A Bifactor Approximation Algorithm for Cloudlet Placement in Edge Computing

Dixit Bhatta, *Student Member, IEEE* and Lena Mashayekhy, *Senior Member, IEEE*

Abstract—Emerging applications with low-latency requirements such as real-time analytics, immersive media applications, and intelligent virtual assistants have rendered Edge Computing as a critical computing infrastructure. Existing studies have explored the cloudlet placement problem in a homogeneous scenario with different goals such as latency minimization, load balancing, energy efficiency, and placement cost minimization. However, placing cloudlets in a highly heterogeneous deployment scenario considering the next-generation 5G networks and IoT applications is still an open challenge. The novel requirements of these applications indicate that there is still a gap in ensuring low-latency service guarantees when deploying cloudlets. Furthermore, deploying cloudlets in a cost-effective manner and ensuring full coverage for all users in edge computing are other critical conflicting issues. In this article, we address these issues by designing a bifactor approximation algorithm to solve the heterogeneous cloudlet placement problem to guarantee a bounded latency and placement cost, while fully mapping user applications to appropriate cloudlets. We first formulate the problem as a multi-objective integer programming model and show that it is a computationally NP-hard problem. We then propose a bifactor approximation algorithm, ACP, to tackle its intractability. We investigate the effectiveness of ACP by performing extensive theoretical analysis and experiments on multiple deployment scenarios based on New York City OpenData. We prove that ACP provides a (2,4)-approximation ratio for the latency and the placement cost. The experimental results show that ACP obtains near-optimal results in a polynomial running time making it suitable for both short-term and long-term cloudlet placement in heterogeneous deployment scenarios.

Index Terms—Edge computing, cloudlets, placement cost, latency, full coverage, approximation algorithm

1 Introduction

The advances in wireless network technologies and computational capabilities of smart connected devices have now made it possible to use many novel and sophisticated applications not feasible before. Domains such as healthcare, connected vehicles, and smart cities have generally been beneficiaries of this innovation and so have contemporary applications like live streaming on social media apps and games on virtual reality headsets [1]. Despite massive improvements in mobile hardware capabilities over past few years, it is still a challenge to run computation and data-intensive applications on mobile devices since they are restricted by weight, size, battery life, and heat dissipation [2]. These restrictions impose limitations on processing power, memory, and storage capacities of these devices.

Edge computing is a relatively new computing paradigm that provides a distributed computing solution at the edge of the network. As such, mobile users are able to consume the computing resources in their vicinity. These resource-rich components placed closer to the users are called "Cloudlets" or "micro data centers" [3], [4]. The mobile devices can offload their resource-intensive applications to cloudlets to augment their resources while experiencing significantly reduced latency compared to the conventional cloud [5], [6]. Since cloudlets are geo-distributed, a challenge lies in strategically placing them in an area to reduce the placement cost while providing low-latency edge services.

Despite existing research literature in cloudlet placement in edge computing, very few view the problem from the heterogeneous perspective and evolving needs of the next-generation 5G networks and IoT applications. As we move into 5G networks, presence of micro network operators (MNOs), smaller cell towers, and short-range millimeter wave communication lead to a highly localized and heterogeneous deployment environment [7]. Cloudlet placement is at the core of providing ultra-low latency services by the likes of Verizon Inc. with their 5G Edge service [8]. Ensuring consistent low-latency across the region and providing full coverage to all users is computationally and economically expensive in such scenarios. Therefore, many factors such as cost, latency, capacity, and coverage must be considered for heterogeneous cloudlet placement. One-track approach to optimize only one of these objectives, persistently presented by multiple studies, is very limited in heterogeneous real world scenarios. As a result, they fall short of addressing all the practical aspects of the cloudlet placement problem.

In this paper, we address the placement challenges comprehensively by designing a bicriteria approximation algorithm. Our goal is to deploy cloudlets with heterogeneous capacities and coverage radius in a region to provide edge services to heterogeneous devices in order to guarantee bounded service latency, placement cost, and full coverage. We formulate a thorough representation of the optimal

• *The authors are with the Department of Computer and Information Sciences, University of Delaware, Newark, DE 19716 USA. E-mail: {dixit, mlena}@udel.edu.*

Manuscript received 5 Jan. 2021; revised 11 June 2021; accepted 5 July 2021. Date of publication 9 Nov. 2021; date of current version 9 Dec. 2021. This work was supported in part by NSF under Grant CNS-1755913. (Corresponding author: Lena Mashayekhy.) Recommended for acceptance by R. Tolosana. Digital Object Identifier no. 10.1109/TPDS.2021.3126256

heterogeneous cloudlet placement problem as an Integer Program (IP).

The cloudlet placement problem is NP-hard as we prove that the decision version of the problem is NP-complete (see Theorem 1). For any NP-hard problem, it is not possible to design algorithms that can simultaneously achieve optimal results in polynomial time and for all instances of the problem [9]. When designing a solution for such problems, we must choose any two out of these three properties. Designing either a meta-heuristic approach or an approximation algorithm is an effective way to solve such hard problems. While a fast converging meta-heuristic approach is useful, one of the major limitations of such approaches is the apparent lack of guarantees on its solutions.

An approximation algorithm is a stronger notion where we can formally claim that our proposed approximate cloudlet placement (ACP) always results in near-optimal solution, with simultaneously guaranteed theoretical upper-bounds on both latency and cost, and ensures full device coverage and consistent performance over all instances. We use the linear programming (LP) relaxation of our IP formulation to design ACP as a "bifactor" approximation algorithm, i.e., it provides separate worst-case bounds for the two objectives. Having a bifactor approximation allows us to isolate and independently analyze the worst-case scenario for each objective instead of the overall approximation of their linear combination.

We perform extensive theoretical analysis and several experiments to show the effectiveness of ACP in finding approximate solutions in polynomial time for different deployment scenarios. We prove that ACP provides a (2,4)-approximation ratio for the latency and the placement cost, while providing full coverage. Our experiments are designed based on real data containing the latest WiFi hotspot locations and usage statistics obtained from NYC OpenData [10], provided by NSF COSMOS [11]. The results show that ACP efficiently finds near-optimal solutions in these real world-based scenarios.

1.1 Related Work

We have divided our related work section into two categories: classical placement problem approaches and edge computing placement approaches.

1.1.1 Classical Placement Problem Approaches

The classical approaches to tackle the placement problems include approaches for facility location problems, clustering problems, and assignment problems. The most studied among them is the metric uncapacitated facility location problem. Multiple studies [12], [13], [14], [15], [16] have proposed approximation algorithms for this problem. Byrka and Aardal [15] provided a bifactor approximation algorithm for the facility cost and the connection cost, which are analogous to our placement costs and latency. However, there are no capacity constraints. Mahdian *et al.* [12] additionally provided a 2-approximation algorithm for the soft-capacitated version of the problem. However, capacity is a hard constraint in our problem. Thus, these approaches are not directly applicable to our problem.

For capacitated k-facility location problem, Aardal *et al.* [17] provided a $(7+\epsilon)$ approximation algorithm for uniform opening costs and non-uniform capacities by placing at most $2k$ facilities. However, as full coverage service is significantly important in edge computing, it is not known that k cloudlets will suffice in any scenario. Also, the costs depend on multiple parameters such as the capacity of a cloudlet and coverage radius. General clustering approaches such as k-means and k-medoids do not apply well to our problem for the same reasons. Raghavan *et al.* [18] did the first study to formulate integer programming models for capacitated mobile facility location problem. They provided algorithms to find fractional LP solutions and an LP rounding heuristic that works well with homogeneous facilities. However, the heuristic is not guaranteed to terminate with a feasible integer solution.

Lin and Vitter [19] is a useful study since they provided important assignment properties for an LP-based approximation algorithm for finding geometric means. Efficient approximation algorithms are available for Generalized Assignment Problems as presented in studies such as [20], [21] which have hard capacity constraints. However, they are more suitable for task assignment with profit maximization.

1.1.2 Edge Computing Placement Approaches

Multiple clustering-based approaches have been proposed to place cloudlets. Kang *et al.* [22] used a greedy geographic clustering approach to balance workload and collaborative scheduling to reduce access delay. Jia *et al.* [23] proposed density-based clustering and k-means clustering of users to minimize response time. Zhang *et al.* [24] focused on reducing latency by greedily selecting cloudlets with minimum distance to the center of the user cluster. However, none of them consider placement costs and heterogeneity of cloudlets in their approach. Furthermore, a cloudlet is most often placed at the centroid of each cluster of users. This is unlike real world where the connectivity requirements of the users must be met individually.

Further greedy approaches include Zeng *et al.* [25], who proposed a greedy-based algorithm that minimizes the number of cloudlets to be placed considering latency requirements. A greedy heuristic approach was proposed in [26] to reduce access delay for users served from nearby access points. Li *et al.* [27] proposed energy-aware placement of cloudlets using swarm optimization, while assuming users are mapped to cloudlets through base stations. Yao *et al.* [28] presented a greedy approach with heterogeneous cloudlets and placement costs. However, they considered user coverage in terms of probability of user contact with access points. Only the last study directly considers the placement cost and none of these studies consider full coverage of individual users.

