Algorithms for Data Sharing-Aware Task Allocation in Edge Computing Systems

Sanaz Rabinia, *Student Member, IEEE*, Niloofar Didar, *Student Member, IEEE*, Marco Brocanelli, *Member, IEEE*, and Daniel Grosu, *Senior Member, IEEE*

***Abstract*—Edge computing has been developed as a low-latency data driven computation paradigm close to the end user to maximize profit, and/or minimize energy consumption. Edge computing allows each user's task to analyze locally-acquired sensor data at the edge to reduce the resource congestion and improve the efficiency of data processing. To reduce application latency and data transferred to edge servers it is essential to consider data sharing for some user tasks that operate on the same data items. In this article, we formulate the data sharing-aware allocation problem which has as objectives the maximization of profit and minimization of network traffic by considering data-sharing characteristics of tasks on servers. Because the problem is NP − hard, we design the DSTA algorithm to find a feasible solution in polynomial time. We investigate the approximation guarantees of DSTA by determining the approximation ratios with respect to the total profit and the amount of total data traffic in the edge network. We also design a variant of DSTA, called DSTAR that uses a smart rearrangement of tasks to allocate some of the unallocated tasks for increased total profit. We perform extensive experiments to investigate the performance of DSTA and DSTAR, and compare them with a representative greedy baseline that only maximizes profit. Our experimental analysis shows that, compared to the baseline, DSTA reduces the total data traffic in the edge network by up to 20% across 45 case study instances at a small profit loss. In addition, DSTAR increases the total profit by up to 27% and the number of allocated tasks by 25% compared to DSTA, all while limiting the increase of total data traffic in the network.**

***Index Terms*—Edge computing, data sharing, task allocation, profit maximization, network load minimization.**

I. Introduction

EDGE computing facilitates the operations of nearby resource-limited mobile devices such as smartphones, tablets, autonomous mobile robots, drones, and connected vehicles at lower transmission latency compared to the cloud. In fact, many data-driven applications running on mobile devices need computational support to analyze locally-acquired sensor data (e.g., a video or an image from camera, an audio trace from microphone). Typical tasks include face recognition [1], image classification [2], and object tracking [3]. To offload a task, each device must transmit all the data items to be analyzed (e.g., camera frames) to one of the nearby available edge servers. On the other hand, given the possibly large number of end-user devices in the edge system and the even larger number of requests, it is important to ensure the scalability of edge resources with respect to the number of tasks and data being offloaded.

Task allocation in edge computing has been intensively studied during recent years. Due to the limited computing/energy availability of end-user devices, a significant proportion of related work has focused on offloading task execution to edge servers for lowering end-user energy consumption at a maximum latency requirement [4], [5], [6], [7], [8], [9], [10]. Some studies have focused on maximizing the quality of service for end-users via task offloading within edge resource constraints [11], [12], [13], [14]. Other studies focus on maximizing the profit for executing tasks on the edge servers [15], [16], [17], [18]. However, when allocating tasks to edge servers it is very important to consider how much data is being transferred over the edge network to avoid overloading. Our observation is that, in edge computing systems, multiple tasks from end users may share the same data, which can help reducing the total data traffic in the network if those tasks are allocated to the same server. To the best of our knowledge, there is no previous work studying how to design task allocation algorithms in edge computing for joint maximization of profit and minimization of total data size being transferred on the network.

In this paper, we address this challenge by formulating the sharing-aware task allocation problem in edge computing as a bi-objective multi-linear mixed integer program and design two greedy task allocation algorithms, DSTA (*Data Sharing-Aware Task Allocation*) and DSTAR (*DSTA Reallocation*), that solve it. We consider two objectives for the problem, maximizing the profit derived from the execution of tasks on the edge servers, and minimizing the amount of data transferred through the edge network. This paper is an extended version of [19] in which we presented preliminary results on our DSTA algorithm. Compared to a representative baseline that only focus on profit maximization (i.e., P−Greedy), DSTA can reduce the total data traffic in the network at a small profit loss. Different from our previous work [19], in this paper we investigate the approximation guarantees of DSTA by determining the approximation ratios with respect to profit and the amount of total data traffic

Received 17 June 2022; revised 15 October 2024; accepted 22 October 2024. Date of publication 24 October 2024; date of current version 19 November 2024. This work was supported by the US National Science Foundation under Grant CCF-2118202, Grant CNS-2142406, and Grant CNS-2231523. Recommended for acceptance by V. Chaudhary. *(Sanaz Rabinia and Niloofar Didar contributed equally to this work.) (Corresponding author: Daniel Grosu.)*

Sanaz Rabinia, Niloofar Didar, and Daniel Grosu are with the Department of Computer Science, Wayne State University, Detroit, MI 48202 USA (e-mail: srabin@wayne.edu; niloofar_didar@wayne.edu; dgrosu@wayne.edu).

Marco Brocanelli is with the Electrical and Computer Engineering Department, The Ohio State University, Columbus, OH 43210 USA (e-mail: brocanelli.1@osu.edu).

Digital Object Identifier 10.1109/TPDS.2024.3486184

in the edge network. In addition, we propose a new variant of DSTA, called DSTAR, that analyzes the server utilization of DSTA and then performs task reallocation to further increase the total profit at a minimal increase in total data traffic in the edge network.

This paper makes the following contributions:

- We *formulate the data sharing-aware task allocation problem* as a bi-objective mixed-integer multilinear program that jointly maximizes the task allocation profit and minimizes the network load. We develop a *novel analytical model for capturing the sharing among tasks* and use it to derive the objective function that corresponds to the second objective of the problem.
- The data sharing-aware task allocation problem is NP-hard. Thus, in order to provide a feasible solution in polynomial time, we *design a greedy algorithm*, called DSTA, that considers the tasks' data-sharing characteristics and allocates them to edge servers to maximize the profit and minimize the network load.
- We investigate the approximation guarantees of DSTA by *determining the approximation ratios* with respect to total profit and amount of data traffic in the network.
- We *design a reallocation algorithm*, called DSTAR, that uses DSTA's allocation and reallocates tasks on servers to obtain higher total profit at a minimal increase of the amount of data transferred in the network.
- Our *extensive experimental analysis* shows that, compared to the baseline algorithm (P – Greedy), DSTA reduces the amount of data in the edge network at a small profit loss. In addition, DSTAR increases the total profit and number of allocated tasks while limiting the increase of the amount of data in the network.

The rest of the paper is organized as follows. Section II describes the related work. Section III formulates the data sharing-aware task allocation problem (DSTA). Section IV describes the DSTA algorithm. Section V provides the approximation guarantees of DSTA. Section VI presents DSTAR, a variant of DSTA with reallocation. Section VII presents the performance analysis of DSTA and DSTAR. Section VIII concludes the paper.

II. Related Work

In the following, we discuss the related work on task allocation in edge computing and in application-specific edge computing.

A. Task Allocation in Edge Computing

Task allocation in edge computing systems has been extensively studied in the past. In some scenarios, edge servers or nearby users may receive some form of profit for providing edge resources for task offloading. Thus, studies have focused on the topic of finding the best task allocation that maximizes the profit of the edge system. Zhang et al. [15] introduced a novel game-theoretic model with a a dynamic feedback incentive mechanism for task allocation to maximize the performance of edge computing devices. Zhang and Wang [16] introduced a bottom-up game-theoretic model that allows the edge devices to specify their task preferences in a way that maximizes their profits. Kiani and Ansari [17] proposed a new hierarchical model for cloudlets to maximize the profit using an auction-based approach. Yuan and Zhou [18] proposed a task offloading model to maximize the edge system's profit by offloading usersâ tasks to a cloud data center layer and an edge computing layer. Chen et al. [20] designed a task allocation scheme to maximize the overall performance by introducing the concept of task importance in the edge system. Meng et al. [21] studied the online deadline-aware task dispatching and scheduling problem in edge computing to maximize the number of completed tasks. Most of these studies simply consider the transmission time of data items associated with each offloaded task on the total offloading latency estimation.

Some studies provided a more accurate consideration of network packet scheduling for allocating tasks to edge servers in the context of data-driven transfer learning. Sahni et al. [22], [23] studied the data-aware task allocation problem to minimize the overall completion time of the application by jointly scheduling tasks and network flows in collaborative edge computing. Zhang et al. [24] designed a joint scheduling and containerization scheme in edge computing to improve the efficiency of container operations and enable efficient task execution. Li et al. [25] proposed a combined optimal placement of data blocks and scheduling of tasks to reduce the computation delay and response time in edge computing. Breitbach et al. [26] proposed a data management approach for edge computing environments that decouples data placement from task scheduling to optimize the trade-off between execution latency, data management overhead, and response time.

B. Task Allocation in Application-Specific Edge Computing

Task allocation and offloading has been investigated in various application-specific edge computing settings such as Internet of Things (IoT), Mobile Edge Computing (MEC), and Vehicular Edge Computing (VEC).

In the area of IoT and MEC, Shao et al. [27] proposed a replica selection and placement technique in IoT to reduce the data access and response time. Xing et al. [28] minimized the local userâs computation latency by jointly optimizing the task assignment and the power allocation in a multi-user cooperative MEC system. Chen et al. [29] proposed an algorithm that minimizes the total cost of a MEC system with a hybrid energy supply by efficiently scheduling vehicle application tasks across vehicles, edge servers, and the base station. However, they do not study how to reduce the data traffic while minimizing the cost.

Some studies considered the dependencies between tasks, the tasks' sizes, and the capacity of resources when developing task offloading and scheduling algorithms that minimize the energy consumption and the delay. Zhang et al. [8] proposed a vehicle task offloading and scheduling approach in MEC to minimize energy consumption and the delay. Cui et al. [9] proposed a multi-user fine-grained task offloading and scheduling method in MEC where the task is considered to be a directed acyclic graph (DAG) and the objectives are to minimize the energy consumption and the delay.

In the area of VEC, Cao et al. [30] investigated the mobility-aware multi-objective task offloading within a digital twin environment, developing a five-objective optimization model. Tan and Hu [31] proposed a deep reinforcement learning approach to jointly optimize resource allocation of communication, caching, and computing in VEC. They considered the mobility of vehicles to improve caching and computing policies. Zhang et al. [32] proposed a coordination graph-driven vehicular task offloading scheme in VEC based on a multi-agent deep reinforcement learning approach to minimize the offloading cost.

To the best of our knowledge, none of the existing solutions have considered the fact that multiple tasks from the same user may have to perform an analysis on the same data item. For example, the same camera frame can be used by a task for face recognition and by another task for object detection. Thus, allocating those tasks to different servers without considering that they share data items may lead to the necessity to send the same data item to both servers. On the other hand, allocating those tasks to the same server can help reduce the network load since only one copy of that shared data item needs to be transmitted.

