The following publication Sahni, Y., Cao, J., Yang, L., & Ji, Y. (2020). Multi-hop multi-task partial computation offloading in collaborative edge computing. IEEE Transactions on Parallel and Distributed Systems, 32(5), 1133-1145 is available at https://doi.org/10.1109/TPDS.2020.3042224.

Multi-Hop Multi-Task Partial Computation Offloading in Collaborative Edge Computing

Yuvraj Sahni, Jiannong Cao, *Fellow, IEEE,* Lei Yang, and Yusheng Ji, *Senior Member, IEEE*

Abstract—Collaborative edge computing (CEC) is a recent popular paradigm where different edge devices collaborate by sharing data and computation resources. One of the fundamental issues in CEC is to make task offloading decision. However, it is a challenging problem to solve as tasks can be offloaded to a device at multi-hop distance leading to conflicting network flows due to limited bandwidth constraint. There are some works on multi-hop computation offloading problem in the literature. However, existing works have not jointly considered multi-hop partial computation offloading and network flow scheduling that can cause network congestion and inefficient performance in terms of completion time. This paper formulates the joint multi-task partial computation offloading and network flow scheduling problem to minimize the average completion time of all tasks. The formulated problem optimizes several dependent decision variables including partial offloading ratio, remote offloading device, start time of tasks, routing path and start time of network flows. The problem is formulated as an MINLP optimization problem and shown to be NP-hard. We propose a joint partial offloading and flow scheduling heuristic (JPOFH) that decides partial offloading ratio by considering both waiting times at the devices and start time of network flows. We also do the relaxation of formulated MINLP problem to an LP problem using McCormick envelope to give a lower bound solution. Performance comparison done using simulation shows that JPOFH leads to up to 32% improvement in average completion time compared to benchmark solutions which do not make a joint decision.

Index Terms—Scheduling and task partitioning, Network flow scheduling, Collaborative Edge Computing, Internet of Things.

1 Introduction

The proliferation of IoT devices with improved computation, communication, and storage capacities has lead to the development of applications such as Industry IoT, multi-robot systems, autonomous cars, etc [1] [2]. These applications have critical requirements such as real-time processing, mobility, context awareness, etc. which cannot be supported by centralized cloud computing. Edge computing aims to meet the needs of such applications by pushing computation within the network close to data sources. However, as applications evolve, there is need to support collaboration among different types of devices so that different applications and stakeholders can coexist. Therefore, the concept of collaborative edge computing has been introduced recently to support applications which have such requirements.

We have defined collaborative edge computing as a computing paradigm where multiple stakeholders (IoT devices, edge devices, or cloud) collaborate with each other by sharing data and computation resources to satisfy individual and/or global goals. This definition includes both vertical collaboration where different layers such as device layer, edge computing layer, and cloud layer collaborate with each other and horizontal collaboration where different edge devices in edge computing layer collaborate with each other [2]. There are many major challenges in collaborative edge computing including collaboration space formation, social trust-based incentive policies, cooperation policies, inter-domain cooperation, intra-domain cooperation, smart collaborative networking, and mobility management [2].

There are many existing works such as [3], [4], [5], [6], [7] etc. which are based on the principle of collaborative edge computing. The work in [5] proposes CVEC to support large-scale vehicular services by using both horizontal and vertical collaboration. The work in [7] proposes collaborative edge computing among small-cell base stations (SBSs) by forming SBS coalitions to share computation resources with each other. We have also proposed Edge Mesh [8] where decision-making is done inside the network by sharing data and computation tasks among mesh network of edge devices instead of sending all the data to a centralized server. One of the fundamental issues in collaborative edge computing is distributing tasks among heterogeneous edge devices. This includes deciding on whether, when, and where to offload the tasks and how to optimally utilize the bandwidth and computation resources of the underlying network.

This paper studies the problem of multi-hop multi-task partial computation offloading in collaborative edge computing with the objective of minimizing the average completion of all tasks. The problem jointly considers partial offloading of independent tasks generated at different edge devices to any other edge device at a multi-hop distance and network flow scheduling. The partial offloading of a task

- *This work was supported in part by the Research Grant Council (RGC) General Research Fund under Grant PolyU 15217919, and in part by the RGC Research Impact Fund (RIF) under Grant R5034-18.*
- *Yuvraj Sahni and Jiannong Cao are with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. (E-mail: {csysahni,csjcao}@comp.polyu.edu.hk)*
- *Lei Yang is with the School of Software Engineering, South China University of Technology, Guangzhou, China. (E-mail: sely@scut.edu.cn).*
- *Yusheng Ji is with the Information Systems Architecture Research Division, National Institute of Informatics, Tokyo, Japan and also with the Department of Informatics, SOKENDAI (The Graduate University for Advanced Studies), Tokyo, Japan. (E-mail: kei@nii.ac.jp).*

implies that a task is partitioned into two components where one component is locally executed and other is offloaded to a remote device. Network flow scheduling includes not only making a decision on routing path but also start time of input data flows. The decision of partial offloading is interdependent on network flow scheduling.

Existing works in literature usually solve the task offloading problem for single-hop network. Few recent works such as [9], [10], [11], [12], etc. have studied multi-hop computation offloading problem. Multi-hop offloading has an advantage over single-hop by enabling the use of underutilized resources in a mesh network of devices. Multi-hop offloading is useful for application scenarios such as unmanned aerial vehicle (UAV) robot swarms, autonomous vehicles, etc. that have few ground stations or road-side units which can be connected through a multi-hop network with the other devices. Partial offloading problem has also been studied in previous works such as [13] and [14]. However, these works do not consider both partial offloading in a network with multiple remote edge devices and network flow scheduling, which leads to an increase in the number of decision variables and makes the problem more challenging. Another main challenging issue is that data transmission cost is no longer constant and difficult to estimate as it depends on the partial offloading decision. The partial offloading decision can affect the start time of input data flows which is included in data transmission cost. This problem is useful for applications such as large-scale multi-camera video analytics where video and image data from multiple cameras is used for generating situational awareness. Another application scenario is optical character recognition (OCR) images which can be arbitrarily divided as assumed in this work [15].

The main contribution made in this paper are:

1) We have mathematically formulated the joint multi-hop multi-task partial computation offloading and network flow scheduling problem in collaborative edge computing with the objective of minimizing the average completion time of independent tasks. We consider the heterogeneity in independent tasks where each task has different computation load, input data, and release time. We also consider heterogeneity in device processing speed and link bandwidth.
2) We show the problem is NP-hard and therefore, propose a joint partial offloading and flow scheduling heuristic (JPOFH) which creates a priority of tasks considering release time and computation load and calculates the partial offloading ratio considering the waiting time at the devices and start time of input data flows. We also relaxed the formulated MINLP problem to an LP problem using McCormick envelope and solve it using Mosek solver in CVX to give a lower bound solution.
3) We have conducted simulation experiments to evaluate and compare the performance, in terms of average completion time of tasks and running time, of JPOFH against benchmark solutions. The benchmark solutions consider different possibilities, including local execution, remote execution, and separate partial offloading and flow scheduling solution. We make a comprehensive performance comparison by changing the values of different input parameters, including the number of tasks, number of devices, number of routing paths, and amount of input data. The performance comparison shows that JPOFH leads to up to 32% improvement in completion time compared to benchmark solutions.

The rest of the paper is as follows. In Section 2, we discuss some related works. In Section 3, we give the system model and problem formulation. In Section 4, we discuss relaxation of formulated problem and the proposed solution, JPOFH, for the multi-hop multi-task partial computation offloading problem. In Section 5, we have done the performance evaluation. Finally, we give the conclusion in Section 6.

