Multi-institutional deep learning modeling without sharing patient data: A feasibility study on brain tumor segmentation," in *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*, A. Crimi, S. Bakas, H. Kuijf, F. Keyvan, M. Reyes, and T. van Walsum, Eds. Cham: Springer International Publishing, 2019, pp. 92–104.

[8] R. Brum, L. Drummond, M. C. Castro, and G. Teodoro, "Towards optimizing computational costs of federated learning in clouds," in *2021 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)*, Oct 2021, pp. 35–40.

[9] S. Park, G. Kim, J. Kim, B. Kim, and J. C. Ye, "Federated split task-agnostic vision transformer for covid-19 cxr diagnosis," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 24 617–24 630.

[10] D. Amodei and D. Hernandez, "Ai and compute," 2018, accessed: 2022-11-30. [Online]. Available: https://openai.com/blog/ai-and-compute/

[11] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green ai," *Commun. ACM*, vol. 63, no. 12, p. 54–63, nov 2020. [Online]. Available: https://doi.org/10.1145/3381831

[12] P. Henderson, J. Hu, J. Romoff, E. Brunskill, D. Jurafsky, and J. Pineau, "Towards the systematic reporting of the energy and carbon footprints of machine learning," *Journal of Machine Learning Research*, vol. 21, no. 248, pp. 1–43, 2020.

[13] X. Qiu, T. Parcollet, J. Fernandez-Marques, P. P. B. de Gusmao, D. J. Beutel, T. Topal, A. Mathur, and N. D. Lane, "A first look into the carbon footprint of federated learning," *arXiv preprint arXiv:2102.07627*, 2021.

[14] Y. G. Kim and C.-J. Wu, "Autofl: Enabling heterogeneity-aware energy efficient federated learning," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 183–198.

[15] Z. Xu, L. Li, and W. Zou, "Exploring federated learning on battery-powered devices," in *Proceedings of the ACM Turing Celebration Conference - China*, ser. ACM TURC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3321408.3323080

[16] L. Li, H. Xiong, Z. Guo, J. Wang, and C.-Z. Xu, "Smartpc: Hierarchical pace control in real-time federated learning system," in *2019 IEEE Real-Time Systems Symposium (RTSS)*, Dec 2019, pp. 406–418.

[17] N. H. Tran, W. Bao, A. Zomaya, M. N. H. Nguyen, and C. S. Hong, "Federated learning over wireless networks: Optimization model design and analysis," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, April 2019, pp. 1387–1395.

[18] V.-D. Nguyen, S. K. Sharma, T. X. Vu, S. Chatzinotas, and B. Ottersten, "Efficient federated learning algorithm for resource allocation in wireless iot networks," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3394–3409, 2021.

[19] T. T. Anh, N. C. Luong, D. Niyato, D. I. Kim, and L.-C. Wang, "Efficient training management for mobile crowd-machine learning: A deep reinforcement learning approach," *IEEE Wireless Communications Letters*, vol. 8, no. 5, pp. 1345–1348, Oct 2019.

[20] J. Kang, Z. Xiong, D. Niyato, H. Yu, Y.-C. Liang, and D. I. Kim, "Incentive design for efficient federated learning in mobile networks: A contract theory approach," in *2019 IEEE VTS Asia Pacific Wireless Communications Symposium (APWCS)*, Aug 2019, pp. 1–5.

[21] L. Li, D. Shi, R. Hou, H. Li, M. Pan, and Z. Han, "To talk or to work: Flexible communication compression for energy efficient federated learning over heterogeneous mobile edge devices," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.

[22] C. W. Zaw, S. R. Pandey, K. Kim, and C. S. Hong, "Energy-aware resource management for federated learning in multi-access edge computing systems," *IEEE Access*, vol. 9, pp. 34 938–34 950, 2021.

[23] Z. Yang, M. Chen, W. Saad, C. S. Hong, and M. Shikh-Bahaei, "Energy efficient federated learning over wireless communication networks," *IEEE Transactions on Wireless Communications*, vol. 20, no. 3, pp. 1935–1949, March 2021.

[24] J. Zhang, Z. Qu, C. Chen, H. Wang, Y. Zhan, B. Ye, and S. Guo, "Edge learning: The enabling technology for distributed big data analytics in the edge," *ACM Comput. Surv.*, vol. 54, no. 7, jul 2021. [Online]. Available: https://doi.org/10.1145/3464419

[25] C. Wang, X. Wei, and P. Zhou, "Optimize scheduling of federated learning on battery-powered mobile devices," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2020, pp. 212–221.