III. Data Sharing-Aware Task Allocation Problem

We consider an edge computing system composed of a set $\mathcal{S} = \{S_1, S_2, \ldots, S_M\}$ of M distributed servers, where each server S_j has a limited capacity C_j of computational resources (i.e., CPU cycles). These edge servers serve a set $\mathcal{T} = \{T_1, T_2, \ldots, T_N\}$ of N tasks originating from end-user devices. The set of tasks $\mathcal{T}$ has an associated set of data items, $\mathcal{D} = \{D_1, D_2, \ldots, D_D\}$, that are needed to execute the tasks. We denote the size of data item D_k by d_k, where $k = 1, 2, \ldots, D$. Each task T_i is characterized by a tuple $(r_i, p_i, [A]_{i,*})$, where r_i is the amount of computational resources required by T_i, p_i is the profit for executing T_i, and $[A]_{i,*}$ is the ith row of the task-data matrix, A. The *task-data matrix* A is a $N \times D$ matrix, where $a_{ik} = d_k$, if task T_i requires data item D_k, and 0, otherwise. The tasks need to be allocated to the servers such that the total profit obtained from executing the tasks is maximized and the total amount of data transferred in the network is minimized.

We formulate the *data sharing-aware task allocation problem* (DSTAP) as a bi-objective mixed-integer multi-linear program

$$\text{maximize:} \quad \sum_{j=1}^{M}\sum_{i=1}^{N} p_i x_{ij} \tag{1}$$

$$\text{minimize:} \quad \sum_{\mathcal{I}\in\mathcal{P}(\mathcal{T})} (-1)^{(|\mathcal{I}|+1)} \sigma_{\mathcal{I}} \sum_{j=1}^{M} \prod_{i\in\mathcal{I}} x_{ij} \tag{2}$$

subject to:

$$\sum_{i=1}^{N} r_i x_{ij} \le C_j, \quad \forall j \in \{1, \ldots, M\} \tag{3}$$

$$\sum_{j=1}^{M} x_{ij} \le 1, \quad \forall i \in \{1, \ldots, N\} \tag{4}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i, \forall j \tag{5}$$

TABLE I
Notation

Notation	Description
$\mathcal{T}$	Set of tasks.
N	Number of tasks.
T_i	Task i.
r_i	Requested amount of CPU resource by T_i.
p_i	Profit of task T_i.
$\mathcal{S}$	Set of servers.
M	Number of servers.
S_j	Server j.
C_j	CPU capacity of server S_j.
$\mathcal{D}$	Set of data items.
D	Number of data items.
D_k	Data item k.
d_k	Size of data item D_k.
A	Task-data matrix (a_{ik}; $i = 1, \ldots, N$; $k = 1, \ldots, D$).
$\sigma_{\mathcal{I}}$	Sharing parameter.
$\mathcal{T}^c$	Set of candidate tasks.
$\mathcal{D}^c$	Set of candidate data which is assigned to servers.
s_k	Sum of column k entries of matrix A.
$\text{supp}([A]_{*,k})$	Support of column k in matrix A.

Fig. 1. Task-data matrix and its associated bipartite graph for a DSTAP instance with three tasks, three data items.

The first objective (1) is to maximize the total profit. The decision variable x_{ij} is 1, if task T_i is allocated to server S_j, and 0, otherwise. The second objective (2) is to minimize the total amount of data offloaded from user devices to the servers, which depends on the decision variables x_{ij} and on the data sharing among tasks. Here, $\mathcal{P}(\mathcal{T})$, is the power set of the set of indices of the tasks in $\mathcal{T}$, and $\mathcal{I}$ is an element of the power set. We define the *sharing parameter*, $\sigma_{\mathcal{I}}$, as the total amount of data shared among the tasks whose indices are in set $\mathcal{I}$.

In the next paragraph, we give more details on how the sharing parameter is computed and explain how (2) captures the sharing of data and gives the total amount of data in the network. Constraint (3) ensures that the total allocated computational requests to a server does not exceed the capacity of the server. Constraint (4) ensures that each task is allocated to only one server, while Constraint (5) guarantees the integrality of the decision variables. Table I summarizes the notation of this paper.

To explain how the data sharing is captured in (2), we use a small example consisting of a set of three tasks $\mathcal{T} = \{T_1, T_2, T_3\}$, a set of three data items $\mathcal{D} = \{D_1, D_2, D_3\}$, and a set of three servers $\mathcal{S} = \{S_1, S_2, S_3\}$. For this example, we consider that T_1 needs D_1, D_2, T_2 needs D_1, D_2, D_3, and T_3 needs D_2, D_3. The task-data matrix A and the associated bipartite graph capturing the sharing for this example is given in Fig. 1. One partition of the bipartite graph consists of vertices corresponding to the tasks, while the other partition consists of vertices corresponding to the data items. The sharing parameter $\sigma_{\mathcal{I}}$ in (2) is the amount of data shared by the tasks whose indices are in set $\mathcal{I}$. For our example, we have $\sigma_1 = 10, \sigma_2 = 15$, $\sigma_3 = 11, \sigma_{12} = 10, \sigma_{13} = 6, \sigma_{23} = 11$, and $\sigma_{123} = 6$. Suppose

that T_1 and T_2 are allocated to S_1, T_3 is allocated to S_2, and no task is allocated to S_3, then we have $x_{11} = x_{21} = x_{32} = 1$ and $x_{12} = x_{13} = x_{22} = x_{23} = x_{31} = x_{33} = 0$. Thus, using (2) we have

$$\begin{aligned}
&\sum_{\mathcal{I}\in\mathcal{P}(\mathcal{T})} (-1)^{(|\mathcal{I}|+1)}\sigma_{\mathcal{I}} \sum_{j=1}^{3}\prod_{i\in\mathcal{I}} x_{ij} = \\
&(+1)\left[\sigma_1(x_{11}+x_{12}+x_{13}) + \sigma_2(x_{21}+x_{22}+x_{23})\right. \\
&\left.+\sigma_3(x_{31}+x_{32}+x_{33})\right] + \\
&(-1)\left[\sigma_{12}(x_{11}x_{21}+x_{12}x_{22}+x_{13}x_{23})\right. \\
&+\sigma_{13}(x_{11}x_{31}+x_{12}x_{32}+x_{13}x_{33}) \\
&\left.+\sigma_{23}(x_{21}x_{31}+x_{22}x_{32}+x_{23}x_{33})\right] + \\
&(+1)\left[\sigma_{123}(x_{11}x_{21}x_{31}+x_{12}x_{22}x_{32}+x_{13}x_{23}x_{33})\right].
\end{aligned}$$

Plugging in the values of x_{ij} and $\sigma_{\mathcal{I}}$ in the above equation we obtain: $\sigma_1 + \sigma_2 + \sigma_3 - \sigma_{12} = 10 + 15 + 11 - 10 = 26$. We can easily check that the total amount of data offloaded to servers is 26. Since T_1 and T_2 are assigned to the same server D_1, D_2, and D_3 need to be offloaded to server S_1 and the amount of data offloaded to S_1 is 15. T_3 is assigned to server S_2 and it needs D_2 and D_3, thus the total amount of data offloaded to S_2 is 11. Therefore, the total amount of data offloaded in the system is 26.

The Knapsack problem is a special case of DSTAP, that is, by removing the second objective from DSTAP, the problem becomes the Knapsack problem. The Knapsack is a known NP-hard problem [33] and since it is a special case of DSTAP, we have that DSTAP is NP-hard. Thus, there is no polynomial time algorithm that obtains an optimal solution of DSTAP, unless P = NP. Therefore, in the next sections, we design two greedy algorithms that find feasible solutions in polynomial time.

IV. DSTA Algorithm

We design a greedy algorithm, called DSTA, for solving DSTAP. DSTA takes into account the data sharing characteristics of the tasks when deciding which tasks to allocate on the edge servers. That is, it iteratively selects a subset of tasks that share the highest amounts of data with the tasks that are already allocated. In each iteration, it establishes a greedy order among these tasks, that is induced by a function which prioritizes high-profit and light-workload tasks for allocation to the most suitable edge server. DSTA is given in Algorithm 1. The input of DSTA consists of the set of tasks, $\mathcal{T}$; the set of servers, $\mathcal{S}$; and the task-data matrix, A.

Initialization (Lines 1–8): DSTA initializes the set of candidate tasks $\mathcal{T}^c$ and the set of candidate data items $\mathcal{D}^c$ to the empty set, and the allocation matrix X to zero (Lines 1 to 3). Here, the allocation matrix has as entries the variables x_{ij}, where $x_{ij} = 1$, if task T_i is allocated to server S_j, and 0, otherwise. Next (Line 4), it sorts the servers in non-increasing order of their capacities. The ordering of the servers after sorting is given by the permutation $\beta(j)$.

DSTA uses an array s, whose entries $s_k = \sum_{i=1}^{N} a_{ik}$, for $k = 1, \ldots, D$. That is, s_k is the sum of the entries of column k of task-data matrix A. Since the column k of A corresponds to data item k, s_k is the total amount of data item k needed by the tasks without considering sharing. DSTA computes the entries of s in Lines 6 to 8.

Allocation Strategy Overview: The while loop in Lines 9 to 39 is executed until the set of tasks $\mathcal{T}$ becomes empty. In each iteration of the loop, the algorithm determines which data item is the most shared and the tasks that share it, computes the efficiency metric that is used to establish the greedy order among those tasks, and allocates the tasks to servers in the order given by the greedy order. In the following, we describe these operations in more details.

Data Sharing Analysis (Lines 10–20): The algorithm uses an additional array s' whose entry s'_k is set to 1 if a task in the candidate set is using the data item D_k and D_k has not been assigned to a server yet (Lines 11 to 15). The algorithm determines the *support* of the array s', denoted here by supp(s'), which is defined as the set of all indices corresponding to nonzero entries in s' (Line 16). If the size of the support of s' is greater than zero, there are data items that have not been assigned to servers yet, thus the algorithm places in set $\mathcal{K}$ the indices of the data items that have not been assigned yet and have the largest value of s_k (Line 17). If the size of the support of s' is equal to zero, then no data item was allocated yet (i.e., this is the first iteration of the while loop). Thus, DSTA places in set $\mathcal{K}$ the indices of the items that have the largest value of s_k (Line 19). Next, DSTA determines the index $\tilde{k}$ of the items in set $\mathcal{K}$ whose corresponding column in the task-data matrix A has the largest support, i.e., the item shared by the largest number of tasks (Line 20).

