# EFFICIENT AND SCALABLE CLOUDLET PLACEMENT APPROACHES FOR EDGE COMPUTING IN NEXT-GENERATION NETWORKS

by

Dixit Bhatta

A dissertation submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science

Summer 2022

**EFFICIENT AND SCALABLE CLOUDLET PLACEMENT APPROACHES FOR EDGE COMPUTING IN NEXT-GENERATION NETWORKS**

by

Dixit Bhatta

Approved: _____

Rudolf Eigenmann, Ph.D.
Interim Chair of the Department of Computer and Information Sciences

Approved: _____

Levi T. Thompson, Ph.D.
Dean of the College of Engineering

Approved: _____

Louis F. Rossi, Ph.D.
Vice Provost for Graduate and Professional Education and
Dean of the Graduate College

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Lena Mashayekhy, Ph.D.
Professor in charge of dissertation

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Chien-Chung Shen, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Guangmo Tong, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Hana Khamfroush, Ph.D.
Member of dissertation committee

## ACKNOWLEDGEMENTS

The completion of this dissertation would not have been possible without continuous guidance and support of my advisor, Dr. Lena Mashayekhy. I would also like to thank Dr. Shen, Dr. Tong, and Dr. Khamfroush for your generous contributions and support as my dissertation committee members.

I express my gratitude to the University of Delaware Graduate College for providing me Doctoral and Dissertation fellowships to cover the most important years of my research. My sincere thanks to the National Research Platform for providing large-scale computing resources to conduct experiments and simulations.

Last but not the least, I would like to thank my parents Mr. Hem Raj Bhatta and Mrs. Dipa Bhatta, and my brother Mr. Kshitiz Bhatta for always being there; none of this would have been possible without you.

## TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

## 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. Since large data centers cannot be placed everywhere, placing cloudlets nearby the users provides an effective solution to these requirements. 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 networks and IoT applications is still an open challenge.

The goal of this dissertation is to specifically tackle strategic placement of cloudlets to reduce costs associated with providing edge services (deployment cost, energy consumption) as well as improving metrics important to the users such as coverage, service stability, and access latency. These problems are significantly challenging because the cloudlets have limited capacities whereas users have dynamic demands. Moreover, both users and cloudlets can be heterogeneous. As we move into next-generation networks and IoT, the placements need to be done in even denser and highly heterogeneous scenarios. Another challenging aspect is the continuous user mobility. These several placement challenges in edge computing systems are hence addressed by mathematically modeling diverse placement scenarios based on real-life constraints. However, the complexities of these problems make the mathematical models inherently NP-hard, even with the simplifying assumptions. As a result, the core of the research lies at solving these problems by designing efficient and scalable solution approaches.

# Chapter 1

# INTRODUCTION

## 1.1 Background

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 [56]. As per Gartner Newsroom around 6.3 billion connected "Things" were used in 2016, and this number was projected to be 20.4 billion at the end of 2020 [3]. Likewise, AT&T stated in 2019 that data traffic on their mobile network has increased by about 470,000% since 2007. They expect 4K video, autonomous cars, drones, VR/AR, and mobile gaming to drive as much as 75% of their data traffic in 2022 [2]. 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 [46]. These restrictions impose limitations on processing power, memory, and storage capacities of these devices.

Edge computing, sometimes used interchangeably with the term Fog 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" [63, 64]. 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 [23, 45].

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 networks and IoT applications. It is now clear that there will be many novel, cutting-edge, and large-scale applications on billions of connected devices and IoT (Internet of Things) endpoints that will be pushing boundaries of the existing infrastructure [49, 67]. As we move into and beyond 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]. Cloudlets here can be used to complement available cloud resources as well as provide quick ad hoc solutions to areas with low coverage [23].

Cloudlet placement is at the core of providing ultra-low latency services by the likes of Verizon Inc. with their 5G Edge service [4]. 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 the next section, we explore the current research gap and how the approaches we design address those concerns.

### 1.2 Research Gap

To realize full potential of edge computing, efficient and scalable approaches for placing cloudlets are needed. Different approaches have been used to tackle the cloudlet placement problem, however, they do not fully address all important aspects of cloudlet placement in edge computing systems and next-generation networks. We have divided

the research gap into two categories based on the scope of the proposed solutions: classical placement problem approaches and edge computing placement approaches.

### 1.2.1 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 [15, 16, 39, 48, 66] 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 parameters important to our research such as placement costs and latency. However, there are no capacity constraints. Mahdian *et al.* [48] 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.* [6] 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.* [62] 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.

There are some studies which are too broad to considered an edge computing placement approach and too specific to be considered a classical placement problem approach. These studies lie in the intersection of our two categories. A study by Liu *et al* [41] uses k-means clustering on spatial distribution of base stations in 5G networks.

Their work is brief and does not consider heterogeneity, capacities, and placement costs. Jiao *et al.* [32] proposed a sequential heuristic algorithm that runs in polynomial time. However, their approach applies to only tree-structured networks. Likewise, they have guarantees on convergence but no performance bound. All of these reasons necessitate specific approaches that directly address unique requirements of edge computing in next-generation networks.

### 1.2.2 Edge Computing Placement Approaches

#### 1.2.2.1 Static Placement Approaches

Multiple clustering-based approaches have been proposed to place cloudlets. Kang *et al.* [33] used a greedy geographic clustering approach to balance workload and collaborative scheduling to reduce access delay. Jia *et al.* [31] proposed density-based clustering and k-means clustering of users to minimize response time. Zhang *et al.* [82] 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 without consideration for individual user requirement. This is unlike real world where the connectivity requirements of the users must be met individually.

A few studies concentrate on placing edge computing resources for one specific application such as big data processing [22, 68] or virtual reality [10]. Fan and Ansari [22] considered the placement for big data processing based on a load allocation scheme. Veith *et al.* [68] also presented a study for data stream analytics at edge. Likewise, Bastug *et al.* [10] discussed edge server placement for augmented and virtual reality applications. Extremely specific properties of the system are leveraged in these studies rendering them ineffective for general placement of cloudlets.

Further greedy approaches include Zeng *et al.* [81], who proposed a greedy-based algorithm that minimizes the number of cloudlets to be placed considering latency requirements. A greedy heuristic approach is proposed in [77] to reduce access delay for

users served from nearby access points. Li *et al.* [40] proposed energy-aware placement of cloudlets using swarm optimization, while assuming users are mapped to cloudlets through base stations. Yao *et al.* [78] presented a greedy approach with heterogeneous cloudlets and placement costs. However, they consider 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.* [72] proposing a binary-based differential evolution cuckoo search (BDECS) algorithm. They emphasize on deployment of cloudlets based on cost and latency for Internet of Things (IoT) scenario. However, they consider fixed placement cost ignoring heterogeneity. Another study [69] 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. In short, none of these studies investigates a general heterogeneous deployment scenario or considers the mapping of individual users or devices to cloudlets. Therefore, we have proposed two static cloudlet placement approaches that directly address these issues.

#### 1.2.2.2 Dynamic Placement Approaches

The static approaches described in the last two sections do not consider the mobility of the users, the cloudlets, or both since they are primarily designed for permanent placement of cloudlets, hence, are not suitable for scenarios where placements need to be updated in very short time intervals.

When it comes to dynamic placement of mobile computing resources, there are related studies on UAV positioning and mobility models in flying ad-hoc networks (FANET) and node placement in wireless sensor networks (WSN) [58, 79]. Studies on WSN node placement focus on maximizing surveillance area or geographical coverage of the nodes [79]. However, our problem is concerned with covering actual devices that need computing resources and connectivity to services. Moreover, higher geographic

coverage may not even lead to better coverage of users or devices in a dynamic environment where multiple sub-regions may not even have any users. Likewise, FANET positioning and mobility models are essentially geographical deployments, and they too do not follow the users or their demands. Many models are in fact based on time-based, topological, and even random positioning of UAVs, agnostic of user mobility or demand [58]. Hence, existing studies in both of these domains insufficiently address the dynamic mobile cloudlet placement problem.

Specific to mobile cloudlet placements, there are extremely limited studies on dynamic or online placement of cloudlets since the primary infrastructure has been most often perceived to be static and perennial. However, mobile cloudlets have become increasingly relevant due to ubiquitous mobile applications today, rendering a necessity for more studies. Most studies instead focus on online user allocation based on offloading or QoS requirements to already placed cloudlets [37, 59, 61, 75]. These are user-centric approaches which address only part of our overall placement problem. Xiang *et al.* [76] proposed an adaptive cloudlet placement approach for mobile applications. Their approach identifies gathering regions of mobile devices using position clustering and generates mobility paths of the cloudlets to new locations based on the shortest distance. Their approach inherently does not capture the mobility of each cloudlet as a part of the main decision, which is to determine the cluster centers instead. They recalculate cluster centers in each time slot and only calculate the mobility paths after the new cluster centers are established. Moreover, their approach looks at the placement from a 2D perspective with homogeneous cloudlets and is computationally-heavy, which limits its applicability significantly, especially for real-time scenarios. Zhang *et al.* [83] presented another adaptive cloudlet placement approach that directly improves on the previous study. They proposed a covering-based clustering technique to determine cloudlet placement locations. They also made the covering algorithm parallel on Spark to speed it up. However, their approach again relies on generating the shortest mobility path, and the limitations stay the same since their core assumptions are exactly the same as the previous study.

