Offloading Dependent Tasks in Mobile Edge Computing with Service Caching

Gongming Zhao[1,3] Hongli Xu*[1,3] Yangming Zhao[2] Chunming Qiao[2] Liusheng Huang[1,3]
[1]School of Computer Science and Technology, University of Science and Technology of China
[2]Department of Computer Science and Engineering, University at Buffalo, The State University of New York
[3]Suzhou Institute for Advanced Study, University of Science and Technology of China

***Abstract*—In Mobile Edge Computing (MEC), many tasks require specific service support for execution and in addition, have a dependent order of execution among the tasks. However, previous works often ignore the impact of having limited services cached at the edge nodes on (dependent) task offloading, thus may lead to an infeasible offloading decision or a longer completion time. To bridge the gap, this paper studies how to efficiently offload dependent tasks to edge nodes with limited (and predetermined) service caching. We formally define the problem of offloading dependent tasks with service caching (ODT-SC), and prove that there exists no algorithm with constant approximation for this hard problem. Then, we design an efficient convex programming based algorithm (CP) to solve this problem. Moreover, we study a special case with a homogeneous MEC and propose a favorite successor based algorithm (FS) to solve this special case with a competitive ratio of $O(1)$. Extensive simulation results using Google data traces show that our proposed algorithms can significantly reduce applications' completion time by about 27-51% compared with other alternatives.**

***Index Terms*—Mobile Edge Computing, Task Offloading, Service Caching, Dependency, Approximation.**

I. Introduction

The Internet of Things and widespread use of mobile devices are driving the development of many delay-sensitive and resource-intensive applications, such as virtual/augmented reality, face recognition and data stream processing [1] [2] [3]. Currently, these applications are processed or performed on either mobile devices or on a cloud platform. On one hand, mobile device has too little computational resource for many applications [4]. On the other hand, running resource-intensive applications on a cloud platform often requires massive data be transferred between mobile devices and remote servers in the cloud, leading to unpredictable communication delay [5] [6]. As a result, Mobile Edge Computing (MEC) has emerged as a promising solution to overcome the above disadvantages [1] [7] [8] [9] [10].

However, there still exist many challenges in MEC. We take the face recognition application as an example. Basically, a face recognition application can be divided into five dependent tasks: object acquisition, face detection, preprocessing, feature extraction and classification [11]. When these tasks are offloaded to edge nodes, we need to take the following factors into considerations: 1) *Service caching*. Task execution may require the support of specific services. That means tasks can only be offloaded to edge nodes configured with corresponding services. For example, tasks "feature extraction" can only be offloaded to the edge nodes configured with trained machine learning model. 2) *Dependency*. There may be dependencies between tasks. For example, the output of task "feature extraction" is the input of task "classification". Thus, task "classification" can start only if task "feature extraction" has completed.

Actually, both *service caching* and *dependency* will impact the performance of task offloading. If we do not consider *service caching* or *dependency* when offloading these applications, the applications may not be performed successfully [2] [12]. Existing works on service caching often focus on the problem of joint optimization of service placement and task offloading in MEC [2] [4] [13] [14]. Xu *et al.* [2] formulated an online and decentralized algorithm, with the objective of computation latency minimization, to jointly optimize service caching and task offloading. Ouyang *et al.* [4] studied the dynamic service placement problem and proposed a Thompson-sampling based online learning algorithm to make adaptive service placement decisions. In fact, service placement/update may incur higher operation cost than task execution, and hurt the system stability [13]. For example, object database and trained machine learning models require a nontrivial amount of data and are time-consuming if we migrate these services [13]. Thus, service placement often occurs at long-term time scale. If we jointly update the service placement and task offloading at long-term time scale (*e.g.*, [4] [14]), due to task dynamics and uncertainty [15] [16], the offloading solutions may lead to computation congestion on some edge nodes. Different from the previous works, we assume that services have been placed/cached on edge nodes according to the existing methods such as [4] [14]. We will consider *the impact of service caching on the applications' performance (e.g., the completion time) of dynamic task offloading*.

Due to the limited memory resource, only a subset of services can be cached on an edge node [1]. The status of service caching (*i.e.*, where the services are hosted) will influence the decisions of task offloading. We give an example as shown in Fig. 1. Three tasks need to be offloaded to two edge nodes. Task 1 must finish and send corresponding data to task 2 before task 2 can start. Task 3 is independent of tasks 1 and 2. For simplicity, the processing delay for any task on any node edge is set as 1 and the

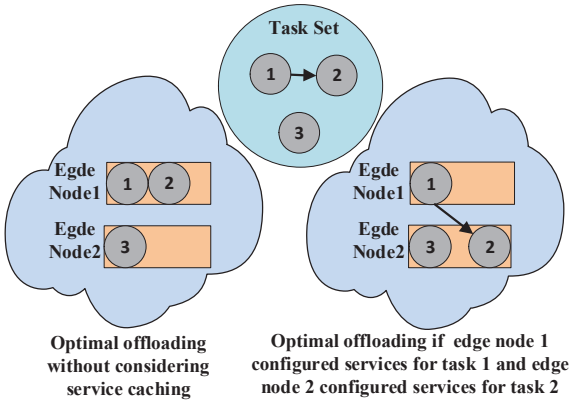


Fig. 1: A Motivation example. We assume three tasks need to be offloaded as illustrated in the top plot. The optimal offloading solution is shown in the left plot without considering service caching. The right plot is the offloading solution if only edge node 1 caches services for task 1 and only edge node 2 caches services for task 2.

communication delay is set as 0.5 for data transmission from task 1 to task 2 if these two tasks are offloaded to different edge nodes. If we do not consider the constraint of service caching, the optimal offloading solution is shown in the left plot of Fig. 1 and the makespan is 2. However, if only edge node 1 caches services for task 1 and only edge node 2 caches services for task 2, the offloading solution is shown in the right plot of Fig. 1 and the makespan is 2.5. In this case, the offloading strategy of the left plot is infeasible/inefficient due to lack of the services support (service migration is time-consuming). Thus, this paper focuses on *offloading dependent tasks with service caching*.

We should note that the problem of offloading dependent tasks in MEC is complicated and only some works considered the dependency constraints in MEC, such as [12] [17] [18]. Hermes *et al.* [17] designed a polynomial time approximation algorithm to minimize the makespan under resource constraints when offloading dependent tasks. Fan *et al.* [18] formulated the dependent task offloading problem to minimize the overall cost of all applications under each application's completion time constraints. Due to the complexity of offloading dependent tasks, only a few works consider the service caching constraints at the same time. The most related work is GenDoc [19], which jointly considered the problem of dependent task offloading and service caching placement with the objective of application completion time minimization. However, GenDoc does not consider the computing resource constraints when offloading tasks to edge nodes. In fact, mobile edge nodes are resource-sensitive and GenDoc may cause irrational use of limited computing resources. The main contributions of this paper are as follows:

1) We formally define the problem of offloading dependent tasks in MEC while considering service caching (ODT-SC), and prove its NP-hardness. We also analyze that the ODT-SC problem cannot be solved in polynomial time with a constant approximation algorithm.
2) We present a convex programming based algorithm, called CP, for the ODT-SC problem.
3) Moreover, we design a favorite successor based algorithm, called FS, for the special case (*i.e.*, homogeneous edge nodes), and prove that FS can achieve an approximation ratio of $O(1)$.
4) We conduct extensive simulations using real-world applications (from [20]) and data traces (from [21]) to show that CP and FS help reduce applications' completion time by about 27-51% compared with other alternatives.