Other approaches include Wang *et al.* [29] proposing a binary-based differential evolution cuckoo search (BDECS) algorithm. They emphasized on deployment of cloudlets based on cost and latency for Internet of Things (IoT) scenario. However, they considered fixed placement cost ignoring heterogeneity. Fan and Ansari [30] proposed a Lagrangian heuristic algorithm for cost aware placement to obtain sub-optimal solutions. Their approach only indirectly addresses

TABLE 1
Comparison With Existing Research

Study	Cost Obj.	Latency Obj.	Het. Cloudlets	Het. Devices	Capacity Const.	Real Dataset	Full Coverage	Approximation
Kang *et al.* [22]		✓			✓			
Zhang *et al.* [24]		✓						
Zeng *et al.* [25]	✓				✓			
Xu *et al.* [26]		✓		✓	✓			✓
Li *et al.* [27]						✓		
Yao *et al.* [28]	✓		✓		✓			
Wang *et al.* [29]	✓	✓						
Fan and Ansari [30]	✓	✓			✓			
Mondal *et al.* [31]	✓		✓		✓			
Wang *et al.* [32]	✓							
Wang *et al.* [33]		✓			✓	✓		
Guo *et al.* [34]		✓			✓	✓		
Ma *et al.* [35]		✓						
Meng *et al.* [36]	✓	✓		✓	✓			✓
Our Study	✓	✓	✓	✓	✓	✓	✓	✓

cloudlet heterogeneity and lacks device heterogeneity entirely. Mondal *et al.* [31] proposed a hybrid cost-optimization framework for optimal cloudlet placement specifically for three-tier fiber wireless network topology. However, they only proposed a mixed integer non-linear programming model as their solution. Another study [32] investigates deployment of edge servers cost-effectively so that the collective area of edge servers is maximized. Their focus is primarily on maximizing coverage at a reduced placement cost.

Wang *et al.* [33] used Mixed Integer Programming (MIP) to find K locations to place edge servers. Their approach uses CPLEX, an optimization solver, to get results. The work has been extended in [34] by combining K-means clustering and MIP. The placed servers are still homogeneous. Ma *et al.* [35] used Particle Swarm Optimization (PSO) to place K cloudlets at K access points (APs) to reduce access delay. Like most other studies, they lack capacity considerations, placement costs, and heterogeneity.

There have been a few studies that provide approximation algorithms for the cloudlet placement problem. Meng *et al.* [36] use direct reduction of capacitated k-facility placement problem [17] to their problem formulation to provide a $(7+\epsilon)$ approximation bound. This is the best bound provided by them which assumes placement of up to $2k$ cloudlets with identical capacities. Xu *et al.* [26], which we already discussed under greedy approaches, have also provided an approximation algorithm with a bound of $16(1+\epsilon)$ on average access delay.

As summarized in Table 1, none of these studies provide approximate solutions with worst-case performance guarantees on both the placement cost and the latency. Moreover, they do not investigate heterogeneous deployment. Another point to consider is that the experiments performed in these studies are based on either synthetic or randomly generated networks. Only a few studies have used scenarios based on real datasets for their experiments (cellular base stations data: [27], [33], [34], transportation network data: [23]). Finally, none of these studies consider full mapping of individual users or devices to cloudlets.

1.2 Organization

The rest of the paper is organized as follows. In Section 2, we introduce the cloudlet placement problem and provide a mathematical optimization model. In Section 3, we present our proposed LP-based approximation approach, ACP, in detail. In Section 4, we evaluate the performance of ACP by extensive experiments. In Section 5, we summarize our results and present possible directions for future research.

2 Cloudlet Placement Problem

We aim to efficiently place cloudlets to specific locations in a region to serve the demands of all the end devices (IoT) that require edge services. We model the region as a two-dimensional space (grid), where cloudlets and devices can exist. The devices could be at any point in the space. On the other hand, we assume only a set of candidate points within the grid are available where the cloudlets can be placed and the devices can be best served from. The candidate points are selected based on the load of user requests and the location of user demands over a long period. The determination of candidate points are also constrained by technological and economic aspects such as available locations in the existing infrastructure, network bottlenecks, lack of sufficient space, unavailable private property, and high placement and operating expenses. The set of candidate points is hence defined as $\mathcal{P} = \{\rho_1, \rho_2, \ldots, \rho_n\}$, where each refers to a preselected, feasible placement location in the grid (in coordinate axes).

A set of cloudlets is denoted by $\mathcal{C} = (c_1, c_2, \ldots, c_w)$. The cloudlets are heterogeneous, and each $c_j \in \mathcal{C}$ is represented by a 4-tuple $c_j = \{\Pi_j, M_j, S_j, r_j\}$ denoting its attributes: processor capacity Π_j (in GHz), RAM M_j (in GB), storage capacity S_j (in GB), and coverage radius r_j (euclidean distance units). We define a distance function $d(a, b)$ for calculating the euclidean distance between points a and b. A set of heterogeneous devices requiring edge computing services is denoted by $\mathcal{E} = (e_1, e_2, \ldots, e_v)$. Each $e_i \in \mathcal{E}$ is represented by a 4-tuple $e_i = \{\pi_i, m_i, s_i, \lambda_i\}$ denoting its attributes, where π_i is the processing demand (in GHz), m_i is its memory demand (in GB), s_i is its storage demand (in GB), and λ_i represents the location of the device (in coordinate axes).

The incurred cost of placing cloudlet $c_j \in \mathcal{C}$ at a candidate point $\rho_k \in \mathcal{P}$ on the grid is defined by a cost function $\Phi(c_j, \rho_k)$ (simply, ϕ_{jk}). The cost may include procurement cost, space (rented, public) cost, and maintenance costs as needed. We consider that the cost function is linearly correlated with cloudlet capacities and coverage radius. While our approach is applicable for a non-linear cost function, we cannot theoretically bound the obtained cost in that case (see Theorem 4).

Devices may experience delays in receiving their computing services based on the availability of bandwidth and distance metrics [37]. To capture that, we define a latency function $L(e_i, \rho_k)$ (simply, l_{ik}) that represents the latency when device $e_i \in \mathcal{E}$ is served by a cloudlet placed at candidate point $\rho_k \in \mathcal{P}$ of the grid. We assume a homogeneous bandwidth across the grid in our model. As a result, latency is predominantly affected by distance. Well-known models have

TABLE 2
Notations

Symbol	Description
$\mathcal{P}$	set of candidate points
$\mathcal{C}$	set of cloudlets
$\mathcal{E}$	set of devices
ρ_k	a candidate point in $\mathcal{P}$
c_j	a cloudlet in $\mathcal{C}$
e_i	a device in $\mathcal{E}$
Π_j, M_j, S_j	capacity parameters of c_j
r_j	coverage radius of c_j
π_i, m_i, s_i	demand parameters of e_i
λ_i	coordinate location of e_i
$\Phi(c_j, \rho_k)$ or ϕ_{jk}	cost of placing c_j at ρ_k
$L(e_i, \rho_k)$ or l_{ik}	latency between e_i and ρ_k

shown that latency is logarithmically related with distance in wireless networks, including 5G. Our model supports any latency function increasing on distance. The notations used in our system model are summarized in Table 2.

Our goal is to simultaneously minimize the cost of deploying heterogeneous cloudlets in the region and minimize the latency in accessing edge services, while providing the services to all devices (full coverage). This is a bicriteria optimization problem and computationally NP-hard.