2 Related Works

Computation offloading problem in edge computing has been well-studied. Some works such as [20] have also given performance guaranteed solution for computation offloading using an optimization framework. Most existing works study the computation offloading problem for single-hop. However, there are few recent works such as [9], [10], [11], [12], etc. that have studied multi-hop computation offloading problem. The work in [9] gave a distributed solution to make full task offloading decision to minimize the energy consumption. The work in [10] proposes an iterative algorithm for task assignment to devices in a multi-hop cooperative network. However, the work in [10] makes many assumptions in deriving the routing cost and does not consider partial offloading. The work in [11] jointly formulates the computation offloading and routing decision problem and solves using a game-theoretic approach by showing the existence of Nash equilibrium. The problem in [11] is solved for both unsliceable and sliceable tasks. The work in [12] also proposes a game-theoretic solution for multi-hop computation offloading problem; however, it considers local, edge, and cloud computing for the execution of tasks. Compared to these existing works which usually assume offloading of tasks to a single remote edge or cloud device, our work considers partial offloading to all available edge devices. Our work also considers network flow scheduling where we make decisions not only on the routing path but also start time of input data flows.

There are some works such as [21] and [16] which have also considered task allocation problem for multi-hop wireless sensor networks (WSN). However, these works do not consider partial offloading and network flow scheduling done in our work. The partial offloading problem has been addressed for single-hop in many works such as [13] and [14]. The work in [13] studied multi-user partial computation offloading problem considering both edge and cloud. They also design iterative heuristic algorithm to make the offloading decision dynamically. The work in [14] studied the partial computation offloading problem where mobile devices can partially offload the tasks to both single or multiple cloud servers with the objective of optimizing the energy consumption and latency of the application. There are some works such as [22], which also solve the computation peer offloading problem between small-cell base stations. Some works such as [17], [23], [24], [25], etc. have considered network condition while scheduling tasks.

TABLE 1: Comparison of existing works

Existing works	Multiple tasks	Partial Offloading	Bandwidth-aware	Joint network flow decision
[16] (2011)	Yes	No	Yes	Yes
[17] (2016)	Yes	No	Yes	No
[14] (2016)	Yes	Yes	No	No
[18] (2018)	Yes	No	No	No
[19] (2018)	Yes	No	Yes	Yes
[13] (2018)	Yes	Yes	No	No
[10] (2019)	Yes	No	Yes	No
[11] (2019)	Yes	Yes	Yes	No
This paper	Yes	Yes	Yes	Yes

The work in [17] proposes a task scheduling framework that utilizes the underlying network scheduler to make task placement decisions. Another work [23] solves the joint reducer placement and coflow bandwidth scheduling problem. Both works in [24] and [25] considered network bandwidth to make the task scheduling decision.

The different related works in literature solve some combination of partial computation offloading, peer offloading, multi-server offloading, or network flow scheduling problem.Table 1 shows the comparison of existing works. To the best of our knowledge, there is no related work which has addressed all the concerns. The work in this paper jointly formulates the partial computation offloading and network flow scheduling problem for multi-hop collaborative edge computing. Our previous work also studied data-aware task allocation problem in collaborative edge computing [19] where dependent tasks with input data at different devices are placed in a multi-hop mesh network. The data-aware task allocation problem can be mapped to a multi-hop full computation offloading for dependent tasks. However, compared to this work, data-aware task allocation did not consider partial offloading of tasks, the release time of tasks, and decision on the routing path. We also consider additional constraints for network flow scheduling, including no flows can pass through a link at the same time, whereas the work in [19] just considered the limited link bandwidth constraint.

The formulation methodology used in this work is similar to that in [18], where the authors study the problem of offloading dependent task and give a lower bound solution. However, compared to the work in [18], we study the multi-hop multi-task partial offloading computation for independent tasks.

3 System Model and Problem Formulation

This section first describes the system model including network and application model and then the problem formulation.

3.1 System Model

Fig 1 shows the system architecture of Edge Mesh based on collaborative edge computing paradigm where the intelligence is distributed and pushed within the network by sharing computation resources and data between mesh network of edge devices [8]. Edge devices is such an architecture can be heterogeneous in computation capacity and can also serve as routers, as shown in Fig 1. The computation tasks are generated at different edge devices and can be offloaded to other edge devices, even at a multi-hop distance.

The decision to partition and offload the tasks is made at a centralized SDN controller which is assumed to have global knowledge by collecting network and task-related information from all the edge devices and routers. The SDN controller includes different functional components, resource discovery and traffic rules registry, that are responsible for collecting information. This information is utilized by the functional components, task scheduler and flow scheduler in the controller, to make the offloading decisions. In practice, multiple SDN controllers could be used to enable scalability and fault tolerance. We have not considered the communication overhead to collect network knowledge in this model. SDN controller has been used in some previous works such as [26], [27], etc. to make scheduling decisions in wireless networks. The work in [28] proposed meSDN to extend the control of SDN to mobile devices. Another work [29] implemented a prototype for the algorithm proposed in [27].

Although the system model assumes that edge devices are connected using a wireless network, we have not fully considered all the issues due to dynamics in a wireless network such as spatial and temporal variation of wireless channel conditions, interference of wireless transmission among neighbouring devices [19]. Nevertheless, these issues in a wireless network should be considered as part of future work. The problem formulation in this work has been done assuming a static network condition.

The objective of the problem is to minimize the average completion time of all the tasks. We have included both communication and computation cost to make task offloading and network flow scheduling decisions. The computation cost includes both waiting time at the devices and time to execute the task. The communication cost includes waiting time to start the data transmission, and data transmission cost. We do not include propagation time in communication cost as it is usually very small. Further, we also ignore the switching cost for routing between subsequent links in the multi-hop path. In practice, these costs would influence the total cost; however, we ignore these costs to simplify the system model. Other works such as [18], [11], [30] etc. have used similar assumptions to calculate the total cost.

The network and application model used in formulating the problem is:

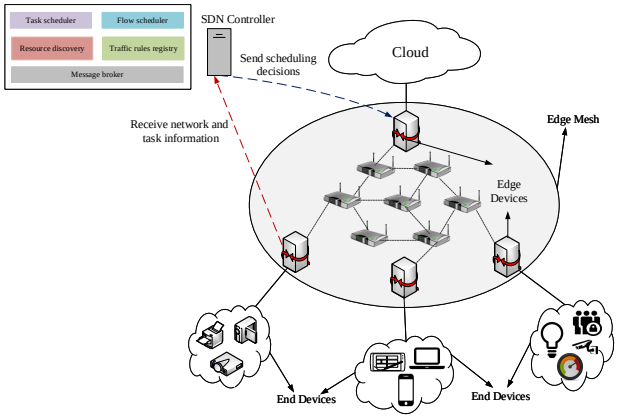


Fig. 1: System Architecture

Network model: The communication network is a mesh network of edge devices, shown to Edge Mesh circle in Fig 1, connected to each other using a multi-hop path. The communication network is modelled as a connected graph G = (V, E), where V is the set of devices, V = $\{j|1 \leq j \leq M\}$, and E is the set of links connecting different devices, E = $\{e_{\text{jv}}|j, v \in V\}$. Here, M is the total number of devices. In the problem description, we sometimes neglect the subscript and denote the link as e. The weight of each device is PS_j, which represents the processing speed of each device j. The devices can be heterogeneous in terms of processing speed. Each device is assumed to have a queue with positions equal to the number of tasks, i.e. N.

Any two devices are assumed to have maximum K available routing paths between them. A binary parameter W_{jvk} (1 for yes, 0 for no) is used to represents whether there is a k^{th} routing path between devices j and v. Another binary parameter Y_{kejv} (1 for yes, 0 for no) is used to represent if an edge e lies on k^{th} routing path between device j to device v. The number of hops and bandwidth of the k^{th}routing path between devices j and v is represented by H_{jvk} and R_{jvk} respectively.

Application model: An application is composed of a set of independent tasks, A = $\{i|1 \leq i \leq N\}$, where N is the total number of tasks. Each task i is associated with a computation load of CL_i and an amount of input data equal to D_i. Each task i is assumed to be generated at an edge device $z_i|z_i \in V$ at release time $Trel_i$. When a task i is partially offloaded to a device j, the input data corresponding to task i is also sent to device j. This transfer of input data is referred to as an input data flow.

