Combinatorial Optimization Approach to Client Scheduling for Federated Learning

Tomohito Omori[1] and Kenji Kashima[1], *Senior Member, IEEE*

***Abstract*— For machine learning in situations where data is scattered and cannot be aggregated, federated learning, in which aggregators and agents send and receive model parameters, is one of the most promising methods. The scheduling problem of deciding which agents to communicate with has been studied in various ways, but it is not easy to solve due to its combinatorial optimization nature. In this letter, we attempt to solve this scheduling problem using combinatorial optimization theory. Specifically, we propose an efficient exact solution method based on dynamic programming and a greedy method whose superiority is confirmed by numerical examples. We also discuss the applicability of the proposed methods to a more dynamic and uncertain environment.**

I. Introduction

In recent years, there have been many situations where system models are learned from data. In addition, it is often impossible to consolidate data on a single machine due to privacy and computing resources. In these situations, *federated learning* is a method of finding the optimal model parameters for the entire data by building a centralized network and sending and receiving only the estimated model parameters, rather than sending the data to a central server [1]. Various studies have been conducted on federated learning [2], including one that formulated the client scheduling problem and discussed convergence [3] and how to implement federated learning on mobile devices [4]–[6].

Given the recent strong interest in data-driven control, there is no doubt that federated learning can be applied to many problems in the control community; See, e.g., Example 1. Such federated learning in dynamical and uncertain environment requires more efficient scheduling algorithms. The main feature of this letter is to propose an approach based on combinatorial optimization theory [7] for this purpose. Specifically, we give an efficient exact solution and a greedy method for the client scheduling problem in [8]. The proposed method outperforms conventional methods in terms of computational complexity and accuracy and can also be deployed in a more realistic federated learning problem setting.

This letter is organized as follows: Section II provides an overview of the federated learning and client scheduling problem and shows that the problem is NP-hard. In Section III, we consider the scheduling problem as a Knapsack problem and propose an efficient exact solution using dynamic programming. In Section IV, we propose a greedy algorithm whose superiority is confirmed by numerical examples. In Section V, we summarize this letter and discuss the applicability of the obtained result to federated learning in an uncertain environment.

Notation For integers k, l, let

$$[k,l] := \begin{cases} \{k, k+1, \dots, l\} & \text{if } k \le l \\ \emptyset & \text{otherwise.} \end{cases}$$

For a vector c and an integer k, let $c(k)$ be a k-th element of c. For an ordered set $\boldsymbol{I} := \langle i_1, \dots, i_m \rangle$, $\boldsymbol{I}^j$ (resp. ${}^j\boldsymbol{I}$) is the j-letter prefix (resp. $(m-j+1)$-letter suffix) of $\boldsymbol{I}$, i.e.,

$$\boldsymbol{I}^j := \langle i_1, \dots, i_j \rangle \tag{1}$$

$${}^j\boldsymbol{I} := \langle i_j, \dots, i_m \rangle \tag{2}$$

and $\boldsymbol{I}^0 = {}^{m+1}\boldsymbol{I} = \langle\rangle$. For two ordered sets $\boldsymbol{I}_1 = \langle i_1, \dots, i_a \rangle$, $\boldsymbol{I}_2 = \langle j_1, \dots, j_b \rangle$ with $\boldsymbol{I}_1 \cap \boldsymbol{I}_2 = \emptyset$, let $\boldsymbol{I}_1 \oplus \boldsymbol{I}_2$ be the ordered set $\langle i_1, \dots, i_a, j_1, \dots, j_b \rangle$.

II. Preliminaries: Client scheduling for federated learning

A. Federated Learning

Suppose there is an aggregator that serves as the central server, multiple agents communicate with it, and each agent stores a data set. The objective is to learn the model parameters from the data without collecting the data in the aggregator.

Example 1: Suppose a vendor offers the same system to its agents, which is expressed as the FIR model

$$y_k = \sum_{l=0}^{N-1} w_l u_{k-l} + d_k, \tag{3}$$

where y_k is the output, u_k is the input, d_k is the white noise, and w_l (common among agents) is the coefficients. Suppose that the agent i has input time series $\{u_{i,k}\}_{k=1}^{D_i}$, output time series $\{y_{i,k}\}_{k=1}^{D_i}$. Our goal is to identify the common system parameter w suitable for all data $\cup_{i=1}^{n}(\{u_{i,k}\}_{k=1}^{D_i}, \{y_{i,k}\}_{k=1}^{D_i})$ by minimizing the loss function (log-likelihood)

$$F(w) := \frac{1}{D_{\text{all}}} \sum_{i=1}^{n} D_i F_i(w), \quad D_{\text{all}} := \sum_{i=1}^{n} D_i,$$

where

$$F_i(w) := \frac{1}{2D_i} \sum_{k=N}^{D_i} \| y_{i,k} - w^\top u_{i,k-N+1:k} \|^2.$$

[1]T. Omori and K. Kashima are with the Graduate School of Informatics, Kyoto University, Kyoto, Japan omori.tomohito.58z@st.kyoto-u.ac.jp; kk@i.kyoto-u.ac.jp This work was supported by JSPS KAKENHI Grant Number JP21H04875.