2.1 Optimal Cloudlet Placement

We now formulate the heterogeneous cloudlet placement problem as a bicriteria optimization model. We define the following decision variables:

$$y_{jk} = \begin{cases} 1 & \text{if cloudlet } c_j \text{ is placed at candidate point } \rho_k, \\ 0 & \text{otherwise,} \end{cases}$$

and

$$a_{ik} = \begin{cases} 1 & \text{if device } e_i \text{ is served from candidate point } \rho_k, \\ 0 & \text{otherwise.} \end{cases}$$

We mathematically formulate the optimal cloudlet placement (OCP) as an Integer Program (IP) as follows:

$$\begin{cases} \boldsymbol{\Phi} = \min \sum_{j=1}^{w} \sum_{k=1}^{n} \phi_{jk}\, y_{jk} \\ \boldsymbol{L} = \min \sum_{i=1}^{v} \sum_{k=1}^{n} l_{ik}\, a_{ik} \end{cases}. \tag{1}$$

Subject to

$$\sum_{j=1}^{w} \sum_{k=1}^{n} y_{jk} \leq |\mathcal{C}| \tag{2}$$

$$d(\lambda_i, \rho_k) a_{ik} \leq \sum_{j=1}^{w} r_j y_{jk} \quad \forall e_i \in \mathcal{E}, \rho_k \in \mathcal{P} \tag{3}$$

$$\sum_{i=1}^{v} m_i a_{ik} \leq \sum_{j=1}^{w} M_j y_{jk} \quad \forall \rho_k \in \mathcal{P} \tag{4}$$

$$\sum_{i=1}^{v} s_i a_{ik} \leq \sum_{j=1}^{w} S_j y_{jk} \quad \forall \rho_k \in \mathcal{P} \tag{5}$$

$$\sum_{i=1}^{v} \pi_i a_{ik} \leq \sum_{j=1}^{w} \Pi_j y_{jk} \quad \forall \rho_k \in \mathcal{P} \tag{6}$$

$$a_{ik} \leq \sum_{j=1}^{w} y_{jk} \quad \forall e_i \in \mathcal{E}, \rho_k \in \mathcal{P} \tag{7}$$

$$\sum_{j=1}^{w} y_{jk} \leq 1 \quad \forall \rho_k \in \mathcal{P} \tag{8}$$

$$\sum_{k=1}^{n} y_{jk} \leq 1 \quad \forall c_j \in \mathcal{C} \tag{9}$$

$$\sum_{k=1}^{n} a_{ik} = 1 \quad \forall e_i \in \mathcal{E} \tag{10}$$

$$y_{jk} \in \{0, 1\} \quad \forall c_j \in \mathcal{C}, \rho_k \in \mathcal{P} \tag{11}$$

$$a_{ik} \in \{0, 1\} \quad \forall e_i \in \mathcal{E}, \rho_k \in \mathcal{P}. \tag{12}$$

The objective functions shown in Eq. (1) minimize the total cost of placing the cloudlets and the total latency suffered by the devices. Constraint (2) ensures that the total number of cloudlets placed in the grid does not exceed the number of available cloudlets. Constraints (3) guarantee that each device must be within the coverage range of some cloudlet. Constraints (4), (5), and (6) satisfy supply and demand in terms of memory, storage, and processing requirements. Constraints (7) guarantee that a device can be served from a candidate point only if at least one cloudlet is placed there. Constraints (8) ensure that at most one cloudlet is placed at any candidate point. Constraints (9) ensure that a cloudlet can only be placed at a single candidate point. Constraints (10) guarantee that all devices must be served, and each is served from exactly one candidate point. Finally, constraints (11) and (12) ensure the integrality requirements of the decision variables.

OCP finds the optimal placement of the cloudlets minimizing both the placement cost and the service latency, while guaranteeing full coverage. Our goal is to solve OCP in the presence of trade-offs between these two conflicting objectives. OCP can run equally well or better without the full coverage constraint or a lower coverage (e.g., 90%) threshold. However, our goal is to look into low-latency edge services for all devices, where latency of connection to the cloud is unbearable. The cost obtained is an appropriate representation of the worst cost at the time of planning. Next, using an example we show the ϵ-constraint method and its limitations in solving our multi-objective problem.

2.2 Optimization Example

Fig. 1a represents a scenario with 20 grid points, 7 candidate points, 25 devices, and 5 cloudlets to be placed such that the optimization criteria of OCP are met. In this figure, the candidate points are denoted by numbered circles with their respective index. Likewise, low-demand devices are represented by *smartwatch* icons, and high-demand devices are shown as *cellphone* icons. For this example, the specifications of cloudlets and devices are provided in Tables 3 and 4, respectively.

In this example, among the 5 available cloudlets, 1 is large, 3 are medium, and 1 is of small size, according to their capacity and coverage radius factor.

We now use the ϵ-constraint method for solving our multi-objective problem. In the ϵ-constraint method, we optimize one of the objective functions and use the other objective functions as constraints [38]. We use this approach to find a set of Pareto-optimal solutions, where none of the

(a) A Cloudlet Placement Scenario (b) Optimal Cost Placement (c) Optimal Latency Placement

Fig. 1. A cloudlet placement scenario along with single-objective optimal cloudlet placement solutions.

objective functions can be improved in value without degrading the other objective value.

Minimizing Cost (OCP-Cost). Fig. 1b shows the perspective of minimizing only the cost (i.e., latency objective is ignored), while providing full coverage. In the figure, the sizes of cloudlets are depicted by the radius around the candidate point. The larger the radius, the larger is the cloudlet placed at the candidate point. The devices can be observed to be within the coverage of multiple cloudlets but they are assigned optimally. The figure shows that only three (2 medium, 1 large) cloudlets are placed to serve all the devices, and the minimum cost is 127. However, this solution does not lead to the minimum latency for the users (the obtained latency is 48).

Minimizing Latency (OCP-Latency). Fig. 1c depicts the perspective of only minimizing the latency (i.e., cost is relaxed), while providing full coverage. Unlike the solution of OCP-Cost, the optimal solution of OCP-Latency consists of all available cloudlets. The minimum latency is 22.

Cost-Latency Tradeoff. We now investigate how the Pareto-optimal solutions behave when using the ϵ-constraint method. Note that the other objective becomes an additional constraint with different threshold values in OCP.

For our analysis, we use the latency value of OCP-Cost solution as the initial threshold, which is an upper bound on latency in the added constraint. We then gradually decrease the threshold value until there is no solution to OCP-Cost, meaning that we reach the optimal solution of the latency objective. This is shown in Fig. 2. When the minimum cost obtained by OCP-Cost is 127, the value of latency is 48, but the best value of latency at this cost is 37 (see green "x"). By adding a new hard constraint on latency (e.g., 25), the best achievable cost increases to 145. If we further decrease the latency constraint to 22 (optimal latency), the best cost becomes 175 (see red "x").

All of these suggest a clear trade-off between cost and latency, and optimizing one does not give the best value for the other. It is even more challenging when the problem size is large since the trade-off range becomes even larger. Therefore, approximation of any one objective is insufficient, which is why we need to design a bifactor approximation algorithm. We discuss our proposed approach to address this challenge in the upcoming sections.

TABLE 3
Specifications of Cloudlets

Type	Π_j	M_j	S_j	r_j	ϕ_{jk}
Large	20	20	20	3	63
Medium	10	10	10	2	32
Small	5	5	5	1	16

TABLE 4
Specifications of Devices

Type	π_i	m_i	s_i
High-Demand (Cellphone)	2	2	2
Low-Demand (Smartwatch)	1	1	1

3 Bifactor Approximation of Cloudlet Placement

We design a bifactor approximation algorithm for the cloudlet placement problem as shown in Algorithm 1. The algorithm, called ACP, uses linear programming (LP) relaxation of OCP as a guide to obtain a feasible solution and to provide separate bounds on the total placement cost and the total latency. To achieve this, ACP runs in three phases (two parts): 1) filtering, 2) rounding (the first part), and 3) supplementing (the second part).

ACP receives the set of candidate points, cloudlets and users, and the cost and latency matrices as inputs. It then solves $\texttt{LP}-\texttt{OCP}-\texttt{Cost}()$ to obtain fractional solutions $y^*_{jk}, a^*_{ik} \in \mathbb{R}_{\geq 0}$ (line 2). Two sets $A_{\mathcal{C}}$ and $A_{\mathcal{E}}$ are defined to store the mappings of cloudlets and devices to the candidate points, respectively (lines 3-4). Each set holds a pair that shows such a mapping. Next, ACP calculates the total fractional latency D_i for each device $e_i \in \mathcal{E}$ to all candidate points and creates a set of nearby candidates N_i such that N_i is the set of fractionally assigned (non-zero) candidate points within $2D_i$ of the device

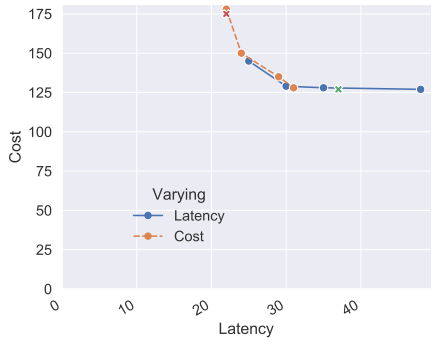


Fig. 2. Cost versus latency trade-off in OCP.

(line 5-7). This is the "filtering" phase of the algorithm and has a property that at least half of every device is assigned to candidates in N_i of the device (see Lemma 2). ACP then creates a skipped set S and a temporary unassigned devices set T. ACP uses the temporary devices set to iteratively assign devices to candidate points or skip them (lines 10-28).