Some of the assumptions made in the problem formulation are:

1) The local and offloaded component of a task can be executed independently.
2) The bandwidth for each flow is assumed to be given and equal to the minimum bandwidth of a link in the routing path.
3) The problem is solved for a static condition where the values of different parameters are known beforehand and there is no device or network failure during task execution.

TABLE 2: Notations used in the Paper

G = (V, E)	network model, where V is the set of devices and E is the set of edges
e_{jv}	link connecting device j and v
PS_j	processing speed of device j
W_{jvk}	binary parameter to represent if there is a k^{th} routing path between devices j and v
Y_{kejv}	binary parameter to represent if an edge e lies on k^{th} routing path between device j to device v
H_{jvk}	number of hops in the k^{th}routing path between devices j and v
R_{jvk}	bandwidth of the k^{th}routing path between devices j and v
A	set of tasks
CL_i	computation load of task i
D_i	amount of input data of task i
z_i	device where task i is generated
$Trel_i$	release time of task i
M	number of devices
N	number of tasks
K	number of routing paths
X_{ijr}	variable to represent ratio of task i executed on device j at position r
F_{ijk}	binary variable to represent if k^{th} routing path is used for input data flow of task i to device j
S_{ijk}	variable to represent start time of input data flow of task i offloaded to device j using k^{th} routing path
Ft_{ijr}	variable to represent finish time of component of task i executed on device j at position r
Tft_i	variable to represent total finish time of task i
$T_{ijk_{loc}}$	local execution time for task i partially offloaded to device j using k^{th} routing path
$T_{ijk_{loc}}$	offloaded execution time for task i partially offloaded to device j using k^{th} routing path
x_ijk	ratio of task i offloaded at device j using k^{th} routing path
$Tbusy_j$	waiting time at device j
$rank_i$	rank of task i in the priority list
i, u	index used to represent task i and u
j, v	index used to represent device j and v
k, w	index used to represent k^{th} and w^{th} routing path
r, s	index used to represent r^{th} and s^{th} position on the queue of device

4) Each device can execute one task at a time, while other tasks wait in the queue at the device.
5) Preemptive scheduling of tasks is not allowed.
6) No two flows are allowed to pass through a link at the same time to consider the interference among simultaneous wireless transmissions.

3.2 Problem Formulation

This section describes the constraints and formulates the problem as an optimization problem. Table 2 summarizes the notations used in the paper.

3.2.1 Constraints

There are several constraints in the problem related to task offloading and flow scheduling decision. Equation (1)-(4) represent constraints corresponding to task offloading decision. The problem considers that each task is offloaded only to a single remote device as represented by equation (1).

$$\sum_{r=1}^{N} X_{ijr} * \sum_{s=1}^{N} X_{ivs} = 0, \quad \forall i \in A, \quad j \in V \backslash \{z_i\}, \quad v \in V \backslash \{z_i, j\} \tag{1}$$

Each task consisting of local and offloaded component is placed only once as represented by equation (2).

$$\sum_{j=1}^{M} \sum_{r=1}^{N} X_{ijr} = 1, \quad \forall i \in A \tag{2}$$

Since the number of positions at the queue on each device is equal to the number of tasks, each task can occupy only one position on each device. This is represented by equation (3).

$$X_{ijr} * X_{ijs} = 0, \quad \forall i \in A, \quad j \in V, \quad r = 1, ...N, \quad s = 1, ...r-1, r+1, ...N \tag{3}$$

Furthermore, each queue position on each device can be occupied by only one task as represented by equation (4).

$$X_{ijr} * X_{ujr} = 0, \quad \forall i \in A, \quad u \in A \backslash \{i\}, \quad j \in V, \quad r = 1, ...N \tag{4}$$

The constraints corresponding to flow scheduling decision are represented by equation (5)-(8). There is an input data flow corresponding to offloading the task to a remote device as represented by equation (5).

$$\sum_{r=1}^{N} X_{ijr} * (\sum_{k=1}^{K} F_{ijk} - 1) = 0, \quad \forall i \in A \quad j \in V \tag{5}$$

The input data flow can start only after the task has been released as represented by equation (6).

$$S_{ijk} \geq Trel_i, \quad \forall i \in A, \quad j \in V, \quad k = 1, ...K \tag{6}$$

The problem assumes that a single routing path is used to transmit each input data flow as represented by equation (7).

$$\sum_{k=1}^{K} F_{ijk} \leq 1, \quad \forall i \in A, \quad j \in V \tag{7}$$

As mentioned earlier, it is assumed that no two flows are allowed to pass through a link at the same time. Therefore, if there are two flows passing through a link, one of the flows is delayed until the other one is finished as represented by equation (8).

$$\begin{aligned} &|Y_{kez_ij} * F_{ijk} - Y_{wez_uv} * F_{uvw}| * L \quad + \\ &(S_{ijk} - S_{uvw} - (H_{z_uvw} * \frac{D_u}{R_{z_uvw}} * \sum_{s=1}^{N} X_{uvs} * F_{uvw})) * \\ &(S_{ijk} + (H_{z_ijk} * \frac{D_i}{R_{z_ijk}} * \sum_{r=1}^{N} X_{ijr} * F_{ijk}) - S_{uvw}) >= 0 \\ &\forall i, u \in A, \quad j, v \in V, \quad k, w = 1, ...K, \\ &\forall e \in E, \quad r, s = 1, ...N \end{aligned} \tag{8}$$

where, L represents a large number.

There are also some constraints on the finish time of the task. Equation (9) represents that finish time of component of task i executed on device j at position r is greater than the start time of input data flow of task i component offloaded to device j through k^{th} routing path by the sum of computation time of component task i executed at device j and time to send the input data to execute tasks i component at device j.

$$\begin{aligned} &Ft_{ijr} - S_{ijk} \geq X_{ijr} * CT_{ij} + \\ &W_{z_ijk} * H_{z_ijk} * \frac{D_i}{R_{z_ijk}} * \sum_{r=1}^{N} X_{ijr} * F_{ijk} \\ &\forall i \in A, \quad j \in V, \quad k = 1, ...K, \quad r = 1, ...N \end{aligned} \tag{9}$$

The finish time of a successive task executed on a device is at least equal to the sum of the finish time of preceding task executed on that device at a previous position and computation time of the task. This constraint is represented by equation (10).

$$Ft_{ijr} >= Ft_{uj(r-1)} + X_{ijr} * CT_{ij} \quad \forall i, u \in A, \quad j \in V, \quad r = 1, ...N \tag{10}$$

The total finish time of task is greater than the finish time of any component of a task as represented by equation (11).

$$Tft_i >= Ft_{ijr} \quad \forall i \in A, \quad j \in V, \quad r = 1, ...N \tag{11}$$

Equation (12) - (16) represent the range of each decision variable.

$$0 \leq X_{ijr} \leq 1, \quad \forall i \in A, \quad j \in V, \quad r = 1, ...N \tag{12}$$

$$F_{ijk} = \{0, 1\}, \quad \forall i \in A, \quad j \in V, \quad k = 1, ...K \tag{13}$$

$$S_{ijk} \geq 0, \quad \forall i \in A, \quad j \in V, \quad k = 1, ...K \tag{14}$$

$$Ft_{ijr} \geq 0, \quad \forall i \in A, \quad j \in V, \quad r = 1, ...N \tag{15}$$

$$Tft_i \geq 0, \quad \forall i \in A \tag{16}$$

3.2.2 Optimization Problem

The objective function of the problem is to minimize the average completion time of all tasks. The objective function considers the time instance at which each task is finished (Tft_i) rather than the time period ($Tft_i - Trel_i$) to complete the task. As $Trel_i$ is a constant value, it is equivalent to minimizing the time period to complete the task. The multi-hop multi-task partial computation offloading problem, **P1**, can be formulated as:

$$\underset{X_{ijr}, F_{ijk}, S_{ijk}, Ft_{ijr}, Tft_i}{\text{minimize}} \frac{\sum_{i=1}^{N} Tft_i}{N}, \tag{17}$$

subject to (1) - (16)

$$\forall i \in A, \quad j \in V, \quad k = 1, ...K, \quad r = 1, ...N$$

This problem is NP-hard since it includes the Generalized Assignment Problem (GAP) as a special case (for a fully connected network). Since we cannot find an optimal

solution in polynomial time. Therefore, we have proposed a heuristic solution JPOFH (joint partial offloading and flow scheduling heuristic).