Wang *et al.* [70] proposed an online algorithm that dispatches UAV-mounted edge servers by identifying UAV hover locations to complement existing infrastructure during heavy usage by maximizing the number of served tasks. However, they do not consider separating distance between the UAVs and possible interference when they are placed too close to each other, around the same location. Although the UAVs hover at different heights, all devices are assumed to be on the ground. Hence, the coverage is observed in terms of the 2D influence radius on the ground. Moreover, they do not consider the concept of service stability. A dynamic approach for mobile environments by Yuan *et al.* [80] uses deep-learning for virtual edge node placements in edge cloud systems. Their combination of deep-learning-based predictions and hierarchical-clustering-based placement approach is catered more towards dynamic service placement than the actual placement of mobile cloudlets. Service or application placement approaches have inherently different properties which are not suitable to our problem. Moreover, user mobility is highly uncertain and even deep-learning may lead to inaccuracy and consequently high latency services. Furthermore, considering the importance of user privacy, we aim to design an approach that does not rely on learning or predicting user mobility patterns.

Apart from [70], none of the existing dynamic placement approaches consider cloudlet capacities or user to cloudlet assignments as a part of their problem formulation. Most importantly, in all existing studies, the movement of the individual cloudlets is not directly guided by changes in device positions and demands in their immediate surrounding. In our dynamic placement approach, the mobile cloudlets are aware of and respond to changes around them in real-time. There is no prediction involved, which reduces the run-time overhead. Similarly, sensitive long-term mobility data required for those predictions do not need to be acquired or processed by our approach.

#### 1.2.2.3 Distributed Placement Approaches

Among existing approaches, there are several approaches related to resource and server placement in distributed systems and networks, but very few approaches

that make placement decisions in a distributed manner. Enyioha *et al.* [20] proposed a distributed resource allocation strategy to control the spread of a process in a network similar to controlling the spread of a contagion. This is different from our problem with no spread but high mobility. Huang *et al.* [28] proposed a distributed resource allocation approach for network slicing in fog computing using similar ideas to the previous study. Our problem is again fundamentally different from network slicing.

Clustering-based approaches have been used to establish centers of placement of cloudlets in static placement approaches. Consequently, there are various distributed clustering approaches [8, 35] which can potentially be used to perform distributed cloudlet placement. Binder *et al.* [14] also provided a distributed approach to gravitational clustering in their work. However, mobile cloudlet placement has heterogeneous cloudlets and demands, which are not sufficiently addressed by clustering. Furthermore, clustering repeatedly over different time slots leads to the same issues we described for clustering-based dynamic placement approaches. When it comes to approaches specifically designed for mobile cloudlet placement, there is a lack of distributed placement approaches in the research literature. To the best of our knowledge, we will provide one of the first works to explore placement of mobile cloudlets in a distributed manner in this dissertation.

## 1.3 Organization

The remainder of this dissertation is structured as follows. In Chapter 2, we mathematically formulate the static cloudlet problem and introduce OCP, a detailed mathematical model for Optimal Cloudlet Placement. In Chapter 3, GACP, a meta-heuristic Genetic Algorithm-based Clouldet Placement approach for fast generalized heterogeneous cloudlet placements is presented. In Chapter 4, we further extend our study on static cloudlet placement to design, ACP, a bifactor approximation algorithm for cloudlet placements in highly heterogeneous scenario. Chapter 5 presents, DCP, a comprehensive system model and the mathematical formulation of the optimal dynamic mobile cloudlet placement problem. In Chapter 6, we present PMCP, an efficient

physics-inspired dynamic mobile cloudlet placement approach. Chapter 7 proposes an efficient and scalable distributed approach to dynamic mobile cloudlet placement problem. We finally conclude the dissertation in Chapter 8.

# Chapter 2

# THE STATIC CLOUDLET PLACEMENT PROBLEM

## 2.1 Background

The edge computing infrastructure is just taking shape. One of the most important aspects of building the infrastructure is to place resources to cater long-term user demands. There are studies on perennial placement of edge servers or cloudlets to address these needs. However, many existing studies consider the static placement from homogeneous resource perspective, indirect user connectivity, and with no flexibility to extend the idea to systems with high mobility. Hence, we begin by exploring the static placement of cloudlets in edge computing systems where heterogeneity, user connectivity, and the potential to handle mobility are the foremost priorities. In this chapter, we address this static cloudlet placement challenge by designing a generalized cost-aware cloudlet placement model. We formulate a general representation of the cloudlet placement problem considering a set of heterogeneous cloudlets and multiple user devices with the objectives to minimize the cloudlet placement cost and to ensure the placed cloudlets can cover all the users' requests guaranteeing minimum latency. Furthermore, our model also finds the full mapping of devices to the placed cloudlets. Finally, the design of the approach provides an elegant premise for handling mobility through extension of the model.

### 2.1.1 Static System Model

We aim to efficiently place mobile cloudlets to specific locations in a region to serve the demands of all the end (mobile) devices that require edge services. We model the region as a two-dimensional space (grid), where cloudlets and end devices can exist. The end 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. The candidate points are selected based on the load of user requests and the location of user demands over a long period. The cloudlet placement itself is not a long-term decision and is repeated over time.

The set of candidate points is defined as $P = \{\rho_1, \rho_2, \ldots, \rho_n\}$, and each refers to a location in the grid (in coordinate axes). A set of cloudlets is denoted by $C = (c_1, c_2, \ldots, c_w)$. The cloudlets are heterogeneous and each $c_j \in C$ is represented by a 4-tuple $c_j = \{\Pi_j, M_j, S_j, r_j\}$ denoting its attributes: VM specifications (processor capacity $\Pi_j$ (in GHz), RAM $M_j$ (in GB), storage capacity $S_j$ (in GB)) and coverage radius $r_j$ (Euclidean distance units).

A set of heterogeneous end devices requiring edge computing services is denoted by $E = (e_1, e_2, \ldots, e_v)$. Each $e_i \in E$ is represented by a 4-tuple $e_i = \{\pi_i, m_i, s_i, \lambda_i\}$ denoting its attributes, where $\pi_i$ is the processing demand (in GHz), $m_i$ is its memory demand (in GB), $s_i$ is its storage demand (in GB), $\lambda_i$ represents the location of the device (in coordinate axes). In addition, we define distance function $d(a, b)$ for calculating the Euclidean distance between points $a$ and $b$.

The incurred cost of placing cloudlet $c_j \in C$ at a candidate point $\rho_k \in P$ on the grid is defined by a cost function $\Phi(c_j, \rho_k)$ (simply, $\phi_{jk}$). The cost may include procurement cost, space (rented, public) cost, mobility cost, and maintenance costs.

End devices may experience delays in receiving their computing services based on the availability of bandwidth and distance metrics [26]. To capture that, we define a latency function $\mathrm{L}(e_i, \rho_k)$ (simply, $l_{ik}$) that represents the latency when end device $e_i \in E$ is serviced from a cloudlet placed at candidate point $\rho_k \in P$ of the grid.

Our goal is to minimize the cost of deploying cloudlets in the region and simultaneously minimize the latency in accessing edge services, while providing the services to all end devices (full coverage). This is a bicriteria optimization problem and computationally NP-hard [71]. We next provide the mathematical optimization model of this problem.

## 2.2 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:

$$\begin{cases} \boldsymbol{\Phi} = \min \sum_{j=1}^{w} \sum_{k=1}^{n} \phi_{jk}\, y_{jk} \\ \boldsymbol{L} = \min \sum_{i=1}^{v} \sum_{k=1}^{n} l_{ik}\, a_{ik} \end{cases} \tag{2.1}$$

Subject to:

$$\sum_{j=1}^{w} \sum_{k=1}^{n} y_{jk} \leq |\mathcal{C}| \tag{2.2}$$

$$d(\lambda_i, \rho_k) a_{ik} \leq \sum_{j=1}^{w} r_j y_{jk} \qquad \forall e_i \in \mathcal{E}, \rho_k \in \mathcal{P} \tag{2.3}$$

$$\sum_{i=1}^{v} m_i a_{ik} \leq \sum_{j=1}^{w} M_j y_{jk} \qquad \forall \rho_k \in \mathcal{P} \tag{2.4}$$

$$\sum_{i=1}^{v} s_i a_{ik} \leq \sum_{j=1}^{w} S_j y_{jk} \qquad \forall \rho_k \in \mathcal{P} \tag{2.5}$$

$$\sum_{i=1}^{v} \pi_i a_{ik} \leq \sum_{j=1}^{w} \Pi_j y_{jk} \qquad \forall \rho_k \in \mathcal{P} \tag{2.6}$$

$$a_{ik} \leq \sum_{j=1}^{w} y_{jk} \qquad \forall e_i \in \mathcal{E}, \rho_k \in \mathcal{P} \tag{2.7}$$

$$\sum_{j=1}^{w} y_{jk} \leq 1 \qquad \forall \rho_k \in \mathcal{P} \tag{2.8}$$

$$\sum_{k=1}^{n} y_{jk} \leq 1 \qquad \forall c_j \in \mathcal{C} \tag{2.9}$$

$$\sum_{k=1}^{n} a_{ik} = 1 \qquad \forall e_i \in \mathcal{E} \tag{2.10}$$

$$y_{jk} \in \{0, 1\} \qquad \forall c_j \in \mathcal{C}, \rho_k \in \mathcal{P} \tag{2.11}$$

$$a_{ik} \in \{0, 1\} \qquad \forall e_i \in \mathcal{E}, \rho_k \in \mathcal{P} \tag{2.12}$$