Algorithm 1. ACP: Approximate Cloudlet Placement

1: **Input:** $\mathcal{C}$, $\mathcal{E}$, $\mathcal{P}$, Φ, L
2: $(y^*_{jk}, a^*_{ik}) \leftarrow$ `LP-OCP-Cost()`
3: $A_\mathcal{C} = \emptyset$ ▷ cloudlet to candidate mapping
4: $A_\mathcal{E} = \emptyset$ ▷ device to candidate mapping
/* *Filtering* */
5: **for** $e_i \in \mathcal{E}$ **do**
6: $D_i = \sum_{\forall \rho_k \in \mathcal{P}} l_{ik}\, a^*_{ik}$ ▷ Fractional total latency
7: $N_i \leftarrow$ All $\rho_k \in \mathcal{P}$ with $a^*_{ik} > 0$ within $2D_i$ of e_i
8: $S = \emptyset$ ▷ Set of skipped devices
9: $T = \mathcal{E}$ ▷ Temporary set of devices
/* *Rouding* */
10: **while** $|T| > 0$ **do**
11: $e_i \leftarrow$ `MaxD(`T`)` ▷ Device with the largest D_i
12: $\rho_i \leftarrow$ `BestCandidate(`e_i`)` ▷ Closest $\rho_k \in N_i$ to e_i
13: $c_i \leftarrow$ `BestCloudlet(`e_i, ρ_i`)` ▷ Selected cloudlet
14: **if** $c_i = \emptyset$ **then**
15: $S = S \cup \{e_i\}$
16: **else**
17: $A_\mathcal{E} = A_\mathcal{E} \cup \{(e_i, \rho_i)\}$
18: $A_\mathcal{C} = A_\mathcal{C} \cup \{(c_i, \rho_i)\}$
19: Adjust remaining capacities of c_i
20: $E_i \leftarrow$ All $e_v \in \mathcal{E}$ with $\rho_i \in N_v$ ▷ Extended set
21: **while** $|E_i| > 0$ **do**
22: $e_v \leftarrow$ `MaxD(`E_i`)` ▷Device with the largest D_v
23: **if** `RangeCap(`e_v, c_i`)` **then**
24: $A_\mathcal{E} = A_\mathcal{E} \cup \{(e_v, \rho_i)\}$
25: Adjust remaining capacities of c_i
26: $T = T \setminus \{e_v\}$
27: $E_i = E_i \setminus \{e_v\}$
28: $T = T \setminus \{e_i\}$
/* *Supplementing* */
29: **for** $e_i \in S$ **do**
30: $(c^*_i, \rho^*_i) \leftarrow$ `BestFeasiblePair(`$e_i, A_\mathcal{C}$`)`
31: **if** $(c^*_i, \rho^*_i) = \emptyset$ **then**
32: $(c^*_i, \rho^*_i) \leftarrow$ `UpgradeCloudlet(`ρ^*_i`)`
33: $A_\mathcal{C} = A_\mathcal{C} \cup \{(c^*_i, \rho^*_i)\}$
34: $A_\mathcal{E} = A_\mathcal{E} \cup \{(e_i, \rho^*_i)\}$
35: Adjust remaining capacities of c^*_i
36: Calculate $\widehat{\mathbf{\Phi}}, \widehat{L}$
37: **Output:** $A_\mathcal{E}$, $A_\mathcal{C}$, $\widehat{\mathbf{\Phi}}, \widehat{L}$

Algorithm 2. BestCloudlet(e_i, ρ_i)

1: $c_i \leftarrow \emptyset$
2: **if** $\{(c^*_i, \rho_i)\} \in A_\mathcal{C}$ **then** ▷ Cloudlet already exists
3: **if** `RangeCap(`e_i, c^*_i`)` **then**
4: $c_i \leftarrow c^*_i$
5: **else**
6: $c_i \leftarrow$ `SmallestFeasibleCloudlet()`
7: **return** c_i

In each iteration, `MaxD()` function takes the temporary devices set T as an input and returns a device, e_i, that has the largest value of total fractional latency, D_i (line 11). This is to ensure the worst possible cases are handled first, which simplifies the assignment problem as the algorithm progresses. Next, ACP selects candidate point ρ_i from set N_i that leads to the lowest latency for e_i by calling the `BestCandidate(`e_i`)` function. Then, the best cloudlet is selected using the `BestCloudlet(`e_i, ρ_i`)` function, given in Algorithm 2. If a cloudlet c^*_i with sufficient capacity and range is already placed at candidate point ρ_i (Algorithm 2, lines 3-4), c^*_i is selected and returned as the best cloudlet c_i. Otherwise, the smallest possible cloudlet that can cover the device is selected to be placed at ρ_i (Algorithm 2, lines 7-8). If no such a cloudlet is found, i.e., `BestCloudlet(`e_i, ρ_i`)` returns $null$ ($\emptyset$), ACP simply skips the device by adding it to S (lines 14-15). The skipped devices are handled in the second part of the algorithm. The successful mappings of a cloudlet to a candidate point and a device to a candidate point are added to the respective solution sets $A_\mathcal{C}$ and $A_\mathcal{E}$ (lines 17-18). The demands (processing, memory, and storage) of the assigned device e_i are then subtracted from the the capacity of the selected cloudlet c_i.

For every device with a successfully selected candidate point and feasible cloudlet, ACP creates an extended neighborhood set E_i containing all unassigned devices, $e_v \in T$, that have candidate point ρ_i in their neighborhood set N_v (line 20). ACP then assigns all devices in E_i that are within the radius and capacity of cloudlet c_i and removes them from T (line 22-26). These devices are prioritized based on the value of their total fractional latencies. Device e_i is finally removed from the temporary set (line 28). This concludes the "rounding" phase, where the assignments are finalized for all devices except the skipped devices kept in S.

The second part of the algorithm (lines 29-35) is for all skipped devices added to S. Here, ACP assigns each individual device e_i to the best existing candidate-cloudlet pair $\{c^*_i, \rho^*_i\}$. That is, the candidate point with the least latency among the ones having feasible cloudlets for device e_i is chosen. If no feasible candidate-cloudlet pair is found for the device, ACP then upgrades the cloudlet at the least-latency candidate point ρ^*_i so that it can meet the device's demand and cover it sufficiently (line 32). The upgrade here is the addition of the capacity and radius equivalent to the smallest cloudlet to the existing cloudlet. This is the "supplementing" phase of ACP.

Once the mappings are completed for all devices, ACP calculates the approximate cost ($\widehat{\mathbf{\Phi}}$) and the approximate latency ($\widehat{L}$) based on the final mappings in $A_\mathcal{C}$ and $A_\mathcal{E}$, respectively (line 36). The algorithm terminates by displaying the mapping sets and the approximate values (line 37).

ACP has two notable properties: the latency for a device does not go beyond the initial $2D_i$ bound since it is still assigned within $2D_i$, and the upgraded cloudlets do not increase the overall cost by more than the cost of filtering and rounding. We prove both bounds in Theorems 3 and Theorem 4, and demonstrate that ACP is a (2,4)-approximation algorithm for the OCP problem. Our proposed algorithm always finds a feasible solution with performance guarantees if `LP-OCP-Cost()` has a feasible solution. Given the cloudlet upgrades, ACP eventually covers all devices skipped due to insufficient capacity or radius. We structure our algorithm in this manner since checking if a feasible solution exists for OCP is NP-complete. For that, we

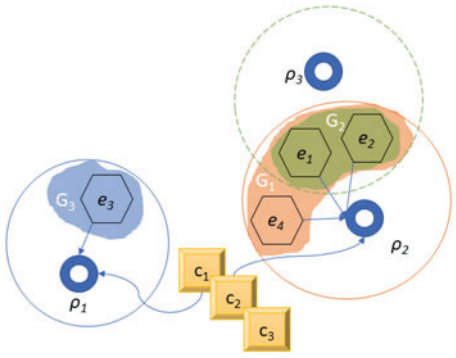


Fig. 3. Reduction of set cover decision problem to OCP.

need to strictly know, for every instance, if w cloudlets placed across n candidate points are sufficient for covering v devices. We prove this NP-completeness next.

Theorem 1 *Checking feasibility of the OCP problem is NP-complete.*

Proof Given an instance and a feasible solution of the OCP problem, we can verify if all devices are covered by the deployed cloudlets in polynomial time. It takes $O(|\mathcal{E}|)$ time to check coverage for all devices. However, given an arbitrary instance with w cloudlets, n candidate points, and v devices, it is NP-complete to check if a feasible solution exists. We prove this by reducing the Set Cover decision problem to OCP and vice versa in polynomial time.

We first briefly explain the Set Cover problem. Given a set of elements $U = \{1, 2, \ldots, u\}$ called the universe and a collection G of $|G|$ sets whose union equals the universe, the Set Cover decision problem is to identify whether there is a sub-collection of G with k sets $G^k \subseteq G$ whose union equals the universe U [39]. The decision version of the set cover problem is NP-complete.

For a given instance of the Set Cover decision problem, we can always construct an equivalent instance of the OCP problem. In doing so, we construct a collection of candidate point-to-device assignments, then the universe is $A_{\mathcal{E}}$, which equals to all assignments to cover the devices. Each device is required to be covered by at least one cloudlet. For every set (and their elements) in the collection set of set-cover G, we construct a set of corresponding assignments in $A_{\mathcal{E}}$ and assume it is the set of assignments that can be covered by deploying a cloudlet c on a candidate point ρ. Intuitively, the union of all such sets equals to $A_{\mathcal{E}}$. The set of cloudlets placed at the candidate points in $A_{\mathcal{E}}$ to cover all devices is hence $A_{\mathcal{C}}$.