4 Joint Partial Offloading and Flow Scheduling Heuristic (JPOFH)

This section first gives detail about the proposed JPOFH algorithm and then gives a lower bound solution by relaxing the formulated problem.

4.1 JPOFH Algorithm

JPOFH determines the execution schedule specifying the selected device, finish time, the partial offloading ratio for each task, and the start time and selected path of each input data flow. JPOFH is developed considering three principles:

1) Using a priority list of task to determine the execution schedule.
2) Scheduling each task for execution only after the previous task in the priority list is executed.
3) Determining the partial offloading ratio considering both the waiting time at the device and start time of the input data flow.

Each task in JPOFH is scheduled with the objective of minimizing the finish time, which is defined in equation (18). The terms $T_{ijk_{loc}}$ and $T_{ijk_{off}}$ represent the local execution time and offloaded execution time for task i partially offloaded to device j using k^{th} routing path. $T_{ijk_{loc}}$ is defined in equation (19) as the sum of computation time at the local device and waiting time to execute the task at the local device. The waiting time at the local device is represented by $Tbusy_{z_i}$ where z_i stands for the local device for task i. $T_{ijk_{off}}$ is defined in equation (20) as the sum of computation time at the offloaded device, and maximum of time to send the input data to the offloaded device and waiting time to execute the task at offloaded device. The waiting time at the offloaded device is represented by $Tbusy_j$ where j stands for device j where task i is offloaded.

$$Tft_i = \min_{j\in V, k=1,...K} \max\{T_{ijk_{loc}}, T_{ijk_{off}}\}, \quad \forall i \in A \tag{18}$$

$$T_{ijk_{loc}} = (1 - x_{ijk}) * CT_{iz_i} + \max\{Tbusy_{z_i}, Trel_i\}, \quad \forall i \in A, \quad j \in V, \quad k = 1, ...K \tag{19}$$

$$T_{ijk_{off}} = x_{ijk} * CT_{ij} \quad + \max\{Tbusy_j, Trel_i, S_{ijk} + x_{ijk} * H_{z_ijk} * \frac{D_i}{R_{z_ijk}}\} \quad \forall i \in A, \quad j \in V, \quad k = 1, ...K \tag{20}$$

The equation (22) determines the partial offloading ratio, x_{ijk}, for task i to be offloaded at device j using k^{th} routing path. The value of x_{ijk} is determined by setting local execution time equal to the upper bound of offloaded execution time defined in equations (19) and (21) respectively. The work in [31] used a similar idea to derive the optimal solution for computation offloading using Nash equilibrium. However, [31] did not consider network flow scheduling required for multi-hop partial offloading.

$$T_{ijk_{off}} \leq x_{ijk} * CT_{ij} + x_{ijk} * H_{z_ijk} * \frac{D_i}{R_{z_ijk}} + \max\{Tbusy_j, Trel_i, S_{ijk}\} \quad \forall i \in A, \quad j \in V, \quad k = 1, ...K \tag{21}$$

$$x_{ijk} = \frac{CT_{iz_i} + \max\{Tbusy_{z_i}, Trel_i\} - \max\{Tbusy_j, S_{ijk}, Trel_i\}}{CT_{iz_i} + CT_{ij} + H_{z_ijk} * \frac{D_i}{R_{z_ijk}}} \quad \forall i \in A, \quad j \in V, \quad k = 1, ...K \tag{22}$$

Both offloaded execution time and partial offloading ratio require the value of S_{ijk}. We calculate S_{ijk} using equation (23) which is based on the assumption that no two flows can pass through the same link at the same time. Equation (23) represents that if the input data flow for task i passes through the same link in the routing path of any other input data, then it can start only after previous tasks have finished their flow.

$$S_{ijk} = \max_{u=1,...i, p\in P_{ijk}} (Trel_i, Y_{kpz_ij} * Y_{wpz_uv} * (S_{uvw} + x_{uvw} * H_{z_uvw} * \frac{D_i}{R_{z_uvw}})) \quad \forall i \in A\backslash\{1\}, \quad j \in V, \quad k = 1, ...K \tag{23}$$

where, P_{ijk} denotes the edges in the k^{th} routing path for an input data flow of task i offloaded to device j and v, w are the offloaded device and routing path selected for task u.

The details of JPOFH are given in Algorithm 1. Before starting the algorithm, JPOFH initializes the value of S_{ijk} to be equal to $Trel_i$ (line 2). JPOFH starts by creating a priority list of tasks based on increasing order of rank metric defined in equation (24). The rank metric considers both the release time and computation load of a task.

$$rank_i = Trel_i + \frac{CL_i}{\overline{PS}} \quad \forall i \in A, \quad j \in V, \quad k = 1, ...K \tag{24}$$

where, $\overline{PS}$ is the average value of PS.

Starting with the first task in the priority list, JPOFH calculates the value of S_{ijk} (line 9). Then, JPOFH calculates the value of x_{ijk}, $T_{ijk_{loc}}$, and $T_{ijk_{off}}$ (line 10 -11). The selected device for offloading, j_i^*, and routing path, k_i^*, each task i is determined based on the $\min_{j\in M, k\in K}\{T_{ijk_{loc}}\}$ value (line 18), which is also equal to finish time of the task i (line 19). The algorithm returns the selected device j_i^*, finish time Tft_i, partial offloading ratio $x_{ij_i^*k_i^*}$, start time of input data flow $S_{ij_i^*k_i^*}$, and selected routing path k_i^* for each task i (line 23).

The computation complexity of JPOFH is $\mathcal{O}(N^2 * M^2 * K)$. The computation complexity is calculated by considering the most complex operation in JPOFH algorithm, which is the calculation of S_{ijk} in line 9. There are three for loops for calculating S_{ijk} one loop each corresponding to the number of tasks (N), number of devices (M), and number of routing paths (K). The calculation of S_{ijk} in equation (23)

Algorithm 1: Joint Partial Offloading and Flow Scheduling Heuristic (JPOFH)

Input: The set of N tasks, the network of M edge devices, and k routing paths between all device pairs

Output: The execution schedule specifying the selected device, finish time, offloading ratio for each task, and the start time and selected routing path of each input data flow

$S_{ijk} \leftarrow Trel_i, \forall i \in A, j \in V, k = 1, ...K$;
Create an index I of tasks in increasing order of rank metric;
for $t \leftarrow 1$ *to* N **do**
  $i \leftarrow I(t)$;
  for $j \leftarrow 1$ *to* M **do**
    for $k \leftarrow 1$ *to* K **do**
      if $W_{ijk} \neq 0$ **then**
        if $t \geq 2$ **then**
          Calculate S_{ijk} using equation (23);
        Calculate x_{ijk} using equation (22);
        Calculate $T_{ijk_{loc}}$ and $T_{ijk_{off}}$ using equations (19) and (20) respectively;
      else
        $x_{ijk} \leftarrow 0$;
        $T_{ijk_{loc}} \leftarrow CT_{iz_i} + \max\{Tbusy_{z_i}, Trel_i\}$;
        $T_{ijk_{off}} \leftarrow Inf$;
    end
  end
  Find j_i^* and k_i^* based on $\min_{j \in M, k \in K}\{T_{ijk_{loc}}\}$;
  $Tft_i \leftarrow \min_{j \in M, k \in K}\{T_{ijk_{loc}}\}$;
  $Tbusy_i \leftarrow Tft_i$;
  $Tbusy_{j_i^*} \leftarrow Tft_i$;
  return $j_i^*, Tft_i, x_{ij_i^*k_i^*}, S_{ij_i^*k_i^*}, k_i^*$;
end

also requires maximum operation over different values of previous tasks in the priority list, i.e. $N-1$ in the worst case, and all the edges in the routing path, i.e. $M-1$ in the worst case. Hence, the complexity of JPOFH is $\mathcal{O}(N^2 * M^2 * K)$.