Algorithm 1 Federated Learning

```
Require: $\tau$, $\kappa$
Ensure: $w_f$
1: Initialize $w_f, w(1)$ to the same value
2: for $t = 1$ to $\tau$ do
3:     Set $\tilde{w}_i \leftarrow w(t)$ as agent $i$'s initial value
4:     for $k = 1$ to $\kappa$ do
5:         $\tilde{w}_i \leftarrow \tilde{w}_i - \eta \nabla F_i(\tilde{w}_i)$ for all agent $i$
6:     end for
7:     Set $w(t+1) \leftarrow \frac{1}{D_{\text{all}}} \sum_{i=1}^{n} D_i \tilde{w}_i$ as aggregator's next value
8:     $w_f \leftarrow \arg\min_{w \in \{w_f, w(t)\}} F(w)$
9: end for
```

However, this optimization must be done without direct data sharing of $\{u_{i,k}\}$, $\{y_{i,k}\}$ between the agent and vendor. □

The learning algorithm is as in Algorithm 1. Learning consists of τ iterations. In the t-th iteration, the aggregator sends the estimated model parameter $w(t)$ to all agents. Each agent takes $w(t)$ as its initial value and updates it κ times in gradient steps to make $F_i(w)$ smaller. Afterward, each agent sends the updated estimated model parameters $\tilde{w}_i$ to the aggregator. The aggregator takes these estimated model parameters and uses their weighted average over the number of data as the estimated model parameters $w(t+1)$ in the next iteration. After repeating this iteration τ times, the estimated parameter w_f is the one with the minor overall loss function $F(w(t))$ among the estimated parameters $w(t)$ stored in the aggregator so far and is output as the parameter estimated by the training.

B. Client scheduling

In prior work [8], when the computation time of the agent and the communication speed between the agent and the aggregator were different from each other, they formulated the problem of which agents to collect data from and in which order, instead of collecting data from the entire agent, in the part where the aggregator receives $\tilde{w}_i$ from each agent. Specifically, given the computation time $c(i)$ for agent i, the time $u(i)$ required to send model parameters to the aggregator, and the number of data D_i stored in agent i in advance, the problem is formulated in such a way that the data collection time $H(\boldsymbol{I})$ does not exceed T. The problem is finding the order $\boldsymbol{I}$ to collect the data from the agents so that the time to collect the data $H(\boldsymbol{I})$ is at most T.

Problem 1: Given $D \in \mathbb{Z}^n_{\geq 0}$, $u, c \in \mathbb{R}^n_{\geq 0}$, and $T \in \mathbb{R}_{\geq 0}$,

$$\begin{aligned} \underset{\boldsymbol{I}}{\text{maximize}} \quad & \sum_{i \in \boldsymbol{I}} D_i \\ \text{subject to} \quad & H(\boldsymbol{I}) \leq T, \end{aligned}$$

where $H(\langle\rangle) := 0$, and

$$H(\boldsymbol{I}^k) := \max(H(\boldsymbol{I}^{k-1}), c(i_k)) + u(i_k). \tag{4}$$

△

The expression (4) reflects that if data is to be collected from agent $i_1, \ldots, i_k$ in this order, it is necessary to collect from agent i_k that data from agent $i_1, \ldots, i_{k-1}$ has already been collected, and only after agent i_k is calculated. The previous study treated this problem as a Submodular Cost Submodular Knapsack problem (SCSK), i.e.

$$\max g(X) \text{ s.t. } f(X) \leq b, X \subseteq \{1, \ldots, n\} \tag{5}$$

where f and g are monotone non-decreasing submodular functions, and proposed an approximate solution using the greedy method.

C. NP-hardness

In the previous study [8], Problem 1 is explained as a particular case of the SCSK problem, which is generally NP-hard [7, p.410]. Indeed, the max k-cover problem [9], an example of the SCSK problem, is known to be NP-hard. However, not all SCSK problems are NP-hard; see the problem with $f(X) = g(X) = |X|$ in the equation (5), which is also an example of SCSK problem and is clearly solvable in polynomial time. In view of this, we rigorously prove the NP-hardness.

Theorem 1: Problem 1 is NP-hard.