We claim that there is a sub-collection of G with k sets, whose union equals the universe, if and only if there exist k candidate points to deploy cloudlets, which can cover all devices. We illustrate the reduction using an example of Set Cover decision problem with 4 elements $\{e_1, \ldots, e_4\}$, and OCP with 4 devices, 3 cloudlets, and 3 candidate points (as shown in Fig. 3). In the example, the sub-collections of G are $G_1 = \{e_2, e_1, e_4\}$, $G_2 = \{e_2, e_1\}$, and $G_3 = \{e_3\}$. The minimum number of subsets whose union equals the universe set is $k = 2$, with $G1$ and $G3$. Now, in the corresponding OCP problem, $\{e_3\}$ can be covered from ρ_1, $\{e_4, e_1, e_4\}$ are the devices that can be covered from ρ_2, and $\{e_1, e_2\}$ can be covered from ρ_3. Placing cloudlets c_1 and c_2 on ρ_1 and ρ_2 respectively is sufficient to cover all devices. Thus, the minimum number of candidate points to deploy cloudlets is 2. The assignment set is $A_{\mathcal{E}} = \{(\rho_1, e_3), (\rho_2, e_2), (\rho_2, e_1), (\rho_2, e_4)\}$, and the cloudlet assignment set is $A_{\mathcal{C}} = \{(\rho_1, c_1), (\rho_2, c_2)\}$.

Conversely, if we can deploy cloudlets on k candidate points to cover devices, then we can select corresponding k sets from G. As all devices are covered, the union of the sub-collection sets of G equals the universe. Therefore, since the Set Cover decision problem is an NP-complete problem, checking feasibility of the OCP problem is NP-complete. □

To prove the bounds, we first need to prove the following lemma.

Lemma 2 *Using* `LP−OCP−Cost()`*, at least half of every device $e_i \in \mathcal{E}$ is assigned to the candidate points in its neighborhood set N_i.*

Proof We need to prove $\sum_{\rho_k \in N_i} a^*_{ik} \geq 1/2$. Let $X_i(\beta)$ denote the subset of candidate points to which device e_i is fractionally assigned and are more than βD_i latency from e_i. Therefore, N_i in our formulation is equivalent to remaining candidate points to which e_i is fractionally assigned, i.e., $\forall \rho_k \notin X_i(\beta)$, where $\beta = 2$.

Let $z_{ik} = \sum_{\rho_k \in \mathcal{P}} a^*_{ik}$. We know that for any feasible solution, $z_{ik} = 1$. Suppose if $\sum_{\rho_k \in X_i(\beta)} a^*_{ik} > \frac{z_{ik}}{\beta}$. Then,

$$\begin{aligned} D_i &= \sum_{\rho_k \in X_i(\beta)} l_{ik}\, a^*_{ik} + \sum_{\rho_k \notin X_i(\beta)} l_{ik}\, a^*_{ik} \\ &\geq \sum_{\rho_k \in X_i(\beta)} l_{ik}\, a^*_{ik} \\ &\geq \beta D_i \sum_{\rho_k \in X_i(\beta)} a^*_{ik} \\ &> \beta D_i \frac{z_{ik}}{\beta} \\ &> D_i, \end{aligned}$$

which is a contradiction. Therefore, $\sum_{\rho_k \in X_i(\beta)} a^*_{ik} \leq \frac{z_{ik}}{\beta}$. And by extension, we have

$$\sum_{\rho_k \notin X_i(\beta)} a^*_{ik} \geq z_{ik}\left(1 - \frac{1}{\beta}\right) \geq 1 - \frac{1}{\beta}.$$

If we set $\beta = 2$, which is our condition for $2D_i$, $\sum_{\rho_k \notin X_i(2)} a^*_{ik} \geq 1/2$. This proves $\sum_{\rho_k \in N_i} a^*_{ik} \geq 1/2$. In other words, the sum of non-zero fractional assignments to filtered candidate points in N_i is at least $1/2$. □

Theorem 3 *ACP is a 2-approximation algorithm for latency ($\widehat{L} \leq 2L$).*

Proof ACP begins by assigning device e_i with the largest D_i to the closest candidate point ρ_i in its neighborhood set N_i (within $2D_i$ of e_i). Any device selected using this criterion is always assigned to a candidate point within $2D_i$.

Next, it builds an extended set for every candidate point ρ_i selected above. The extended set contains all devices $e_v \in \mathcal{E}$ that have ρ_i in their respective neighborhood set N_v. All devices that can be covered by cloudlet c_i placed at ρ_i are assigned to ρ_i. Since this ρ_i is

in N_v of any device e_v (by the definition of the extended set), all such devices are within $2D_v$ from ρ_i.

Any skipped device $e_i \in S$ is assigned in the second part of the algorithm. These devices are evaluated against cloudlet-candidate point pairs established in the solution set A_C in the first part. An important observation here is that the devices are skipped if and only if they were selected as the initial device (in line 11) with the largest D_i, i.e., they are never skipped from an extended set. This means an empty (*null*) cloudlet was returned by `BestCloudlet()` function due to capacity or radius constraint. Since we can upgrade the cloudlet when the `BestFeasiblePair()` has insufficient capacity or radius, any cloudlet-candidate pair assigned in the second part should be either in the device's N_i or not in N_i.

If it is in N_i, we know the latency is within $2D_i$. If it is not in N_i, it means `BestFeasiblePair()`, which always selects the candidate point with the least latency among available, has found a candidate point with better latency than the candidate points in N_i. So, the latency is still within $2D_i$. Note that if no candidate point is feasible, ACP will upgrade the cloudlet, and the latency will remain in $2D_i$. Therefore, all skipped devices are within $2D_i$ as well. Mathematically, we have

$$\widehat{\boldsymbol{L}} \leq \sum_{e_i \in \mathcal{E}} 2D_i. \tag{13}$$

We know that

$$D_i = \sum_{\rho_k \in \mathcal{P}} l_{ik}\, a^*_{ik}. \tag{14}$$

Since all individual devices are within $2D_i$, the sum over all devices is also within twice the value of LP latency, denoted by $\boldsymbol{L}^*$.

$$\begin{aligned}
&\sum_{e_i \in \mathcal{E}} D_i \leq \sum_{\rho_k \in \mathcal{P}} \sum_{e_i \in \mathcal{E}} l_{ik}\, a^*_{ik} \\
&\sum_{e_i \in \mathcal{E}} D_i \leq \boldsymbol{L}^* \\
&2 \sum_{e_i \in \mathcal{E}} D_i \leq 2\boldsymbol{L}^* \\
&\widehat{\boldsymbol{L}} \leq 2\boldsymbol{L}^*.
\end{aligned}$$

Since the LP latency ($\boldsymbol{L}^*$) is a lower-bound on the optimal latency value of the OCP problem ($\boldsymbol{L}^* \leq \boldsymbol{L}$), ACP is a 2-approximation algorithm for latency ($\widehat{\boldsymbol{L}} \leq 2\boldsymbol{L}$). □

Theorem 4 *ACP is a 4-approximation algorithm for placement cost ($\widehat{\boldsymbol{\Phi}} \leq 4\boldsymbol{\Phi}$).*

Proof We prove the placement cost bound for ACP using properties of the `LP-OCP-Cost()` solution, linearity of the cost function, and nature of the upgrades.

The solution of ACP is based on an important assignment property, where for every device e_i, at least half of it is fractionally assigned to its neighborhood set N_i (see Lemma 2).

$$\sum_{\rho_k \in N_i} a^*_{ik} \geq 1/2. \tag{15}$$

As we start assigning devices to the candidate points in their N_i and cloudlets to those candidate points, we round the fractional a^*_{ik} and y^*_{ik} values to either 0 or 1 to obtain the approximate assignments $\widehat{a}_{ik}$ and $\widehat{y}_{jk}$, respectively (Algorithm 1, line 17-18). Since every device is assigned to exactly one candidate point in ACP (also in OCP, constraint (12)), we have that the approximate device to cloudlet assignment $\sum_{\rho_k \in N_i} \widehat{a}_{ik} = 1$, $\forall e_i \in \mathcal{E}$. Combining with Eq. (15), this leads us to

$$\sum_{\rho_k \in N_i} \widehat{a}_{ik} \leq 2 \sum_{\rho_k \in N_i} a^*_{ik}. \tag{16}$$

There is also a mathematical relationship between the two decision variables based on constraint (7) of OCP. The `LP-OCP-Cost()` solution must follow this constraint for a feasible solution. Thus, we have

$$a^*_{ik} \leq \sum_{c_j \in \mathcal{C}} y^*_{jk} \qquad \forall e_i \in \mathcal{E}, \rho_k \in \mathcal{P}. \tag{17}$$

Since the relation above is true for every candidate point $\rho_k \in \mathcal{P}$, it is true for sum over all $\rho_k \in N_i$. Thus,

$$\sum_{\rho_k \in N_i} a^*_{ik} \leq \sum_{\rho_k \in N_i} \sum_{c_j \in \mathcal{C}} y^*_{jk} \qquad \forall e_i \in \mathcal{E}. \tag{18}$$

Combining Eqs. (16) and (18), we get

$$\sum_{\rho_k \in N_i} \widehat{a}_{ik} \leq 2 \sum_{\rho_k \in N_i} \sum_{c_j \in \mathcal{C}} y^*_{jk} \qquad \forall e_i \in \mathcal{E}. \tag{19}$$

In ACP, exactly one cloudlet is placed at ρ_k to which a device e_i is assigned, and a device is assigned to exactly one ρ_k, which gives us the relation

$$\sum_{c_j \in \mathcal{C}} \widehat{y}_{jk} = \sum_{\rho_k \in \mathcal{P}} \widehat{a}_{ik} \qquad \forall e_i \in \mathcal{E}. \tag{20}$$

This equation implies that every device is served by a single cloudlet uniquely placed at a candidate point.