4.2 Lower Bound Solution

We have also proposed a lower bound solution by relaxing the formulated MINLP problem. The problem is non-convex due to the following three issues:

1) Bilinear terms such as $F_{ijk}. * \sum_{r=1}^{N} X_{ijr}$, $X_{ijr} * X_{ujr}$, $\sum_{r=1}^{N} X_{ijr} * \sum_{s=1}^{N} X_{ivs}$, and $X_{ijr} * X_{ijs}$, which make the problem non-convex.
2) Equation (8) is non-convex as it is of the form *quadratic expression* $\geq$ *constant*
3) Binary decision variable, F_{ijk}

The three issues in **P1** can be addressed by relaxing the problem in three stages.

In first stage, we can relax bilinear terms in **P1** into linear terms. The bilinear term $F_{ijk}. * \sum_{r=1}^{N} X_{ijr}$ can be relaxed by using McCormick envelope. The bilinear term is replaced by a new decision variable $Z_{ijk}, \forall i \in A, j \in V, k = 1, ...K$ and by adding four additional constraints. The equations (5) and (9) can be changed to equations (25), and (26) respectively:

$$\sum_{k=1}^{K} Z_{ijk} = \sum_{r=1}^{N} X_{ijr}, \qquad \forall i \in A, \quad j \in V \tag{25}$$

$$Ft_{ijr} - S_{ijk} \geq X_{ijr} * CT_{ij} + W_{z_ijk} * H_{z_ijk} * \frac{D_i}{R_{z_ijk}} * Z_{ijk} \quad \forall i \in A, \quad j \in V, \quad k = 1, ...K, \quad r = 1, ...N \tag{26}$$

The four additional constraints to replace bilinear term $F_{ijk}. * \sum_{r=1}^{N} X_{ijr}$ are:

$$Z_{ijk} \geq 0, \forall i \in A, \quad j \in V, \quad k = 1, ...K \tag{27}$$

$$Z_{ijk} \geq F_{ijk} + \sum_{r=1}^{N} X_{ijr} - 1, \quad \forall i \in A, \quad j \in V, \quad k = 1, ...K \tag{28}$$

$$Z_{ijk} \leq F_{ijk}, \quad \forall i \in A, \quad j \in V, \quad k = 1, ...K \tag{29}$$

$$Z_{ijk} \leq \sum_{r=1}^{N} X_{ijr}, \quad \forall i \in A, \quad j \in V \tag{30}$$

Same relaxation method can be applied to other bilinear terms. The details about the relaxation of all bilinear terms are shown in Appendix A.

In the second stage, we can relax the non-convex constraint in equation (8). We first expand and change the equation (13) to equation (31) by considering the relaxation done previously. The absolute function in equation (8) is also relaxed to a bilinear term in equation (31).

$$\begin{aligned}
&(Y_{kez_ij} * F_{ijk} * Y_{wez_uv} * F_{uvw}) * L + S_{ijk} * S_{ijk} + \\
&H_{z_ijk} * \frac{D_i}{R_{z_ijk}} * Z_{ijk} * S_{ijk} - 2 * S_{ijk} * S_{uvw} - \\
&H_{z_ijk} * \frac{D_i}{R_{z_ijk}} * Z_{ijk} * S_{uvw} + S_{uvw} * S_{uvw} - (H_{z_uvw} * \\
&\frac{D_u}{R_{z_uvw}} * Z_{uvw} * S_{ijk}) - (H_{z_uvw} * \frac{D_u}{R_{z_uvw}} * Z_{uvw} * H_{z_ijk} * \\
&\frac{D_i}{R_{z_ijk}} * Z_{ijk}) + H_{z_uvw} * \frac{D_u}{R_{z_uvw}} * Z_{uvw} * S_{uvw} >= 0 \\
&\quad \forall i, u \in A, \quad j, v \in V, \quad k, w = 1, ...K, \\
&\quad \forall e \in E, \quad r, s = 1, ...N
\end{aligned} \tag{31}$$

Equation (31) has many bilinear terms that can be relaxed to linear decision variables by using McCormick envelope. The details about the relaxation of all bilinear terms are shown in Appendix A.

In the third stage, we relax the binary decision variable, F_{ijk}, to a continuous variable. The range of continuous decision variable F_{ijk} can be set as:

$$0 \leq F_{ijk} \leq 1, \quad \forall i \in A, \quad j \in V, \quad k = 1, ...K \tag{32}$$

The new relaxed problem, **P2**, can be defined with the same objective function and replacing the constraints having bilinear terms with relaxed constraints. See the details in Appendix A. The relaxed problem, **P2**, is a convex linear

programming (LP) problem that has been solved using CVX [32] to give a lower bound solution. However, this lower bound solution is not feasible as many constraints are violated. The lower bound solution can still serve as a benchmark solution for performance evaluation.

5 EVALUATION

We have done simulation to evaluate and compare the performance of JPOFH with other benchmark solutions. The simulation experiments have been done on a MacBook Pro with 2.7 GHz Dual-Core Intel Core i5 processor. The performance evaluation has been done for two performance metrics: average completion time and running time of the algorithm. The parameters used for simulation are in similar range to the one used previously in [18] and [33].

5.1 Simulation Setting

Parameters for Network Model: We generate a network of edge devices where devices are deployed randomly using a uniform distribution. The size of the area is selected to be $M \times M$ square units, and any two devices less than $2*M/5$ units apart are connected to each other. The distance between devices is set to be in a similar range as done in previous works such as [33] and [19]. However, compared to the fixed-size area used in these works, a variable area size makes it easier to create connected mesh network topology even with a low number of devices. Besides, maintaining a similar network density using variable area size helps in avoiding network topology with too little or too much network links. All the devices are connected to each other using a multi-hop path to form a connected graph. Each vertex in the graph represents a device and its weight represents the processing power of the device. The weight of the link (edge) connecting two devices (vertices) represents the bandwidth capacity of the link (edge). The devices are heterogeneous in terms of processing power which is selected from a normal distribution with mean 50MCPS (Million Cycles Per Second) and variance 30%. The bandwidth of each link is selected from a normal distribution with mean 20Mbps and variance 30%.

Parameters for Application Model: The N tasks in the application model are generated at a device selected randomly. The computation load of each task is selected from a normal distribution with mean 300KCC (Kilo Clock Cycles) and variance 30% and the input data for each task transferred is selected from a normal distribution with mean 50 Kb and variance 30%. The release time of each task is selected from a normal distribution with mean 6 ms and variance 30%. The value of mean release time is calculated as the ratio of mean computation load of tasks to mean processing speed of devices.

5.2 Benchmark solutions

We have compared the performance the JPOFH algorithm with four benchmark solutions.