Efficiency Function Evaluation and Task Selection (Lines 21–24): In Lines 21 to 24, DSTA computes the *efficiency function*, which is used to establish the greedy order among the tasks. The efficiency function is computed only for the tasks that share the data item $\tilde{k}$, which is shared by the highest number of tasks and that has the greatest corresponding total amount of data. Thus, the efficiency function is computed only for the tasks that are not allocated yet and have $a_{i\tilde{k}} \neq 0$ (Line 23). The efficiency function is defined by

$$E_i = \frac{p_i}{\sqrt{\frac{r_i}{\sum_{j=1}^{M} C_j}}}. \tag{6}$$

The efficiency function for a given task can be viewed as a density measure, computed as the profit obtained from executing the task divided by the square root of the relative size of the request. Here, the relative size of the request is with respect to the total capacity of the servers in the system. This efficiency function allows the algorithm to allocate the tasks in the order of their highest profit density and therefore obtain high values for the total profit gained from executing the tasks.

Allocation of the Selected Tasks (Lines 25–39): Once the efficiency function is determined, DSTA sorts the tasks in non-increasing order of E_i (Line 25). The ordering of the tasks after sorting is given by the permutation $\alpha(i)$. The algorithm goes over the tasks with $E_i > 0$ (i.e., the tasks with high data sharing for which the efficiency metric was determined) in the order given by

Algorithm 1: DSTA Algorithm.

Input: $\mathcal{T}$: set of tasks;
$\mathcal{S}$: set of edge servers;
A: task-data matrix.

1: $\mathcal{T}^c \leftarrow \emptyset$
2: $\mathcal{D}^c \leftarrow \emptyset$
3: $X \leftarrow [0]$
4: Sort servers in non-increasing order of capacity C_j
Let $S_{\beta(1)}, S_{\beta(2)}, \ldots, S_{\beta(M)}$ be the order
5: $s \leftarrow [0]$
6: **for** $k = 1, \ldots, D$ **do**
7: **for** $i = 1, \ldots, N$ **do**
8: $s_k \leftarrow s_k + a_{ik}$
9: **while** $|\mathcal{T}| > 0$ **do**
10: $s' \leftarrow [0]$
11: **for** $i = 1, \ldots, N$ **do**
12: **if** $T_i \in \mathcal{T}^c$ **then**
13: **for** $k = 1, \ldots, D$ **do**
14: **if** $D_k \notin \mathcal{D}^c$ **and** $a_{ik} \neq 0$ **then**
15: $s'_k \leftarrow 1$
16: **if** $|\text{supp}(s')| > 0$ **then**
17: $\mathcal{K} = \{k | \forall\, l \in \{1, \ldots, D\} : s_l s'_l \leq s_k s'_k\}$
18: **else**
19: $\mathcal{K} = \{k | \forall\, l \in \{1, \ldots, D\} : s_l \leq s_k\}$
20: $\bar{k} \leftarrow \text{argmax}_{k \in \mathcal{K}} \{|\text{supp}([A]_{\star,k})|\}$
21: **for** $i = 1, \ldots, N$ **do**
22: $E_i \leftarrow 0$
23: **if** $a_{i\bar{k}} \neq 0$ and $T_i \in \mathcal{T}$ **then**
24: $E_i \leftarrow \frac{p_i}{r_i \sqrt{\sum_{j=1}^{M} C_{\beta(j)}}}$
25: Sort tasks in non-increasing order of E_i
Let $T_{\alpha(1)}, T_{\alpha(2)}, \ldots, T_{\alpha(N)}$ be the order
26: **for** $i = 1, \ldots, N$ **do**
27: **if** $E_{\alpha(i)} > 0$ **then**
28: **for** $j = 1, \ldots, M$ **do**
29: **if** $C_{\beta(j)} - r_{\alpha(i)} \geq 0$ **then**
30: $C_{\beta(j)} \leftarrow C_{\beta(j)} - r_{\alpha(i)}$
31: $x_{\alpha(i)\beta(j)} \leftarrow 1$
32: $\mathcal{T}^c \leftarrow \mathcal{T}^c \cup \{T_{\alpha(i)}\}$
33: $\mathcal{T} \leftarrow \mathcal{T} \backslash \{T_{\alpha(i)}\}$
34: **break**
35: **else**
36: **if** $j = M$ **then**
37: $\mathcal{T} \leftarrow \mathcal{T} \backslash \{T_{\alpha(i)}\}$
38: $\mathcal{D}^c \leftarrow \mathcal{D}^c \cup \{D_{\bar{k}}\}$
39: $s_{\bar{k}} \leftarrow 0$
40: **output:** X

permutation $\alpha(i)$ and attempts to allocate them to the available servers. In Lines 28 to 37, the servers are considered for task allocation in the non-increasing order of their capacities (given by the permutation $\beta(j)$). DSTA checks if the given server has enough capacity to handle the request. If it has enough capacity, the capacity of the server is decreased by the size of the request, the entry corresponding to task $T_{\alpha(i)}$ and server $S_{\beta(j)}$ in the allocation matrix X is set to 1, the task is added to the set of candidate tasks $\mathcal{T}^c$ and removed from the set of tasks $\mathcal{T}$. Once a task is allocated the algorithm exits from the for loop in Line 34. If the server does not have enough capacity the algorithm considers for allocation the next server in the order. If none of the servers have enough capacity to allocate the task then the task is removed from the set of tasks $\mathcal{T}$ (Lines 36 to 37). Finally, at the end of each while loop iteration (Lines 38 and 39), DSTA adds data item $D_{\bar{k}}$ to the candidate data set and sets $s_{\bar{k}}$ to 0, to avoid reconsidering data item $D_{\bar{k}}$ in the next iterations.

Complexity of DSTA : The while loop (Lines 9 to 39) determines the *time complexity* of DSTA which is $\mathcal{O}(N^2(D+M))$. This is mainly due to the running time of the for loop in Lines 26 to 37, which takes $\mathcal{O}(NM)$, and the computation of $\bar{k}$ in Line 20 which takes $\mathcal{O}(ND)$. In the worst case, the while loop is executed $\mathcal{O}(N)$ times, and thus the running time of the while loop is $\mathcal{O}(N^2(D+M))$. Therefore, DSTA has a time complexity of $\mathcal{O}(N^2(D+M))$.

The space complexity of DSTA is determined by the space needed to store the input (i.e., $\mathcal{O}(N)$ for the tasks' information, $\mathcal{O}(M)$ for the servers' information, and $\mathcal{O}(ND)$ for the task-data matrix A) and the allocation matrix X (i.e., $\mathcal{O}(NM)$). The amount of space required to store the other variables used in the algorithm is dominated by the amount of space required to store the input and the allocation matrix. Therefore, the *space complexity* of DSTA is $\mathcal{O}(N(D+M))$.

DSTA*: Illustrative Example*. We use a small example to clarify how DSTA works. Consider a set of six tasks $\mathcal{T} = \{T_1, T_2, T_3, T_4, T_5, T_6\}$, a set of six data items $\mathcal{D} = \{D_1, D_2, D_3, D_4, D_5, D_6\}$ with data sizes $\{200, 100, 40, 20, 30, 400\}$, respectively. We also have a set of three servers $\mathcal{S} = \{S_1, S_2, S_3\}$ with capacities $\{25, 20, 12\}$ in non-increasing order. The amount of computational resource and the profit of each task is shown in Table II. Tasks T_1 and T_6 share D_1 and D_6, tasks T_2 and T_5 share data items D_4 and D_5, and tasks T_3 and T_4 share data items D_2 and D_3. As a result, the task-data matrix A is

TABLE II
DSTA EXAMPLE PARAMETERS

Task	T_1	T_2	T_3	T_4	T_5	T_6
p_i	15	8	10	7	10	10
r_i	10	9	5	7	10	11

$$A = \begin{array}{c} \\ T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \end{array} \begin{array}{c} \begin{array}{cccccc} D_1 & D_2 & D_3 & D_4 & D_5 & D_6 \end{array} \\ \begin{bmatrix} 200 & 0 & 0 & 0 & 0 & 400 \\ 0 & 0 & 0 & 20 & 30 & 0 \\ 0 & 100 & 40 & 0 & 0 & 0 \\ 0 & 100 & 40 & 0 & 0 & 0 \\ 0 & 0 & 0 & 20 & 30 & 0 \\ 200 & 0 & 0 & 0 & 0 & 400. \end{bmatrix} \end{array} \tag{7}$$

In Line 8 of DSTA the entries of s_k are calculated, which indicates that the maximum amount of shared data is associated with D_6 (i.e., $s_6 = 800$). T_1 and T_6 share D_6, thus DSTA calculates their efficiency function (Line 24) and sorts them (Line 25). Specifically, $E_1 > E_6$, and thus, the allocation order is T_1 followed by T_6. The server with the highest capacity (i.e.,

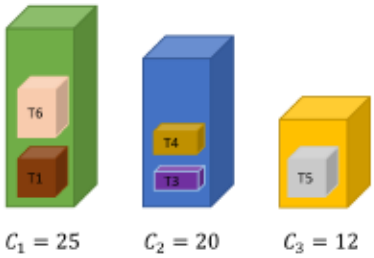


Fig. 2. DSTA illustrative example.

25) is S_1. Because it has enough capacity to host both T_1 and T_6, they are both allocated to S_1, whose capacity decreases from 25 to 4 (i.e., $25 - 10 - 11 = 4$). Then, the allocation matrix X is updated, and T_1 and T_6 are removed from the set of initial tasks $\mathcal{T}$ (Line 33). After this, $\mathcal{T} = \{T_2, T_3, T_4, T_5\}$. Before continuing with another iteration of the while loop and allocate other tasks, data item D_6 is added to the set of candidate data items $\mathcal{D}^c$ (Line 38) and s_6 is set to 0, so that D_6 is not considered in the rest of the execution.

Since $s_1 = 400$ is currently the highest value entry of s, D_1 is the item selected in the second iteration of the while loop (Line 20). On the other hand, only tasks T_1 and T_6 use this data item, thus the rest of the algorithm simply adds D_1 to the set of candidate data items $\mathcal{D}^c$ and sets s_1 to 0. As a result, in the third iteration of the while loop the highest value entry in s is $s_2 = 200$, thus all tasks requiring D_2, i.e., T_3 and T_4, are selected for allocation. DSTA calculates their efficiency and determines that $E_3 > E_4$, thus T_3 is allocated before T_4. The remaining capacity of S_1 is 4, which is not sufficient to host any of the two selected tasks, thus S_2, whose current capacity is 20 is selected. Because S_2 has enough capacity, both T_3 and T_4 are allocated to it. Then, DSTA updates the capacity of S_2 to 8 (i.e., $20 - 5 - 7 = 8$). In the fourth iteration, D_2 and D_3 are added to the set of candidate data items $\mathcal{D}^c$. In the last iteration, data item D_4 is selected for allocation. Both the unallocated tasks T_2 and T_5 use this data item and $E_5 > E_2$. Thus, DSTA first allocates T_5 to the only server able to host it, i.e., S_3. However, T_2 requesting 9 units remains unallocated since none of the servers have enough capacity ($C_1 = 4$, $C_2 = 8$, and $C_3 = 2$). Fig. 2 shows the final allocation.