[26] C. Wang, Y. Yang, and P. Zhou, "Towards efficient scheduling of federated mobile devices under computational and statistical heterogeneity," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 394–410, Feb 2021.

[27] L. Lima Pilla, "Optimal task assignment for heterogeneous federated learning devices," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2021, pp. 661–670.

[28] H. Khaleghzadeh, R. R. Manumachu, and A. Lastovetsky, "A novel data-partitioning algorithm for performance optimization of data-parallel applications on heterogeneous hpc platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 10, pp. 2176–2190, Oct 2018.

[29] H. Khaleghzadeh, M. Fahad, A. Shahid, R. R. Manumachu, and A. Lastovetsky, "Bi-objective optimization of data-parallel applications on heterogeneous hpc platforms for performance and energy through workload distribution," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 543–560, March 2021.

[30] T. Huang, W. Lin, W. Wu, L. He, K. Li, and A. Y. Zomaya, "An efficiency-boosting client selection scheme for federated learning with fairness guarantee," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1552–1564, July 2021.

[31] L. U. Khan, S. R. Pandey, N. H. Tran, W. Saad, Z. Han, M. N. H. Nguyen, and C. S. Hong, "Federated learning for edge networks: Resource optimization and incentive mechanism," *IEEE Communications Magazine*, vol. 58, no. 10, pp. 88–93, October 2020.

[32] B. Luo, X. Li, S. Wang, J. Huang, and L. Tassiulas, "Cost-effective federated learning design," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, May 2021, pp. 1–10.

[33] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, and F. Kawsar, "An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices," in *Proceedings of the 2015 International Workshop on Internet of Things towards Applications*, ser. IoT-App '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 7–12. [Online]. Available: https://doi.org/10.1145/2820975.2820980

[34] J. Jang, H. Ha, D. Jung, and S. Yoon, "Fedclassavg: Local representation learning for personalized federated learning on heterogeneous neural networks," in *51th International Conference on Parallel Processing*, ser. ICPP 2022. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3545008.3545073

[35] M. J. Walker, S. Diestelhorst, A. Hansson, A. K. Das, S. Yang, B. M. Al-Hashimi, and G. V. Merrett, "Accurate and stable run-time power modeling for mobile and embedded cpus," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 1, pp. 106–119, 2017.

[36] G. Damaskinos, R. Guerraoui, A.-M. Kermarrec, V. Nitu, R. Patra, and F. Taiani, "Fleet: Online federated learning via staleness awareness and performance prediction," in *Proceedings of the 21st International Middleware Conference*, ser. Middleware '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 163–177. [Online]. Available: https://doi.org/10.1145/3423211.3425685

[37] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão, and N. D. Lane, "FLOWER: A FRIENDLY FEDERATED LEARNING FRAMEWORK," Mar. 2022, open-Source, mobile-friendly Federated Learning framework. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03601230

[38] W. Zhao, X. Qiu, J. Fernandez-Marques, P. P. de Gusmão, and N. D. Lane, "Protea: Client profiling within federated systems using flower," *arXiv preprint arXiv:2207.01053*, 2022.

[39] E. García-Martín, C. F. Rodrigues, G. Riley, and H. Grahn, "Estimation of energy consumption in machine learning," *Journal of Parallel and Distributed Computing*, vol. 134, pp. 75–88, 2019.

[40] H. Casanova, A. Legrand, and Y. Robert, *Parallel algorithms*. CRC Press, 2008.

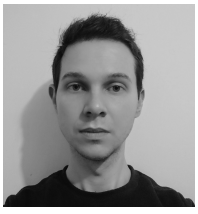
[41] H. Kellerer, U. Pferschy, and D. Pisinger, "The multiple-choice knapsack problem," in *Knapsack Problems*. Springer, 2004, pp. 317–347.

[42] F. Furini, I. Ljubić, and M. Sinnl, "An effective dynamic programming algorithm for the minimum-cost maximal knapsack packing problem," *European Journal of Operational Research*, vol. 262, no. 2, pp. 438–448, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221717302928

[43] V. Cacchiani, M. Iori, A. Locatelli, and S. Martello, "Knapsack problems — an overview of recent advances. part ii: Multiple, multidimensional, and quadratic knapsack problems," *Computers & Operations Research*, vol. 143, p. 105693, 2022.