The objective functions shown in Eq (2.1) minimize the total cost of placing the cloudlets and the total latency suffered by the devices. Constraint (2.2) ensures that the total number of cloudlets placed in the grid does not exceed the number of available cloudlets. Constraints (2.3) guarantee that each device must be within the coverage range of some cloudlet. Constraints (2.4-2.6) satisfy supply and demand in terms of memory, storage, and processing requirements. Constraints (2.7) guarantee that a device can be served from a candidate point only if at least one cloudlet is placed there. Constraints (2.8) ensure that at most one cloudlet is placed at any candidate point. Constraints (2.9) ensure that a cloudlet can only be placed at a single candidate point. Constraints (2.10) guarantee that all devices must be served, and each is served from exactly one candidate point. Finally, constraints (2.11-2.12) 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. Next, using an example we show the $\epsilon$-constraint method and its limitations in solving our multi-objective problem.

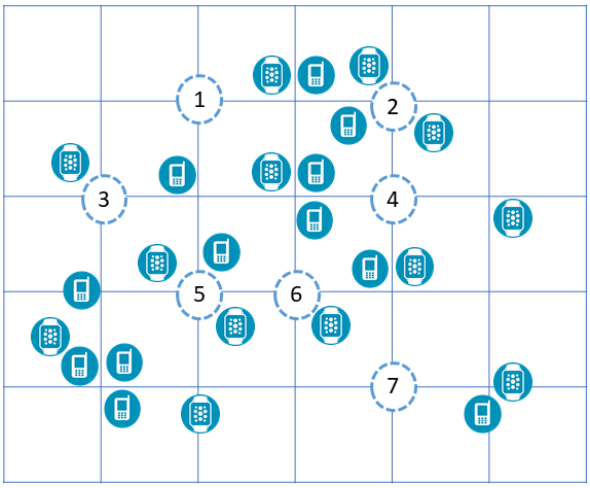



Figure 2.1: An OCP Cloudlet Placement Scenario

### 2.2.1 Optimization Example

Figure 2.1 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 Table 2.1 and Table 2.2, respectively.

Table 2.1: OCP Example: Specifications of Cloudlets

| **Type** | $\Pi_j$ | $M_j$ | $S_j$ | $r_j$ | $\phi_{jk}$ |
|---|---|---|---|---|---|
| Large | 20 | 20 | 20 | 3 | 63 |
| Medium | 10 | 10 | 10 | 2 | 32 |
| Small | 5 | 5 | 5 | 1 | 16 |

Table 2.2: OCP Example: Specifications of Devices

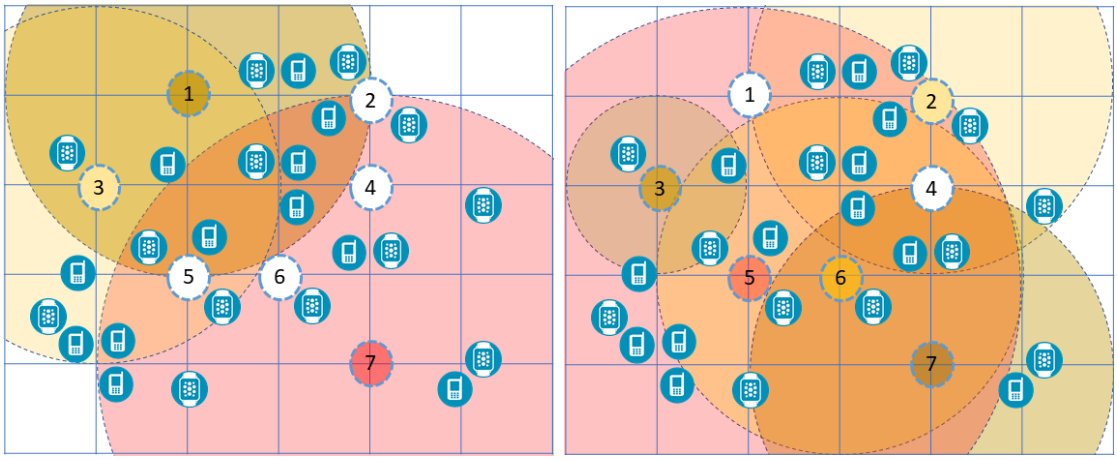
| **Type** | $\pi_i$ | $m_i$ | $s_i$ |
|---|---|---|---|
| High-Demand (Cellphone) | 2 | 2 | 2 |
| Low-Demand (Smartwatch) | 1 | 1 | 1 |

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 $\epsilon$-constraint method for solving our multi-objective problem. In the $\epsilon$-constraint method, we optimize one of the objective functions and use the other objective functions as constraints [50]. We use this approach to find a set of Pareto-optimal solutions, where none of the objective functions can be improved in value without degrading the other objective value.

#### 2.2.1.1 Minimizing Cost (OCP-Cost)

Figure 2.2a 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).




(a) OCP: Optimal Cost Placement (b) OCP: Optimal Latency Placement

Figure 2.2: Single-objective Optimal Cloudlet Placement Solutions

#### 2.2.1.2 Minimizing Latency (OCP-Latency)

Figure 2.2b 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.

#### 2.2.1.3 Cost-Latency Tradeoff

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

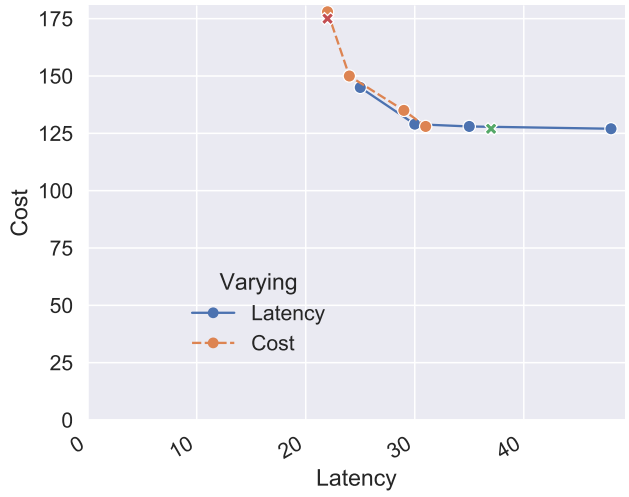



Figure 2.3: OCP: Cost versus Latency Trade-off

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 Figure 2.3. 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. This is why we need to design efficient approaches to solve this problem based on different real-life scenarios.

## 2.3 Chapter Summary

In this chapter, we mathematically formulated a general system model considering heterogeneous cloudlets and user devices which allowed us to optimize cost and latency while ensuring all devices are serviced. However, it is not possible to optimize both objectives simultaneously, and an inherent trade-off exists between the objectives. As a result, we propose efficient approaches to solve this problem in the next two chapters.

# Chapter 3

# GENETIC ALGORITHM-BASED CLOUDLET PLACEMENT

## 3.1 Background

Deploying cloudlets is costly and may not be feasible in many situations (e.g., disaster situations, emergency rescue, unexpected surge in user demand) and regions with sparse or no infrastructure of wireless access points such as remote rural areas [19]. For these situations, an emerging method called Vehicular Edge Computing (VEC) has been introduced, where smart vehicles (SV) such as Unmanned Aerial Vehicles (UAVs) and Connected and Autonomous Vehicles (CAVs) are considered as computational cloudlets due to their inherent attributes such as mobility, low operating costs, flexible deployment, and wireless communication ability [44, 84]. SV-mounted cloudlets can augment edge computing capacity, enhance its coverage, improve latency by executing the offloaded applications. While VEC can bring many opportunities, efficiently placing cloudlets brings new research challenges as it is a dynamic decision to balance the load and improve latency.

The cloudlet placement problem is known to be NP-hard [71], and an efficient way to solve large-scale placement problems is by using heuristic and meta-heuristic approaches [25]. In particular, Genetic Algorithm (GA) is a discrete technique that is suitable for combinatorial problems such as grouping, ordering, and assignments. This meta-heuristic approach uses population-based search by relying on bio-inspired operations such as mutation, crossover, and selection [54]. With a well-designed fitness function, diverse initial population, and suitable termination, a genetic algorithm is known to find high-quality solutions in a short amount of time. The design of genetic operations is also flexible and allows parameter tuning to find better results. Moreover,

since covering the end devices is crucial in our problem, GA is a suitable approach that can lead to high coverage [24].

In this chapter, we propose a GA-based cloudlet placement approach, which allows a smooth trade-off between the objectives to obtain close to optimal solutions at a markedly lower running time. We ensure that the devices are covered by the placed cloudlets considering the cloudlet capacities and latency requirements of the applications.