V. Approximation Guarantees of DSTA

In this section, we determine the approximation ratios of DSTA with respect to the total profit and the total amount of data offloaded. An algorithm has an *approximation ratio* ρ, if for all instances of the problem, it produces a solution whose value is within ρ from the value of the optimal solution. Here, we follow the standard convention that $\rho < 1$ for maximization problems, and $\rho > 1$ for minimization problems. That is, for minimization problems, $\rho = 2$ means that the value of the solutions obtained by the algorithm is at most twice the optimal value. First, we determine, α, the approximation ratio of DSTA with respect to the total profit and then, β, the approximation ratio with respect to the total data offloaded.

Theorem 1: The approximation ratio of DSTA with respect to total profit is $\alpha = \frac{1}{N \cdot \pi}$, where $\pi = \frac{\max_{i:T_i \in \mathcal{T}} \{p_i\}}{\min_{i:T_i \in \mathcal{T}} \{p_i\}}$.

Proof: To determine the approximation ratio of DSTA for the maximization objective (e.g., maximizing the total profit), we ignore the second objective and consider the worst case instance for DSTA consisting of a set of tasks $\mathcal{T}^W$ and a set of servers $\mathcal{S}^W$. The set of tasks $\mathcal{T}^W$ has the following characteristics:

i) There is no data sharing among any of the tasks.
ii) Task $T_{\hat{i}}$ needs the largest data item $D_{\hat{i}}$ only, while the other tasks need smaller data items, one for each task, i.e., $\forall T_i \in \mathcal{T}^W \setminus \{T_{\hat{i}}\}$, $d_i < d_{\hat{i}}$.
iii) Task $T_{\hat{i}}$ has the largest request size $r_{\hat{i}}$ while the other tasks have smaller requests, i.e., $\forall T_i \in \mathcal{T}^W \setminus \{T_{\hat{i}}\}, r_i < r_{\hat{i}}$. In addition, the sum of the requests of all the tasks in $\mathcal{T}^W$ except $T_{\hat{i}}$ is less than or equal to the request of $T_{\hat{i}}$, i.e., $\sum_{i:T_i \in \mathcal{T}^W \setminus \{T_{\hat{i}}\}} r_i \leq r_{\hat{i}}$.
iv) Task $T_{\hat{i}}$ has the minimum profit among the tasks, i.e., $p_{\hat{i}} = p_{\min}$, while the other tasks have the largest profit, $p_{\max}$, i.e., $\forall T_i \in \mathcal{T}^W \setminus \{T_{\hat{i}}\}$, $p_i = p_{\max}$.

The set of servers $\mathcal{S}^W$ has the following characteristics:

i) Server $S_{\hat{j}}$ has a capacity $C_{\hat{j}} = r_{\hat{i}}$. That is, server $S_{\hat{j}}$ has enough capacity to execute task $T_{\hat{i}}$ only, or all the other tasks together, except $T_{\hat{i}}$. This is because $\sum_{i:T_i \in \mathcal{T}^W \setminus \{T_{\hat{i}}\}} r_i \leq r_{\hat{i}}$.
ii) The other servers have capacities that are smaller than the smallest task request, and thus, they cannot execute any of the tasks, i.e., $\forall S_j \in \mathcal{S}^W \setminus \{S_{\hat{j}}\}$, $C_j < \min_{i:T_i \in \mathcal{T}^W} \{r_i\}$.

Since the tasks do not share any data items and task $T_{\hat{i}}$ has the largest data item, DSTA allocates $T_{\hat{i}}$ to server $S_{\hat{j}}$. The other tasks are not allocated by DSTA because there is no capacity left on server $S_{\hat{j}}$, and the other servers do not have enough capacity to execute them. Thus, the profit P obtained by DSTA is $P = p_{\hat{i}} = p_{\min}$ which is the minimum profit among the tasks.

The optimal algorithm that maximizes the profit allocates all the tasks except $T_{\hat{i}}$ on server $S_{\hat{j}}$. This is possible because $S_{\hat{j}}$ has enough capacity to execute all these tasks. Therefore, the profit obtained by the optimal algorithm is

$$P^* = \sum_{i:T_i \in \mathcal{T}^W} p_i = (N-1)p_{\max}. \tag{8}$$

Thus,

$$\frac{P}{P^*} = \frac{p_{\min}}{(N-1)p_{\max}} \geq \frac{p_{\min}}{N \cdot p_{\max}}. \tag{9}$$

Let $\pi = \frac{p_{\max}}{p_{\min}}$, then we have $P \geq \frac{1}{N \cdot \pi} P^*$ and the approximation ratio of DSTA is $\alpha = \frac{1}{N \cdot \pi}$. □

Theorem 2: The approximation ratio of DSTA with respect to the total amount of data in the edge network is $\beta = 2$.

Proof: To determine the approximation ratio of DSTA for the minimization objective (e.g., minimizing the total amount of offloaded data), we ignore the profit as a second objective. To build the worst case instance with respect to this objective for DSTA, we divide the N tasks into K groups, where each group has common shared data with the following properties: i) There is no data sharing across groups; ii) Each server has

enough capacity to host all the tasks from the same group; and iii) Each server does not have enough capacity to host all the tasks of more than one group, but can host an additional task from another group. An example of such case is given in the task-data matrix in (7), where the three task groups $\{T_1, T_6\}$, $\{T_2, T_5\}$, and $\{T_3, T_4\}$ do not share data with each other.

DSTA sorts all the servers in non-increasing order. We also can sort the total amount of data of each group in non-increasing order and consider the total amount of data of a group has a slight difference of ε, from its previous and next group. Thus, if the total amount of data for group 1 is D_{G_1}, then the total amount of data for group 2 is $D_{G_1} - \varepsilon$, the total amount of data for group 3 is $D_{G_1} - 2\varepsilon$ and so on. Considering this setup, DSTA picks tasks from group 1 first and assigns them to the same server. Then it picks tasks from the second group and assigns them accordingly.

Let the total amount of data obtained from an optimal solution and DSTA be denoted by D^* and D, respectively. For the optimal solution, since each server has enough capacity to host an entire group, it assigns all tasks from the same group to the same server. We have at most $K = M$ groups so the total amount of data is $D^* = M \cdot D_{G_1} - \frac{M(M-1)}{2}\varepsilon$. DSTA assigns all tasks from the first group to the first sever. According to the third property of our server setup it can assign one task from the second group to the first server. It assigns the remaining tasks of group 2 to the second server along with one task from the third group and continues the assignment of tasks to servers. We have at most $K = M$ groups, thus the total amount of data for DSTA is $D = (2M \cdot D_{G_1} - D_{G_1}) - M(M-1)\varepsilon$. Therefore, $D \leq 2D^*$ and the approximation ratio with respect to the total amount of data is $\beta = 2$. □

VI. DSTAR: DSTA Reallocation Algorithm

The DSTA algorithm does not consider that rearranging the allocated tasks to the servers may free up enough capacity to allocate some of the unallocated tasks for increased profit. For instance, consider the illustrative example of DSTA in Section IV. Task T_2 remains unallocated even though, as we will show in this section, a smart rearrangement of task allocation would allow all the tasks to be allocated for increased total profit at the same network data size. Note that increasing the number of allocated tasks for higher profit may minimally increase the total data size on the network compared to DSTA's allocation depending on the sharing data characteristics of the unallocated tasks. However, DSTAR guarantees that the data size on the network increases only if more tasks are allocated or if a higher profit can be obtained with same or fewer allocated tasks. In many cases, as we will show in the evaluation section, DSTAR can even maintain the total profit while decreasing the data size on the network. To do so, the DSTAR algorithm compares the total profit and total resource utilization of tasks allocated by DSTA with those of the unallocated tasks and then re-allocates them in groups without breaking the tasks-server coupling decisions of DSTA. DSTAR is given in Algorithm 2. It has as input the set of tasks $\mathcal{T}$, the set of edge servers $\mathcal{S}$, and the task-data matrix A, as defined in the DSTA algorithm.

Algorithm 2: DSTAR Algorithm.

Input: $\mathcal{T}$: set of tasks;
$\mathcal{S}$: set of edge servers;
A: task-data matrix.

1: $X \leftarrow \text{DSTA}(\mathcal{T}, \mathcal{S}, A)$
2: $F = [0]$
3: $\mathcal{G} \leftarrow \emptyset$
4: **for** $j = 1, \ldots, M$ **do**
5: $\quad\mathcal{E} \leftarrow \emptyset$
6: $\quad$**for** $i = 1, \ldots, N$ **do**
7: $\quad\quad$**if** $x_{ij} \neq 0$ **then**
8: $\quad\quad\quad\mathcal{E} \leftarrow \mathcal{E} \cup \{i\}$
9: $\quad\quad\quad F_i \leftarrow 1$
10: $\quad\mathcal{G} \leftarrow \mathcal{G} \cup \{\mathcal{E}\}$
11: **for** $i = 1, \ldots, N$ **do**
12: $\quad$**if** $F_i = 0$ **then**
13: $\quad\quad\mathcal{G} \leftarrow \mathcal{G} \cup \{i\}$
14: Sort servers in non-decreasing order of capacity C_j
Let $S_{\beta(1)}, S_{\beta(2)}, \ldots, S_{\beta(M)}$ be the order
15: **for** $s = 1, \ldots, |\mathcal{G}|$ **do**
16: $\quad\tilde{E}_s \leftarrow \dfrac{\sum_{i \in \mathcal{G}_s} p_i}{\sqrt{\dfrac{\sum_{i \in \mathcal{G}_s} r_i}{\sum_{j=1}^{M} C_{\beta(j)}}}}$
17: Sort task sets in non-increasing order of efficiency $\tilde{E}_s$
Let $\mathcal{G}_{\alpha(1)}, \mathcal{G}_{\alpha(2)}, ldots, \mathcal{G}_{\alpha(|\mathcal{G}|)}$ be the order
18: $\tilde{X} \leftarrow 0$
19: **for** $s = 1, \ldots, |\mathcal{G}|$ **do**
20: $\quad$**for** $j = 1, \ldots, M$ **do**
21: $\quad\quad$**if** $C_{\beta(j)} - \sum_{k \in \mathcal{G}_{\alpha(s)}} r_k \geq 0$ **then**
22: $\quad\quad\quad$**for** $i \in \mathcal{G}_{\alpha(s)}$ **do**
23: $\quad\quad\quad\quad C_{\beta(j)} \leftarrow C_{\beta(j)} - r_i$
24: $\quad\quad\quad\quad \tilde{x}_{i\beta(j)} \leftarrow 1$
25: $\quad\quad\quad$**break**
26: **if** $\sum_{i=1}^{N}\sum_{j=1}^{M} x_{ij}p_i \leq \sum_{i=1}^{N}\sum_{j=1}^{M} \tilde{x}_{ij}p_i$ **then**
27: $\quad X \leftarrow \tilde{X}$
28: **output:** X

Initialization (Lines 1–14): DSTAR initializes the allocation matrix X by executing DSTA. Then, it initializes to 0 each element of a flags array F, which indicates whether each task is assigned to a server by DSTA. $\mathcal{G}$ in Line 3 is a set of task sets that is initially empty and is then updated in Lines 4 to 13 according to the allocation decisions of DSTA. Specifically, starting from the first server (Line 4), DSTAR places the indexes of all the tasks allocated by DSTA to this server into a set $\mathcal{E}$ (Lines 5 to 9). Then, in Line 10 the task set $\mathcal{E}$ becomes a subset of $\mathcal{G}$. Note that in Line 9 the algorithm updates the array flag F, where $F_i = 1$ if task T_i has already been assigned by DSTA. This flag is used in Lines 11 to 13 to add the remaining unallocated tasks to $\mathcal{G}$ as subsets of cardinality one. As a result of executing Lines 4-13, the updated set $\mathcal{G}$ is composed of subsets with cardinality ≥ 1, each one grouping the indexes of all the tasks allocated to the same server by DSTA, or subsets of cardinality 1, each corresponding to an unallocated task. Next, DSTAR

sorts the servers in non-decreasing order of their capacities (Line 14). The ordering of the servers after sorting is given by the permutation $\beta(j)$.