1) Lower Bound (LB): LB is the solution described in section 4 for the relaxed problem. Compared to JPOFH, LB is obtained after relaxing many constraints including offloading each task to only one remote device, executing no more than one task at each device, and not allowing any two flows to pass through a link at the same time. Due to these relaxations, LB gives an infeasible solution; however, it provides a loose lower bound for performance comparison.
2) Local execution (LE): LE is obtained by executing all the tasks at the local devices where the tasks are generated. LE is easy to obtain as it does not require consideration of flow scheduling. The finish time and waiting time for local execution are defined according to equation (33) and (34) respectively.

$$Tft_i = CT_{iz_i} + \max\{Tbusy_{z_i}, Trel_i\} \tag{33}$$

$$Tbusy_{z_i} = Tft_i, \quad \forall i \in A, \quad j \in V, \quad k = 1, ...K \tag{34}$$

3) Remote execution (RE): RE is obtained by executing the tasks at a remote device. Compared to JPOFH, RE only considers full offloading of tasks where the offloading is done using a greedy heuristic based on the priority list used in JPOFH. The flow scheduling in RE is done based on the priority order of tasks similar to JPOFH algorithm.
4) Separate offloading and flow scheduling (SOFS): SOFS is obtained by separating the partial task offloading and flow scheduling problem. Compared to JPOFH, where the tasks are executed on devices based on priority order, SOFS follows the first-come-first-serve in executing the tasks. Besides, the partial offloading ratio for SOFS, shown in equation (35), does not consider the start time of input data flows and waiting time at the devices. The flow scheduling is done in SOFS based on the priority order of the earliest deadline first (EDF) approach, used in flow scheduling algorithm PDQ [34]. Here, the deadline of the flow is based on the priority order of the tasks. Compared to JPOFH, flow scheduling in SOFS only requires calculation for the selected device and routing path for offloading the task.

$$x_{ijk} = \frac{CT_{iz_i}}{CT_{iz_i} + CT_{ij} + H_{z_ijk} * \frac{D_i}{R_{z_ijk}}} \quad \forall i \in A, \quad j \in V, \quad k = 1, ...K \tag{35}$$

5.3 Simulation Results

The default parameters used for the performance comparison are number of tasks as 5, number of devices as 10, number of routing paths between any source-destination pair as 3, and the amount of input data selected from a normal distribution with mean 50 Kb and variance 30%. The evaluation has been done under two different settings of selecting the device where each task is generated: fixed device and random device. In the fixed device setting, each task is generated on a separate device. In the simulation, we assume task i is generated on device j, where $i = j$. On the other hand, in the random device setting, the tasks can be generated randomly at any device. This implies that each device may have more than one task being generated in the random device setting. Table 3 and 4 show the performance

TABLE 3: Performance Comparison for default parameters (fixed device)

Metric	LB	LE	RE	SOFS	JPOFH
Completion time (sec)	0.0079 ± 0.0003	0.0138 ± 0.0008	0.0128 ± 0.0005	0.0115 ± 0.0008	0.0103 ± 0.0003
Running time (sec)	9479.8 ± 199.13	8.70E-05 ± 6.80E-05	0.0087 ± 0.0018	0.0021± 0.0018	0.0101 ± 0.0024

TABLE 4: Performance Comparison for default parameters (random device)

Metric	LB	LE	RE	SOFS	JPOFH
Completion time (sec)	0.0076 ± 0.0002	0.0139 ± 0.0008	0.0124 ± 0.0004	0.0109 ± 0.0006	0.0100 ± 0.0004
Running time (sec)	9907.8 ± 162.22	1.51E-04 ± 0.0001	0.0099 ±0.0023	0.0025 ± 0.0023	0.011 ±0.0027

comparison between JPOFH and benchmark solutions for fixed device and random device setting respectively. The results in both tables have been averaged for 30 iterations and the error margin is calculated for 95% confidence interval. The input values for the application and network model in each iteration are different and generated randomly.

It can be observed from both Table 3 and 4 that JPOFH performs better than the three benchmark solutions (LE, RE, and SOFS) in terms of average completion time. JPOFH is around 8% to 26% better in terms of average completion time depending on benchmark solution for both fixed and random device setting. The error margin for completion is less than 5% for 95% confidence interval. The running time of JPOFH is higher than the three benchmark solutions; however, it is still within the same range as other solutions. The value of running time changes depending on the system being used for simulation and the algorithm implementation. However, we can make some observations based on the trend in the values. The value of running time also shows large variation as we consider values from all iterations, even including outlier values. It should be noted that we have shown LB in the table for comparison; however, as mentioned previously, LB gives an infeasible solution due to relaxations of various constraints. We used Mosek solver in CVX to find the LB solution.

The rest of this section gives a detailed performance comparison by changing the values of the number of tasks, the number of devices, the number of routing paths, and the amount of input data. These parameters were chosen because the number of tasks, the number of devices, and the number of routing paths directly impact the algorithm complexity whereas varying the amount of input data helps in considering the change in communication cost. Similar to the results shown in Table 3 and 4, the following results have been averaged for 30 iterations and the error margin is calculated for 95% confidence interval. In order to provide better insight, we have used the fixed device setting for further evaluation unless specified otherwise. By fixing the device where each task is generated, we can study the trend in the values of average completion time and running time for a specific parameter.

5.3.1 Effect of number of tasks

We evaluate the effect of the number of tasks by changing the tasks from 2 to 20 as shown in Fig 2. We use the fixed device setting when the number of tasks is less than 10 and the random device setting after the tenth task. The average completion time increases with the number of tasks as both the waiting time at the devices and start time of network flows are increased. The difference in completion time between JPOFH and LE decreases with an increase in the number of tasks from 27.9% at 2 tasks to 16.6% at 20 tasks. This decrease in the gap can be explained as LE does not require to include waiting time at the devices and the start time of network flows which are both considered in JPOFH. However, as we increase the number of tasks to be more than the number of devices, there is some waiting time involved which slightly increases the performance difference between LE and JPOFH. RE shows a similar performance trend as LE, where the difference in the average completion time of JPOFH decreases from 20.7% at 2 tasks to 4% at 20 tasks. RE fully offloads the tasks while considering the waiting time at devices and start time of flows which leads to better performance than LE for a high number of tasks. JPOFH shows a continuous increase in performance difference with SOFS from 2% at 2 tasks to 25.5% at 20 tasks. This is because the cost of waiting time at devices and start time of network flows, which are considered separately in SOFS, is not significant when the number of devices is large compared to the number of tasks. However, as the number of tasks starts to become more than the number of devices, SOFS performs worse than JPOFH. This shows that joint decision making of partial offloading and flow scheduling, as done in JPOFH, leads to better performance in terms of average completion time compared to different benchmark solutions.

The effect of the number of tasks on running time of the algorithm is shown in Fig 3. We have shown the comparison in running time on Log10 scale as there is a huge difference in values of running between LE and LB. The running time of JPOFH is more than LE, RE, and SOFS. The running time of both RE and JPOFH increases with the number of tasks as they are similar in implementation except for the calculation of partial offloading ratio in JPOFH. LE does not show much variation in running time on increasing the number of tasks as there is no significant calculation involved. The increase in running time of SOFS is less compared to JPOFH as SOFS requires to calculate the start time of input data flow only for the selected device and routing path which decreases the running time cost significantly. The running time of LB is significantly higher than other algorithms as it includes not only the time of solving the convex optimization problem but also specifying all the possible constraint equations. Although this time can be decreased by using a different

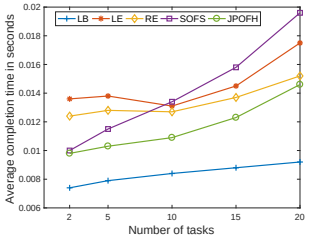


Fig. 2: Effect of number of tasks on completion time

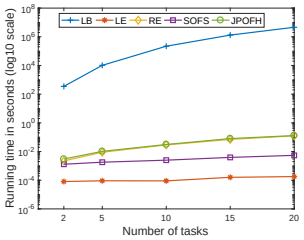


Fig. 3: Effect of number of tasks on running time

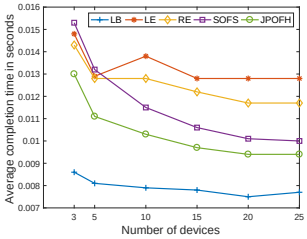


Fig. 4: Effect of number of devices on completion time

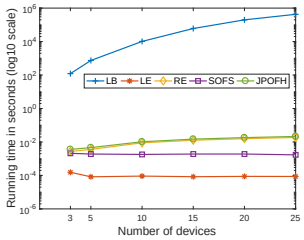


Fig. 5: Effect of number of devices on running time

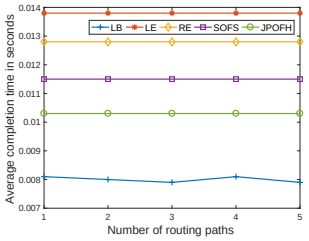


Fig. 6: Effect of number of routing paths on completion time

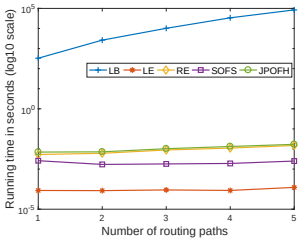


Fig. 7: Effect of number of routing paths on running time

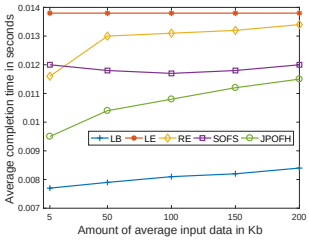


Fig. 8: Effect of amount of input data on completion time

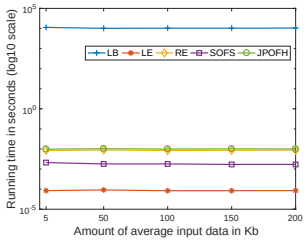


Fig. 9: Effect of number of routing paths on running time

solver or implementation approach, we can still observe the increase in running time of LB with an increase in the number of tasks.