We perform extensive experiments to show the effectiveness of our proposed approach in finding close to optimal cloudlet placements in different deployment scenarios and different degrees of freedom in placing them. Our experiments are designed based on real data containing WiFi hotspot locations and usage statistics obtained from NYC OpenData [17].

### 3.2 The GACP Algorithm

In this section, we propose a novel Genetic Algorithm-based method, called GACP, to find a *close to optimal* solution for the cloudlet placement problem.

GACP, given in Algorithm 1, receives the input parameters $C$, $E$, $P$, $\Phi$, $L$, $\delta$, $R$, $X_p$, $M_p$, where $\delta$ is a device coverage threshold, $R$ represents the initial size of the placement set, $X_p$ is a crossover probability, and $M_p$ is a mutation probability. GACP uses an estimated value of cost ($\widehat{OPT}$) and initial number of cloudlets needed ($min_c$) by calling the LP relaxation of OCP-Cost, `LP-OCP-Cost()` (line 2). This is a pre-processing step to obtain a lower-bound on cost and initial number of cloudlets in polynomial time to be used in the fitness function and to produce initial placements, respectively. GACP creates a set of random cloudlet placements to the candidate points, called *Cloudlet Candidate Placement (CCP)* (line 3). The goal of GACP is to improve this set iteratively (i.e., generation of genetic operations) to find a better solution. To keep track of intermediate solutions, GACP uses an initially empty set $B$ representing the *Best Placement Set* obtained at the end of each iteration. The iterations continue until GACP finds at least $R$ assignments in $B$ from which the best one is selected.

**Algorithm 1** GACP

1: **Input:** $C$, $E$, $P$, $\Phi$, $L$, $\delta$, $R$, $X_p$, $M_p$
2: $(\widehat{OPT}, min_c) \leftarrow$ `LP-OCP-Cost()`
3: CCP $\leftarrow$ `RandPlace(`$R, min_c$`)`
4: DCA $\leftarrow$ `InitDeviceAssign()`
5: **repeat**
6: $\quad\mathcal{Q} \leftarrow$ PriorityQueue(CCP)
7: $\quad B = \emptyset$
8: $\quad$**while** $|B| \leq R$ **do**
9: $\quad\quad y_1 \leftarrow \mathcal{Q}$.pop()
10: $\quad\quad y_2 \leftarrow \mathcal{Q}$.pop()
11: $\quad\quad$**if** $(rand(0,1) <= X_p)$ **then**
12: $\quad\quad\quad O_1, O_2 \leftarrow$ `Crossover`$(y_1, y_2)$
13: $\quad\quad$**else**
14: $\quad\quad\quad O_1, O_2 \leftarrow y_1, y_2$
15: $\quad\quad O_1 \leftarrow$ `Mutate`$(O_1)$
16: $\quad\quad O_2 \leftarrow$ `Mutate`$(O_2)$
17: $\quad\quad F_P = \min($`Fitness`$(y_1),$ (`Fitness`$(y_2))$
18: $\quad\quad F_O = \min($`Fitness`$(O_1),$ (`Fitness`$(O_2))$
19: $\quad\quad V_P = \min($`Coverage`$(y_1),$ `Coverage`$(y_2))$
20: $\quad\quad V_O = \min($`Coverage`$(O_1),$ `Coverage`$(O_2))$
21: $\quad\quad$**if** $(F_O < F_P)$ or $(F_O = F_P$ and $V_O \geq V_P)$**then**
22: $\quad\quad\quad$**for** $O \in \{O_1, O_2\}$ **do**
23: $\quad\quad\quad\quad$**if** `Coverage`$(O) \geq \delta$ **then**
24: $\quad\quad\quad\quad\quad B \leftarrow B \cup \{O\}$
25: $\quad\quad$**else**
26: $\quad\quad\quad$**for** $y \in \{y_1, y_2\}$ **do**
27: $\quad\quad\quad\quad$**if** `Coverage`$(y) \geq \delta$ **then**
28: $\quad\quad\quad\quad\quad B \leftarrow B \cup \{y\}$
29: $\quad$CCP $\leftarrow B$
30: $\quad Y^* \leftarrow$ `SelectLeastCost`(CCP)
31: $\quad$DCA_NEW $\leftarrow$ `DeviceAssign()`
32: **until** DCA_NEW != $\emptyset$
33: **Output:** $Y^*$, DCA_NEW

GACP finds an initial assignment of the devices to the candidate points, called *Device Candidate Assignment (DCA)* (line 4). In doing so, each device $e_i$ is initially assigned to its closest candidate point to receive a computing service, $\min d(e_i, \rho_k), \forall \rho_k \in P$. This approach helps in minimizing latency and also in providing a consistent benchmark for calculating device coverage for a cloudlet placement.

**Algorithm 2** Crossover($y_1, y_2$)

1: mid_point = $\lceil |y_1|/2 \rceil$
2: **for** $i$ = mid_point **to** $|y_1|$ **do**
3: swap($y_1[i], y_2[i]$)
4: `Validate`($y_1$)
5: `Validate`($y_2$)
6: **return** $y_1, y_2$

In each iteration, GACP selects two least cost cloudlet placements from the priority queue ($\mathcal{Q}$) as parent placements (lines 9-10). GACP then crossovers these placements by probability $X_p$ and calling `Crossover`() function (line 12, detailed in Algorithm 2), to obtain their two offspring placements. `Crossover`() function also validates the two offspring using function `Validate`() to ensure that the offspring placements satisfy the requirements (do exceed the available number of cloudlets of each type). If an assignment is not valid, extra cloudlets are removed randomly. Then, GACP mutates the two offspring placements (lines 15-16, detailed in Algorithm 3). Generally, crossovers happen more frequently and mutations very rarely.

To obtain lower placement cost, our fitness function is defined to select a placement that has less difference with the optimal placement cost (minimum optimality gap). Since the optimal cost is not known due to NP-hardness, GACP uses $\widehat{OPT}$. The `Fitness`() function for a cloudlet placement $y$ is calculated as (detailed in Algorithm 4):

$$\texttt{Fitness}(y) = |\widehat{OPT} - \text{PLACEMENT\_COST}(y)| \tag{3.1}$$

Once the fitness values are calculated (lines 17-18), GACP finds the device coverage of parent placements and offspring placements (lines 19-20). The `coverage`()

**Algorithm 3** Mutate($y$)

1: **for** $c_i \in y$ **do**
2: **if** ($rand(0,1) <= M_p$) **then**
3: $c_i \leftarrow$ int_rand(0, $|C|$)
4: `Validate`($y$)
5: **return** $y$

**Algorithm 4** Fitness($y$)

1: $f_y = 0$
2: **for** k = 1 **to** $|y|$ **do**
3: $\quad c_j = y[k]$
4: $\quad f_y \mathrel{+}= \phi_{jk}$
5: $f_y = |f_y - \widehat{OPT}|$
6: **return** $f_y$

function (given in Algorithm 5) uses the device assignments and the current cloudlet placement considering memory, processing, storage, and coverage radius constraints (by calling function `InRange&Capacity`()) to check whether a device can be served (covered) by a cloudlet. This function returns a fraction of covered devices by a cloudlet placement $y$:

$$\texttt{Coverage}(y) = \frac{\text{DEVICES\_COVERED}(y)}{|E|} \tag{3.2}$$

If the offspring have strictly better fitness, or equal fitness with better device coverage than their parents, GACP adds the offspring to $B$ (lines 21-24). Otherwise, the parents are added (lines 26-28). Please note that the offspring and parents are only added to $B$ if they satisfy a certain coverage (at least the specified threshold, $\delta$); otherwise, GACP simply moves to the next iteration.

At the end of a generation (line 29), GACP adds the new generation of the best placements to CCP. The algorithm chooses the least cost placement ($Y^*$) and finds the actual device assignments (lines 31-32). `DeviceAssign()` is a fail-fast device assignment function to validate the full coverage of the obtained solution. If the function does not return a solution (line 32), GACP repeats the procedure to obtain a new generation of best assignments with $B$ from the last generation is now regarded as CCP. At the end, GACP outputs the final cloudlet placement and the device assignment.

### 3.2.1 GACP: Example

In this section, we present and describe an example which encompasses one full generation of genetic operations in GACP including but not limited to the key ideas such as

**Algorithm 5** Coverage($y$)

```
1: covered = 0
2: for i = 1 to |E| do
3:     ρ_k = DCA[i]
4:     c_j = y[ρ_k]
5:     if (c_j = 0) then      /*no cloudlet is placed at ρ_k*/
6:         min_dist = ∞
7:         for k = 1 to |P| do
8:             c_k = y[ρ_k]
9:             if (c_k != 0 &&
10:                InRange&Capacity(e_i, c_k)) then
11:                if (d(e_i, ρ_k) < min_dist) then
12:                    min_dist ← d(e_i, ρ_k)
13:                    index ← k
14:        if min_dist ≠ ∞ then
15:            covered += 1
16:            Π_index -= π_i
17:            M_index -= m_i
18:            S_index -= s_i
19:    else
20:        if InRange&Capacity(e_i, c_j)) then
21:            covered += 1
22:            Π_j -= π_i
23:            M_j -= m_i
24:            S_j -= s_i
25: return covered/|E|
```

selection, crossover, mutation, and fitness. This example is based on the same scenario as presented in Section 2.2.1.