Efficiency Function and Task-Set Selection (Lines 15–27): In Lines 15 to 16, DSTAR computes a new *efficiency function* $\tilde{E}_s$ for each task set $s \in \mathcal{G}$ as follows:

$$\tilde{E}_s = \frac{\sum_{t \in \mathcal{G}_s} p_t}{\sqrt{\frac{\sum_{i \in \mathcal{G}_s} r_i}{\sum_{j=1}^{M} C_j}}}. \tag{10}$$

DSTAR's efficiency function is calculated similarly to the efficiency of DSTA, but the profit at the numerator and the resource requests at the denominator are the aggregated profit and resource requests of all tasks in the subset $s \in \mathcal{G}$. This efficiency function helps evaluate the efficiency of the set of allocated tasks compared to that of the unallocated tasks. In Line 17, DSTAR sorts the task sets in non-increasing order of efficiency $\tilde{E}_s$. The ordering of the task sets after sorting is given by the permutation $\alpha(s)$. Finally, in Lines 18 to 25 DSTAR greedily allocates the tasks in each set in the defined order. In particular, starting from the task set with the highest efficiency, i.e., $\tilde{E}_{\alpha(1)}$ (Line 19), DSTAR finds the server with the smallest capacity able to host all the tasks of the selected task set, which avoids breaking tasks sharing data into different servers. If a server is found, the tasks are allocated, the server capacity is updated, and the loop is broken to allocate the tasks of the next task set (Lines 20 to 25). DSTAR guarantees that the profit obtained is at least equal or higher than the one obtained by DSTA by overwriting X with its solution $\tilde{X}$ in such cases (Lines 26 to 27). As a result, if DSTA already allocates all the tasks, DSTAR also maintains the guarantee of DSTA with respect to data. However, if DSTA does not allocate all tasks the guarantee with respect to the data might be violated since DSTAR increases the number of allocated tasks. We will investigate this property in the experimental results section.

Complexity of DSTAR*:* The time complexity of DSTAR is determined by that of DSTA (Line 1), $\mathcal{O}(N^2(D+M))$, the complexity of initializing the group of task sets $\mathcal{G}$ (Lines 4 to 10) which is $\mathcal{O}(NM)$, the complexity of sorting the servers, $\mathcal{O}(M \log M)$, the complexity of sorting the task-sets, $\mathcal{O}(N \log N)$, the complexity of for loops in Lines 19 to 25, $\mathcal{O}(NM)$, and the complexity for comparing the profits of DSTA and DSTAR in Lines 26 to 27, $\mathcal{O}(NM)$. Thus, DSTAR has the same time complexity as DSTA. Considering the same reasoning as in the case of DSTA, the *space complexity* of DSTAR is the same as that of DSTA, i.e., $\mathcal{O}(N(D+M))$.

DSTAR. *Illustrative Example:* We illustrate how DSTAR operates by considering the same example we used for DSTA in Section IV. First, DSTAR uses DSTA to obtain the allocation matrix X (Line 1). For our example, the matrix X returned by DSTA has $x_{11} = x_{61} = x_{32} = x_{42} = x_{53} = 1$. Only task T_2 is not allocated by DSTA. In Lines 4 to 13, DSTAR uses the allocation X to update the set $\mathcal{G}$. In the first iteration of Lines 4 to 10, DSTAR adds the subset $\{1,6\}$ to $\mathcal{G}$, since both T_1 and T_6 are allocated to server S_1, and updates the flag array F to $[1,0,0,0,0,1]$. After two more iterations the loop in Lines 4 to 10 ends and $\mathcal{G} = \{\{1,6\},\{3,4\},\{5\}\}$, while the flag array $F = [1,0,1,1,1,1]$. Then, in Lines 11 to 13 DSTAR adds the index of unallocated task T_2 to $\mathcal{G}$, which becomes $\mathcal{G} = \{\{1,6\},\{3,4\},\{5\},\{2\}\}$.

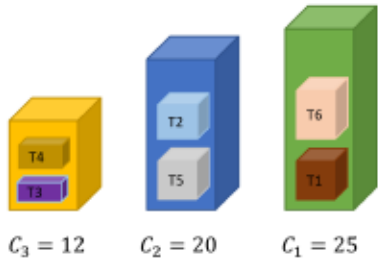


Fig. 3. DSTAR illustrative example.

Next, DSTAR sorts the servers in non-decreasing order of their capacities, i.e., $\{S_3, S_2, S_1\}$ (Line 14), and the four sets of tasks in $\mathcal{G}$ in non-increasing order of their efficiency $\tilde{E}$ (Lines 15 to 17). Using the task parameters from Table II, the server capacities $C_1 = 25$, $C_2 = 20$, and $C_3 = 12$, and the task sets in $\mathcal{G}$, the non-increasing order of efficiency is $\tilde{E}_1 > \tilde{E}_2 > \tilde{E}_3 > \tilde{E}_4$. As a result, DSTAR allocates the set of tasks $\{1,6\}$ first, then $\{3,4\}$, then $\{5\}$, and finally, $\{2\}$.

In Lines 18 to 27, DSTAR allocates one set of tasks at the time. The first set is $\{1,6\}$, i.e., T_1 and T_6. The first server considered is S_3, which has a total capacity of 12. Because the total request for the two tasks is 21 (10+11 according to Table II), S_3 could host only one of the two tasks in the set. However, DSTAR is not allowed to break tasks from the same set to avoid increasing the data size on the network. Thus, DSTAR checks the other two servers to see if they have enough capacity to allocate T_1 and T_6. S_1 has enough capacity, thus both T_1 and T_6 are allocated to S_1. A similar procedure is followed for the second set $\{3,4\}$, i.e., T_3 and T_4. DSTA previously allocated these tasks to S_2. However, doing so left T_2 unallocated. In particular, DSTAR finds that server S_3, the first one checked for capacity, is the one with minimum capacity able to hold T_3 and T_4. Thus, these tasks are allocated to S_3, which leaves the three servers with capacities $C_1 = 4$, $C_2 = 20$, and $C_3 = 0$. The only server able to allocate the remaining two sets of tasks is S_2 since its capacity 20 is larger than the requested demand of both the remaining sets, i.e., T_5 demands 10 and T_2 demands 9 for a total of 19. As a result, in the last two iterations, first T_5 and then T_2 are allocated to server S_2.

Finally, DSTAR checks whether the new allocation has improved the total profit compared to that of DSTA. Since DSTAR has allocated one more task compared to DSTA, its profit is higher and its allocation is placed in X. Fig. 3 shows the final allocation of tasks by DSTAR.

VII. Experimental Analysis

In this section, we investigate the performance of the proposed algorithms, DSTA and DSTAR, and a baseline algorithm that only maximizes profit. We implement the algorithms in Java and run the simulation experiments on a system with 4 cores, Intel (R) Core(TM) i5-8365U at 1.60 GHz, 8 GB of memory,

Algorithm 3: P – Greedy Algorithm.

Input: $\mathcal{T}$: set of tasks;
$\mathcal{S}$: set of edge servers.
1: $X \leftarrow [0]$
2: Sort servers in non-increasing order of capacity C_j. Let $S_{\beta(1)}, S_{\beta(2)}, \ldots, S_{\beta(M)}$ be the order.
3: **while** $|\mathcal{T}| > 0$ **do**
4: $\mathcal{I} = \{i | \forall\, l \in \{1, \ldots, N\} : p_l \leq p_i\}$
5: $\tilde{i} \leftarrow \text{argmin}_{i \in \mathcal{I}} \{r_i\}$
6: **for** $j = 1, \ldots, M$ **do**
7: **if** $C_{\beta(j)} - r_{\tilde{i}} \geq 0$ **then**
8: $\mathcal{T} \leftarrow \mathcal{T} \backslash \{T_{\tilde{i}}\}$
9: $C_{\beta(j)} \leftarrow C_{\beta(j)} - r_{\tilde{i}}$
10: $x_{\tilde{i}\beta(j)} \leftarrow 1$
11: **break**
12: **else**
13: **if** $j = M$ **then**
14: $\mathcal{T} \leftarrow \mathcal{T} \backslash \{T_{\tilde{i}}\}$
15: **output:** X

and 250 GB SSD of storage. First, we describe the baseline algorithm used for comparison and how we generate the taskset and data sharing characteristics. Then, we analyze the results.

A. Baseline Algorithm

The proposed DSTA and DSTAR algorithms jointly maximize the total profit and minimize the network data load by considering the data sharing characteristics of the tasks. Thus, for comparison purposes, we define the P – Greedy baseline algorithm that, similarly to existing work [15], [16], [17], [18], allocates tasks with the objective of maximizing the profit without considering the network data load. Other possible baseline algorithms that can be considered include Round-Robin and Random algorithms. The *Round-Robin algorithm* allocates tasks to servers in a circular order until all the tasks are allocated. However, Round-Robin does not focus on profit maximization, and thus, high-profit tasks may not get the appropriate resources, leading to sub-optimal profit. The *Random algorithm* assigns tasks to servers randomly. It promotes load balancing and is simple to implement, but it does not consider the profit associated with each task when making allocation decisions. This can lead to situations where high-profit tasks are assigned to less capable servers or delayed unnecessarily. The P – Greedy algorithm prioritizes tasks based on their profit, ensuring that the most profitable tasks are processed first. This direct focus on maximizing profit is better aligned with the objective of the task allocation problem considered here than either Round-Robin or Random. Thus, in our experiments, we consider only P – Greedy as the baseline algorithm.