[44] K. Dudziński and S. Walukiewicz, "Exact methods for the knapsack problem and its generalizations," *European Journal of Operational Research*, vol. 28, no. 1, pp. 3–21, 1987. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0377221787901652

[45] L. L. Pilla, "llpilla/energy-optimal-federated-learning: Full set of algorithms: (MC)²MKP, MarIn, MarCo, MarDec, and MarDecUn," Nov. 2022. [Online]. Available: https://doi.org/10.5281/zenodo.7377419



Laércio Lima Pilla is a tenured Research Scientist working for the French National Centre for Scientific Research (CNRS) in the *Laboratoire Bordelais de Recherche en Informatique* (LaBRI). He is a member of the STORM team in the Inria centre at the University of Bordeaux. He obtained his Ph.D. from the Federal University of Rio Grande do Sul (UFRGS) and Univ. Grenoble Alpes in 2014. His research is mostly focused on scheduling algorithms for parallel and distributed systems. More information is available on his website at https://www.labri.fr/perso/llimapilla/.

Supplemental Material to "Scheduling Algorithms for Federated Learning with Minimal Energy Consumption"

Laércio Lima Pilla

This document details the experimental evaluation campaign carried out in the context of the article "Scheduling Algorithms for Federated Learning with Minimal Energy Consumption". Appendix A describes the experimental environment of the campaign. Appendix B provides the total cost achieved by different scheduling algorithms for different numbers of tasks and resources, and different cost function behaviors. Appendix C presents the execution time required by the different scheduling algorithms to compute their solutions. All code, analysis scripts, and data generated during the experiments are freely available online [1].

APPENDIX A
EXPERIMENTAL ENVIRONMENT

In order to visualize the differences in solution quality and execution time of the proposed scheduling algorithms, we have organized an experimental evaluation based on simulation. Prototypes of the different algorithms were implemented using Python 3 to profit from the language's ease of implementation and execution (to help reproduce our results).

Our experiments include six scheduling algorithms (five of which are proposed in the main article), four kinds of cost functions for the resources, and scheduling scenarios with different numbers of resources and tasks. The algorithms are compared based on the quality of their schedules (achieved total costs) and their execution times. In the next sections, we provide details of the scheduling algorithms, cost functions, lower and upper limits, hardware, and software used in the experiments.

A.1 Scheduling Algorithms

Our experiments include the six algorithms listed below.

1) **(MC)^{2}MKP**, described in Section 4.2, proved optimal for all kinds of cost functions;
2) **MarIn**, described in Section 5.3, proved optimal for monotonically increasing cost functions with increasing marginal costs;
3) **MarCo**, described in Section 5.4, proved optimal for monotonically increasing cost functions with constant marginal costs;
4) **MarDecUn**, described in Section 5.5, proved optimal for monotonically increasing cost functions with decreasing marginal costs and no upper limits;
5) **MarDec**, described in Section 5.6, proved optimal for monotonically increasing cost functions with decreasing marginal costs; and
6) **FedAvg**, proposed by McMahan et al. [2], which sets an equal number of tasks to all resources.

• *L. Lima Pilla is with Univ. Bordeaux, CNRS, Bordeaux INP, Inria, LaBRI, UMR 5800, F-33400 Talence, France.*
E-mail: laercio.lima-pilla@labri.fr

A.2 Cost Functions

We simulate resources whose cost functions follow four possible behaviors. For the functions using parameters α and β, their values were randomly chosen from a uniform distribution in the interval $[1, 10)$.

- **Random** follows a function $f(x) = \gamma$, where γ is randomly chosen from a uniform distribution in the interval $[0, 5001)$. Although not realistic, this kind of cost function stresses the capability of our scheduling algorithms to find good or optimal solutions.
- **Nlogn** follows a function $f(x) = \alpha + \beta x \log x$ with increasing marginal costs.
- **Linear** follows a function $f(x) = \alpha + \beta x$ with constant marginal costs.
- **Logn** follows a function $f(x) = \alpha + \beta \log x$ with decreasing marginal costs.