5.3.2 Effect of number of devices

Fig 4 shows the performance comparison, in terms of average completion time, on changing the number of devices from 3 to 25. The comparison covers the full range from when the number of devices is less than the number of tasks to when the number of devices is 5 times the number of tasks. The average completion time of tasks decreases an increase in the number of devices as waiting time at the devices is reduced. The performance difference between JPOFH and LE increases from 12.2% at 3 devices to 26.6% at 25 devices. JPOFH performs even better when the number of devices is more than the number of tasks as there are more options of partially offloading the tasks leading to a decrease in waiting time. In contrast, LE does not leverage the resources available on other devices. JPOFH also shows an increase in performance difference between RE from 9.1% at 3 devices to 19.7% at 25 devices. Although RE can leverage resources on other devices, partially offloading the tasks in JPOFH leads to better performance as the different components of the tasks can be executed simultaneously. There is a decrease in performance difference between JPOFH and SOFS from 15% at 3 devices to 6% at 25 devices. As explained earlier, the cost of waiting time at the devices and start time of network flows is very low when the number of devices is significantly high compared to the number of tasks. However, JPOFH still performs better than SOFS, even for a large number of devices.

The performance comparison, in terms of running time of the algorithm, is shown in Fig 5. The running time of all the algorithms, expect LE, increases with an increase in the number of devices. SOFS also does not show significant variance in running time as it does not have to calculate start time of input data flow for all devices and routing paths which is the most computation-intensive part of JPOFH algorithm.

5.3.3 Effect of number of routing paths

Fig 6 shows the performance comparison, in terms of average completion time, on changing the number of routing paths between any source-destination pair from 1 to 5. We find K shortest paths for all source-destination pairs using Yen's algorithm [35]. JPOFH uses one of the routing paths between source and destination device to send the input data for a task. Although we initially expected to have an improvement in completion time for an increased number of routing paths, however, we find that there is no significant change in the value of average completion time for a different number of routing paths available between any two devices. This is because we use predetermined routing paths with increasing value of communication delay for random network topology. Besides, we measure the performance for a specific case of the number of tasks and number of devices. In order to observe the effect of the number of routing path, we need to see the performance for specific network topology with increased network traffic. However, our aim in this work is to study the performance of the proposed solution for random network topologies and not for a specific network topology usually considered for data centers. The evaluation for specific network topologies in data center networks can be considered in future work.

The performance comparison, in terms of running time of algorithm, is shown in Fig Fig 7. We observe an increase in the running time of all algorithms, except LE for reasons explained previously, on increasing the number of routing paths. SOFS does not show significant variance in running time as it considers a selected device and routing path to calculate the start time of input data flow.

5.3.4 Effect of amount of input data

Fig 8 shows the performance comparison, in terms of average completion time, on changing the mean of normal distribution used to select the amount of input data from 5 Kb to 200 Kb. The change in the amount of input data reflects the change in the ratio of communication to computation cost. The comparison covers the full range from computation-intensive tasks (5 Kb input data) to communication-intensive tasks (200 Kb input data). We observe an increase in average completion time on increasing the amount of input data as communication cost is increased. The performance difference between JPOFH and LE decreases from 31.2% at average input data of 5 Kb to 16.7% at average input data of 200 Kb. This is because the increase in input data leads to an increase in communication cost only for JPOFH as tasks are executed on local devices in LE. JPOFH also shows a decrease in performance difference between RE from 18.1% at average input data of 5 Kb to 14.2% at average input data of 200 Kb. This is because the benefit of executing the partitioned components of tasks simultaneously in JPOFH, compared to RE, is limited by high communication cost in case of high average input data. There is also a decrease in performance difference between JPOFH and SOFS from 20.8% at average input data of 5 Kb to 4.2% at average input data of 200 Kb. Compared to SOFS, JPOFH considers the order of tasks for execution at the devices based on their priority which leads to better performance when average input data is low. However, the effect is ordering the tasks is limited by high communication cost in case of high average input data leading to a decrease in performance difference between JPOFH and SOFS.

The performance comparison, in terms of running time of algorithm, is shown in Fig Fig 9. There is no significant difference in running time of algorithms on changing the amount of input data as computation complexity of both JPOFH and benchmark solutions is independent of the amount of input data.

6 Conclusion

This paper studies the multi-hop multi-task partial computation offloading problem in collaborative edge computing where heterogeneous independent tasks generated at different heterogeneous devices at different release time are partially offloaded to a remote device with the objective of minimizing the average completion time of all tasks. We need to make a decision considering both partial offloading and network flow scheduling as tasks can be offloaded to a device which is multi-hop away. We formulate the problem as an MINLP optimization problem which is proven to be NP-hard. Therefore, we propose a JPOFH algorithm which jointly solves the partial offloading and network flow scheduling problem. The MINLP problem is also relaxed to an LP problem using McCormick envelope and the solution to the relaxed problem acts as lower bound for performance comparison. We have done simulation experiments to evaluate the efficacy of the JPOFH by comparing it against benchmark solutions, including local execution, remote execution, and separate offloading and flow scheduling. We have done a comprehensive performance comparison of JPOFH with benchmark solutions by varying different input parameters, including the number of tasks, number of devices, number of routing paths, and the amount of input data. Performance comparison shows that JPOFH leads to up to 32% improvement in average completion time compared to benchmark solutions.

We have solved the problem for an offline setting and evaluated it using simulation experiments. In the future work, we will implement the solution for an online setting considering network dynamics and wireless interference. Another direction for future work is to consider issues related to SDN controller and implement the scheduling and network monitoring components in the SDN controller. We also plan to develop a real-world prototype to illustrate the efficacy of the proposed solution. Furthermore, the proposed approach assumes a centralized controller with global knowledge which can be addressed in a future work by proposing a distributed scheduling solution.

References

[1] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the internet of things," *IEEE access*, vol. 6, pp. 6900–6919, 2017.

[2] L. U. Khan, I. Yaqoob, N. H. Tran, S. Kazmi, T. N. Dang, and C. S. Hong, "Edge computing enabled smart cities: A comprehensive survey," *arXiv preprint arXiv:1909.08747*, 2019.

[3] A. Saeed, M. Ammar, K. A. Harras, and E. Zegura, "Vision: The case for symbiosis in the internet of things," in *Proceedings of the 6th International Workshop on Mobile Cloud Computing and Services*. ACM, 2015, pp. 23–27.

[4] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative mobile edge computing in 5g networks: New paradigms, scenarios, and challenges," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 54–61, 2017.

[5] K. Wang, H. Yin, W. Quan, and G. Min, "Enabling collaborative edge computing for software defined vehicular networks," *IEEE Network*, vol. 32, no. 5, pp. 112–117, 2018.

[6] H. Zhang, P. Dong, W. Quan, and B. Hu, "Promoting efficient communications for high-speed railway using smart collaborative networking," *IEEE wireless communications*, vol. 22, no. 6, pp. 92–97, 2015.