Let $c_1$, $c_2$, and $c_3$ represent different types of cloudlets. This is not to be confused with cloudlet at index $j$ $c_j$. So, for simplicity, we assume $c_1$ as a large cloudlet, $c_2$ as a medium, and $c_3$ as a small cloudlet. Assignments will be a vector showing the cloudlets placed in different candidate points. Having 0 in any index of the vector represents that the candidate point is empty (i.e., no cloudlet is placed there).

#### 3.2.1.1 Initial Random Assignments

This is first step of our genetic algorithm. In this step all available cloudlets are randomly placed across the candidate points as shown in Figure 3.1. GACP will consistently add/remove/validate cloudlets to optimize our objectives by improving fitness of the assignments at each generation. For this example we set the size of the random assignment set to 5 ($R = 5$). Thus, CCP set has 5 different cloudlet assignment vectors and $B$, Best Assignments set, is empty as shown below:

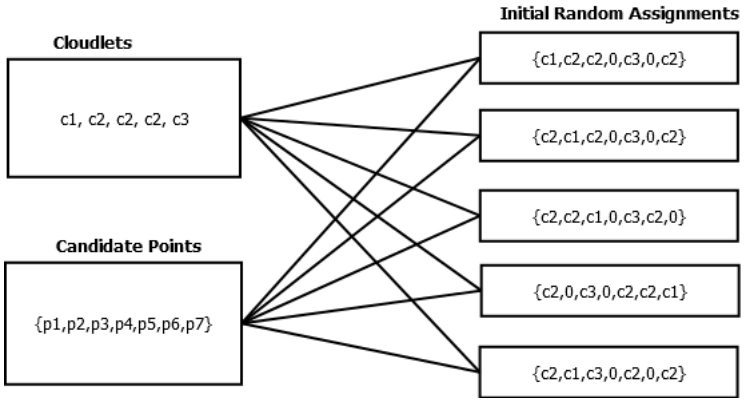



Figure 3.1: GACP: Initial Random Cloudlet Assignments.

$$\begin{aligned}\text{CCP} \leftarrow\ & [\{c_1, c_2, c_2, 0, c_3, 0, c_2\},\\ & \{c_2, c_1, c_2, 0, c_3, 0, c_2\},\\ & \{c_2, c_2, c_1, 0, c_3, c_2, 0\},\\ & \{c_2, 0, c_3, 0, c_2, c_2, c_1\},\\ & \{c_2, c_1, c_3, 0, c_2, 0, c_2\}]\end{aligned}$$

$$\text{B} = \emptyset$$

Note that we keep the initial set intact and use a priority queue, $\mathcal{Q}$, for efficient selection of cloudlets. The costs of the 5 random assignments, in order of their appearance in the initial set, are [14, 14, 14, 13, 14]. The priority queue will be ordered like below:

$$\mathcal{Q} \leftarrow\ [\{c_2, 0, c_3, 0, c_2, c_2, c_1\}(13),$$

$\{c_1, c_2, c_2, 0, c_3, 0, c_2\}(14)$,
$\{c_2, c_1, c_2, 0, c_3, 0, c_2\}(14)$,
$\{c_2, c_2, c_1, 0, c_3, c_2, 0\}(14)$,
$\{c_2, c_1, c_3, 0, c_2, 0, c_2\}(14)]$

#### 3.2.1.2 Device Assignments

Essential part of our problem is not only to place the cloudlets but also to assign each end device to be served by one of those cloudlets satisfying their quality of services. For an initial device assignments, we ignore the cloudlets and simply focus on the locations of the candidate points. To be precise, every device is assigned to the closest candidate point. The intuition here is to begin with the least possible latency. Many of these devices will be naturally covered by randomly placed cloudlets, and we use this minimal latency scenario to calculate coverage of the cloudlet assignments. Only later in the process, any uncovered devices is mapped to the cloudlets by trading off the coverage with latency. In this example, the initial device assignments for our 25 end devices spread across the grid looks like below:

DCA $\leftarrow$ [1, 1, 2, 2, 2, 3, 1, 4, 4, 4, 4, 3, 5,
5, 5, 6, 4, 4, 5, 5, 5, 5, 5, 7, 7]

#### 3.2.1.3 Selection

GACP continues with two assignments with the least total placement costs (lines 11-12).
$y_1 \leftarrow \mathcal{Q}$.pop()
$y_2 \leftarrow \mathcal{Q}$.pop()

In case of a tie, one that appears first in the priority queue set is selected. Hence, below are the two cloudlet assignments selected from the priority queue.

$y_1 = \{c_2, 0, c_3, 0, c_2, c_2, c_1\}$

$y_2 = \{c_1, c_2, c_2, 0, c_3, 0, c_2\}$

Then, the priority queue will have three elements left, two of which are chosen in the next round.

$\mathcal{Q} = [\{c_2, c_1, c_2, 0, c_3, 0, c_2\}(14),$
$\{c_2, c_2, c_1, 0, c_3, c_2, 0\}(14),$
$\{c_2, c_1, c_3, 0, c_2, 0, c_2\}(14)]$

#### 3.2.1.4 Crossover

Here, we use a single-point crossover. GACP crossovers the cloudlet assignments at their midpoint such that second halves of both the assignments are swapped as shown in Figure 3.2.

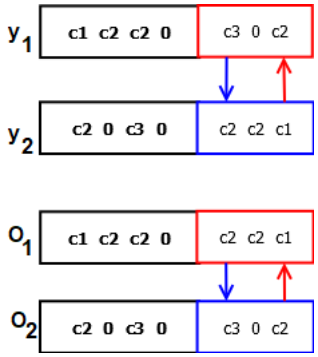



Figure 3.2: GACP: Crossover Operation

The offspring are validated after crossover (checking number of cloudlets of each size), and the offspring are as follows:

$O_1 = \{c_2, 0, 0, 0, c_3, 0, c_2\}$

$O_2 = \{0, 0, c_2, 0, c_2, c_2, c_1\}$

#### 3.2.1.5 Mutate

The Mutate procedure randomly changes cloudlet types at candidate points in an assignment so that it represents a different assignment set. It may also stay the same since mutation probability is generally low. The mutation changes are underlined below.

$O_1 = \{c_2, 0, 0, 0, c_3, 0, c_2\}$
$O_1 \leftarrow \text{Mutate}(O_1) = \{c_2, \underline{c_1}, 0, 0, c_3, 0, c_2\}$

$O_2 = \{0, 0, c_2, 0, c_2, c_2, c_1\}$
$O_2 \leftarrow \text{Mutate}(O_2) = \{0, 0, \underline{c_3}, 0, \underline{0}, c_2, c_1\}$

No validation is required in this case because the number of cloudlets are consistent after the mutation operation.

#### 3.2.1.6 Fitness

The fitness function takes an assignment set as an input and returns its fitness value based on the estimated optimal cost. It is worth noting that the estimated optimal cost is when we place the least number of cloudlets to meet all demands of the devices. In this example we have estimated optimal cost, $\widehat{OPT} = 9$. Our assignments should naturally have some deviation from this estimated optimal cost.

$F_P = \min(\text{fitness}(y_1), (\text{fitness}(y_2)) = \min(4, 5) = 4$
$F_O = \min(\text{fitness}(O_1), (\text{fitness}(O_2)) = \min(2, 1) = 1$

$V_P = \min(\text{coverage}(y_1), \text{coverage}(y_2))$
$\quad = \min(0.64, 0.52) = 0.52$
$V_O = \min(\text{coverage}(O_1), \text{coverage}(O_2))$

$= \min(0.44, 0.24) = 0.24$

In case of a tie, one with higher coverage is chosen.

#### 3.2.1.7 Comparison

Since $F_O < F_P$, the offspring are considered for the best set $B$. The next check is to compare their coverage values with threshold $\delta = 0.2$. Now that the coverage values of both $O_1$ and $O_2$ are more than $\delta$, we add them both to $B$.

$B \leftarrow B \cup \{O_1, O_2\}$

$B = [\ \{c_2, c_1, 0, 0, c_3, 0, c_2\}, \{0, 0, c_3, 0, 0, c_2, c_1\}\ ]$

*Note: The coverage value $\delta$ is used as 0.2 only for illustration since we want to show how the best set is created. The actual values used are quite high (e.g., 0.95) since we want to maximize coverage.*

#### 3.2.1.8 Repeat

GACP continues the loop by picking up the next two least cost assignments from the priority queue and performing the same operations with the chosen assignments until the set of best assignments $B$ has at least $|R|$ assignments satisfying the coverage threshold $\delta$. Then, GACP checks if all assignments in $B$ are under the desired coverage threshold $\delta$. If they are not, the best assignments set $B$ becomes the new Cloudlet Candidate Assignments (CCP) set, and we run a new generation of genetic operations. Once all of them are under the desired coverage threshold, we select the assignment with the least cost $Y^*$ as the solution. In this case, $Y^* = \{0, c_3, 0, 0, c_1, c_2, c_2\}$.