In each experiment, all simulated resources are represented by cost functions with the same behavior generated with different random number generator (RNG) seeds. The exact RNG seeds used in each experiment are presented in Tables 1 and 2.

A.3 Lower and Upper Limits

In our model, each resource i may have a lower limit L_i and an upper limit U_i on the number of tasks it may receive.

In the case of the total cost minimization experiments in Appendix B, all resources have a lower limit equal to 5. The first half of the resources has no upper limit, while the second half cannot receive more than $2\frac{T}{n}$ tasks, for T

TABLE 1
Random number generator seeds used in the experiments of Appendix B.

Cost function	RNG Seeds
Random	[400,499]
Nlogn (increasing marginal costs)	[200,299]
Linear (constant marginal costs)	[100,199]
Logn (decreasing marginal costs)	[300,399]
Linear without upper limits	[500,599]

TABLE 2
Random number generator seeds used in the experiments of Appendix C.

Experiment	RNG Seeds
Fixed resources: Linear Costs	[100,199]
Fixed tasks: Linear Costs	[0,79]

tasks being scheduled and n resources. As this upper limit represents twice the average number of tasks per resource possible, FedAvg is guaranteed to provide valid schedules. Additionally, in the specific case of the experiments without upper limits, no resource has an upper limit.

In the case of the scheduling time experiments in Appendix C, all resources have a lower limit equal to 1 and the same upper limit behavior described for Appendix B. The presence of upper limits can lead to invalid schedules computed by MarDecUn, but this is not an issue in this situation, as we only care for execution times.

A.4 Hardware and Software

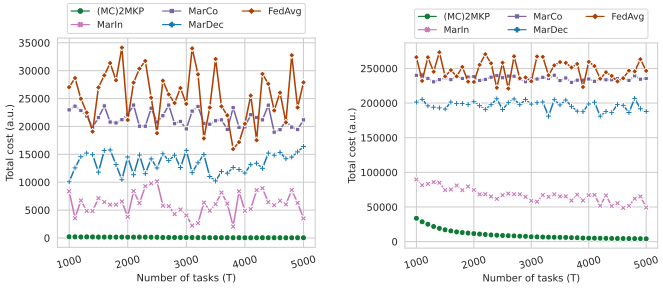
Hardware: experiments were executed on a Dell Latitude 7420 notebook with an 11th Gen Intel(R) Core(TM) i7-1185G7 processor, 32GB of LPDDR4X RAM (2133MHz), and a Western Digital PC SN530 NVMe WDC 512GB SSD. The computer was plugged to a power source at all times.

Software: the computer runs Ubuntu 20.04.5 LTS (kernel version 5.14.0-1054-oem). We used Python 3.8.10 with `numpy` version 1.22.2 for the experiments. Modules `matplotlib` (3.5.1), `pandas` (1.4.3), `seaborn` (0.11.2), and `scipy` (1.7.1) were used for the visualization and statistical analysis of results. During scheduling time experiments, no other applications were open besides a terminal.

Appendix B
Total Cost Minimization Results

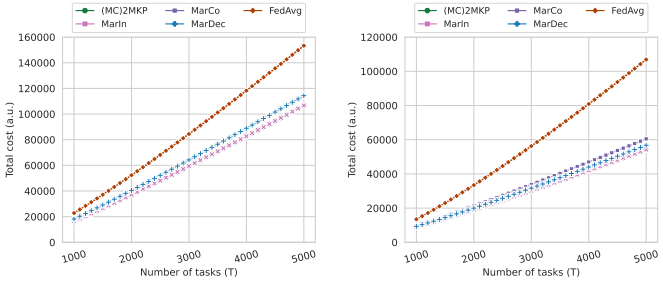
In this experiment, the algorithms have to schedule from $1,000$ to $5,000$ tasks (in increments of 100) over 10 or 100 resources. The heterogeneous resources are organized in five groups with the same cost function behavior. The first four groups contain random, nlogn, linear, and logn cost functions with lower and upper limits. The last group is composed of resources with linear cost functions and no upper limits in order to enable experiments with MarDecUn.

Figs. 1 to 5 summarize the total costs achieved by all scheduling algorithms in these scenarios. Each figure represents the results for one group and a given number of resources. Figures in the same pair contain results for resources in the same group of cost functions for different numbers of resources. The horizontal axis represents the number of tasks scheduled, and the vertical axis represents the total cost ΣC achieved by the algorithms (the lower, the better). Each scheduler is represented by a line connecting the total costs achieved for consecutive numbers of tasks. Each figure has its own scale for the vertical axis due to the particular costs of its group of resources.