Proof: This can be proved by polynomial reduction [7, p.396, p.412] from the Knapsack problem [7, p.471]. The Knapsack problem is written by

$$\begin{aligned} \underset{\mathcal{S} \subseteq [1,n]}{\text{maximize}} \quad & \sum_{i \in \mathcal{S}} \delta_i \\ \text{subject to} \quad & \sum_{i \in \mathcal{S}} \upsilon_i \leq \mathcal{T} \end{aligned} \tag{6}$$

for given $\delta, \upsilon \in \mathbb{Z}^n_{\geq 0}$ and $\mathcal{T} \in \mathbb{Z}_{\geq 0}$. This means that any optimal solution of Problem 1 with $T = \mathcal{T}, c(i) = 0, u(i) = \upsilon_i, D_i = \delta_i$ can be viewed as an optimal solution of the Knapsack problem (6) by regarding it as an unordered set. In other words, Problem 1 involves the Knapsack problem as a special case. Therefore, since the Knapsack problem is NP-hard, Problem 1 is also NP-hard. ■

Therefore, we can say that when P $\neq$ NP, it is impossible to obtain an exact solution in polynomial time with an agent number n. A naïve dynamic programming approach to Problem 1 requires $O(n!T)$ time[1], which is too large for large-scale problems. In the next section, we will propose algorithms of the 1st order with respect to n.

III. Dynamic programming for Knapsack problem

A. Problem conversion via schedule sorting

Problem 1 looks different from the Knapsack problem because the constraints include a max function. We change the constraints to linear and transform the problem so that it is easier to see how to apply the solution method used in the Knapsack problem.

Problem 2: Given $D \in \mathbb{Z}^n_{\geq 0}$, $u, c \in \mathbb{R}^n_{\geq 0}$, and $T \in \mathbb{R}_{\geq 0}$,

$$\begin{aligned} \underset{\boldsymbol{I}}{\text{maximize}} \quad & \sum_{i \in \boldsymbol{I}} D_i \\ \text{subject to} \quad & \sum_{i \in^j \boldsymbol{I}} u(i) \leq T - c(i_j) \quad (\forall j \in [1, |\boldsymbol{I}|]). \end{aligned} \tag{7}$$

[1]For example, calculate the maximum amount $\check{d}(S, t)$ of data that can be computed by t when the entire agent set is S.

△

This problem is equivalent to Problem 1 in the following sense:

Theorem 2: Problems 1 and 2 have the same feasible solution set, that is,

$$H(\boldsymbol{I}) \le T \Leftrightarrow \sum_{i \in^j \boldsymbol{I}} u(i) \le T - c(i_j) \quad (\forall j \in [1, |\boldsymbol{I}|]).$$

□

Proof: ($\Leftarrow$ direction) Note that $H(\langle\rangle) = 0 \le c(i_1)$. This implies we can take the largest index $k \in [0, |\boldsymbol{I}| - 1]$ such that $H(\boldsymbol{I}^k) \le c(i_{k+1})$. Then,

$$H(\boldsymbol{I}) = c(i_{k+1}) + \sum_{j=k+1}^{|\boldsymbol{I}|} u(i_j) = c(i_{k+1}) + \sum_{i \in^{k+1} \boldsymbol{I}} u(i) \le T,$$

which shows the desired result.

($\Rightarrow$ direction) By definition, we have

$$H(\boldsymbol{I}^{k-1}) + u(i_k) \le H(\boldsymbol{I}^k),\ c(i_k) + u(i_k) \le H(\boldsymbol{I}^k),$$

for all k in $[1, |\boldsymbol{I}|]$. Using this repeatedly, it follows that

$$\begin{aligned} \sum_{i \in^j \boldsymbol{I}} u(i) + c(i_j) &\le H(\boldsymbol{I}^j) + \sum_{i \in^{j+1} \boldsymbol{I}} u(i) \\ &\le H(\boldsymbol{I}^{|\boldsymbol{I}|}) = H(\boldsymbol{I}) \le T \end{aligned}$$

for all j in $[1, |\boldsymbol{I}|]$. ■

The following theorem plays a key role to apply the solution of the Knapsack problem to Problem 2.

Theorem 3: Let $\boldsymbol{I}$ be any feasible solution of Problem 2. Then, the ordered set obtained by reordering $\boldsymbol{I}$ in ascending order of $c(i)$ is also feasible for Problem 2. □

Proof: Let $\boldsymbol{I}$ be a feasible solution that is not in ascending order in $c(i)$, that is, $\exists k$ s.t. $c(i_k) \ge c(i_{k+1})$. We first show that the ordered set $\tilde{\boldsymbol{I}} := \langle \tilde{i}_1, \ldots, \tilde{i}_{|\boldsymbol{I}|} \rangle := \langle i_1, \ldots, i_{k-1}, i_{k+1}, i_k, i_{k+2}, \ldots, i_{|\boldsymbol{I}|} \rangle$ is a feasible solution. In other words, we show that

$$\begin{aligned} &\sum_{\pi \in^j \boldsymbol{I}} u(i) \le T - c(i_j) \quad (\forall j \in [1, |\boldsymbol{I}|]) \\ &\Rightarrow \sum_{i \in^{\tilde{j}} \tilde{\boldsymbol{I}}} u(i) \le T - c(\tilde{i}_{\tilde{j}}) \quad (\forall \tilde{j} \in [1, |\boldsymbol{I}|]). \end{aligned} \tag{8}$$