The rest of this paper is organized as follows. Section II defines the problem of offloading dependent tasks in MEC while considering service caching and proves that there exists no approximation algorithm with a constant factor for this hard problem. In Section III, we propose an efficient convex programming based algorithm to solve this problem, called CP. Section IV focuses on the special case of homogeneous edge nodes and proposes an approximation algorithm with bounded approximation ratio. The simulation results are presented in Section V. We conclude the paper in Section VI.

II. Preliminaries and Problem Definition

In this section, we first introduce the system model, including task dependency model and network model. We then formally define the dependent task offloading with service caching (ODT-SC) problem, and prove that there exists no constant approximation algorithm for ODT-SC.

A. System Model

Task Dependency Model: We assume that one or several application(s) (*e.g.*, face recognition, virtual reality) need to be executed at some point in time. These application(s) can be divided into many tasks and each task can only be executed by one edge node. Note that, task execution may require the support of various resources (*e.g.*, storage, CPU, network I/O) and corresponding services (*e.g.*, machine learning mode) [22].

We use $V = \{v_1, v_2, ..., v_n\}$ to denote the set of tasks, where $n = |V|$ is the number of tasks. Some of these tasks may exist dependency. Given the precedence constraints among these tasks, we can use a directed acyclic graph (DAG) $G = (V, E)$ to denote the dependency between tasks, where V denotes the task set and E is set of edges representing the precedence constraints. A sink node of the DAG is a node such that no edge emerges out of it. More specifically, there is an edge from task v to task v' if and only if there exists data transmission from task v to task v' (*i.e.*, task v' can start only if task v is completed and the corresponding data is transmitted to task v'). We use $a_{vv'}$ to denote the amount of data that required to be transferred from task v to task v'.

Network Model: We consider an MEC network containing a set $M = \{m_1, m_2, ..., m_l\}$ of edge nodes, where

$l = |M|$ denotes the number of edge nodes. These edge nodes are interconnect with each other by kinds of network connections (*e.g.*, local-area network [23]). The communication delay per unit data from edge nodes m to m' is denoted by $c_{mm'}$ ($c_{mm'} = 0$ if $m = m'$). Each edge node has cached a subset of services. Let M_v represent the set of edge nodes that meet the service constraints for task $v \in V$. That means task $v \in V$ can only be executed by an edge node in M_v. Moreover, each edge node has limited resources (*e.g.*, CPU cycles, storage, computing, I/O [17] [24] [25]). For the sake of description, we only consider the resource constraints of CPU cycles in this paper. We assume each edge node $m \in M$ only assigns $C(m)$ CPU cycles to execute these tasks due to resource constraints, and it will consume t_{vm} execution time and r_{vm} CPU cycles per second if task v is offloaded to edge node m. Note that, it is easy to extend to other resources constraints [17] [25].

B. Problem Definition

Given a set of available edge nodes and a set of applications (each application consisting of multiple dependent tasks), we define the problem of offloading these dependent tasks with service caching (ODT-SC). We first construct a DAG according to the dependencies among tasks, as described in Section II-A. We then use a binary variable z_v^m to denote whether task $v \in V$ is offloaded to edge node $m \in M$ or not. A feasible offloading solution is defined as a start time t_v and an edge node m_v to execute each task $v \in V$ such that:

1) *All Tasks should be Offloaded*: Each task should be offloaded to exactly one edge node. That means, for each task $v \in V$, $\sum_{m \in M} z_v^m = 1$.
2) *Service Constraint*: Task $v \in V$ can only be offloaded to the edge node configured with corresponding required services, *i.e.*, the edge node in M_v. That means, $\sum_{m \in M_v} z_v^m = 1, \forall v \in V$.
3) *Dependency Constraint*: For any edge $< v, v' > \in E$, task v' can start iff all precedent tasks are completed and the required data is transferred to the edge node $m_{v'}$. That is, for each $< v, v' > \in E$, $t_v + \sum_{m \in M} z_v^m t_{vm} + \sum_{m \in M} \sum_{m' \in M} c_{mm'} a_{vv'} z_v^m z_{v'}^{m'} \leq t_{v'}$.
4) *Execute Tasks in Sequence*: For any pair of tasks $v, v' \in V$, if both tasks are offloaded to the same edge node m (*i.e.*, $m_v = m_{v'} = m$), then $t_v + t_{vm} \leq t_{v'}$ or $t_{v'} + t_{v'm} \leq t_v$. That means, each edge node can only perform one task at a time instance and tasks cannot be interrupted during the execution [12].
5) *Processing Resource Constraints*: The processing resource constraint should be satisfied for every edge node, which can be formulated as $\sum_{v \in V} z_v^m t_{vm} r_{vm} \leq C(m), \forall m \in M$.

The makespan of these offloaded tasks is denoted by $T = \max\{t_v + \sum_{m \in M} z_v^m t_{vm}, v \in V\}$. We aim to find a feasible offloading solution with a minimum makespan. Thus, the ODT-SC problem can be formulated as follows:

$$\min \quad T$$
$$s.t. \begin{cases} \sum_{m \in M} z_v^m = 1, & \forall v \in V \\ \sum_{m \in M_v} z_v^m = 1, & \forall v \in V \\ t_v + \sum_{m \in M} z_v^m t_{vm} + & \\ \sum_{m \in M} \sum_{m' \in M} c_{mm'} a_{vv'} z_v^m z_{v'}^{m'} \leq t_{v'}, & \forall < v, v' > \in E \\ \frac{t_v - t_{v'}}{\chi} < x_{vv'}, & \forall v, v' \in V \\ \chi(3 - z_v^m - z_{v'}^m - x_{vv'}) + & \\ t_v - t_{v'} \geq t_{v'm}, & \forall v, v' \in V, m \in M \\ \sum_{v \in V} z_v^m t_{vm} r_{vm} \leq C(m), & \forall m \in M \\ t_v + \sum_{m \in M} z_v^m t_{vm} \leq T, & \forall v \in V \\ z_v^m \in \{0, 1\}, & \forall m \in M, v \in V \\ t_v \geq 0, & \forall v \in V \\ x_{vv'} \in \{0, 1\}, & \forall v, v' \in V \end{cases} \tag{1}$$

The first set of equations represents that each task should be offloaded on exactly one edge node. The second set of equations denotes the service constraint. The third set of inequalities represents the dependency constraint. We let χ to represent a large number and $x_{vv'} \in \{0, 1\}, \forall v, v' \in V$. The fourth and fifth sets of inequalities denote the feature of edge node executing tasks in sequence. More specifically, for any two tasks v and v', there are two cases: 1) Tasks v and v' are both offloaded to edge node m and without loss of generality, we assume $t_v \geq t_{v'}$. In this case, $x_{vv'}$, z_v^m and $z_{v'}^m$ are all equal to 1. Thus, the fifth inequality can be simplified to $t_v - t_{v'} \geq t_{v'm}$, which guarantees that task v cannot start before task v' is finished. 2) Tasks v and v' are offloaded to different edge nodes, which means z_v^m and $z_{v'}^m$ cannot be equal to 1 at the same time. Under this case, $3 - z_v^m - z_{v'}^m - x_{vv'}$ can be larger than 0 and the fifth inequality holds regardless of the values of t_v and $t_{v'}$ (*i.e.*, there is no constraint between t_v and $t_{v'}$). Thus, the fourth and fifth sets of inequalities can guarantee that all tasks on the same edge node will be executed in sequence. The sixth set of inequalities represents the processing resource constraints. Our objective is to minimize the maekspan, *i.e.*, $\min T$.