P – Greedy is given in Algorithm 3. It has as input the set of tasks and the set of edge servers. First, it initializes the allocation matrix X (Line 1), which is also the output of the algorithm (Line 15). Then, it sorts the servers in non-increasing order of capacity (Line 2) and allocates one by one the tasks to the available servers (Lines 3 to 14). Specifically, in each iteration of allocation P – Greedy extracts the subset of tasks with the highest profit (Line 4) and, among them, selects the one with the lowest computational requirement (Line 5). Then, it finds the server with the highest available capacity to allocate the selected task and, accordingly, updates the server capacity and the allocation matrix entries (Lines 6 to 11). If there is no server with enough capacity, the selected task is left unallocated (Lines 12 to 14).

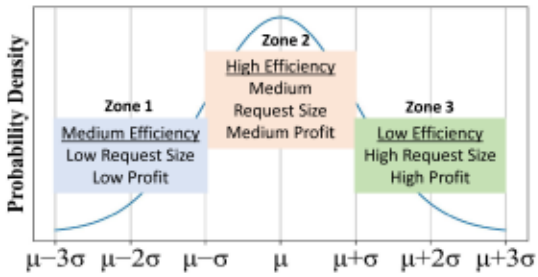


Fig. 4. Normal distribution zones used for generating task profit and request size.

Server Capacity and Taskset: We determine the capacity of the servers based on the total number of instructions that can be executed in a certain amount of time. We define the *demand ratio*, ρ, to characterize the relationship between the total amount of requests from users and the total capacity of the servers as follows:

$$\rho = \frac{\sum_{i=1}^{N} r_i}{\sum_{j=1}^{M} C_j}, \tag{11}$$

Based on the demand ratio, we consider three different cases: *i*) *low demand* ($0.8 < \rho < 1.2$), when the total requested capacity is approximately equal to the total available capacity of the edge servers; *ii*) *medium demand* ($1.8 < \rho < 2.2$), when the total requested capacity is about two times the total available capacity of the edge servers; and *iii*) *high demand* ($2.8 < \rho < 3.2$) when the total requested capacity is almost three times the total capacity available on the edge servers.

In order to generate the task sets, we first fix the desired demand ratio to 0.89, 2.0, and 2.87, corresponding to low, medium, and high demand cases. Second, as we discuss in the next paragraphs, we generate the task request r_i and profit p_i considering $N = 100$ tasks and normal distribution $\mathcal{N}(\mu, \sigma)$ for each case. Finally, we determine the number of servers and their capacities. To do so, we increase the server count by one unit at a time and draw its capacity from the uniform distribution within the interval $[\min(r_i), \max(r_i)]$, where $\min(r_i)$ and $\max(r_i)$ are the minimum and maximum request size of the generated tasks, respectively. We keep increasing the server count until the desired demand ratio (11) is satisfied.

In order to highlight the differences between DSTAR and DSTA, as shown in Fig. 4, we divide the normal distribution into three efficiency zones (according to (6)), i.e., Zone 1 *medium efficiency*, Zone 2 *high efficiency*, and Zone 3 *low efficiency*. As we notice from the figure, most of the N tasks fall into Zone 2 (i.e., high efficiency). Because the efficiency of each task depends on its request size and profit, we need to determine how to automatically generate task sets according to the *cross-zone*

TABLE III
DISTRIBUTIONS AND PARAMETERS USED FOR EVALUATION

Parameter	Distribution/Value
Demand Ratio ρ	0.89 (low), 2.0 (medium), 2.87 (high)
Request Size	$\mathcal{N}(\mu_r = 20, \sigma_r = 6)$
Profit	$\mathcal{N}(\mu_{p_{high}} = 30, \sigma_p = 1.5)$
Sharing Degree θ	0.2 (low), 0.5 (medium), 0.9 (high)
Task set Instances	45
Tasks per Instance	100
Data items	100
Image file size	pixel count: random[100, 400] width, height bit depth: random[1, 64] bit
Video file size	frame size: random[5, 1000] byte frame rate : random[12, 60] fps time: random[1, 20] seconds
Audio file size	bit depth: random[1, 64] bit sample rate: random[1, 16000] Hz audio length: random[10, 1000] seconds channels: random[1, 12] mono, stereo, quad

efficiency relationship, i.e., high probability for tasks in Zone 2 to have the highest efficiency and for those in Zone 3 to have the lowest efficiency. To do so, we draw the task requests sizes from the normal distribution with mean $\mu_r = 20$ and standard deviation $\sigma_r = 6$ (Fig. 4). Then, we assign request sizes and profits starting from Zone 3 tasks. In particular, requests between $r_{high} = \mu_r + \sigma_r$ and $\max(r_t)$ are assigned to Zone 3. To assign the profit values for these tasks, we use a normal distribution for each zone. The profit for tasks in Zone 3 is drawn from the normal distribution with mean $\mu_{p_{high}} = 30$ and standard deviation $\sigma_p = 1.5$.

B. Taskset and Data-Sharing Characteristics

Tasks in Zone 1 are characterized by a medium efficiency. Thus, requests between $\min(r_t)$ and $r_{low} = \mu_r - \sigma_r$ are assigned to Zone 1. In addition, the profit of these tasks is drawn from the normal distribution with mean $\mu_{p_{low}}$ so that $\mu_{p_{low}} > \mu_{p_{high}} \sqrt{\frac{r_{low}}{r_{high}}}$ and the same standard deviation σ_p as that used for Zone 3, which allows to maintain cross-zone efficiency relationship. Finally, tasks in Zone 2 are characterized by a high efficiency. Thus, requests between r_{low} and r_{high} are assigned to Zone 2, which have a mean request size $r_{med} = \mu_r$. The profit of these tasks is drawn from the normal distribution with mean $\mu_{p_{med}}$ so that $\mu_{p_{med}} > \mu_{p_{low}} \sqrt{\frac{r_{med}}{r_{low}}}$ and the same standard deviation σ_p of Zone 3. Table III summarizes the parameters used in the evaluation section.

Data Items and Data Sharing: For the data items and their sizes, we consider three typical data types that can be offloaded to the edge servers for analysis: image, video, and audio files. We use the set of parameters listed in Table III for each data type to calculate their sizes [34]. The table also shows the ranges for each of those parameters. Using those random data sizes, we generate tasksets of $N = 100$ tasks and $D = 100$ data items to allocate on servers.

We leverage the Erdös-Rényi random graph model [35] to generate the bipartite graph characterizing the data sharing pattern for the generated taskset. In the Erdös-Rényi model, for a graph with n vertices and m edges, the probability of generating each edge is given by: $\theta^m(1-\theta)^{\binom{n}{2}-m}$. The parameter $\theta \in [0, 1]$, called the *sharing degree* here, characterizes the sharing among tasks. Larger values of θ correspond to higher number of edges in the graph, i.e., a larger number of tasks share data items with each other. We implement the Erdös-Rényi model using the *igraph* R package. We generate instances for three different cases: *i*) *low sharing* ($0 \leq \theta \leq 0.3$), where only a few tasks share data items; *ii*) *medium sharing* ($0.3 < \theta \leq 0.6$); and *iii*) *high sharing* ($0.6 < \theta \leq 1$), where a large number of tasks share data items. Fig. 5 shows examples of randomly generated bipartite graphs for three different cases with 10 tasks and 10 data items (low sharing, $\theta = 0.2$; medium sharing, $\theta = 0.5$; and high sharing, $\theta = 0.9$).

The data sharing cases considered are realistic and resemble settings such as those presented in [36] and [37]. These references provide real-world examples of application-specific tasks sharing the same data. For example, typical tasksets of autonomous vehicles, as shown in [36], include various tasks that use a camera frame as input for obstacle detection, lane detection, segmentation, and localization. Depending on the scenario, such autonomous vehicles could also execute additional tasks based on those same camera frames. For example, an autonomous police car could analyze license plates in real time or analyze nearby pedestrians to find a suspect. Similarly, in [37] the authors describe tasksets of modern mobile augmented reality apps, which must include camera-based tasks for surface recognition, object detection, image classification, and pose estimation, as well as other app-specific tasks such as face recognition or natural language processing (e.g., Google Translate). Thus, these example tasksets from [36] and [37] currently require tens of tasks periodically analyzing the same camera frames. Considering that our entire dataset includes N = 100 tasks coming from several different users, the low data sharing cases match the current numbers of tasks sharing the same data since they could come from the same user's device, e.g., the same autonomous vehicle or a smartphone executing an augmented reality app. Furthermore, to analyze the performance of our algorithms for future tasksets executing even more tasks on the same input data, we test also the case of medium and high sharing. Finally, the example data inputs from Table III (e.g., image file size) represent the typical input data sizes of real-world applications. These considerations support the relevance and validity of our dataset in practical settings.

C. Experimental Results

To analyze the performance of the proposed algorithms, we consider nine different cases according to the demand and sharing ratios: (low, medium, high) demand × (low, medium, high) sharing. Each task has a certain resource demand and profit generated as discussed in the previous section. For conciseness, Table IV shows the labels used in the figures to refer to each one of the nine cases. For each of the nine cases (e.g., low demand low sharing) the tasksets and data sharing characteristics are generated five times using the same distribution settings of Table III. The results are then averaged for each case and presented in Table V. The results are in terms of i) profit ratio,

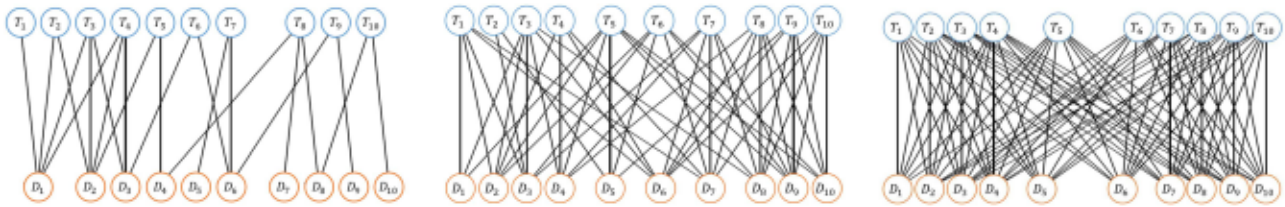


Fig. 5. Examples of bipartite graphs (10 tasks and 10 data items) generated using the Erdös-Rényi random graph model: (a) low data sharing, (b) average data sharing, and (c) high data sharing.

TABLE IV
NOTATION FOR CASE STUDIES

Expression	Description
LDLS	Low demand low sharing.
LDMS	Low demand medium sharing.
LDHS	Low demand high sharing.
MDLS	Medium demand low sharing.
MDMS	Medium demand medium sharing.
MDHS	Medium demand high sharing.
HDLS	High demand low sharing.
HDMS	High demand medium sharing.
HDHS	High demand high sharing.