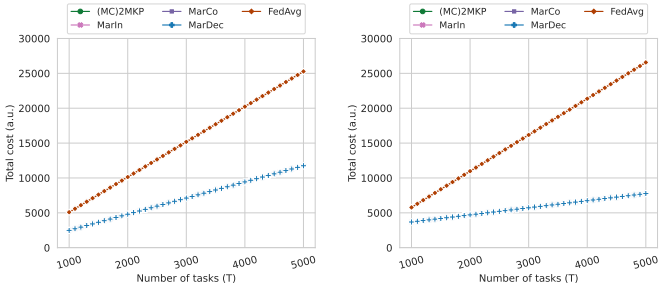
(a) $n = 10$. (b) $n = 100$.

Fig. 1. Total Costs achieved by the different scheduling algorithms for random costs.



(a) $n = 10$. (b) $n = 100$.

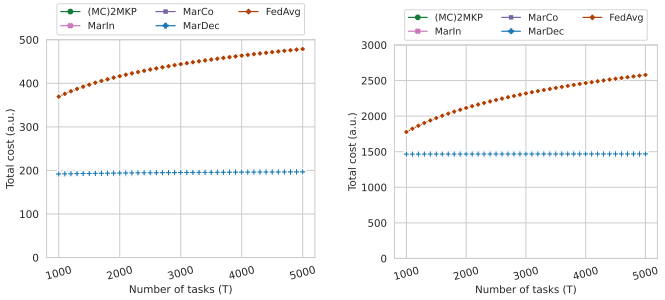
Fig. 2. Total Costs achieved by the different scheduling algorithms for nlogn costs (increasing marginal costs).



(a) $n = 10$. (b) $n = 100$.

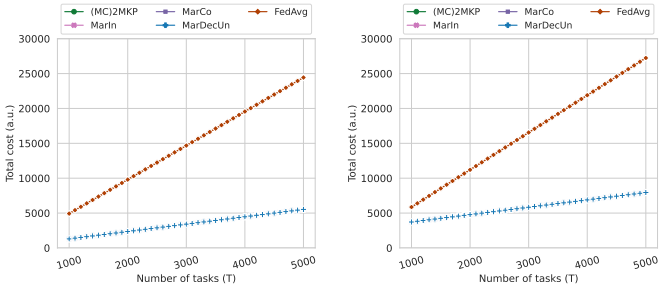
Fig. 3. Total Costs achieved by the different scheduling algorithms for linear costs (constant marginal costs).

At first glance, we may notice three main aspects of these results. First, FedAvg never finds optimal solutions. This is to be expected, as it does not take into consideration the cost functions of our resources. Second, only (MC)^{2}MKP is able to find optimal solutions for random cost functions in Fig. 1. Third, in many situations we are unable to see the total costs found by some schedulers in the figures. This happens because multiple schedulers are able to find optimal solutions



(a) $n = 10$. (b) $n = 100$.

Fig. 4. Total Costs achieved by the different scheduling algorithms for logn costs (decreasing marginal costs).



(a) $n = 10$. (b) $n = 100$.

Fig. 5. Total Costs achieved by the different scheduling algorithms for linear costs without upper limits.

in some scenarios, leading to lines that completely cover each other. Here are the cases where multiple algorithms find optimal solutions:

- **Increasing marginal costs** in Fig. 2: both (MC)^{2}MKP and MarIn find optimal (minimal) solutions.
- **Constant marginal costs** in Fig. 3: all algorithms besides FedAvg find optimal solutions. This happens because constant marginal costs are also monotonically increasing and monotonically decreasing, so MarIn, MarCo, and MarDec are able to optimize the schedule. The same is seen in Fig. 5 using MarDecUn when no upper limits are present.
- **Decreasing marginal costs** in Fig. 4: both (MC)^{2}MKP and MarDec find optimal solutions for both numbers of resources, while MarIn and MarCo are able to find optimal solutions for the scenario with $n = 10$. Nonetheless, we can see that MarIn and MarCo do not find optimal solutions when we take a closer look at the results of Fig. 4b, as illustrated in Fig. 6.