For $\tilde{j} \notin \{k, k+1\}$, this is trivial since

$$\sum_{i \in^{\tilde{j}} \boldsymbol{I}} u(i) = \sum_{i \in^{\tilde{j}} \tilde{\boldsymbol{I}}} u(i),\ c(i_{\tilde{j}}) = c(\tilde{i}_{\tilde{j}}).$$

For $\tilde{j} = k, k+1$,

$$c(i_k) = \max(c(\tilde{i}_k), c(\tilde{i}_{k+1})),\quad \sum_{i \in^{k+1} \tilde{\boldsymbol{I}}} u(i) \le \sum_{i \in^k \tilde{\boldsymbol{I}}} u(i)$$

can be used to obtain

$$\sum_{i \in^k \tilde{\boldsymbol{I}}} u(i) + c(\tilde{i}_k) \le \sum_{i \in^k \boldsymbol{I}} u(i) + c(i_k) \le T, \tag{9}$$

$$\sum_{i \in^{k+1} \tilde{\boldsymbol{I}}} u(i) + c(\tilde{i}_{k+1}) \le \sum_{i \in^k \boldsymbol{I}} u(i) + c(i_k) \le T. \tag{10}$$

This completes the proof of (8).

Repeatedly swapping i_k and i_{k+1} such that $c(i_k) > c(i_{k+1})$, we obtain a feasible solution sorted in ascending order by $c(i)$ like a bubble sort. ■

Theorem 3 shows that it is sufficient to consider only the schedules in ascending order of $c(i)$. For notational simplicity, we assume that $c \in \mathbb{R}^n_{\ge 0}$ is *descending*, i.e.,

$$c(i) \ge c(j) \quad \text{if} \quad i \le j,$$

in the remaining of Section III. This assumption is not restrictive because we only need to relabel i accordingly. Under this assumption, once an agent set to communicate with is determined, the order can be determined in the descending order of i, which leads to the following:

Problem 3: Given $D \in \mathbb{Z}^n_{\ge 0}$, $u \in \mathbb{R}^n_{\ge 0}$, descending $c \in \mathbb{R}^n_{\ge 0}$, and $T \in \mathbb{R}_{\ge 0}$,

$$\begin{aligned} \underset{S \subseteq [1,n]}{\text{maximize}} \quad & \sum_{i \in S} D_i \\ \text{subject to} \quad & \sum_{i \in S, i \le j} u(i) \le T - c(j) \quad (\forall j \in S). \end{aligned} \tag{11}$$

△

This problem is equivalent to Problem 2 in the following sense:

Corollary 1: Let S be any optimal solution of Problem 3. Then, the ordered set $\boldsymbol{I}$ obtained by ordering[2] S in ascending order of $c(i)$ is an optimal solution for Problem 2. □

Since the decision variable is now an unordered set, we can use dynamic programming, which is used as a strict solution method for the Knapsack problem; See Subsections III-B and III-C. Furthermore, a greedy method exploiting this property produces better results than previous studies; See Subsection IV-A.

B. Upload time-based algorithm

In this section, we consider Problem 3 as a Knapsack problem. The standard method for the Knapsack problem [7, p.412, p.475] motivates us to consider the following:

Problem 4: Let $D \in \mathbb{Z}^n_{\ge 0}$, $u \in \mathbb{R}^n_{\ge 0}$, descending $c \in \mathbb{R}^n_{\ge 0}$, and $T \in \mathbb{R}_{\ge 0}$ be given. Then, for an integer $k \in [0, n]$ and a nonnegative real number $y \le T$,

$$\begin{aligned} \underset{S \subseteq [1,k]}{\text{maximize}} \quad & \sum_{i \in S} D_i \\ \text{subject to} \quad & \sum_{i \in S, i \le j} u(i) \le T - c(j) \quad (\forall j \in S), \\ & \sum_{i \in S} u(i) = y. \end{aligned} \tag{12}$$

△

Let $P(k, y)$ be Problem 4 and $d(k, y)$ be its optimal value. Problem 4 differs from Problem 3 in two ways: the sum of upload times is fixed to y, and the entire agent set is limited to $[1, k]$. The optimal value of Problem 3 is equal to

$$\max\{d(n, y) \mid 0 \le y \le T\}. \tag{13}$$

[2] $\boldsymbol{I} = \langle i_1, \ldots, i_{|\boldsymbol{I}|} \rangle$ with $\{i_1, \ldots, i_{|\boldsymbol{I}|}\} = S$ and $c(i_j) \le c(i_k) (\forall j, k \in [1, |\boldsymbol{I}|], j < k)$.