TABLE V
PERFORMANCE COMPARISON BETWEEN DSTAR AND DSTA

Instance	Profit Ratio		Task Allocation %		Datasize Ratio
	DSTAR	DSTA	DSTAR	DSTA	
LDLS	**0.88**	0.72	**88.4%**	73.2%	1.16
LDMS	**0.87**	0.72	**86.9%**	73.6%	1.19
LDHS	**0.85**	0.7	**85.6%**	72.2%	1.21
MDLS	**0.48**	0.44	**50.2%**	46.3%	1.02
MDMS	**0.49**	0.45	**50.8%**	47.2%	1.06
MDHS	**0.5**	0.43	**51.4%**	45.9%	1.11
HDLS	**0.35**	0.32	**36.8%**	34.1%	**0.99**
HDMS	**0.36**	0.33	**37.7%**	35.6%	1.03
HDHS	**0.37**	0.34	**38.6%**	36.4%	1.04

defined as the profit normalized over the maximum achievable profit; ii) task allocation percentage, representing the percentage of allocated tasks over all tasks; and iii) datasize ratio, which is the average total data size ratio of DSTAR relative to DSTA. Further details on the comparison between the algorithms are discussed below.

Profit Analysis: Fig. 6(a) shows the average profit ratio obtained by DSTAR, DSTA, and P – Greedy for the nine demand-sharing cases, which is calculated here as the total profit of each algorithm over the maximum profit achievable if all tasks of the taskset could be allocated.

DSTA reduces the data size by up to 8 times at a minimal profit loss for task sets with demand ratio $\rho \leq 1$, as we have analyzed in [19], i.e., total task set demand is at most similar to the total server capacity. Here, we explore the scenarios for generally higher demand ratios $0.8 < \rho < 3.2$, i.e., the total task set demand is generally higher then the total servers capacity. Compared to P – Greedy, DSTA reduces data size by up to 20% with a profit reduction of 14%. On the other hand, we observe that DSTAR outperforms both algorithms in terms of profit for all the nine cases, by up to 22.84% and 13.9% compared to DSTA and P – Greedy, respectively. This is because DSTAR improves the initial allocation decision from DSTA and runs an efficiency analysis that allows to achieve a more efficient capacity usage that leads to profit increases. As we will discuss in the next paragraph, the increase in profit for DSTAR is mostly due to a larger number of tasks being allocated, which sometimes comes with a slightly larger network data size. Note that for all the three algorithms the profit decreases with the increase in demand ratio because of the lower number of allocatable tasks for higher demand instances.

Network Data Load Analysis: Fig. 6(b) shows the average data size ratio of DSTAR over DSTA and P – Greedy, which is calculated as the ratio between the total data size on the network considering the task allocation decision of DSTAR over that of each of the two baselines. In addition, Fig. 6(c) shows the percentile increase in the number of tasks assigned by DSTAR relative to DSTA and P – Greedy (i.e., a negative increase in the standard deviation bar would mean DSTAR allocates similar/fewer tasks). As it can be seen, in most of the demand-sharing cases, the total size of data corresponding to the allocation obtained by DSTAR is larger than that corresponding to DSTA and P – Greedy. However, we can observe that DSTAR can increase the profit while limiting the total data size for higher data-sharing cases. For example, considering the high demand cases HDLS, HDMS, and HDHS, the data size ratio of DSTAR over P – Greedy is 1.12, 1.04, and 1.0, respectively. Yet, as Fig. 6(c) shows, DSTAR increases the number of allocated tasks compared to P – Greedy in the same high-demand scenarios between 12% and 17%, which leads to an increase in profit, as discussed in the previous paragraph. Compared to DSTA, in a few instances such as MDLS and HDLS, DSTAR also helps reduce the data size while increasing the profit and the number of allocated tasks. The reduction in data size is due to allocating tasks with higher profit and lower data size in place of some other tasks previously allocated by DSTA.

Referring to Fig. 6(c), one can notice that DSTAR succeeds to obtain an increase in total amount of assigned tasks compared to DSTA. The increase is more significant for low-demand cases since there is a higher chance for DSTAR to utilize a larger set of available servers versus high-demand cases where the available servers are limited in terms of count and capacity. On the contrary, we observe that DSTAR performs better in high-demand cases compared to P – Greedy. The reason is that P – Greedy

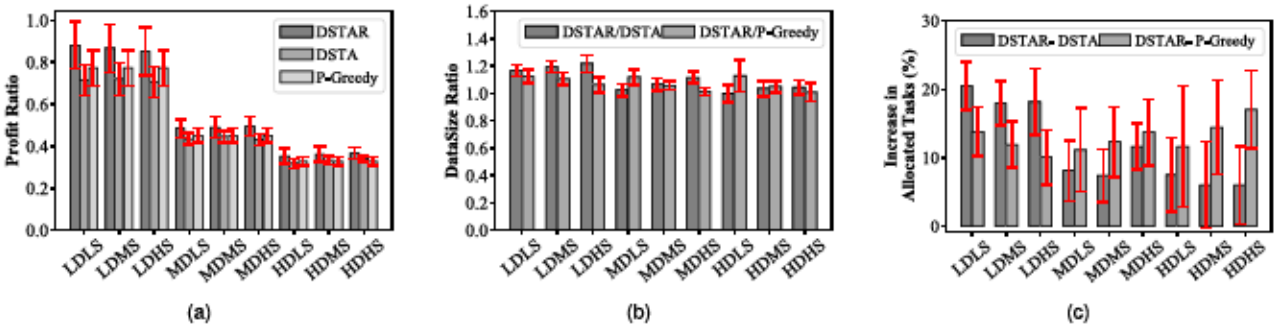


Fig. 6. (a) Average relative profit ratio normalized over the maximum profit and (b) average total data size ratio of DSTAR relative to DSTA and P−Greedy and (c) relative percentage increase in allocated tasks of DSTAR compared to DSTA and P-Greedy for various combinations of workload demand and data sharing characteristics across offloaded tasks.

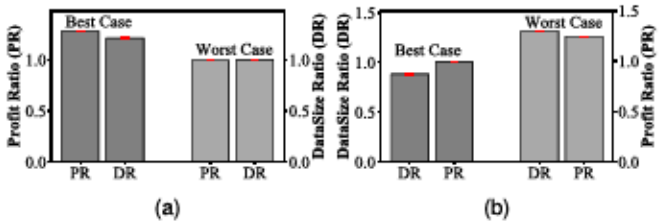


Fig. 7. The best and worst cases of DSTAR compared to DSTA in terms of: (a) profit and (b) data size for various combinations of workload demand and data sharing characteristics across offloaded tasks.

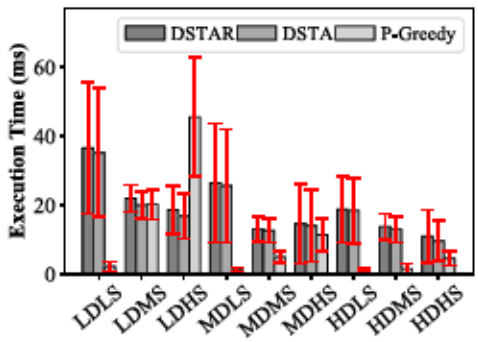


Fig. 8. The execution time of the algorithms for different combination of demand and sharing scenarios.

assigns the tasks with higher profit first and, according to our input data-set discussed in Fig. 4, the tasks with larger profit have also larger request size. Hence, P−Greedy allocates the server's capacity to the tasks in Zone 3, thereby running out of the available capacity faster with lower amount of assigned tasks. In summary, considering the average of the nine demand-sharing cases, DSTAR increases the profit by up to 22.84% and 13.9% compared to DSTA and P−Greedy, respectively. This is achieved by allocating more tasks, which leads to an increase of up to 16.6% and 12.5% of the data size on the network, respectively.

These results show that the network manager could employ DSTA to reduce network data size at a small profit loss or employ DSTAR to increase the total profit while limiting the increase of network data size.

Best Case and Worst Case Analysis: Here, we compare more in depth DSTAR with DSTA. Specifically, Fig. 7 shows two special cases for profit ratio (PR), which is defined here as the profit of DSTAR over DSTA, and data ratio (DR), defined as the data size of DSTAR over DSTA, among the 45 instances evaluated in previous sections. Fig. 7(a) shows that the best case PR is 1.27 with a corresponding DR of 1.21. The reason for the increase in profit and data size is due to a 25% increase in the number of allocated tasks. For the worst case in terms of profit, PR and DR are both equal to 1, which means that, in the worst case from the profit point of view, DSTAR does not degrade the DSTA's solution. Note that we manually verified that this result is not due to Lines 26 to 27 of DSTAR. In fact, in all the evaluated instances DSTAR never lowers the total profit compared to DSTA. Fig. 7(b) shows the best and worst cases from the point of view of data size. For the best case DSTAR leads to DR = 0.86 with a PR = 1, which means that DSTAR is able to find a solution with similar profit and a lower total data size. In the worst case, DR = 1.29 and PR = 1.26. These increases are due to a 24% increase in the allocated tasks.

Execution Time Analysis: Fig. 8 shows the execution time of the three algorithms. As expected, the algorithm with the lowest execution time is P−Greedy. However, as discussed in the previous paragraphs, this baseline leads to a lower profit compared to DSTAR and to a much higher data size on the network at similar profit compared to DSTA. The execution time of DSTAR is largely affected by that of DSTA and the execution of the reallocation algorithm in Lines 2 to 27 of DSTAR. On the other hand, the increase in execution time of DSTAR over DSTA due to the reallocation is minimal with an average of 2.2 ms overhead. For both DSTAR and DSTA, higher demand cases show a lower execution time. This is because, for higher demands there exist a lower chance for the tasks to be assigned to the servers, which lowers the number of allocatable tasks and the algorithms' execution time. In addition, the execution time decreases with larger data-sharing instances, which is due to the higher number of tasks allocated in each allocation round in Lines 26 to 37 of DSTA. These results show that DSTAR is able

to improve the solution of DSTA at a minimal execution time overhead.

VIII. Conclusion and Future Work

In this paper, we studied the data sharing-aware problem in edge computing systems, which we formulated as a bi-objective mixed-integer multilinear program that maximizes the profit derived from executing tasks on edge servers and minimizes the network traffic load by taking into account the data sharing characteristics of tasks. We designed DSTA, a greedy algorithm that considers the task data sharing characteristics to decide which tasks to allocate on the edge servers. In each iteration, DSTA maximizes the profit by prioritizing high-profit/light-workload tasks for allocation to the most suitable edge server. To equip DSTA to resource awareness for each task assignment we designed a reallocation algorithm called DSTAR, which receives the output of DSTA to find a better allocation that maximizes the total profit. Our experimental analysis showed that, compared to a representative greedy baseline that only maximizes profit, DSTA reduces network data size by up to 20% across 45 case study instances at a small profit loss. In addition, in the best-case scenario, DSTAR enhances DSTA, increasing the total profit by 27% and the number of allocated tasks by 25%, while limiting the increase of the total data traffic in the network.