Theorem 1: ODT-SC is one of the most difficult problems in NP-hard class: even finding a k-approximation algorithm (k is a constant) to solve ODT-SC is NP-hard.

Proof: On the one hand, we show that Travelling Salesman Problem (TSP) [26] is a special case of the ODT-SC problem. Due to space limit, we omit the detailed proof here. On the other hand, previous works have proved that finding a k-approximation algorithm for TSP is NP-Hard [26] [27]. Thus we can conclude that finding a k-approximation algorithm (k is a constant) to solve ODT-SC is NP-hard. ■

The above analysis shows the hardness of the ODT-SC problem. Thus, in this paper, we first design algorithms to solve the general ODT-SC problem in Section III and then design an approximation algorithm with bounded approximation factor for the homogenous scenario in Section IV.

III. Convex Programming based Algorithm for ODT-SC

In this section, we present a convex programming based algorithm for ODT-SC, called CP. The workflow of CP is shown in Fig. 2. Specifically, CP offloads dependent tasks through four major steps: 1) *Relaxing the ODT-SC problem* to construct a convex optimization program. 2) *Using progressive rounding* method to obtain feasible solutions for this relaxed problem. 3) *Computing weight* for each task according to the feasible solutions. 4) *Offloading tasks* following the weight values.

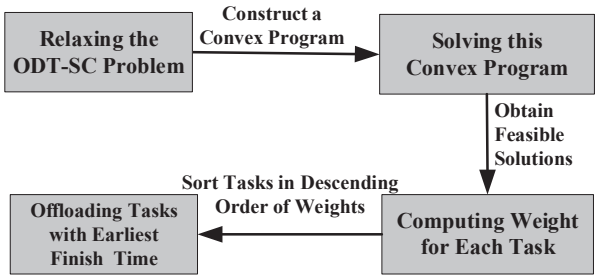


Fig. 2: Workflow of the CP algorithm. CP can be divided into four steps: relaxing the ODT-SC problem to construct convex program, using progressive rounding method to solve this program, computing weight for each task according to the solutions and offloading tasks according to the weights.

Relaxing the ODT-SC Problem. Eq. (1) is non-convex due to the third set of inequations, which increases the difficulty of solving this problem. To eliminate the non-convex in Eq. (1), we leverage the definition of binary variable z_v^m ($\forall v \in V, m \in M$) to give the following modification:

$$t_v + \sum_{m\in M} z_v^m t_{vm} + \sum_{m\in M}\sum_{m'\in M} c_{mm'} a_{vv'} z_v^m z_{v'}^{m'} = t_v + \sum_{m\in M} z_v^m t_{vm} + \sum_{m\in M}\sum_{m'\in M} c_{mm'} a_{vv'} \max[z_v^m + z_{v'}^{m'} - 1, 0] \tag{2}$$

If we then relax the integer constraints of variables z_v^m ($\forall v \in V, m \in M$) and $x_{vv'}(\forall v, v' \in V)$ to be continuous variables, we can solve this problem with a convex programming solver such as CPLEX [28]. However, we find the relaxation results are not good due to the large number χ in the fourth and fifth sets of inequalities. We can not get satisfactory performance if we offload tasks according to these results, which has been testified in Section V.

Thus, we present another method to solve this problem. We assume that edge nodes can perform tasks in parallel and each task $v \in V$ is splittable and can be executed on several edge nodes. In this way, we derive the following convex optimization problem:

$$\min \quad T$$

$$s.t. \begin{cases} \sum_{m\in M} z_v^m = 1, & \forall v \in V \\ \sum_{m\in M_v} z_v^m = 1, & \forall v \in V \\ t_v + \sum_{m\in M} z_v^m t_{vm} + \sum_{m\in M}\sum_{m'\in M} c_{mm'} & \\ a_{vv'} \cdot \max[z_v^m + z_{v'}^{m'} - 1, 0] \le t_{v'}, & \forall < v, v' > \in E \\ \sum_{v\in V} z_m^v t_{vm} r_{vm} \le C(m), & \forall m \in M \\ t_v + \sum_{m\in M} z_v^m t_{vm} \le T, & \forall v \in V \\ z_v^m \in [0,1], & \forall m \in M, v \in V \\ t_v \ge 0, & \forall v \in V \end{cases} \tag{3}$$

Since Eq. (3) is a convex optimization problem, we can solve it in polynomial time with a convex programming solvers such as CPLEX [28]. Assume that the optimal solutions for Eq. (3) are $\widetilde{z}_v^m$ and $\widetilde{t}_v$, $\forall v \in V, m \in M$, and the optimal objective value is $\widetilde{T}$.

Progressive Rounding. This phase obtains integer solutions $\widehat{z}_v^m$ for each $v \in V$ and $m \in M$ using the progressive rounding method [29]. More specifically, in each iteration, we first solve Eq. (3) and obtain fractional solutions $\widetilde{z}_v^m$. Then we choose part of tasks $v \in V$ with large value of $\max\{\widetilde{z}_v^m, m \in M\}$ and use randomized rounding method [30] to derive an integer solution $\widehat{z}_v^m$ for these chosen tasks. We then fix the integer solution in $\widehat{z}_v^m$ (*i.e.*, fix these rounding solutions as known quantities) and solve Eq. (3) again. In this way, we can obtain a feasible solutions $\widehat{z}_v^m$ after several iterations. That means, we get the offloaded edge node m_v for each task $v \in V$ (*i.e.*, $\widehat{z}_v^{m_v} = 1$) while assuming edge node can parallelly execute tasks in this step.

Computing Weights. This step computes the weight for each task. More specifically, we first insert an end task at the bottom of the DAG and connect this task with all sink nodes of the DAG. We then denote the weight of each link $< v, v' > \in E$ as $w < v, v' > = t_{vm_v} + c_{m_v m_{v'}} a_{vv'}$ and the weights of links connected with the end task are set as 0. In this way, we can compute the maximum distance from each task $v \in V$ to the end task, denoted by $W(v)$. It can be easily shown that the descending order of their distance to the end task preserves the dependency constraints and a larger value means potential longer execution time. Thus, we define a list Π and sort all tasks in descending order of their distance to the end task.