### 3.2.2 Convergence of GACP

Since the LP relaxation `LP-OCP-Cost()` gives the lower bound on the optimal cost (OCP-Cost) and it is used as the estimated optimal cost $\widehat{OPT}$ in the fitness function of GACP, cloudlet placements that are closer to the optimal cost placement are favored by GACP. Thus, GACP keeps producing cloudlet placements closer to the

optimal cost in every iteration. As for the device assignments, GACP only keeps the intermediate solutions that meet the coverage criteria. These greedy device assignments are likely to result in full coverage (i.e., all devices are assigned to the placed cloudlets) when a high coverage value (e.g., $\delta = 0.95$) is chosen.

Moreover, every cloudlet placement is validated and the entire genetic improvement process is repeated until full coverage is achieved in final device assignments (line 5-32). If its repeated, the best assignments from the last cycle $B$ are retained and used as the new $CCP$. Since close to optimal costs with high greedy coverage were already achieved in the last cycle, genetic operations in the new cycle will prefer better coverage values without increasing the cost. So, GACP converges fast to a feasible solution with high probability.

## 3.3 Experimental Results

### 3.3.1 Experimental Setup

We perform a set of experiments to investigate the effectiveness of our proposed approach. For our deployment scenario, we use parts of New York City as it has been selected by National Science Foundation (NSF) as a testbed for the new wave of mobile technology [18]. Also, the presence of NYC Open Data [17] allows us to access up-to-date information about implemented hotspot locations and usage statistics. The primary datasets that we utilize for our experiments are: NYC WiFi Hotspot Locations and LinkNYC Usage Statistics. The hotspots can be seen in the map shown in Figure 3.3.

To setup the experiments, we select two neighborhoods in Upper Manhattan: Central Harlem and East Harlem because of their reasonable size and sufficient hotspot locations. The unique hotspot locations in these neighborhoods are 183 and 81, respectively.

Next, we reduce the overall map of these neighborhoods into a 2D grid based on geolocations or coordinates of the hotspots. The reduction preserves the ratio of distance between two points in the grid with actual distance. Out of the available

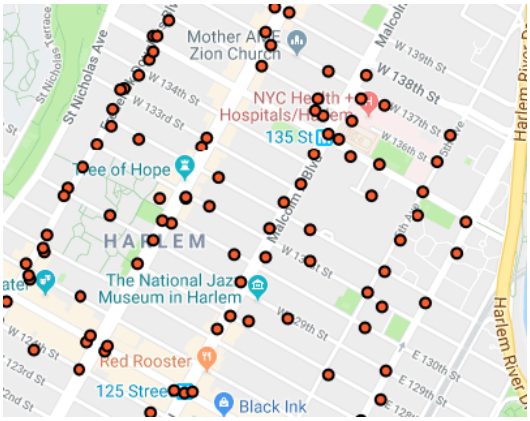



Figure 3.3: WiFi Hotspots in Central Harlem (NYC Open Data).

hotspot locations, we select only a certain percentage as the suitable candidate points. As pointed out in Chapter 2, the candidate points are selected based on the distribution of the users (load), feasibility of placing cloudlets, and other appropriate criteria. The number of devices are selected based on the population data of the neighborhoods and usage statistics, and the devices are placed across the grid based on a uniform distribution.

We finally run three main types of experiments: investigating cost, latency, and running time. We also perform sensitivity analysis on the number of candidate points by choosing different percentages of candidate points, i.e., 10%, 15%, and 20% of hotspot locations on Central Harlem data. The number of cloudlets available for deployment and the number of devices stay the same during the sensitivity analysis.

The optimal results from OCP are found using IBM ILOG CPLEX Concert Technology API for Java [29]. GACP is implemented in the same version of Java and the experiments for both are run on the same JVM on a standalone workstation with Intel Core i7 processor @2.70 GHz and 16 GB of RAM. Please note that we could not compare our results with other approaches because existing studies either do not have the same objectives or have different constraints. 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 a proper benchmark. However, it is not able to obtain any results as the problem size increases due to NP-harness of the problem.

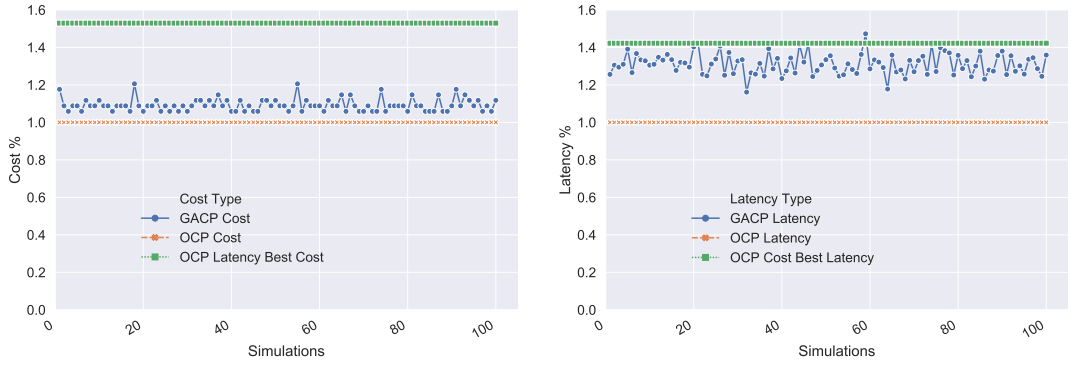
### 3.3.2 Analysis of Results

We first show the results of GACP, OCP-Latency, OCP-Cost in terms of obtained cost, latency, and running time for Central Harlem Data with 10% candidate points. This region has 18 candidate points, 343 devices, and up to 14 cloudlets. Since GACP is meta-heuristic, we run the experiments 100 times to see the distribution of costs and latency. The results are summarized in Figure 3.4.

Figure 3.4a shows that the costs obtained by GACP are very close to the optimal costs (OCP-Cost). The GACP costs on average only 8.8% more than OCP-Cost. Figure 3.4b shows a very effective cost-latency trade-off obtained by GACP as the average GACP Latency is 7.8% less than the OCP-Cost Best Latency.

We capture the running times for 100 runs of GACP, OCP-Cost, and OCP-Latency. The results are shown in Figure 3.5. The GACP running time is significantly less than both optimal approaches. Please note that the extreme spikes in the execution time of the optimal approaches are due to NP-hardness of the problem.

Next, we run experiments for Central Harlem Data to perform sensitivity analysis on different percentages of candidate points and analyze their impacts on cost and latency. As Figure 3.6a shows, the effect on costs is negligible when the number of candidate points increases. This is due to the fact that the same number of cloudlets are able to meet user demands. However, Figure 3.6b shows that spreading the candidate




(a) Cost (b) Latency

Figure 3.4: GACP: Experimental Results for Central Harlem Data (10% Candidates)

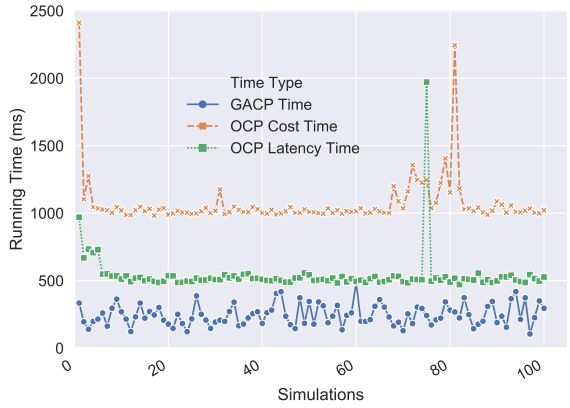
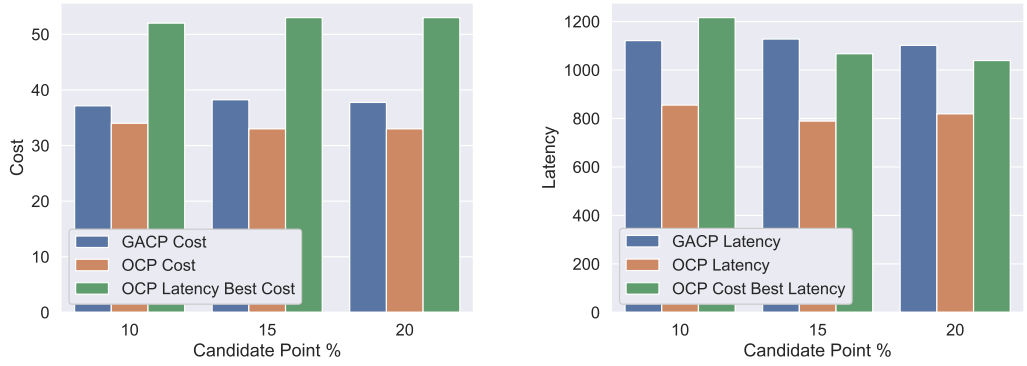