[7] L. Chen and J. Xu, "Socially trusted collaborative edge computing in ultra dense networks," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 2017, p. 9.

[8] Y. Sahni, J. Cao, S. Zhang, and L. Yang, "Edge mesh: A new paradigm to enable distributed intelligence in internet of things," *IEEE access*, vol. 5, pp. 16 441–16 458, 2017.

[9] H. Al-Shatri, S. Müller, and A. Klein, "Distributed algorithm for energy efficient multi-hop computation offloading," in *2016 IEEE International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–6.

[10] C. F. Funai, C. Tapparello, and W. Heinzelman, "Computational offloading for energy constrained devices in multi-hop cooperative networks," *IEEE Transactions on Mobile Computing*, 2019.

[11] Z. Hong, H. Huang, S. Guo, W. Chen, and Z. Zheng, "Qos-aware cooperative computation offloading for robot swarms in cloud robotics," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 4, pp. 4027–4041, 2019.

[12] Z. Hong, W. Chen, H. Huang, S. Guo, and Z. Zheng, "Multi-hop cooperative computation offloading for industrial iot-edge-cloud computing environments," *IEEE Transactions on Parallel and Distributed Systems*, 2019.

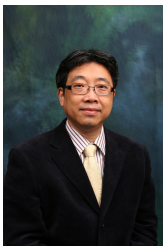
[13] Z. Ning, P. Dong, X. Kong, and F. Xia, "A cooperative partial computation offloading scheme for mobile edge computing enabled internet of things," *IEEE Internet of Things Journal*, 2018.

[14] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Transactions on Communications*, vol. 64, no. 10, pp. 4268–4282, 2016.

[15] B. Li, M. He, W. Wu, A. Sangaiah, and G. Jeon, "Computation offloading algorithm for arbitrarily divisible applications in mobile edge computing environments: An ocr case," *Sustainability*, vol. 10, no. 5, p. 1611, 2018.

[16] Y. Jin, J. Jin, A. Gluhak, K. Moessner, and M. Palaniswami, "An intelligent task allocation scheme for multihop wireless networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 3, pp. 444–451, 2011.
[17] A. Munir, T. He, R. Raghavendra, F. Le, and A. X. Liu, "Network scheduling aware task placement in datacenters," in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*. ACM, 2016, pp. 221–235.
[18] 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.
[19] Y. Sahni, J. Cao, and L. Yang, "Data-aware task allocation for achieving low latency in collaborative edge computing," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3512–3524, 2018.
[20] X. Tao, K. Ota, M. Dong, H. Qi, and K. Li, "Performance guaranteed computation offloading for mobile-edge cloud computing," *IEEE Wireless Communications Letters*, vol. 6, no. 6, pp. 774–777, 2017.
[21] Y. Tian and E. Ekici, "Cross-layer collaborative in-network processing in multihop wireless sensor networks," *IEEE transactions on mobile computing*, vol. 6, no. 3, pp. 297–310, 2007.
[22] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1619–1632, 2018.
[23] Y. Zhao, C. Tian, J. Fan, T. Guan, and C. Qiao, "Rpc: Joint online reducer placement and coflow bandwidth scheduling for clusters," in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. IEEE, 2018, pp. 187–197.
[24] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low latency geo-distributed data analytics," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 421–434, 2015.
[25] L. Rupprecht, W. Culhane, and P. Pietzuch, "Squirreljoin: network-aware distributed join processing with lazy partitioning," *Proceedings of the VLDB Endowment*, vol. 10, no. 11, pp. 1250–1261, 2017.
[26] C.-F. Liu, S. Samarakoon, M. Bennis, and H. V. Poor, "Fronthaul-aware software-defined wireless networks: Resource allocation and user scheduling," *IEEE Transactions on Wireless Communications*, vol. 17, no. 1, pp. 533–547, 2017.
[27] T. De Schepper, S. Latré, and J. Famaey, "A transparent load balancing algorithm for heterogeneous local area networks," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2017, pp. 160–168.
[28] J. Lee, M. Uddin, J. Tourrilhes, S. Sen, S. Banerjee, M. Arndt, K.-H. Kim, and T. Nadeem, "mesdn: Mobile extension of sdn," in *Proceedings of the fifth international workshop on Mobile cloud computing & services*, 2014, pp. 7–14.
[29] T. De Schepper, P. Bosch, E. Zeljkovic, K. De Schepper, C. Hawinkel, S. Latré, and J. Famaey, "Sdn-based transparent flow scheduling for heterogeneous wireless lans," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2017, pp. 901–902.
[30] Y. Liu, S. Wang, Q. Zhao, S. Du, A. Zhou, X. Ma, and F. Yang, "Dependency-aware task scheduling in vehicular edge computing," *IEEE Internet of Things Journal*, 2020.
[31] D. Nowak, T. Mahn, H. Al-Shatri, A. Schwartz, and A. Klein, "A generalized nash game for mobile edge computation offloading," in *2018 6th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*. IEEE, 2018, pp. 95–102.
[32] M. Grant, S. Boyd, and Y. Ye, "Cvx: Matlab software for disciplined convex programming," 2008.
[33] J. Yang, H. Zhang, Y. Ling, C. Pan, and W. Sun, "Task allocation for wireless sensor network using modified binary particle swarm optimization," *IEEE Sensors Journal*, vol. 14, no. 3, pp. 882–892, 2013.
[34] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing flows quickly with preemptive scheduling," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 127–138, 2012.
[35] J. Y. Yen, "Finding the k shortest loopless paths in a network," *management Science*, vol. 17, no. 11, pp. 712–716, 1971.

Yuvraj Sahni received B.E. (Hons) degree in Electrical and Electronics Engineering from Birla Institute of Technology and Science, Pilani, India in 2015. He is currently working towards the Ph.D. degree at Department of Computing, The Hong Kong Polytechnic University. Hong Kong. His research interests include wireless sensor networks, edge computing, and Internet of Things.



Jiannong Cao received the B.Sc. degree in computer science from Nanjing University, China, in 1982, and the M.Sc. and Ph.D. degrees in computer science from Washington State University, USA, in 1986 and 1990 respectively. He is currently the Otto Poon Charitable Foundation Professor in Data Science and the Chair Professor of Distributed and Mobile Computing in the Department of Computing at The Hong Kong Polytechnic University, Hong Kong. He is also the director of the Internet and Mobile Computing Lab in the department and the associate director of University Research Facility in Big Data Analytics. His research interests include parallel and distributed computing, wireless networks and mobile computing, big data and cloud computing, pervasive computing, and fault tolerant computing. He has co-authored 5 books in Mobile Computing and Wireless Sensor Networks, co-edited 9 books, and published over 600 papers in major international journals and conference proceedings. He is a fellow of IEEE, a distinguished member of ACM, a senior member of China Computer Federation (CCF).

Lei Yang received the BSc degree from Wuhan University, in 2007, the MSc degree from the Institute of Computing Technology, Chinese Academy of Sciences, in 2010, and the PhD degree from the Department of Computing, The Hong Kong Polytechnic University, in 2014. He is currently an associate professor at the School of Software Engineering, South China University of Technology, China. His research interest includes mobile cloud computing, Internet of things, and big data analytic. He has published more than 30 papers in conferences and journals. He is a program committee member for many international conferences. He is a member of the IEEE.

Yusheng Ji received the B.E., M.E., and D.E. degrees in electrical engineering from The University of Tokyo. She joined the National Center for Science Information Systems, Japan, in 1990. She is currently a Professor with the National Institute of Informatics and with The Graduate University for Advanced Studies. Her research interests include network architecture, resource management, and quality of service provisioning in wired and wireless communication networks. She is/has been an Editor of the IEEE Transactions on Vehicular Technology, a Symposium Co-Chair of the IEEE GLOBECOM 2012 and the IEEE GLOBECOM 2014, and a Track Co-Chair of the IEEE VTC 2016-Fall and IEEE VTC 2017-Fall.