In our future work, we plan to extend the proposed algorithms to consider constraints such as network and storage capacity, which make the data sharing-aware allocation problem a bi-criteria version of the multi-dimensional knapsack problem.

Acknowledgments

This article is a revised and extended version of [19] presented at IEEE International Conference on Edge Computing, (EDGE 2021).

References

[1] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading tasks with dependency and service caching in mobile edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 11, pp. 2777–2792, Nov. 2021.

[2] E. Kristiani, C.-T. Yang, and C.-Y. Huang, "iSEC: An optimized deep learning model for image classification on edge computing," *IEEE Access*, vol. 8, pp. 27267–27276, 2020.

[3] B. Yang, X. Cao, C. Yuen, and L. Qian, "Offloading optimization in edge computing for deep-learning-enabled target tracking by internet of UAVs," *IEEE Internet Things J.*, vol. 8, no. 12, pp. 9878–9893, Jun. 2021.

[4] X. Huang, L. He, and W. Zhang, "Vehicle speed aware computing task offloading and resource allocation based on multi-agent reinforcement learning in a vehicular edge computing network," in *Proc. IEEE Int. Conf. Edge Comput.*, 2020, pp. 1–8.

[5] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. S. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Trans. Commun.*, vol. 65, no. 8, pp. 3571–3584, Aug. 2017.

[6] Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu, and X. S. Shen, "TOFFEE: Task offloading and frequency scaling for energy efficiency of mobile devices in mobile edge computing," *IEEE Trans. Cloud Comput.*, vol. 9, no. 4, pp. 1634–1644, Fourth Quarter 2021.

[7] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, Jan. 2019.

[8] J. Zhang, M.-J. Piao, D.-G. Zhang, T. Zhang, and W.-M. Dong, "An approach of multi-objective computing task offloading scheduling based NSGS for IOV in 5G," *Cluster Comput.*, vol. 25, no. 6, pp. 4203–4219, 2022.

[9] Y.-Y. Cui, D.-G. Zhang, T. Zhang, J. Zhang, and M. Piao, "A novel offloading scheduling method for mobile application in mobile edge computing," *Wireless Netw.*, vol. 28, no. 6, pp. 2345–2363, 2022.

[10] D. Zhang, L. Cao, H. Zhu, T. Zhang, J. Du, and K. Jiang, "Task offloading method of edge computing in Internet of Vehicles based on deep reinforcement learning," *Cluster Comput.*, vol. 25, no. 2, pp. 1175–1187, 2022.

[11] S. Jošilo and G. Dán, "Decentralized algorithm for randomized task allocation in fog computing systems," *IEEE/ACM Trans. Netw.*, vol. 27, no. 1, pp. 85–97, Feb. 2019.

[12] C. Avasalcai, C. Tsigkanos, and S. Dustdar, "Decentralized resource auctioning for latency-sensitive edge computing," in *Proc. IEEE Int. Conf. Edge Comput.*, 2019, pp. 72–76.

[13] H. Badri, T. Bahreini, D. Grosu, and K. Yang, "Risk-aware application placement in mobile edge computing systems: A learning-based optimization approach," in *Proc. IEEE Int. Conf. Edge Comput.*, 2020, pp. 83–90.

[14] J. Lee, H. Ko, J. Kim, and S. Pack, "DATA: Dependency-aware task allocation scheme in distributed edge clouds," *IEEE Trans. Ind. Informat.*, vol. 16, no. 12, pp. 7782–7790, Dec. 2020.

[15] D. Zhang, Y. Ma, C. Zheng, Y. Zhang, X. S. Hu, and D. Wang, "Cooperative-competitive task allocation in edge computing for delay-sensitive social sensing," in *Proc. IEEE/ACM Symp. Edge Comput.*, 2018, pp. 243–259.

[16] D. Y. Zhang and D. Wang, "An integrated top-down and bottom-up task allocation approach in social sensing based edge computing systems," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 766–774.

[17] A. Kiani and N. Ansari, "Toward hierarchical mobile edge computing: An auction-based profit maximization approach," *IEEE Internet Things J.*, vol. 4, no. 6, pp. 2082–2091, Dec. 2017.

[18] H. Yuan and M. Zhou, "Profit-maximized collaborative computation offloading and resource allocation in distributed cloud and edge computing systems," *IEEE Trans. Automat. Sci. Eng.*, vol. 18, no. 3, pp. 1277–1287, Jul. 2021.

[19] S. Rabinia, H. Mehryar, M. Brocanelli, and D. Grosu, "Data sharing-aware task allocation in edge computing systems," in *Proc. IEEE Int. Conf. Edge Comput.*, 2021, pp. 60–67.

[20] Q. Chen, Z. Zheng, C. Hu, D. Wang, and F. Liu, "Data-driven task allocation for multi-task transfer learning on the edge," in *Proc. 39th IEEE Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 1040–1050.

[21] J. Meng, H. Tan, X.-Y. Li, Z. Han, and B. Li, "Online deadline-aware task dispatching and scheduling in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1270–1286, Jun. 2020.

[22] Y. Sahni, J. Cao, and L. Yang, "Data-aware task allocation for achieving low latency in collaborative edge computing," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 3512–3524, Apr. 2019.

[23] Y. Sahni, J. Cao, L. Yang, and Y. Ji, "Multi-hop multi-task partial computation offloading in collaborative edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 5, pp. 1133–1145, May 2021.

[24] J. Zhang, X. Zhou, T. Ge, X. Wang, and T. Hwang, "Joint task scheduling and containerizing for efficient edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 8, pp. 2086–2100, Aug. 2021.

[25] C. Li, J. Bai, and J. Tang, "Joint optimization of data placement and scheduling for improving user experience in edge computing," *J. Parallel Distrib. Comput.*, vol. 125, pp. 93–105, 2019.

[26] M. Breitbach, D. Schäfer, J. Edinger, and C. Becker, "Context-aware data and task placement in edge computing environments," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, 2019, pp. 1–10.

[27] Z.-L. Shao, C. Huang, and H. Li, "Replica selection and placement techniques on the IoT and edge computing: A deep study," *Wireless Netw.*, vol. 27, no. 7, pp. 5039–5055, 2021.

[28] H. Xing, L. Liu, J. Xu, and A. Nallanathan, "Joint task assignment and wireless resource allocation for cooperative mobile-edge computing," in *Proc. IEEE Int. Conf. Commun.*, 2018, pp. 1–6.

[29] Y. Chen, F. Zhao, X. Chen, and Y. Wu, "Efficient multi-vehicle task offloading for mobile edge computing in 6G networks," *IEEE Trans. Veh. Technol.*, vol. 71, no. 5, pp. 4584–4595, May 2022.

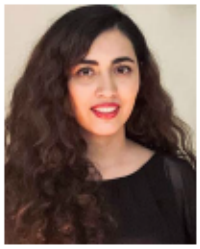
[30] B. Cao, Z. Li, X. Liu, Z. Lv, and H. He, "Mobility-aware multiobjective task offloading for vehicular edge computing in digital twin environment," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 10, pp. 3046–3055, Oct. 2023.

[31] L. T. Tan and R. Q. Hu, "Mobility-aware edge caching and computing in vehicle networks: A deep reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 67, no. 11, pp. 10190–10203, Nov. 2018.

[32] K. Zhang, J. Cao, and Y. Zhang, "Adaptive digital twin and multiagent deep reinforcement learning for vehicular edge computing and networks," *IEEE Trans. Ind. Informat.*, vol. 18, no. 2, pp. 1405–1413, Feb. 2022.

[33] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*. New York, NY, USA: WH Freeman, 1979.
[34] S. K. Alambra and D. Miszewska, "Omni calculator," Oct. 2020. [Online]. Available: https://www.omnicalculator.com/other
[35] P. Erdös and A. Rényi, "On random graphs I," *Publicationes Mathematicae Debrecen*, vol. 6, pp. 290–297, 1959.
[36] L. Liu, Z. Dong, Y. Wang, and W. Shi, "Prophet: Realizing a predictable real-time perception pipeline for autonomous vehicles," in *Proc. IEEE Real-Time Syst. Symp.*, 2022, pp. 305–317.
[37] N. Didar and M. Brocanelli, "Joint AI task allocation and virtual object quality manipulation for improved MAR app performance," in *Proc. IEEE 44th Int. Conf. Distrib. Comput. Syst.*, 2024, pp. 1154–1165.

Sanaz Rabinia (Student Member, IEEE) received the BSc degree in mathematics from Shahid Beheshti University, and the MSc degree in mathematics from the Sharif University of Technology, in 2011 and 2013, respectively. She is currently working toward the PhD degree in computer science with Wayne State University. Her research interests include edge computing, cloud computing, parallel algorithms, distributed systems, and game theory. She is a student member of the ACM.



Niloofar Didar (Student Member, IEEE) received the BE and ME degrees in computer engineering from Shahid Beheshti University, Iran, in 2015 and 2017. She is currently working toward the PhD degree with the Computer Science Department, Wayne State University. As her master thesis, she was working on indoor IoT based navigation systems for evacuation and rescue in fire incidents. Her research interest areas are autonomous systems, edge computing, human-computer interaction, Internet of Things, mobile computing, and smart city.

Marco Brocanelli (Member, IEEE) received the BE and ME degrees in control systems from the University of Rome Tor Vergata, Italy, and the PhD degree from the Electrical and Computer Engineering program of The Ohio State University, in 2018. He is an assistant professor with the Department of Electrical and Computer Engineering, The Ohio State University and the director of the Energy-aware Autonomous Systems Lab (EAS-Lab). His research interests are in the area of cyber-physical systems, energy-aware systems, Internet of Things (IoT), edge computing, embedded and real-time systems. In 2022, he received the CAREER award from NSF.

Daniel Grosu (Senior Member, IEEE) received the diploma degree in engineering (automatic control and industrial informatics) from the Technical University of Iaşi, Romania, in 1994 and the MSc and PhD degrees in computer science from the University of Texas at San Antonio, in 2002 and 2003, respectively. Currently, he is a professor with the Department of Computer Science, Wayne State University, Detroit. His research interests include parallel and distributed computing, approximation algorithms, and topics at the border of computer science, game theory and economics. He has published more than one hundred peer-reviewed papers in the above areas and is an IEEE COMPUTER SOCIETY DISTINGUISHED CONTRIBUTOR. He serves as a senior associate editor for ACM COMPUTING SURVEYS and as an associate editor for IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, and IEEE TRANSACTIONS ON CLOUD COMPUTING. He is a distinguished member of the ACM, a senior member of the IEEE Computer Society.