Figure 3.5: GACP: Central Harlem: Running Time

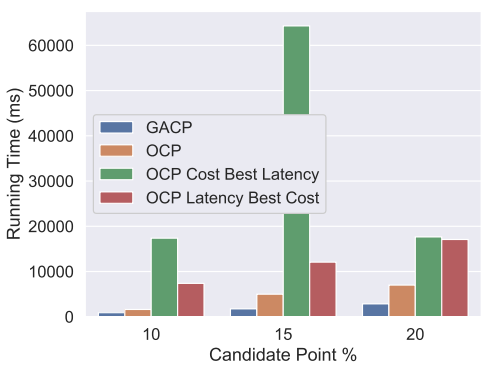



(a) Cost (b) Latency

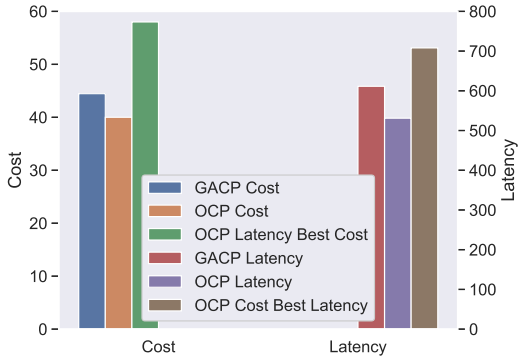
Figure 3.6: GACP: Sensitivity Analysis on the Percentage of Candidate Points

points makes GACP suffer in terms of latency. This is because larger distribution of candidate points makes it harder to achieve low latency at lower costs. Nonetheless, GACP has slight improvement in 20% case, which provides an important insight that designating better candidate points is more important than having more of them. Figure 3.7a shows the running time. While GACP looks linear, OCP-Cost suffers from a slow running time. For instance, for 15% candidate points case, solving OCP-Cost Best Latency takes more than a minute.

Finally, we run experiments on East Harlem data to investigate cost and latency changes with a different size and dimension. Since the populations of Central Harlem and East Harlem are similar, we keep the same number of devices and cloudlets as




(a) Running Time




(b) East Harlem: Cost and Latency

Figure 3.7: GACP: Sensitivity Analysis of Central Harlem Running Time, and East Harlem Cost and Latency Comparisons

in the first experiment. However, at 20% candidate point selection, we have just 16 candidate points. Nonetheless, the results (Figure 3.7b) show that both average GACP cost and average GACP latency are perfectly poised between the optimal value and the best value for the other objective. Since GACP uses cost for fitness, the cost is much closer to the optimal value than latency.

The above results sufficiently demonstrate that GACP is an efficient cost-aware cloudlet placement approach for vehicular edge computing satisfying the desired requirements.

## 3.4 Chapter Summary

Latency suffered by the users can be mitigated by placing resource-rich mobile cloudlets closer to the network edge. However, strategically placing the cloudlets to both reduce costs and ensure all users are covered within their latency requirements is a major challenge. Hence, we proposed a genetic algorithm-based cloudlet placement approach, GACP, to obtain close to optimal solutions with fast execution time. The experimental results based on real data showed that our proposed approach is able to find close to optimal cost placements by trading-off latency in different scenarios.

Early version of this work was accepted as an extended abstract and selected for

poster presentation at the 2019 ACM/IEEE Symposium on Edge Computing (SEC). The resulting research poster was selected and presented as a part of ACM Student Research Competition at the 2019 Supercomputing Conference (SC'19). The completed research paper [11] was accepted and presented at the 2019 IEEE CloudCom where it received the "Best Paper Award". In the next chapter, we design an efficient approximation approach for the cloudlet placement problem..

# Chapter 4

# A BIFACTOR APPROXIMATION ALGORITHM FOR CLOUDLET PLACEMENT

## 4.1 Background

In this chapter, we address the cloudlet 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.

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 [73]. 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

Table 4.1: Comparison of ACP with existing research

| Study | Cost Obj. | Latency Obj. | Het. Cloudlets | Het. Devices | Capacity Const. | Real Dataset | Full Coverage | Approximation |
|---|---|---|---|---|---|---|---|---|
| Kang *et al.* [33] | | ✓ | | | ✓ | | | |
| Zhang *et al.* [82] | | ✓ | | | | | | |
| Zeng *et al.* [81] | ✓ | | | | ✓ | | | |
| Xu *et al.* [77] | | ✓ | | ✓ | ✓ | | | ✓ |
| Li *et al.* [40] | | | | | | ✓ | | |
| Yao *et al.* [78] | ✓ | | ✓ | | ✓ | | | |
| Wang *et al.* [72] | ✓ | ✓ | | | | | | |
| Wang *et al.* [69] | ✓ | | | | | | | |
| Wang *et al.* [71] | | ✓ | | | ✓ | ✓ | | |
| Guo *et al.* [27] | | ✓ | | | ✓ | ✓ | | |
| Ma *et al.* [43] | | ✓ | | | | | | |
| Meng *et al.* [51] | ✓ | ✓ | | ✓ | ✓ | | | ✓ |
| **ACP Approach** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

and independently analyze the worst-case scenario for each objective instead of the overall approximation of their linear combination.

As summarized in Table 4.1, none of existing 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, 40, 71], transportation network data: [31]). Finally, none of these studies consider full mapping of individual users or devices to cloudlets.

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 [17], provided by NSF COSMOS [1]. The results show that ACP efficiently finds near-optimal solutions in these real world-based scenarios.

### 4.2 Bifactor Approximation of Cloudlet Placement

We design a bifactor approximation algorithm for the cloudlet placement problem as shown in Algorithm 6. As we design the algorithm, there is an important assumption: we assume that the cost function is linearly correlated with the cloudlet capacities and coverage radius. 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 `LP-OCP-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 (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 6** ACP: Approximate Cloudlet Placement

**Input:** $\mathcal{C}$, $\mathcal{E}$, $\mathcal{P}$, $\Phi$, $L$
$(y^*_{jk}, a^*_{ik}) \leftarrow$ `LP-OCP-Cost()`
$A_{\mathcal{C}} = \emptyset$ ▷ cloudlet to candidate mapping
$A_{\mathcal{E}} = \emptyset$ ▷ device to candidate mapping
*/* Filtering */*
**for** $e_i \in \mathcal{E}$ **do**
    $D_i = \sum_{\forall \rho_k \in \mathcal{P}} l_{ik}\, a^*_{ik}$ ▷ Fractional total latency
    $N_i \leftarrow$ All $\rho_k \in \mathcal{P}$ with $a^*_{ik} > 0$ within $2D_i$ of $e_i$
$S = \emptyset$ ▷ Set of skipped devices
$T = \mathcal{E}$ ▷ Temporary set of devices
*/* Rounding */*
**while** $|T| > 0$ **do**
    $e_i \leftarrow$ `MaxD(`$T$`)` ▷ Device with the largest $D_i$
    $\rho_i \leftarrow$ `BestCandidate(`$e_i$`)` ▷ Closest $\rho_k \in N_i$ to $e_i$
    $c_i \leftarrow$ `BestCloudlet(`$e_i, \rho_i$`)` ▷ Selected cloudlet
    **if** $c_i = \emptyset$ **then**
        $S = S \cup \{e_i\}$
    **else**
        $A_{\mathcal{E}} = A_{\mathcal{E}} \cup \{(e_i, \rho_i)\}$
        $A_{\mathcal{C}} = A_{\mathcal{C}} \cup \{(c_i, \rho_i)\}$
        Adjust remaining capacities of $c_i$
        $E_i \leftarrow$ All $e_v \in \mathcal{E}$ with $\rho_i \in N_v$ ▷ Extended set
        **while** $|E_i| > 0$ **do**
            $e_v \leftarrow$ `MaxD(`$E_i$`)` ▷ Device with the largest $D_v$
            **if** `RangeCap(`$e_v, c_i$`)` **then**
                $A_{\mathcal{E}} = A_{\mathcal{E}} \cup \{(e_v, \rho_i)\}$
                Adjust remaining capacities of $c_i$
                $T = T \setminus \{e_v\}$
            $E_i = E_i \setminus \{e_v\}$
    $T = T \setminus \{e_i\}$
*/* Supplementing */*
**for** $e_i \in S$ **do**
    $(c^*_i, \rho^*_i) \leftarrow$ `BestFeasiblePair(`$e_i, A_{\mathcal{C}}$`)`
    **if** $(c^*_i, \rho^*_i) = \emptyset$ **then**
        $(c^*_i, \rho^*_i) \leftarrow$ `UpgradeCloudlet(`$\rho^*_i$`)`
    $A_{\mathcal{C}} = A_{\mathcal{C}} \cup \{(c^*_i, \rho^*_i)\}$
    $A_{\mathcal{E}} = A_{\mathcal{E}} \cup \{(e_i, \rho^*_i)\}$
    Adjust remaining capacities of $c^*_i$
Calculate $\widehat{\boldsymbol{\Phi}}, \widehat{\boldsymbol{L}}$
**Output:** $A_{\mathcal{E}}$, $A_{\mathcal{C}}$, $\widehat{\boldsymbol{\Phi}}, \widehat{\boldsymbol{L}}$

**Algorithm 7** BestCloudlet($e_i, \rho_i$)

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 $\rho_i$ 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`$(e_i, \rho_i)$ function, given in Algorithm 7. If a cloudlet $c_i^*$ with sufficient capacity and range is already placed at candidate point $\rho_i$ (Alg. 7, 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 $\rho_i$ (Alg. 7, line 6). If no such a cloudlet is found, i.e., `BestCloudlet`$(e_i, \rho_i)$ 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 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 $\rho_i$ 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 $\rho_i^*$ 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 ($\widehat{\boldsymbol{L}}$) 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 Theorem 3 and Theorem 4, and demonstrate that ACP is a (2,4)-approximation algorithm for the OCP problem.