Offloading Tasks. We offload tasks following the order of list Π in this step. We first define some concepts/variables to facilitate the description of this part. We use $Pred(v)$ and $Succ(v)$ to denote the set of immediate predecessor and successor tasks of task $v \in V$, respectively, which can be obtained according to the DAG. Let $R(m)$ denote the rest processing resources for edge node $m \in M$, initialized as $C(m)$. We use $M_v(R)$ to denote the set of edge nodes that satisfy both processing resource and service constraints for task $v \in V$, formally $M_v(R) = M_v \bigcap \{m \,|\, R(m) \ge r_{vm} t_{vm}, m \in M\}$. Let $T(v, m)$ represent the first date at which there is a larger idle time slot on edge node $m \in M$

than t_{vm}, initialized as 0. Note that, the idle time slot may be between two already-offloaded tasks on edge node m or after the time all tasks offloaded on edge node m are completed. Moreover, for each task $v \in V$, the actual offloaded edge node, the actual start time and the actual finish time are denoted by m_v, $\bar{t_v}$ and $\bar{f_v}$, respectively, both are initialized to 0. With these definitions, if we offload task v on edge node m, we can calculate the earliest start time $EST(v,m)$ and earliest finish time $EFT(v,m)$:

$$EST(v,m) = \max(T(v,m), \max_{v' \in Pred(v)} (\bar{f_{v'}} + c_{m_{v'}m} a_{v'v})) \tag{4}$$

$$EFT(v,m) = EST(v,m) + t_{vm} \tag{5}$$

In each iteration, we choose the first task v left in list Π for offloading. We first compute the earliest finish time $EFT(v,m)$ for each edge node $m \in M_v(R)$ with Eq. (5) and then offload task v on edge node m with $\min_{m \in M_v(R)} EFT(v,m)$. After task v is offloaded, we record the actual offloaded edge node m_v. Besides, we update the actual start time $\bar{t_v}$, the actual finish time $\bar{f_v}$ and the rest processing resources $R(m_v)$ with Eqs. (6), (7) and (8), respectively.

$$\bar{t_v} = EST(v, m_v) \tag{6}$$

$$\bar{f_v} = \bar{t_v} + t_{vm_v} \tag{7}$$

$$R(m_v) = R(m_v) - t_{vm} r_{vm} \tag{8}$$

We repeat these operations until all tasks have been offloaded. The CP algorithm is formally described in Algorithm 1.

Algorithm 1 CP: Convex Programming based Algorithm for ODT-SC

Step 1: Relaxing ODT-SC Problem
Construct a convex optimization program in Eq. (3)
Obtain the optimal solution $\widetilde{z}_v^m$, $\widetilde{t}_v$
Step 2: Progressive Rounding
Derive an integer solution $\widehat{z}_v^m$ by progressive rounding for each $v \in V$, $m \in M$
Step 3: Computing Weights
Compute $W(v)$ for each task $v \in V$
Sort all tasks $v \in V$ in descending order of their distance to the end task and saved in list Π
Step 4: Offloading Tasks
for each task $v \in V$ **do**
  Obtain $Pred(v)$ according to DAG
  Initialize variables $\bar{t_v}$ and $\bar{f_v}$ to 0
for each edge node $m \in M$ **do**
  Initialize variables $R(m)$ to $C(m)$
while offloading list $\Pi \neq \emptyset$ **do**
  Select the first task v from the list Π
  Update $M_v(R) = M_v \bigcap \{m \,|\, R(m) \geq r_{vm} t_{vm}, m \in M\}$
  Compute $EFT(v,m)$ with Eq. (5) for $m \in M_v(R)$
  Offload task v on m with $\min_{m \in M_v(R)} EFT(v,m)$
  Record the offloaded edge node as m_v
  Update $\bar{t_v}$, $\bar{f_v}$ and $R(m_v)$ with Eq. (6), Eq. (7) and Eq. (8), respectively
  Delete task v from offloading list Π

IV. Favorite Successor based Algorithm for Homogeneous Scenario

Section III has proposed the CP algorithm to solve the general ODT-SC problem (*i.e.*, heterogeneous scenario). In practice, many applications will be executed in the homogeneous scenario [31] [32] [33]. In this scenario, we assume that all edge nodes have the same processing speed (*e.g.*, the same CPU capacity) and all links have the same transmission rate. In other words, we use e_v to denote the execution time t_{vm} if task $v \in V$ is offloaded on edge node $m \in M$ (*i.e.*, $e_v = t_{vm}$) and use c to denote the communication delay per unit date $c_{mm'}$ for each pair of edge nodes $m, m' \in M$ (*i.e.*, $c = c_{mm'}$). Besides, considering the processing capacity constraints of mobile edge nodes, we assume that the execution delay is not less than the communication delay for any task [32]. That means, for any task $v \in V$ with a successor $v' \in Succ(v)$, $e_v \geq c \cdot a_{vv'}$. Moreover, we do not consider the resource constraint $C(m)$ for these tasks. For this special case, we present an approximate algorithm with bounded approximation factor.

A. Favorite Successor based Algorithm for Homogeneous Scenario

We first give the definitions of favorite successor and predecessor for this problem.

Definition 1 (Favorite Successor [34]): For any task $v \in V$, if task $v' \in Succ(v)$ satisfies $\bar{t_{v'}} < \bar{t_v} + e_v + ca_{vv'}$, then task v' is called the favorite successor for task v.

Definition 2 (Favorite Predecessor [34]): For any task $v \in V$, if task $v' \in Pred(v)$ satisfies $\bar{t_{v'}} + e_{v'} + ca_{v'v} > \bar{t_v}$, then task v' is called the favorite predecessor for task v.

We can prove that each task $v \in V$ has at most one favorite successor/predecessor. If it exists, the favorite successor/predecessor must be offloaded to the same edge node as task v. Moreover, if task v' is the favorite successor of task v, then task v is the favorite predecessor of task v'. According to the definition of favorite successor, we present a favorite successor based algorithm to solve the special case (FS). Specifically, FS offloads tasks through two major steps: 1) *obtain favorite successor* without considering services and the number of edge nodes constraints. The results can reflect the dependency priority of the DAG. 2) *favorite successor based offloading* while considering the number of edge nodes and service constraints.

Obtaining Favorite Successor. We first attempt to offload tasks to edge nodes without considering services and the number of edge nodes constraints. In this situation, we only need to consider the dependency constraint and we formulate this problem as follows:

$$\min \quad T$$

$$s.t. \begin{cases} t_v + e_v + ca_{vv'} y_{vv'} \leq t_{v'}, & \forall < v, v' > \in E \\ \sum_{v' \in Succ(v)} y_{vv'} \geq |Succ(v)| - 1, & \forall v :< v, v' > \in E \\ \sum_{v' \in Pred(v)} y_{v'v} \geq |Pred(v)| - 1, & \forall v :< v', v > \in E \\ t_v + e_v \leq T, & \forall v \in V \\ t_v \geq 0, & \forall v \in V \\ y_{vv'} \in \{0, 1\}, & \forall < v, v' > \in E \end{cases} \tag{9}$$

where binary variable $y_{vv'}$ denotes whether task v' is the favorite successor of task v. Specifically, $y_{vv'} = 0$ represents that task v' is the favorite successor of task v, otherwise $y_{vv'} = 1$. The first set of inequalities indicates the dependency constraints. Specifically, if $y_{vv'} = 0$, then this set of inequalities turns into $t_v + e_v \leq t_{v'}$ (*i.e.*, no communication delay between task v and task v' if v' is the favorite successor of v). Otherwise, it turns into $t_v + e_v + ca_{vv'} \leq t_{v'}$ (*i.e.*, we need consider communication delay between task v and task v' if v' is not the favorite successor of v). The second and third sets of inequalities indicate that any task has at most one favorite successor/predecessor. Our objective is to minimize the makespan, *i.e.*, min T.