Also, since $[1, k-1] \subset [1, k]$, we have

$$d(k, y) \geq d(k-1, y) \ (\forall k \geq 1, y). \tag{14}$$

Since the constraint in Problem 2 affects only the suffix of the decision variable, we schedule from the end, i.e., in the descending order of c. Under this strategy, whether we can add an agent k to a feasible set S depends only on $\sum_{i \in S} u(i)$, which leads to the following:

Theorem 4: Under notation above, $d(k, y)$ satisfies

$$d(k, y) = \begin{cases} \max(d(k-1, y), d(k-1, y-u(k)) + D_k) \\ \qquad\qquad \text{if } u(k) \leq y \leq T - c(k), \\ d(k-1, y) \\ \quad \text{otherwise}, \end{cases} \tag{15}$$

where

$$d(0, y) = \begin{cases} 0 & \text{if } y = 0, \\ -\infty & \text{otherwise}. \end{cases}$$

□

Proof: Suppose $u(k) \leq y \leq T - c(k)$. Then, for any optimal solution $\tilde{S}$ to $P(k-1, y-u(k))$, $\tilde{S} \cup \{k\}$ is a feasible solution to $P(k, y)$ because $\sum_{i \in \tilde{S} \cup \{k\}} u(i) \leq T - c(k)$ holds. This fact combined with (14) yields that $d(k, y)$ is larger than or equal to the right side of (15).

Next, we show $d(k, y)$ is not larger than the right side of (15) by contradiction. Let $\tilde{S}$ be an optimal solution of problem $P(k, y)$. If $k \notin \tilde{S}$, then $\tilde{S}$ is feasible for problem $P(k-1, y)$. This contradicts the premise $d(k, y) > d(k-1, y)$. If $k \in \tilde{S}$, then $u(k) \leq y \leq T - c(k)$ holds and $\tilde{S} \setminus \{k\}$ is feasible for problem $P(k-1, y-u(k))$. This means $d(k-1, y-u(k)) \geq d(k, y) - D_k$, which is also a contradiction. This completes the proof. ■

This theorem shows that $d(k, y)$ can be obtained by dynamic programming.

For implementation, we impose the following:

Assumption 1: Under the notation above, time-related variables $c(i), u(i), T$ are integer-valued.

Then, we can obtain $d(k, y)$ and an optimal solution of Problem 4 by dynamic programming as summarized in Algorithm 2, which also solves Problem 3 by virtue of (13). The computational complexity of this algorithm is $O(nT)$.

Algorithm 2 Upload time-based algorithm

```
Require: $D \in \mathbb{Z}^n_{\geq 0}$, $u \in \mathbb{Z}^n_{\geq 0}$, descending $c \in \mathbb{Z}^n_{\geq 0}$, $T \in \mathbb{Z}_{\geq 0}$
Ensure: the optimal value $d^*$ and optimal solution $\boldsymbol{I}^*$
 1: $d(0, 0) \leftarrow 0$
 2: $d(0, y) \leftarrow -\infty \quad (1 \leq \forall y \leq T)$
 3: for $k = 1$ to $n$ do
 4:     for $y = 0$ to $T$ do
 5:         if $u(k) \leq y$ and $y + c(k) \leq T$ then
 6:             $d(k, y) \leftarrow \max(d(k-1, y), d(k-1, y-u(k)) + D_k)$
 7:         else
 8:             $d(k, y) \leftarrow d(k-1, y)$
 9:         end if
10:     end for
11: end for
12: $d^* \leftarrow \max_j(d(n, j))$
13: $\boldsymbol{I}^* \leftarrow \langle\rangle$
14: $y \leftarrow \arg\max_j(d(n, j))$
15: for $k = n$ down to $1$ do
16:     if $d(k-1, y) \neq d(k, y)$ then
17:         $y \leftarrow y - u(k)$
18:         $\boldsymbol{I}^* \leftarrow \boldsymbol{I}^* \oplus \langle k \rangle$
19:     end if
20: end for
```

C. Data amount-based algorithm

In this section, we provide another way of dynamic programming where Assumption 1 is not required.

Problem 5: Let $D \in \mathbb{Z}^n_{\geq 0}$, $u \in \mathbb{R}^n_{\geq 0}$, descending $c \in \mathbb{R}^n_{\geq 0}$, and $T \in \mathbb{R}_{\geq 0}$ be given. Then, for integers $k \in [0, n]$ and $z \in [0, D_{\text{all}}]$,

$$\begin{aligned} \underset{S \subseteq [1:k]}{\text{minimize}} \quad & \sum_{i \in S} u(i) \\ \text{subject to} \quad & \sum_{i \in S, i \leq j} u(i) \leq T - c(j) \quad (\forall j \in S), \\ & \sum_{i \in S} D_i = z. \end{aligned} \tag{16}$$