Since $\sum_{c_j \in \mathcal{C}} \widehat{y}_{jk}$ is equal to the value of $\sum_{\rho_k \in \mathcal{P}} \widehat{a}_{ik}$ for each device e_i and from Eq. (19), we have

$$\sum_{\rho_k \in N_i} \sum_{c_j \in \mathcal{C}} \widehat{y}_{jk} \leq 2 \sum_{\rho_k \in N_i} \sum_{c_j \in \mathcal{C}} y^*_{jk}. \tag{21}$$

Having a linear cost function based on capacity and radius of the cloudlets, any change in the objective value (cost) from `LP-OCP-Cost()` to ACP is linearly proportional to a change in y^*_{jk} values over all candidate points and cloudlets (as seen in Eq. (21)). This is because no capacity other than the cloudlets fractionally used in `LP-OCP-Cost()` is used (only rounded) in the first part of ACP. The added capacity, hence the added cost, only comes from the upgrades in the second part of ACP. If the set of the upgrades (c^+_j) added is $\mathcal{L}$, the overall approximate cost is given by

$$\widehat{\boldsymbol{\Phi}} = \sum_{c_j \in \mathcal{C}} \sum_{\rho_k \in N_i} \phi_{jk} \widehat{y}_{jk} + \sum_{c^+_j \in \mathcal{L}} \sum_{\rho_k \in N_i} \phi_{jk} \widehat{y}_{jk}.$$

From Lemma 5, the upgrades do not exceed the overall placement cost of cloudlets placed in the first part of ACP.

$$\widehat{\mathbf{\Phi}} \leq 2 \sum_{c_j \in \mathcal{C}} \sum_{\rho_k \in N_i} \phi_{jk} \widehat{y}_{jk}.$$

As all candidate points not in N_i of devices are never chosen for placement, we can perform the above sum over all candidate points in $\mathcal{P}$, and using Eq. (21) we have

$$\widehat{\mathbf{\Phi}} \leq 2 \times 2 \sum_{j \in \mathcal{C}} \sum_{k \in \mathcal{P}} \phi_{jk} y^*_{jk} \qquad \widehat{\mathbf{\Phi}} \leq 4\mathbf{\Phi}^*$$

Since the LP cost ($\mathbf{\Phi}^*$) is a lower-bound on the cost value of the optimal solution, ACP is a 4-approximation algorithm for the placement cost ($\widehat{\mathbf{\Phi}} \leq 4\mathbf{\Phi}$). □

Lemma 5 *The upgrade cost does not exceed the cost of cloudlets placed in the first part of ACP.*

Proof Based on Lemma 2, at least half-of each device e_i is assigned to the candidate points in their N_i by `LP-OCP-Cost()`. Consequently, at least half of the device demands are met by placing cloudlets in N_i. Since devices are either assigned or skipped in the first part, the upgrades need to meet at most the remaining half of the overall demands. Hence, in the worst case, the overall capacity and radius of the cloudlets have to be doubled by upgrading them. Again, we know that each upgrade is no larger than the size of the smallest cloudlet. If only smallest cloudlets are being used to cover the demands, it leads to the least possible cost for that demand since cloudlet placement is binary (either place whole cloudlet or none). This is at worst as costly as the placements in the first part. Therefore, the worst cost of upgrades does not exceed the cost of the first part of ACP. □

4 Experimental Results

4.1 Experimental Setup

As much as we have theoretically demonstrated the performance guarantees of our approach, the practical scenarios where the approach needs to be implemented can be different from the extreme cases we analyzed in the proofs. Thus, we perform an extensive set of experiments to investigate the effectiveness of ACP. For our deployment scenarios, we use the five boroughs of New York City as it has been selected by National Science Foundation (NSF) as a testbed for the new wave of mobile technology [11], [40]. Also, the presence of NYC Open Data [10] allows us to access the latest information about implemented hotspot locations, types of placement locations, and usage statistics. The primary datasets that we utilize for our experiments are: NYC WiFi Hotspot Locations, LinkNYC Map, and LinkNYC Usage Statistics.

To setup the experiments, we treat the five boroughs of New York City as individual scenarios since they provide sufficient variety in hotspot locations, candidate point selection, and usage metrics. NYC Open Data has equivalent 2D coordinates defined for each geolocation representing a WiFi hotspot or Kiosk. We use the exact locations in our experiments without any simplification or reduction so the experiments represent as close of a scenario to the real world as possible. The hostspot locations represent the device locations in our setup. Likewise, the hotspot locations that represent a feasible space for cloudlet placement such as a subway station or a local library serve as a candidate point.

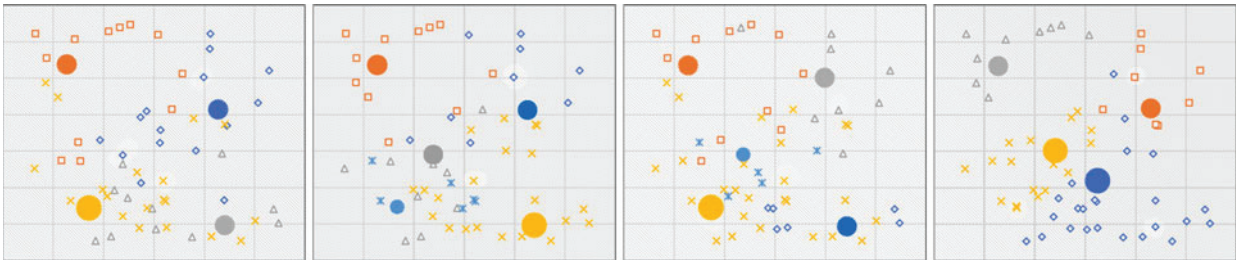
TABLE 5
Experiment Scenarios

Scenario	$\lvert\mathcal{P}\rvert$	$\lvert\mathcal{C}\rvert$	$\lvert\mathcal{E}\rvert$
Staten Island	7-10	5-8	59-71
Bronx	31-40	24-31	190-210
Queens	57-69	44-52	321-349
Brooklyn	81-97	61-73	428-451
Manhattan	113-127	81-96	1042-1091

The device demands (processing, memory, storage) are randomly selected from a uniform distribution, $X \sim U(5, 20)$ to signify heterogeneous demands. This broad range of demands also signifies that we are accounting for multiple devices at some hotspot locations. The cloudlet capacities are subsequently specified based on the total demand of the devices in each scenario. Finally, the coverage radii of the cloudlets is based on the mean latency of the devices from the candidate points. Each scenario represents a set of experiments than a single experiment. We randomly draw fixed number of candidate points and devices from each scenario to create 30 similarly-sized sub-scenarios. This provides further experimental variety and ability to statistically analyze the performance at scenarios of different sizes and scales. The experiment scenarios and their deployment sizes are summarized in Table 5.

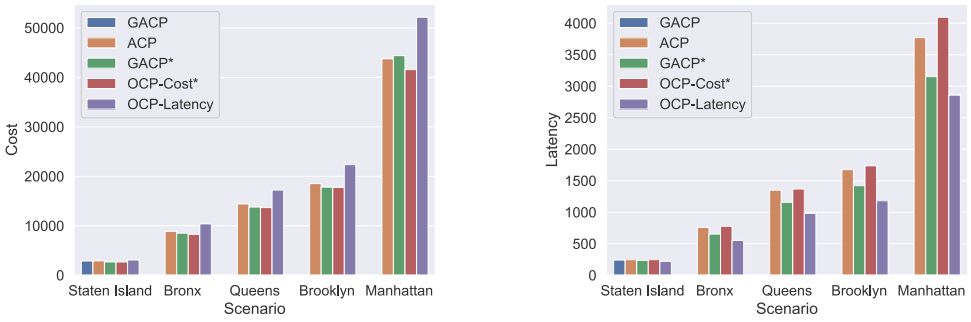
We run four different types of experiments to comprehensively evaluate and compare the performance of the proposed approach ACP to the three related approaches: the Optimal Cost of the Cloudlet Placement (OCP-Cost), the Optimal Latency of the Cloudlet Placement (OCP-Latency), and the Genetic Algorithm-based Cloudlet Placement (GACP), proposed in [41]. The first experiment shows the coverage of each approach, visualizing the device assignments and the obtained locations for the cloudlet placement. The next two experiments show the comparisons for the placement cost and the latency individually. Finally, we investigate the scalability of the approaches.