To solve this program in polynomial time, we relax the sixth set of constraints by setting $y_{vv'} \in [0, 1]$. In this way, we can obtain the optimal solutions $\widetilde{y}_{vv'}$ ($\forall < v, v' > \in E$) with a linear programming solver such as PuLP [35]. The second set of inequalities indicates that at most one successor v' of task $v \in V$ can satisfy $\widetilde{y}_{vv'} < 0.5$. Specifically, if there exists two successors v' and v'' of task $v \in V$ such that $\widetilde{y}_{vv'} < 0.5$ and $\widetilde{y}_{vv''} < 0.5$, then we have $\sum_{v' \in Succ(v)} y_{vv'} < |Succ(v)| - 1$, which is contradicts with the second set of inequalities. Thus, for each $< v, v' > \in E$, let $\widehat{y}_{vv'} = 0$ if $\widetilde{y}_{vv'} < 0.5$, and let $\widehat{y}_{vv'} = 1$ otherwise. In this way, we get the integer solutions, that is, the favorite successor (if exists) for each task.

Favorite Successor based Offloading. In this step, we leverage the results obtained by the first step to offload tasks. We first introduce some definitions for the sake of convenience. We use V_R to denote the set of tasks whose all predecessors have been offloaded, initialized as the set of tasks without predecessor. Let l_m denote the last offloaded task on edge node $m \in M$ and f_m denote the favorite successor of task l_m, both initialized as none. We use $Avail(v) = \max_{v' \in Pred(v)}(\bar{t_{v'}} + e_{v'} + ca_{v'v})$ to denote the available time that task v can be executed on any edge node.

Then we compute the earliest start time $EST(v, m)$ with Eq. (4) for each task $v \in V_R$ and edge node $m \in M_v$. For each edge node $m \in M_{f_m}$, if $EST(f_m, m) < T(f_m, m) + ca_{l_m f_m}$, edge node m can execute the favorite successor f_m earlier than other edge nodes. Thus, we try to reserve edge node m for executing task f_m. For each task $v \in V_R \bigcap Succ(l_m)$ and $v \neq f_m$ and $m \in M_v$ (*i.e.*, v is a potential competing successor), if 1) $Avail(v) \geq EST(f_m, m)$ or 2) there is an edge node $m' \neq m$ such that $EST(v, m') \leq Avail(v)$ and if $m' \in M_{f_{m'}}$ and $EST(f_{m'}, m') < T(f_{m'}, m') + ca_{l_{m'} f_{m'}}$, $v \notin Succ(l_{m'})$ or $v = f_{m'}$, we defer the earliest staring time of task v on edge node m, that is, $EST(v, m) = EST(f_m, m) + e_{f_m}$. Then we choose edge node m' with $\min_{v' \in V_R, m' \in M_{v'}} EST(v', m')$ to offload task v'. After task v' is offloaded, we record the actual offloaded edge node $m_{v'}$ and update $\bar{t_{v'}}$ and $\bar{f_{v'}}$ with Eq. (6) and Eq. (7), respectively. Besides, we update the last offloaded task $l_{m_{v'}}$ as task v' and update the unoffloaded ready set V_R. We repeat this iteration until all tasks have been offloaded. The FS algorithm is formally described in Algorithm 2.

Algorithm 2 FS: Favorite Successor based Algorithm for Homogeneous Scenario

Step 1: Obtaining Favorite Successor
Construct a linear program based on Eq. (9)
Obtain the optimal solution $\widetilde{y}_{vv'}$
Derive an integer solution $\widehat{y}_{vv'}$ for each $< v, v' > \in E$
Record the favorite successor (if exists) for each task based on the integer solution
Step 2: Favorite Successor based Offloading
for each task $v \in V$ **do**
  Obtain $Pred(v)$ and $Succ(v)$ according to DAG
  Initialize variables $\bar{t_v}$, $\bar{f_v}$ and $Avail(v)$ to 0
for each edge node $m \in M$ **do**
  Initialize the last offloaded task l_m to none
Compute the set V_R of unoffloaded tasks that all predecessors are offloaded
while $V_R \neq \emptyset$ **do**
  for each task $v \in V_R$ and edge node $m \in M_v$ **do**
    Compute $EST(v, m)$ with Eq. (4)
  for each edge node $m \in M$ that the last offloaded task l_m have a favorite successor $f_m \in V_R$ **do**
    if $m \in M_{f_m}$ and $EST(f_m, m) < T(f_m, m) + ca_{l_m f_m}$ **then**
      for each task $v \in V_R \bigcap Succ(l_m)$ and $v \neq f_m$ and $m \in M_v$ **do**
        $Avail(v) = \max_{v' \in Pred(v)}(\bar{t_{v'}} + e_{v'} + ca_{v'v})$
        if 1) $Avail(v) \geq EST(f_m, m)$ or 2) there is an edge node $m' \neq m$ such that $EST(v, m') \leq Avail(v)$ and if $m' \in M_{f_{m'}}$ and $EST(f_{m'}, m') < T(f_{m'}, m') + ca_{l_{m'} f_{m'}}$, $v \notin Succ(l_{m'})$ or $v = f_{m'}$ **then**
          Update $EST(v, m) = EST(f_m, m) + e_{f_m}$
  $EST_{min} = \min_{v \in V_R, m \in M_v} EST(v, m)$
  Choose one task $v' \in V_R$ to offloaded to edge node $m' \in M_{v'}$ which satisfying $EST(v', m') = EST_{min}$
  Use $m_{v'}$ to record the offloaded edge node
  Update $\bar{t_{v'}}$ and $\bar{f_{v'}}$ with Eq. (6) and Eq. (7), respectively
  Update the last offloaded task $l_{m_{v'}} = v'$
  Update the unoffloaded task set V_R that all predecessors are offloaded

B. Performance Analysis

This section analyzes the approximate performance of FS. If we do not consider the services and the number of edge nodes constraints, we can offload tasks satisfying the favorite

successor requirement. We first give the approximation ratio for this problem.

Theorem 2: If we do not consider edge node constraints and offload tasks according to the integer solutions obtained by Eq.(9), we can finish all tasks in a makespan at most $\frac{4}{3}$ times of the optimal makespan.

Proof: let T^{wo} denote the actual makespan and t_v^{wo} denote the actual start time of task $v \in V$ using the integer solutions to offload. Let T^{lp} denote the makespan obtained by linear program, which is the lower-bound of the optimal makespan (denoted as T^{opt}). We denote the weight of link $< v, v' > \in E$ as $w < v, v' >= e_v + ca_{vv'} y_{vv'}$.