△

Let $\bar{P}(k, z)$ be Problem 5 and $\bar{d}(k, z)$ be its optimal value. Comparing Problems 4 and 5, the objective function and a constraint are interchanged so that

$$d(k, y) = \max_{z \in [0, D_{\text{all}}]} \{z \mid \bar{d}(k, z) \leq y\}. \tag{17}$$

The optimal value of Problem 3 is equal to

$$z^* := \max\{z \mid \bar{d}(n, z) \leq T\} \tag{18}$$

and any optimal solution to $\bar{P}(n, z^*)$ gives an optimal solution to Problem 3. We also have

$$\bar{d}(k, z) \leq \bar{d}(k-1, z) \ (\forall k \geq 1, z). \tag{19}$$

Problem 5 can also be solved using dynamic programming based on the following:

Theorem 5: Under notation above, $\bar{d}(k, z)$ satisfies

$$\bar{d}(k, z) = \begin{cases} \min(\bar{d}(k-1, z), \bar{d}(k-1, z-D_k) + u(k)) \\ \qquad \text{if } \bar{d}(k-1, z-D_k) + u(k) + c(k) \leq T, \\ \bar{d}(k-1, z) \\ \quad \text{otherwise} \end{cases} \tag{20}$$

where

$$\bar{d}(0, z) = \begin{cases} 0 & \text{if } z = 0, \\ \infty & \text{otherwise}. \end{cases}$$

□

Proof: This theorem can be shown in the same way as Theorem 4. Suppose $\bar{d}(k-1, z-D_k) + u(k) + c(k) \leq T$.

Algorithm 3 Data amount-based algorithm

```
Require: $D \in \mathbb{Z}^n_{\geq 0}$, $u \in \mathbb{R}^n_{\geq 0}$, descending $c \in \mathbb{R}^n_{\geq 0}$, $T \in \mathbb{R}_{\geq 0}$
Ensure: the optimal value $\bar{d}^*$ and an optimal solution $\boldsymbol{I}^*$
 1: $\bar{d}(0,0) \leftarrow 0$
 2: $\bar{d}(0,z) \leftarrow \infty \quad (1 \leq \forall z \leq D_{\text{all}})$
 3: for $k = 1$ to $n$ do
 4:     for $z = 0$ to $D_{\text{all}}$ do
 5:         if $\bar{d}(k-1, z-D_k) + u(k) + c(k) \leq T$ then
 6:             $\bar{d}(k,z) \leftarrow \min(\bar{d}(k-1,z), \bar{d}(k-1,z-D_k) + u(k))$
 7:         else
 8:             $\bar{d}(k,z) \leftarrow \bar{d}(k-1,z)$
 9:         end if
10:     end for
11: end for
12: $\bar{d}^* \leftarrow \max_j(\bar{d}(n,j))$
13: $\boldsymbol{I}^* \leftarrow \langle\rangle$
14: $z \leftarrow \arg\max_j(\bar{d}(n,j))$
15: for $k = n$ down to $1$ do
16:     if $\bar{d}(k-1,z) \neq \bar{d}(k,z)$ then
17:         $z \leftarrow z - u(k)$
18:         $\boldsymbol{I}^* \leftarrow \boldsymbol{I}^* \oplus \langle k \rangle$
19:     end if
20: end for
```

Then, for any optimal solution $\tilde{S}$ to $\bar{P}(k-1, z-D_k)$, $\tilde{S} \cup \{k\}$ is a feasible solution to $\bar{P}(x,y)$ because $\sum_{i \in \tilde{S} \cup \{k\}} u(i) \leq T - c(k)$ holds. This fact combined with (19) yields that $\bar{d}(k,z)$ is less than or equal to the right side of (20).

Next, we show $\bar{d}(k,z)$ is not less than the right side of (20) by contradiction. Let $\tilde{S}$ be an optimal solution of problem $\bar{P}(k,z)$. If $k \notin \tilde{S}$, then $\tilde{S}$ is feasible for problem $\bar{P}(k-1,z)$. This means contradicts the premise $\bar{d}(k,z) < \bar{d}(k-1,z)$. If $k \in \tilde{S}$, then $\bar{d}(k-1,z-D_k) + u(k) + c(k) \leq T$ holds and $\tilde{S} \setminus \{k\}$ is feasible for problem $\bar{P}(k-1, z-D_k)$. Consequenty $\bar{d}(k-1,z-D_k) \leq \bar{d}(k,z) - u(k)$, which is also a contradiction. This completes the proof. ∎

Therefore, the optimal value $\bar{d}(k,z)$ and an optimal solution of Problem 5 can be obtained by dynamic programming as summarized in Algorithm 3, which also solves Problem 3 by (18). The computational complexity of this algorithm is $O(nD_{\text{all}})$.