The optimal results from OCP-Cost and OCP-Latency are found using IBM ILOG CPLEX Concert Technology API for Java [42]. ACP and GACP partially use CPLEX to obtain LP results to guide their respective solution approaches. It is noteworthy that solving any LP optimally (unlike IP) using CPLEX takes polynomial time as it is in P, and other approaches can be used to obtain optimal fractional results. Both ACP and GACP are implemented in the same version of Java, and the experiments for all approaches are run on the same JVM on the Nautilus HyperCluster [43] with 16 CPU cores and 64 GB RAM. Note that we could not compare our results with other existing studies because they either do not have the same objectives or have different constraints as we discussed in Section 1.1. Hence, any direct comparisons would be unfair. It is also noteworthy that CPLEX provides the optimal results for the small cases of the problem, which is used as a proper benchmark. However, it is not able to obtain any results for large-scale problems due to NP-hardness of the problem. GACP likewise struggles to converge with full coverage when the experiment instances have narrow solution spaces. As a result,



(a) OCP-Cost Placement (b) OCP-Latency Placement (c) ACP Placement (d) GACP Placement

Fig. 4. Cloudlet coverage.



(a) Placement Cost of Different Approaches (b) Latency Suffered by Different Approaches

Fig. 5. Summary of Cost and Latency values. *GACP* denotes only the instances unable to obtain full coverage due to indefinite convergence time. OCP-Cost* denotes that CPLEX did not converge to an optimal solution, and the Percentage Gap value from a node-limited CPLEX run has been used to derive the "best estimated" optimal value.*

OCP-Cost and GACP have partial but valid results which are explained further in the analysis section.

4.2 Analysis of Results

We first compare the results of ACP with OCP-Cost, OCP-Latency, and GACP approaches in terms of device coverage. In Fig. 4, we visualize the locations of the placed cloudlets using shaded circles (larger circles represent larger cloudlets) and the devices covered by those cloudlets with different markers (using the same color) in each approach. As we can see, OCP-Cost, shown in Fig. 4a, uses the minimum cloudlet resources to obtain the minimum cost. OCP-Latency, shown in Fig. 4b, covers devices with the minimum latency (the assignment of the devices are mostly to their closest cloudlets, and an additional cloudlet has been used, compared to that of OCP-Cost). GACP, shown in Fig. 4d, effectively clusters the devices around the placed cloudlets while using the same number of cloudlets as OCP-Cost. However, it uses two large cloudlets since a medium cloudlet is mutated by GACP to a large cloudlet. So, it obtains a higher cost. Likewise, the latency is not minimized in GACP since some devices are far away from their mapped cloudlet despite the clustering. In contrast, ACP, shown in Fig. 4c, achieves coverage similar to OCP-Cost and OCP-Latency. The overall coverage scenario of ACP can be seen as a blend of these two optimal benchmarks. ACP reflects the optimal solutions closely while taking significantly less running time. We analyze them in detail next.

Fig. 5a shows the placement costs obtained by the approaches. Before explaining the results, we should note that CPLEX was not able to converge to provably optimal solutions for OCP-Cost for more than 24 hours. Hence, a node-limited solution was used to best estimate the optimal value (presented by OCP-Cost*) based on the solution gap value given by CPLEX. The average OCP-Cost* solution gap for each scenario is presented in Fig. 6b. Likewise, GACP was not able to converge with full coverage for all instances. Hence, solutions with partial (yet high) coverage values are presented for comparison as indicated by GACP*. The amount of coverage GACP* achieved on average is shown in Fig. 6a.

The results show that the costs obtained by ACP are very close (and much lower than the proven theoretical bound) to the optimal costs in all scenarios. Although GACP* costs in some scenarios are lower, it is because GACP* does not cover all devices. For the largest Manhattan scenario, ACP, covering all devices, obtains observably lower cost than GACP*, which only covers 85% devices on average. Costs obtained by ACP are similarly much lower than OCP-Latency costs.

The latency values presented in Fig. 5b show that ACP obtains higher latency than GACP*. However, note that the results of GACP* do not include the high latency values by simply not covering those devices. A direct comparison with full coverage GACP in Staten Island shows no significant difference in latency. In addition, despite the fact that latency values of OCP-Cost* and ACP look comparable for Staten Island, Bronx, and Queens, ACP is able to achieve incrementally lower latency than OCP-Cost* as the problem size grows. Also, all latency values of ACP for the scenarios are still within the proven theoretical bound.

Finally, we perform experiments to observe running time and scalability of the approaches. Since the number of cloudlets is already determined from the device demand, we only consider the number of candidate points and the number of devices in our experiments. Fig. 6c shows running time as a function of the number of candidate points

(a) GACP* Coverage Values (b) OCP-Cost* Solution Gap (c) Running Time

Fig. 6. Partial solution scenarios and running time.

and the number of devices in a 3D plot. The vertical axis uses a logarithmic scale (log_2) that shows the running time in seconds. As we move from left to right, the data points are closer to the viewer's perspective, and they correspond from the smallest to the largest experiment scenarios in Table 5. We can observe that ACP is faster than the compared approaches in all instances except for the largest scenario (Manhattan), where OCP-Latency is faster. Being an NP-hard problem, both OCP-Cost and OCP-Latency have random bursts (peaks) in running time. In addition, both of them were not able to find a feasible solution for some instances. ACP and GACP, on the other hand, can find a solution even in those scenarios because they can perform upgrades and mutations, respectively.

GACP* seems to coincide in terms of running time with ACP in most instances because both ACP and GACP rely on `LP-OCP-Cost()` solution, which makes up the majority portion of their running time. However, purely ACP running time is significantly faster than purely GACP convergence time. Moreover, ACP achieves full coverage which GACP is simply unable for majority of instances. One of the weakness of GACP is that it may never converge for difficult problem instances. This can be seen in Fig. 6a where the coverage values decrease as the problem size increases. Therefore, ACP is the only approach which guarantees full coverage in a polynomial running time.

To summarize, ACP is able to reduce cost and latency by performing smaller upgrades to existing cloudlets instead of provisioning new cloudlets. Practically, if appropriate estimates are made about user demands and their distribution, and a reasonable set of candidate points are established, ACP does not even need the upgrades to find a feasible solution. In that case, we have even tighter bounds on cost and latency. This is clearly observable in our results above. The efficient, polynomial running time makes this method suitable for both short-term and long-term placements.

Our source code and datasets are publicly available online. The experiments can be reproduced using our Code Ocean capsule available at [44].

5 Conclusion

The primary motivation of edge computing is to mitigate latency suffered by the users and save network bandwidth by placing resources closer to where they are consumed. In highly heterogeneous scenarios like 5G networks and IoT, deployment cost becomes an equally important parameter since it is extremely expensive to cover all users. Indiscriminate placement or disregard for heterogeneity necessarily make any approach both inefficient and ineffective in next generation networks. Existing literature lacks considering heterogeneous scenarios while simultaneously reducing cost and latency. In this paper, we address these challenges by designing a bifactor approximation algorithm for the heterogeneous cloudlet placement problem. Our approach, ACP, runs in polynomial time and provides a (2,4) approximation bound for latency and cost, respectively. The rigorous experimental results show that ACP is scalable and achieves close to the optimal results in experimental scenarios. Our approach, however, does not consider mobility factors in the placement decision, which need a more meticulous formulation. Mobility can be handled either proactively (predictive placement) or reactively (adaptive placement), both of which are possible extensions of our work. Further enhancements can also be done by introducing 3D coverage, adaptive user mapping, and distributed placement decisions.

Acknowledgments

This article is an extended version of [41], which received the Best Paper Award of the 2019 CloudCom.

References

[1] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A comprehensive survey on fog computing: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 416–464, First Quarter 2018.

[2] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, Third Quarter 2017.

[3] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[4] N. Sharghivand, F. Derakhshan, L. Mashayekhy, and L. Mohammadkhanli, "An edge computing matching framework with guaranteed quality of service," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2020.3005539.

[5] W. Ma and L. Mashayekhy, "Truthful computation offloading mechanisms for edge computing," in *Proc. 6th IEEE Int. Conf. Edge Comput. Scalable Cloud*, 2020, pp. 1–8.

[6] E. F. Maleki, L. Mashayekhy, and S. M. Nabavinejad, "Mobility-aware computation offloading in edge computing using machine learning," *IEEE Trans. Mobile Comput.*, to be published, doi: 10.1109/TMC.2021.3085527.