Since we assume the system contains an unlimited number of edge nodes, we can offload any task once it receives all data from successors. Thus, T^{wo} equals to the longest path in the DAG. It means:

$$\frac{T^{wo}}{T^{lp}} \leq \max_{<v,v'>\in E} \left(\frac{w^{wo} < v, v' >}{w^{lp} < v, v' >}\right) = \max_{<v,v'>\in E} \left(\frac{e_v + ca_{vv'}\widehat{y}_{vv'}}{e_v + ca_{vv'}\widetilde{y}_{vv'}}\right) \quad (10)$$

If $\widehat{y}_{vv'} = 0$, then $\frac{e_v + ca_{vv'}\widehat{y}_{vv'}}{e_v + ca_{vv'}\widetilde{y}_{vv'}} \leq 1$. Otherwise $\widetilde{y}_{vv'} \geq 0.5$, which means:

$$\frac{e_v + ca_{vv'}\widehat{y}_{vv'}}{e_v + ca_{vv'}\widetilde{y}_{vv'}} \leq \frac{e_v + ca_{vv'}}{e_v + 0.5ca_{vv'}} \leq \frac{4}{3} \quad (11)$$

The last inequality holds because we assume that $e_v \geq ca_{vv'}$ for this special case. Hence we conclude that:

$$\frac{T^{wo}}{T^{opt}} \leq \frac{T^{wo}}{T^{lp}} \leq \max_{<v,v'>\in E} \left(\frac{e_v + ca_{vv'}\widehat{y}_{vv'}}{e_v + ca_{vv'}\widetilde{y}_{vv'}}\right) \leq \frac{4}{3} \quad (12)$$

■

We then give some features of the proposed FS algorithm.

Lemma 3: Let $l' = \min_{v\in V}(|M_v|)$ (*i.e.*, for any task, at least l' edge nodes are configured with required services) and $l'' = l - l'$. We use $T[0, \bar{t_v}]$ to denote the accumulate idle time on all edge nodes before the actual start time $\bar{t_v}$ of task v. Then we have :

$$T[0, \bar{t_v}] \leq (l' - 1)t_v^{wo} + l''\bar{t_v}$$

Proof: In the worst case, the other l'' edge nodes have not been configured with any service and all tasks require the support of service(s). Thus, all tasks can only be offloaded to l' edge nodes while leaving other l'' edge nodes idle. That means, the accumulate idle time on l'' edge nodes equals to $l''\bar{t_v}$. If we can show at most $(l'-1)t_v^{wo}$ accumulate idle time on l' other edge nodes, the proof is finished. This becomes the identical parallel machines scheduling with dependent tasks problem [34]. Due to space limit, we omit this proof here and the reader can see [34] for reference. ■

Lemma 4: Let T_l^{fs} denote the actual makespan by using the FS algorithm. Then we have:

$$T[0, T_l^{fs}] \leq (l' - 1)T^{wo} + l''T_l^{fs}$$

Proof: Let task v be the last completed task by the FS algorithm, by applying Lemma 3, we have

$$\begin{aligned} T[0, T_l^{fs}] &= T[0, \bar{t_v}] + T[\bar{t_v}, T_l^{fs}] \\ &\leq (l' - 1)t_v^{wo} + l''\bar{t_v} + (l - 1)e_v \\ &= (l' - 1)(t_v^{wo} + e_v) + l''(\bar{t_v} + e_v) \\ &\leq (l' - 1)T^{wo} + l''T_l^{fs} \end{aligned} \quad (13)$$

■

Now, we give the approximation performance of our proposed FS algorithm.

Theorem 5: Let T_l^{opt} to denote the optimal makespan for offloading on l edge nodes. We have $\frac{T_l^{fs}}{T_l^{opt}} \leq \frac{l}{l'} + \frac{4}{3}$.

Proof: We know that the accumulate idle time plus the whole tasks execution time is equal to the total time slices. By applying Theorem 2 and Lemma 4, we have:

$$\begin{aligned} &lT_l^{fs} = T[0, T_l^{fs}] + \sum\nolimits_{v\in V} e_v \leq (l' - 1)T^{wo} + l''T_l^{fs} \\ &+ \sum\nolimits_{v\in V} e_v \Rightarrow T_l^{fs} \leq \frac{l' - 1}{l'}T^{wo} + \frac{\sum_{v\in V} e_v}{l'} \leq \\ &\frac{4(l' - 1)}{3l'}T^{opt} + \frac{l}{l'}T_l^{opt} \leq \frac{4}{3}T_l^{opt} + \frac{l}{l'}T_l^{opt} \end{aligned} \quad (14)$$

■

The penultimate inequality holds because $T_l^{opt} \geq \frac{\sum_{v\in V} e_v}{l}$. Thus, we conclude that FS can achieve an approximate ratio of $\frac{l}{l'} + \frac{4}{3}$, where l is the number of edge nodes and $l' = \min_{v\in V}(|M_v|)$ (*i.e.*, for any task, at least l' edge nodes are configured with required services).

V. Performance Evaluation

This section evaluates the performance of the proposed algorithms by comparing with state-of-the-art methods over multiple offloading scenarios and applications using real-world applications (from [20]) and data traces (from [21]).

A. Performance Metrics and Methodology

Due to space limit, we mainly focus on the comparison of makespan in this section, which is one of the most important metrics for the task offloading problem. We compare CP and FS with the following existing approaches.

- The first one is the Individual Time Allocation with Greedy Offloading (ITAGS) algorithm [12], which aims at minimizing the communication and computation costs while satisfying makespan constraint. Specifically, ITAGS first uses a binary-relaxed version of the original problem to allocate a completion deadline for each individual task, and then greedily optimizes the offloading of each task subject to its time allowance. For fair comparison, we modify the objective of ITAGS to makespan minimization while satisfying processing resources constraints. In this way, ITGAS can solve the same problem proposed in this paper.
- The second one is offloading tasks according to the relaxed solutions of Eq. (1) obtained by using traditional relaxing and rounding method. More specifically, we first relax the integer constraints of variables z_v^m ($\forall v \in V, m \in M$) and $x_{vv'}(\forall v, v' \in V)$ in Eq. (1) to be fractional variables and then replace the third

constraints with Eq. (2). In this way, we can solve this problem with CPLEX and obtain fractional solutions $\widetilde{z}_v^m$ and $\widetilde{x}_{vv'}$. In the end, we use progressive rounding method to obtain integer solutions and offload each task according to the integer solution $\widehat{z}_v^m$. That is, we offload task v on edge node m if $\widehat{z}_v^m = 1$. We use Rounding to denote this method.

- The last one is the traditional algorithm, denoted as Greedy. This algorithm picks tasks starting from the top of the DAG to keep dependency. Then it offloads each picked task to the edge node with minimum makespan while satisfying service and resource constraints.

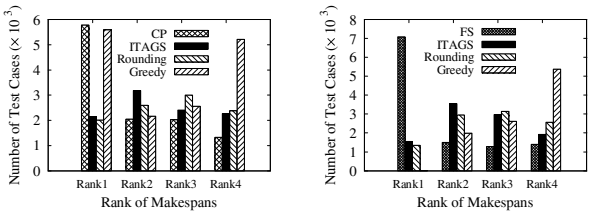
B. Simulation Settings

In this section, we introduce the simulation settings, including the generation methods of DAGs, task set settings, and the scenario settings for simulations.