These results reinforce the importance of taking into consideration the cost behavior of all resources when assigning them tasks, as cost-oblivious algorithms can easily lead to much higher total costs (and, therefore, a higher energy consumption). Finally, whenever the costs are monotonically increasing and their marginal costs follow the same kind of behavior, we can choose to employ other optimal schedulers with lower complexity without fear of missing the best solutions.

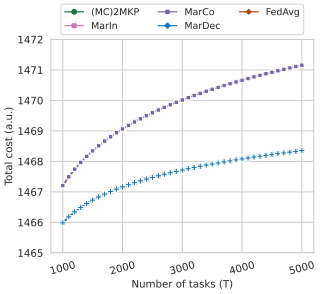
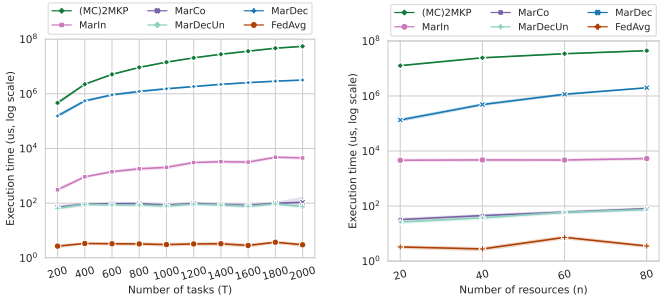


Fig. 6. Zoomed view of the total costs achieved with logn costs and $n = 100$. The upper line contains the results for MarIn and MarCo, while the lower line contains the optimal results found by (MC)^{2}MKP and MarDec.

Appendix C
Scheduling Time Results

The experiments in this scenario measure the impact in execution time seen for our scheduling algorithms when different numbers of tasks and resources are involved. Our intent here is to illustrate how the different time complexities lead to execution times at different scales. We may expect to see shorter execution times when implementing the algorithms using more optimized programming languages instead of Python.

Our experiments here are split into two. We first fix the number of resources at 100, and vary the tasks from 200 to $2,000$ in increments of 200, showing us how the number of tasks influences the execution time. Then, we fix the number of tasks at $2,000$, and vary the number of resources from 20 to 80 in increments of 20. All resources follow linear cost functions, as we assume that their costs should not have a major impact on the performance of the schedulers. For each triple $\langle$scheduler, tasks, resources$\rangle$, we gather 20 samples. Each sample is composed of 5 runs of a scheduler measured using Python's `timeit` module. The order that the samples are collected is randomized to reduce issues with interference and system jitter. The RNG seeds were set to 0 and $1,000$ when fixing the number of resources and tasks, respectively.



(a) $n = 100$. (b) $T = 2,000$

Fig. 7. Average scheduling algorithm times in microseconds (logarithmic scale).

The average execution times for each triple are presented in Fig. 7. The vertical axis represents the execution time for each scheduler (μs, in logarithmic scale), while the horizontal axis represents the number of tasks and the

TABLE 3
Average execution times and standard deviations with $n = 100$. Values are truncated.

	(MC)²MKP - $O(T^2n)$	**MarIn** - $\Theta(T \log n)$	**MarCo** - $\Theta(n \log n)$	**MarDecUn** - $\Theta(n)$	**MarDec** - $O(Tn^2)$	**FedAvg** - $O(n)$
200 tasks (only averages)	$(4.62 \pm 1.50) \times 10^5 \mu s$ $462\ ms$	$(3.08 \pm 1.00) \times 10^2 \mu s$ $308\ \mu s$	$(7.11 \pm 2.45) \times 10^1 \mu s$ $71.1\ \mu s$	$(6.49 \pm 2.02) \times 10^1 \mu s$ $64.9\ \mu s$	$(1.54 \pm 0.51) \times 10^5 \mu s$ $154\ ms$	$(2.68 \pm 0.85) \times 10^0 \mu s$ $2.68\ \mu s$
2,000 tasks (only averages)	$(5.42 \pm 0.07) \times 10^7 \mu s$ $54.2\ s$	$(4.51 \pm 1.34) \times 10^3 \mu s$ $4.51\ ms$	$(1.07 \pm 1.14) \times 10^2 \mu s$ $107\ \mu s$	$(7.42 \pm 2.27) \times 10^1 \mu s$ $74.2\ \mu s$	$(3.17 \pm 0.15) \times 10^6 \mu s$ $3.17\ s$	$(3.01 \pm 0.93) \times 10^0 \mu s$ $3.01\ \mu s$

TABLE 4
Average execution times and standard deviations with $T = 2,000$. Values are truncated.