Remark 1: Note that, differently from Algorithm 2, Assumption 1 is unnecessary for implementing Algorithm 3. The essential reason for this is that D_i is integer-valued. Conversely, Algorithm 2 is applicable to the cases with non-integer valued D_i, e.g., where the value D_i of agent i is quantified in ways other than by its data amount, as long as Assumption 1 is satisfied.

IV. Improvement of Greedy Algorithm

A. Greedy algorithm for Knapsac problem

A well-known approximate solution for the Knapsack Problem (6) is a greedy method that sorts $\frac{v_i}{w_i}$ in descending order and chooses greedily. If we use this method for Problem 2, it becomes like Algorithm 4. This method greedily selects agents in descending order of $\frac{D_i}{u(i)}$. That is, if the solution is feasible when adding it, we include it (if it violates a constraint, we move on to the second largest candidate). It should be emphasized that each examination can be efficiently implemented as follows:

Algorithm 4 Proposed greedy approximation algorithm

```
Require: $D \in \mathbb{Z}^n_{\geq 0}$, $u, c \in \mathbb{R}^n_{\geq 0}$, $T \in \mathbb{R}_{\geq 0}$
Ensure: $S$
1: $W \leftarrow [1, n]$
2: while $W \neq \emptyset$ do
3:     $i \leftarrow \arg\max_{i \in W} \frac{D_i}{u(i)}$
4:     if $S \cup \{i\}$ is feasible to Problem 3 then
5:         $S \leftarrow S \cup \{i\}$
6:     end if
7:     $W \leftarrow W \setminus \{i\}$
8: end while
```

Theorem 6: In Line 4 of Algorithm 4, it can be determined by $O(\log n)$ whether $S \cup \{i\}$ is a feasible solution for Problem 2. Consequently, the overall computational complexity of Algorithm 4 is $O(n \log n)$.

Proof: Due to page limitations, we offer only an outline of the proof. Assume that c is descending. The feasibility can be checked by $\max\{C_i\} \leq T$, owing to recording

$$C_i := \begin{cases} \sum_{j \in S, j \leq i} u_i & (i \notin S), \\ c_i + \sum_{j \in S, j \leq i} u_i & (i \in S). \end{cases}$$

When add an agent i to a set S, we just update that $C_i \leftarrow C_i + u_i + c_i$ and $C_j \leftarrow C_j + u_i$ $(i < \forall j)$. It is known that for a given sequence of length n (e.g., $\{C_i\}_i$), finding its maximum and adding a common value (e.g., u_i) to a consecutive subsequence (e.g., $\{C_j\}_{j>i}$) can be implemented in $O(\log n)$ time by using an appropriate data structure such as Segment Tree [10, p.430]. ∎

This theorem shows that Algorithm 4 is superior to SCSK-based greedy algorithm in [8] regarding computational complexity, whose order is $O(n^2)$. The accuracy will be demonstrated in the next subsection.

B. Numerical example

In this section, we exhibit a numerical experiment. We conducted 50 experiments where we took the number of agents $n = 200$, the time limit $T = 3000$,

$$c(i) = c_a D_i + \alpha c_b, \; u(i) = u_a D_i.$$

The random variables $D_i \sim \text{Uni}(1,100)_{\mathbb{Z}}$, $c_a \sim \text{Uni}(24,27)_{\mathbb{R}}$, $c_b \sim \text{Uni}(1,2)_{\mathbb{R}}$ and u_a obeying the exponential distribution with the mean 0.6 were generated independently for each experiment[3]. The scale parameter α of the computational overhead is $\alpha = 0.1, 50, 400$.

[3]$\text{Uni}(a,b)_{\mathbb{S}}$ denotes the uniform distribution over the set $\{c \in \mathbb{S} | a \leq c \leq b\}$.

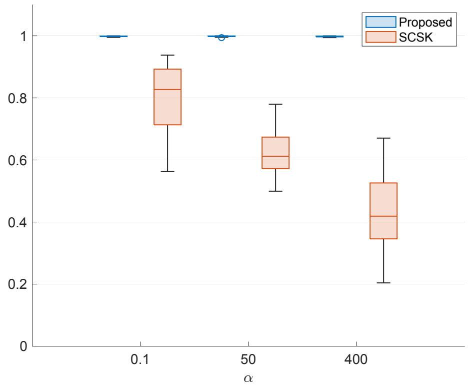

Fig. 1: D_{SCSK}/D^* and D_{greedy}/D^* for $\alpha = 0.1, 50, 400$.

Let us denote the solution obtained by the dynamic programming (Algorithm 3) by $\boldsymbol{I}_{\text{DP}}$, the proposed greedy method (Algorithm 4) by $\boldsymbol{I}_{\text{greedy}}$, and the conventional SCSK-based greedy algorithm by $\boldsymbol{I}_{\text{SCSK}}$. Note that $D^* := \sum_{i\in\boldsymbol{I}_{\text{DP}}} D_i$ is equal to the optimal value of Problem 1. Figure 1 shows $D_{\text{SCSK}} := \sum_{i\in\boldsymbol{I}_{\text{SCSK}}} D_i$ and $D_{\text{greedy}} := \sum_{i\in\boldsymbol{I}_{\text{greedy}}} D_i$ divided by D^*. We observe that the proposed method, which obtained nearly optimal values, was confirmed to have higher performance than the conventional method. In this setting, the larger α is, the larger the difference.