1) DAG Generation: Similar to [12], we use the Gaussian Elimination (GE) algorithm and the Fast Fourier Transform (FFT) algorithm [20] to construct DAGs, respectively. Both generated structures are well known and used in real-world scenarios. The readers can see [20] for the detailed descriptions about the GE and FFT algorithms.

2) Task Set Settings: For the task set, similar to [15] [36], we use the data traces of google clusters [21]. Note that, these data traces only contain the information of processing time, dependency relationship and required processing resources for each task. To be more practical, we randomly generate other required information for our simulations. Specifically, for the general ODT-SC problem, to emulate the heterogeneous environment (*e.g.*, the processing delay of the same job would vary on different edge nodes), we scale the processing time collected from [21] with a factor with uniform distribution in (1,10). The ratio between the communication delay and the processing delay is uniformly randomized in (0.1,10). In other words, for each $< v, v' > \in E$, the communication delay for data transmission from task v to task v' is generated through multiply the processing time for task v with a random number in (0.1,10). Moreover, the required processing resources $t_{vm}r_{vm}$ is drawn uniformly in (1,10) for each task v on edge node m. Moreover, for each task, the percentage of edge nodes that configured with required services are denoted by Ω. We set Ω as 50% and the number of edge nodes as 10 by default. We mainly divide the simulations into three groups and each group of simulations contains two scenarios, one for the heterogeneous scenario (*i.e.*, the general ODT-SC problem) and the other for the homogeneous scenario (*i.e.*, the special case).

3) Simulation Scenario Settings: Actually, the type of network model (heterogeneous or homogeneous) is known in advance. Thus we perform the CP algorithm for the heterogeneous scenario and execute FS for the homogeneous scenario. The simulations are performed under two scenarios. Basically, the first scenario applies to the heterogeneous environment, *i.e.*, the general ODT-SC problem. We test the performance of CP, ITAGS, Rounding and Greedy under this scenario. The second scenario is applied to the homogeneous environment, *i.e.*, the special case illustrated in Section IV. We compare FS with ITAGS, Rounding and Greedy under this scenario.



(a) for the ODT-SC Problem. (b) for the Heterogeneous Scenario.

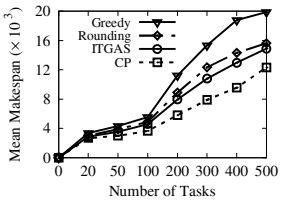
Fig. 3: Number of Test Cases vs. Rank of Makespans.

C. Simulation Results

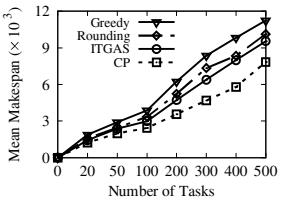
We run three sets of simulations on two different DAGs (*i.e.*, GE and FFT structures) to demonstrate the effectiveness of our proposed algorithms.

Comparison on the Number of Test Cases in Different Rankings: In the first set of simulations, we first randomly generate 200 DAGs using GE and FFT algorithms with the number of tasks from 5 to 500. For each DAG, 50 different random settings are generated with different numbers of edge nodes and different computation/communication/resources/services requirements for the general ODT-SC problem. Thus, we generate totally 10,000 random test cases. We evaluate CP, ITAGS, Rounding and Greedy on these test cases and rank the algorithms according to their makespan. The results are shown in Fig. 3(a). We observe that CP produces the minimum makespan in 57.83% (5,783 out of 10,000) of test cases. By comparison, ITAGS, Rounding and Greedy are in rank 1 in 2145, 2016 and 56 test cases, respectively. The results show that CP outperforms other algorithms on most of the test cases. Similarly, for each DAG, we generate 50 different random settings that meet the requirements of the special case. We test FS, ITAGS, Rounding and Greedy on these 10,000 test cases. As shown in Fig. 3(b), FS produces the shortest makespan in 70.85% (7,085 out of 10,000) of test cases and longest makespan on very little small portion (less than 1%) of the test cases. By comparison, ITAGS is in rank 2 for most test cases, Rounding is in rank 3 for most cases and Greedy produces the longest makespan for most cases. The results indicate our proposed FS algorithm outperforms other state-of-the-art solutions on most test cases.

Impact of the Number of Tasks on Makespan: The second set of simulations investigates the mean makespan by changing the number of tasks. We execute each simulation 100 times and average the numerical results. The results are shown in Figs. 4-5. Fig. 4 shows the results for the general ODT-SC problem with different DAG structures. As the number of tasks increases, the mean makespan increases for all algorithms. CP can always achieve lower mean makespan compared with the other three algorithms. For example, when there are 400 tasks in the FFT structure, the mean

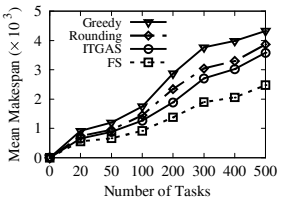


(a) for the GE Structure.

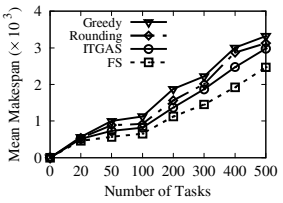


(b) for the FFT Structure.

Fig. 4: Mean Makespan vs. Number of Tasks for Heterogeneous Scenario.

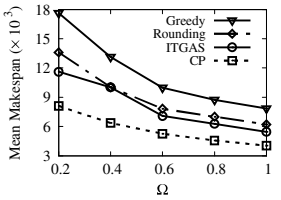


(a) for the GE Structure.

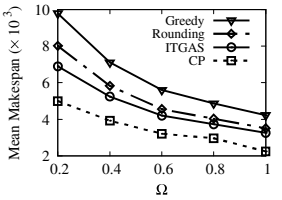


(b) for the FFT Structure.

Fig. 5: Mean Makespan vs. Number of Tasks for Homogeneous Scenario.

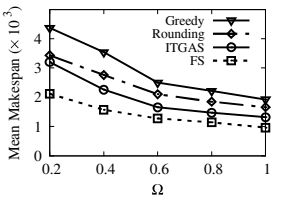


(a) for the GE Structure.

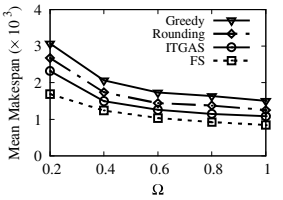


(b) for the FFT Structure.

Fig. 6: Mean Makespan vs. the Value of Ω for Heterogeneous Scenario.



(a) for the GE Structure.



(b) for the FFT Structure.

Fig. 7: Mean Makespan vs. the Value of Ω for Homogeneous Scenario.

makespan under CP is 5,783 while 7982, 8345 and 9834 under ITAGS, Rounding and Greedy, respectively. In other words, CP can decrease mean makespan by about 28%, 31% and 42% compared with ITAGS, Rounding and Greedy, respectively. Fig. 5 gives the results for the special case with different DAG structures. We observe that our proposed FS algorithm always outperforms three benchmarks. Basically, FS achieves the shortest mean makespan while ITAGS is in rank 2, Rounding is in rank 3 and Greedy produces the longest mean makespan. For example, when there are 300 tasks in the GE structure, our proposed FS algorithm can reduce the mean makespan by about 30%, 38% and 49% compared with ITAGS, Rounding and Greedy, respectively.