[7] P. Ahokangas *et al*., "Business models for local 5G micro operators," *IEEE Trans. Cogn. Commun. Netw.*, vol. 5, no. 3, pp. 730–740, Sep. 2019.
[8] *Harness the power of 5G edge*. 2017. Accessed: Dec. 31, 2020. [Online]. Available: https://enterprise.verizon.com/business/learn/edge-computing/
[9] D. P. Williamson and D. Shmoys, *The Design of Approximation Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2011, ch. 1, pp. 10–20.
[10] *City of New York*, "NYC open data," 2017. Accessed: Feb. 22, 2019. [Online]. Available: https://opendata.cityofnewyork.us/data/
[11] *Cloud enhanced open software defined mobile wireless testbed for city-scale deployment (COSMOS)*. Accessed: Dec. 29, 2020. [Online]. Available: https://www.cosmos-lab.org/
[12] M. Mahdian, Y. Ye, and J. Zhang, "Approximation algorithms for metric facility location problems," *SIAM J. Comput.*, vol. 36, no. 2, pp. 411–432, 2006.
[13] D. Shmoys, "Approximation algorithms for facility location problems," in *Proc. Int. Workshop Approximation Algorithms Combinatorial Optim.*, 2000, pp. 27–33.
[14] F. A. Chudak and D. B. Shmoys, "Improved approximation algorithms for the uncapacitated facility location problem," *SIAM J. Comput.*, vol. 33, no. 1, pp. 1–25, 2004.
[15] J. Byrka and K. Aardal, "An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem," *SIAM J. Comput.*, vol. 39, no. 6, pp. 2212–2231, 2010.
[16] S. Li, "A 1.488 approximation algorithm for the uncapacitated facility location problem," *Inf. Comput.*, vol. 222, pp. 45–58, 2013.
[17] K. Aardal, P. L. van den Berg, D. Gijswijt, and S. Li, "Approximation algorithms for hard capacitated k-facility location problems," *Eur. J. Oper. Res.*, vol. 242, no. 2, pp. 358–368, 2015.
[18] S. Raghavan, M. Sahin, and F. S. Salman, "The capacitated mobile facility location problem," *Eur. J. Oper. Res.*, vol. 277, no. 2, pp. 507–520, 2019.
[19] J.-H. Lin and J. S. Vitter, "Approximation algorithms for geometric median problems," *Inf. Process. Lett.*, vol. 44, no. 5, pp. 245–249, 1992.
[20] R. Cohen, L. Katzir, and D. Raz, "An efficient approximation for the generalized assignment problem," *Inf. Process. Lett.*, vol. 100, no. 4, pp. 162–166, 2006.
[21] D. B. Shmoys and É. Tardos, "An approximation algorithm for the generalized assignment problem," *Math. Program.*, vol. 62, no. 1/3, pp. 461–474, 1993.
[22] S. Kang, L. Ruan, S. Guo, W. Li, and X. Qiu, "Geographic clustering based mobile edge computing resource allocation optimization mechanism," in *Proc. 15th Int. Conf. Netw. Service Manage.*, 2019, pp. 1–5.
[23] M. Jia, J. Cao, and W. Liang, "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks," *IEEE Trans. Cloud Comput.*, vol. 5, no. 4, pp. 725–737, Fourth Quarter 2017.
[24] Y. Zhang, K. Wang, Y. Zhou, and Q. He, "Enhanced adaptive cloudlet placement approach for mobile application on spark," *Secur. Commun. Netw.*, vol. 2018, pp. 1–12, 2018.
[25] F. Zeng, Y. Ren, X. Deng, and W. Li, "Cost-effective edge server placement in wireless metropolitan area networks," *Sensors*, vol. 19, 2018, Art. no. 32.
[26] Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo, "Efficient algorithms for capacitated cloudlet placements," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 10, pp. 2866–2880, Oct. 2016.
[27] Y. Li and S. Wang, "An energy-aware edge server placement algorithm in mobile edge computing," in *Proc. IEEE Int. Conf. Edge Comput.*, 2018, pp. 66–73.
[28] H. Yao, C. Bai, M. Xiong, D. Zeng, and Z. Fu, "Heterogeneous cloudlet deployment and user-cloudlet association toward cost effective fog computing," *Concurrency Comput., Pract. Experience*, vol. 29, no. 16, pp. 1–9, 2017.
[29] Z. Wang, F. Gao, and X. Jin, "Optimal deployment of cloudlets based on cost and latency in Internet of Things networks," *Wireless Netw.*, vol. 26, no. 8, pp. 6077–6093, 2020.
[30] Q. Fan and N. Ansari, "On cost aware cloudlet placement for mobile edge computing," *IEEE/CAA J. Automatica Sinica*, vol. 6, no. 4, pp. 926–937, Jul. 2019.
[31] S. Mondal, G. Das, and E. Wong, "Cost-optimal cloudlet placement frameworks over fiber-wireless access networks for low-latency applications," *J. Netw. Comput. Appl.*, vol. 138, pp. 27–38, 2019.
[32] F. Wang, X. Huang, H. Nian, Q. He, Y. Yang, and C. Zhang, "Cost-effective edge server placement in edge computing," in *Proc. 5th Int. Conf. Syst. Control Commun.*, 2019, pp. 6–10.
[33] S. Wang, Y. Zhao, J. Xu, J. Yuan, and C.-H. Hsu, "Edge server placement in mobile edge computing," *J. Parallel Distrib. Comput.*, vol. 127, pp. 160–168, 2019.
[34] Y. Guo, S. Wang, A. Zhou, J. Xu, J. Yuan, and C.-H. Hsu, "User allocation-aware edge cloud placement in mobile edge computing," *Softw., Pract. Experience*, vol. 50, no. 5, pp. 489–502, 2020.
[35] L. Ma, J. Wu, L. Chen, and Z. Liu, "Fast algorithms for capacitated cloudlet placements," in *Proc. IEEE 21st Int. Conf. Comput. Supported Cooperative Work Des.*, 2017, pp. 439–444.
[36] J. Meng, W. Shi, H. Tan, and X. Li, "Cloudlet placement and minimum-delay routing in cloudlet computing," in *Proc. 3rd Int. Conf. Big Data Comput. Commun.*, 2017, pp. 297–304.
[37] R. Goonatilake and R. Bachnak, "Modeling latency in a network distribution," *Netw. Commun. Technol.*, vol. 1, no. 2, pp. 1–11, 2012.
[38] G. Mavrotas, "Effective implementation of the ϵ-constraint method in multi-objective mathematical programming problems," *Appl. Math. Comput.*, vol. 213, no. 2, pp. 455–465, 2009.
[39] R. Karp, "Reducibility among combinatorial problems," *Complexity Comput. Comput.*, vol. 40, pp. 85–103, Jan. 1972.
[40] *Columbia engineering*, "NSF announces New York City as testbed for new wave of mobile technology. Accessed: Jan. 20, 2019, 2018. [Online]. Available: https://engineering.columbia.edu/news/nsf-cosmos-testbed
[41] D. Bhatta and L. Mashayekhy, "Generalized cost-aware cloudlet placement for vehicular edge computing systems," in *Proc. 11th IEEE Int. Conf. Cloud Comput. Technol. Sci.*, 2019, pp. 159–166.
[42] *IBM*, "Overview (CPLEX Java API Reference Manual). 2017. Accessed: Feb. 25, 2019. [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSSA5P_12.8.0/ilog.odms.cplex.help/refjavacplex/html/index.html
[43] *Nautilus documentation*. Accessed: Nov. 30, 2020. [Online]. Available: https://ucsd-prp.gitlab.io/nautilus/
[44] D. Bhatta and L. Mashayekhy, "Bifactor approx cloudlet placement," 2021. [Online]. Available: https://doi.org/10.24433/CO.4921212.v1

Dixit Bhatta (Student Member, IEEE) received the BSc degree in computer science and information technology from Tribhuvan University (St. Xavier's College), Nepal, in 2015, and the second BS degree in computer science from the University of the People (Online), Pasadena, California, in 2016. He is currently working toward the PhD degree and a doctoral fellow in the Department of Computer and Information Sciences, University of Delaware, Newark, Delaware. He received the Best Paper Award at the IEEE CloudCom 2019 and the 2020 Outstanding Graduate Student Award. His research interests include edge/cloud computing, Internet of Things, and distributed systems. He is a student member of the ACM.

Lena Mashayekhy (Senior Member, IEEE) is an assistant professor in the Department of Computer and Information Sciences, University of Delaware, Newark, Delaware. Her research interests include edge/cloud computing, data-intensive computing, Internet of Things, and algorithmic game theory. Her doctoral dissertation received the 2016 IEEE TCSC Outstanding PhD Dissertation Award. She is also a recipient of the 2017 IEEE TCSC Award for Excellence in Scalable Computing for Early Career Researchers. She has published more than forty five peer-reviewed papers in venues such as the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Cloud Computing*. She is a member of the ACM.