	(MC)²MKP - $O(T^2n)$	**MarIn** - $\Theta(T \log n)$	**MarCo** - $\Theta(n \log n)$	**MarDecUn** - $\Theta(n)$	**MarDec** - $O(Tn^2)$	**FedAvg** - $O(n)$
20 resources (only averages)	$(1.26 \pm 0.02) \times 10^7 \mu s$ $12.6\ s$	$(4.62 \pm 1.19) \times 10^3 \mu s$ $4.62\ ms$	$(3.22 \pm 0.87) \times 10^1 \mu s$ $32.2\ \mu s$	$(2.61 \pm 0.73) \times 10^1 \mu s$ $26.1\ \mu s$	$(1.34 \pm 0.36) \times 10^5 \mu s$ $134\ ms$	$(3.26 \pm 0.82) \times 10^0 \mu s$ $3.26\ \mu s$
80 resources (only averages)	$(4.41 \pm 0.02) \times 10^7 \mu s$ $44.1\ s$	$(5.31 \pm 1.20) \times 10^3 \mu s$ $5.31\ ms$	$(7.99 \pm 1.69) \times 10^1 \mu s$ $79.9\ \mu s$	$(7.44 \pm 1.79) \times 10^1 \mu s$ $74.4\ \mu s$	$(1.98 \pm 0.04) \times 10^6 \mu s$ $1.98\ s$	$(3.53 \pm 0.84) \times 10^0 \mu s$ $3.53\ \mu s$

number of resources in Figs. 7a and 7b, respectively. Each scheduler is represented by a line connecting their execution times achieved for consecutive cases. The average times for each scheduler for the smallest and largest cases are also presented in Tables 3 and 4, together with their standard deviations. For a complete view of the times, please check the dataset available online [1].

As it can be noticed in Fig. 7 and Tables 3 and 4, the algorithms' execution times vary by over seven orders of magnitude for the tested numbers of tasks and resources. This is to be expected given the huge differences in their time complexities. Additionally, all execution times follow the expected behaviors given the algorithms' complexities.

(MC)^{2}MKP ($O(T^2n)$) is visibly the slowest algorithm, with execution times varying between the hundreds of milliseconds and the tens of seconds. When we compare it to MarDec ($O(Tn^2)$), we can see that their times are similar when the number of tasks and resources are similar too (first row of Table 3). However, when we increase the number of tasks by a factor of ten (third row in the same table), (MC)^{2}MKP's time increases by a factor of a hundred, while MarDec's time only increases by a factor of ten.

MarDecUn and FedAvg show a difference of about one order of magnitude in their execution times, even though they are both linear in the number of resources. This happens because they require very different operations. MarDecUn requires looping over the resources to assign the lower limits to all resources and to find the one with the smallest marginal cost. Meanwhile, FedAvg can directly assign the same number of tasks to all resources using an optimized `numpy` operation, leading to a faster execution.

We may also notice that the execution times achieved by MarDecUn and MarCo are visually very similar. Owing to their similarity, we have chosen to employ statistical methods to verify if there are performance differences between the two algorithms. Using the Mann-Whitney U test with 5% confidence, we have concluded that their execution times are different (i.e., all comparisons reject H0 with p-values < 0.05, meaning that they come from different distributions). This non-parametric test was chosen because most of the sampled results did not come from normal distributions (Kolmogorov-Smirnov tests with p-values < 0.05).

These results, in addition to the results from Appendix B, really emphasize the benefits of choosing the more optimized algorithms from Section 5 whenever possible. Not only do they provide optimal solutions when the cost functions behave according to their assumptions, they are also capable of doing so from one to six orders of magnitude faster than our general optimal algorithm.

REFERENCES

[1] L. L. Pilla, "llpilla/energy-optimal-federated-learning: Full set of algorithms: (MC)²MKP, MarIn, MarCo, MarDec, and MarDecUn," Nov. 2022. [Online]. Available: https://doi.org/10.5281/zenodo.7377419

[2] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Singh and J. Zhu, Eds., vol. 54. PMLR, 20–22 Apr 2017, pp. 1273–1282. [Online]. Available: https://proceedings.mlr.press/v54/mcmahan17a.html