V. Conclusion

A. Summary

In this letter, based on combinatorial optimization theory, we proposed efficient exact and greedy methods for the client scheduling problem for federated learning. Their superiority was confirmed by numerical examples. Since this is only the first step in our combinatorial optimization approach to network control system design, there are many interesting topics to be tackled. As a direct application of the contributions of this letter, we are currently working on a more realistic federated learning setting, which will be briefly discussed in the next subsection, and multi-agent reinforcement learning.

In the system control field, many interesting problems have been formulated as combinatorial optimization problems in recent years, such as the leader selection problem in multi-agent systems [11]–[13]. Our work suggests that the active use of combinatorial optimization theory is promising to go beyond greedy methods utilizing submodularity [14], [15] or transformation into convex problems [16], [17].

B. Client rescheduling in an uncertain environment

We have so far assumed that the calculation and upload times are known accurately in advance. In practice, however, the calculation and upload times estimated in advance may be inaccurate.

Example 2: Suppose there are three agents, each with upload times of $u(1) = 5$, $u(2) = 10$, and $u(3) = 15$, with computation times of $c(1) = 5$, $c(2) = 10$, and $c(3) = 15$ and the number of data $D_1 = 10$, $D_2 = 15$, and $D_3 = 20$. When the time limit is $T = 40$, the optimal data collection plan is the order $\langle 1, 2, 3\rangle$, which motivates us first to upload the data of the agent 1. What if the upload took 20 instead of $u(1) = 5$? Collecting data in the remaining order $\langle 2, 3\rangle$ violates the time limit. In this case, we should consider how best to collect data in the remaining time. That is, it is better to collect data from only agent 3. $\square$

As in this example, consider that the time taken after receiving data from agent $\mathbf{i}$ is not the estimated time $u(\mathbf{i})$. In the existing result, we must reschedule from scratch to find the best solution when we encounter a situation like this. On the other hand, if data collection planning is done using dynamic programming in Subsection III-B or III-C, efficient rescheduling can be done using $d(x, y)$ obtained as a byproduct. This will be presented in a future publication.

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 *Artificial Intelligence and Statistics*. PMLR, 2017, pp. 1273–1282.

[2] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[3] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, 2019.

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

[5] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–7.

[6] H. H. Yang, Z. Liu, T. Q. Quek, and H. V. Poor, "Scheduling policies for federated learning in wireless networks," *IEEE Transactions on Communications*, vol. 68, no. 1, pp. 317–333, 2019.

[7] B. Korte and J. Vygen, *Combinatorial Optimization*, 6th ed. Springer, 2018.

[8] L. Ye and V. Gupta, "Client scheduling for federated learning over wireless networks: A submodular optimization approach," in *60th IEEE Conference on Decision and Control (CDC)*. IEEE, 2021, pp. 63–68.

[9] U. Feige, "A threshold of $\ln n$ for approximating set cover," *Journal of the ACM (JACM)*, vol. 45, no. 4, pp. 634–652, 1998.

[10] H. Samet, *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006.

[11] W. Yang, X. Wang, and H. Shi, "Optimal control nodes selection for consensus in multi-agent systems," *IFAC Proceedings*, vol. 47, no. 3, pp. 11 697–11 702, 2014.

[12] A. Clark, L. Bushnell, and R. Poovendran, "On leader selection for performance and controllability in multi-agent systems," in *51st IEEE Conference on Decision and Control (CDC)*. IEEE, 2012, pp. 86–93.

[13] F. Pasqualetti, S. Zampieri, and F. Bullo, "Controllability metrics, limitations and algorithms for complex networks," *IEEE Transactions on Control of Network Systems*, vol. 1, no. 1, pp. 40–52, 2014.

[14] T. H. Summers, F. L. Cortesi, and J. Lygeros, "On submodularity and controllability in complex dynamical networks," *IEEE Transactions on Control of Network Systems*, vol. 3, no. 1, pp. 91–101, 2015.

[15] A. Clark, B. Alomair, L. Bushnell, and R. Poovendran, *Submodularity in Dynamics and Control of Networked Systems*. Springer, 2016.

[16] T. Ikeda and K. Kashima, "Sparsity-constrained controllability maximization with application to time-varying control node selection," *IEEE Control Systems Letters*, vol. 2, no. 3, pp. 321–326, 2018.

[17] ——, "On sparse optimal control for general linear systems," *IEEE Transactions on Automatic Control*, vol. 64, no. 5, pp. 2077–2083, 2019.