Impact of the Value of Ω on Makespan: The last set of simulations shows the mean makespan by changing the value of Ω, *i.e.*, for each task, the percentage of edge nodes that are configured with required services. The results are shown in Figs. 6-7, where the horizontal axes are the value of Ω. As the value of Ω increases, the mean makespan decreases under all algorithms and CP/FS always achieve lower mean makespan than other algorithms. Fig. 6 shows the results for the general ODT-SC problem with different DAG structures. We observe that CP always achieves the lower makespan compared with other algorithms. For example, for each task, if we only install required services on 40% of edge nodes for the GE structure, CP will achieve mean makespan of 6368, while ITAGS, Rounding and Greedy can achieve mean makespans of 10024, 10045 and 13141, respectively. That means CP reduces the mean makespan by about 36.4%, 36.7% and 51.5% compared with ITAGS, Rounding and Greedy, respectively. The results for the special case are shown in Fig. 7. Regardless of the proportion of the deployed services, FS always achieves lower makespan compared with other algorithms. For example, when the value of Ω is 0.2 in Fig. 7(b), FS reduces the mean makespan by about 27.1%, 36.9% and 44.9% compared with ITAGS, Rounding and Greedy, respectively.

From these simulation results, we find that CP/FS substantially outperform other algorithms, over a wide range of parameters in the number of edge nodes and the value of Ω. Note that, we also simulate the mean makespan by changing the communication-to-computation ratio and the number of edge nodes, the conclusion agrees with the above simulation results. Due to the limited space, we omit the description of these simulation results.

VI. Conclusion

In this paper, we have studied the problem of offloading dependent tasks with service caching to minimize the makespan (ODT-SC). We have proved that there exists no constant approximation algorithm for ODT-SC. A convex programming based algorithm has been designed to solve this problem. Moreover, we have studied the special case for the ODT-SC problem (*i.e.*, homogeneous scenario) and proposed an approximate algorithm with bounded approximation factor to solve this practical case. Extensive simulation results have shown the high efficiency of our proposed algorithms.

VII. Acknowledgement

This research of Zhao, Xu and Huang is partially supported by the National Science Foundation of China (NSFC) under Grants 61822210, U1709217, and 61936015; by Anhui Initiative in Quantum Information Technologies under No. AHY150300. The research of Qiao is supported in part by National Science Foundation (NSF) Grant CNS-1626374.

REFERENCES

[1] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[2] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 207–215.

[3] Y. Abe, R. Geambasu, K. Joshi, H. A. Lagar-Cavilla, and M. Satyanarayanan, "vtube: efficient streaming of virtual appliances over last-mile networks," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 16.

[4] T. Ouyang, R. Li, X. Chen, Z. Zhou, and X. Tang, "Adaptive user-managed service placement for mobile edge computing: An online learning approach," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1468–1476.

[5] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[6] M.-H. Chen, B. Liang, and M. Dong, "Joint offloading and resource allocation for computation and communication in mobile cloud with computing access point," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[7] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

[8] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2017.

[9] Y. Mao, J. Zhang, and K. B. Letaief, "Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems," in *2017 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2017, pp. 1–6.

[10] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[11] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld, "Face recognition: A literature survey," *ACM computing surveys (CSUR)*, vol. 35, no. 4, pp. 399–458, 2003.

[12] S. Sundar and B. Liang, "Offloading dependent tasks with communication delay and deadline constraint," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 37–45.

[13] V. Farhadi, F. Mehmeti, T. He, T. La Porta, H. Khamfroush, S. Wang, and K. S. Chan, "Service placement and request scheduling for data-intensive applications in edge clouds," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1279–1287.

[14] S. Pasteris, S. Wang, M. Herbster, and T. He, "Service placement with provable guarantees in heterogeneous edge computing systems," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 514–522.

[15] N. Eshraghi and B. Liang, "Joint offloading decision and resource allocation with uncertain task computing requirement," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1414–1422.

[16] Z. Meng, H. Xu, L. Huang, P. Xi, and S. Yang, "Achieving energy efficiency through dynamic computing offloading in mobile edge-clouds," in *2018 IEEE 15th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE, 2018, pp. 175–183.

[17] Y.-H. Kao, B. Krishnamachari, M.-R. Ra, and F. Bai, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," *IEEE Transactions on Mobile Computing*, vol. 16, no. 11, pp. 3056–3069, 2017.

[18] Y. Fan, L. Zhai, and H. Wang, "Cost-efficient dependent task offloading for multiusers," *IEEE Access*, vol. 7, pp. 115 843–115 856, 2019.

[19] L. Liu, H. Tan, S. H.-C. Jiang, Z. Han, X.-Y. Li, and H. Huang, "Dependent task placement and scheduling with function configuration in edge computing," in *Proceedings of the International Symposium on Quality of Service*. ACM, 2019, p. 20.

[20] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 682–694, 2013.

[21] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proceedings of the Third ACM Symposium on Cloud Computing*. ACM, 2012, p. 7.

[22] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "Mobile edge computing: Survey and research outlook," *arXiv preprint arXiv:1701.01090*, 2017.

[23] N. Chen, Y. Yang, T. Zhang, M.-T. Zhou, X. Luo, and J. K. Zao, "Fog as a service technology," *IEEE Communications Magazine*, vol. 56, no. 11, pp. 95–101, 2018.

[24] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 1, no. 2, pp. 89–103, 2015.

[25] Y. Mao, J. Zhang, S. Song, and K. B. Letaief, "Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 16, no. 9, pp. 5994–6009, 2017.

[26] M. Dorigo and L. M. Gambardella, "Ant colonies for the travelling salesman problem," *biosystems*, vol. 43, no. 2, pp. 73–81, 1997.

[27] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, Tech. Rep., 1976.

[28] I. I. CPLEX, "V12. 1: Users manual for cplex," *International Business Machines Corporation*, vol. 46, no. 53, p. 157, 2009.

[29] Y. Zhao, M. Pithapur, and C. Qiao, "On progressive recovery in interdependent cyber physical systems," in *2016 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2016, pp. 1–6.

[30] P. Raghavan and C. D. Tompson, "Randomized rounding: a technique for provably good algorithms and algorithmic proofs," *Combinatorica*, vol. 7, no. 4, pp. 365–374, 1987.

[31] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 856–868, 2018.

[32] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[33] E. Ahmed, A. Gani, M. K. Khan, R. Buyya, and S. U. Khan, "Seamless application execution in mobile cloud computing: Motivation, taxonomy, and open challenges," *Journal of Network and Computer Applications*, vol. 52, pp. 154–172, 2015.

[34] C. Hanen and A. Munier, "An approximation algorithm for scheduling dependent tasks on m processors with small communication delays," *Discrete Applied Mathematics*, vol. 108, no. 3, pp. 239–257, 2001.

[35] S. Mitchell, M. OSullivan, and I. Dunning, "Pulp: a linear programming toolkit for python," *The University of Auckland, Auckland, New Zealand*, 2011.

[36] H. Tan, Z. Han, X.-Y. Li, and F. C. Lau, "Online job dispatching and scheduling in edge-clouds," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.