*ACP Properties.* 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 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. Practically, if appropriate estimates are made about user demands and their distribution, and a reasonable set of candidate points

are established, ACP does not need the upgrades in the second part to find a feasible solution.

**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,...,$u$} 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 $U$ [34]. 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 $\rho$. 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 $\{e_1, ..., e_4\}$, and OCP with 4 devices, 3 cloudlets, and 3 candidate points (as shown in Figure 4.1). In the example, the sub-collections

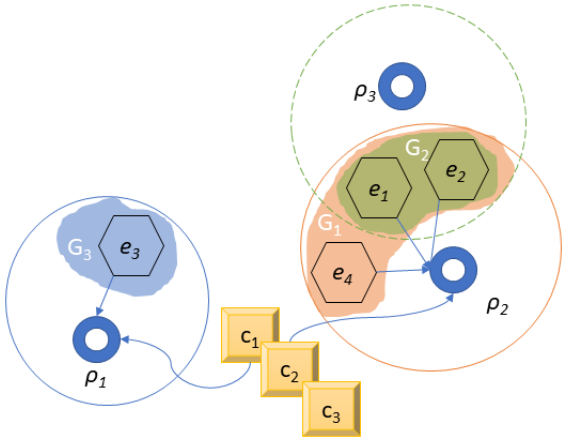



Figure 4.1: Reduction of Set Cover decision problem to OCP

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 $\rho_1$, $\{e_4, e_1, e_4\}$ are the devices that can be covered from $\rho_2$ , and $\{e_1, e_2\}$ can be covered from $\rho_3$. Placing cloudlets $c_1$ and $c_2$ on $\rho_1$ and $\rho_2$ 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 $\beta D_i$ 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:

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

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 ($\widehat{\boldsymbol{L}} \leq 2\boldsymbol{L}$).*

*Proof.* ACP begins by assigning device $e_i$ with the largest $D_i$ to the closest candidate point $\rho_i$ 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 $\rho_i$ selected above. The extended set contains all devices $e_v \in \mathcal{E}$ that have $\rho_i$ in their respective neighborhood set $N_v$. All devices that can be covered by cloudlet $c_i$ placed at $\rho_i$ are assigned to $\rho_i$.

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

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 $A_{\mathcal{C}}$ 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:

$$\widehat{\boldsymbol{L}} \leq \sum_{e_i \in \mathcal{E}} 2D_i. \tag{4.1}$$

We know that:

$$D_i = \sum_{\rho_k \in \mathcal{P}} l_{ik}\, a^*_{ik} \tag{4.2}$$

Since all individual devices are within $2D_i$, the sum over all devices is also within twice

the value of LP latency, denoted by $\boldsymbol{L}^*$.

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

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).

$$\sum_{\rho_k \in N_i} a^*_{ik} \geq 1/2 \tag{4.3}$$

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 (Alg 6, line 17-18). Since every device is assigned to exactly one candidate point in ACP (also in OCP, constraint 2.12), 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 (4.3), this leads us to:

$$\sum_{\rho_k \in N_i} \widehat{a}_{ik} \leq 2 \sum_{\rho_k \in N_i} a^*_{ik} \tag{4.4}$$

There is also a mathematical relationship between the two decision variables based on constraint (2.7) of OCP. The `LP-Cost-OCP()` solution must follow this constraint for a feasible solution. Thus, we have:

$$a^*_{ik} \leq \sum_{c_j \in \mathcal{C}} y^*_{jk} \qquad \forall e_i \in \mathcal{E}, \rho_k \in \mathcal{P} \tag{4.5}$$

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,

$$\sum_{\rho_k \in N_i} a^*_{ik} \leq \sum_{\rho_k \in N_i} \sum_{c_j \in \mathcal{C}} y^*_{jk} \qquad \forall e_i \in \mathcal{E} \tag{4.6}$$

Combining Eq (4.4) and (4.6), we get:

$$\sum_{\rho_k \in N_i} \widehat{a}_{ik} \leq 2 \sum_{\rho_k \in N_i} \sum_{c_j \in \mathcal{C}} y^*_{jk} \qquad \forall e_i \in \mathcal{E} \tag{4.7}$$

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

$$\sum_{c_j \in \mathcal{C}} \widehat{y}_{jk} = \sum_{\rho_k \in \mathcal{P}} \widehat{a}_{ik} \qquad \forall e_i \in \mathcal{E} \tag{4.8}$$

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 (4.7), we have:

$$\sum_{\rho_k \in N_i} \sum_{c_j \in \mathcal{C}} \widehat{y}_{jk} \leq 2 \sum_{\rho_k \in N_i} \sum_{c_j \in \mathcal{C}} y^*_{jk} \tag{4.9}$$

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 4.9). 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:

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

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 (4.9) we have:

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

$$\widehat{\mathbf{\Phi}} \leq 4\mathbf{\Phi}^*$$

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. □

## 4.3 Experimental Results

### 4.3.1 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 [1, 18]. Also, the presence of NYC Open Data [17] 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.

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 4.2.

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

Table 4.2: Experiment Scenarios - ACP

| **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 |

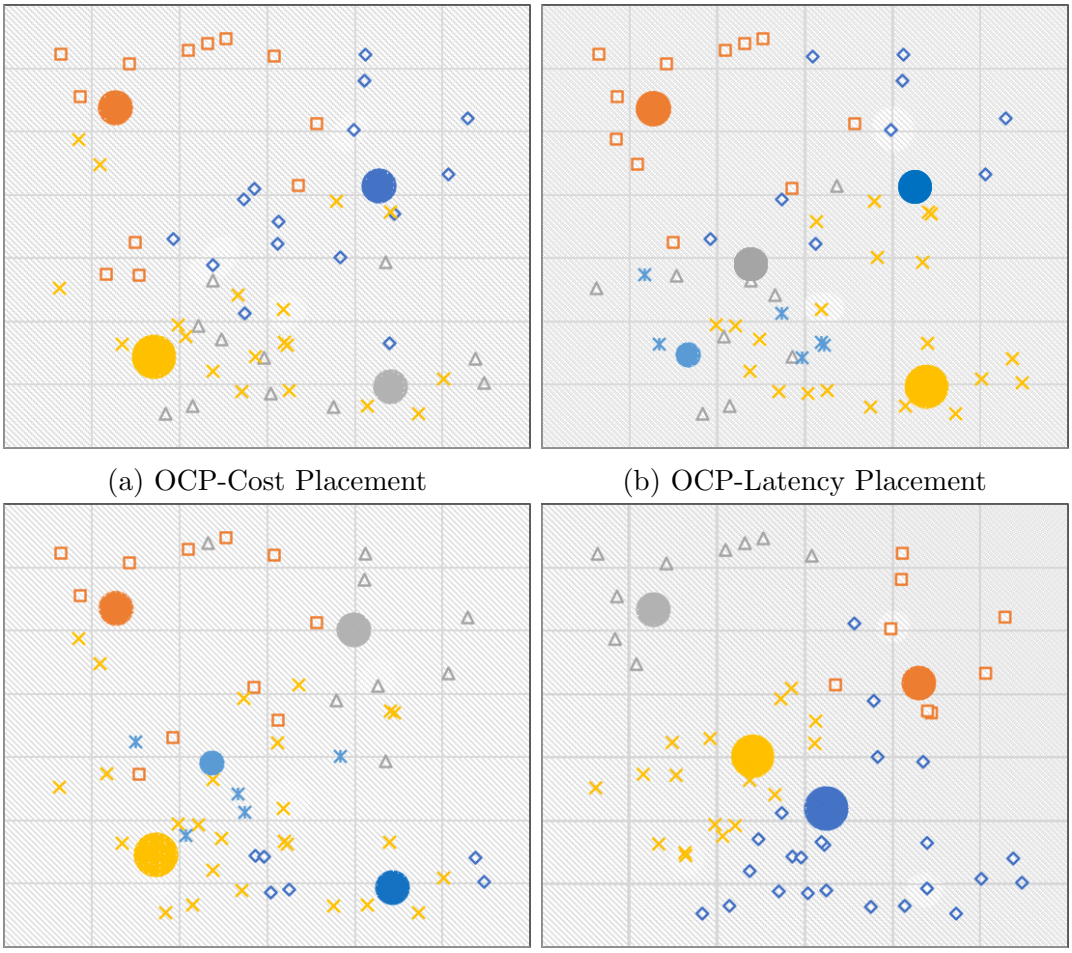
Placement (GACP), proposed in Chapter 3.

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 [29]. 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 [5] 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 Chapter 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, OCP-Cost and GACP have partial but valid results which are explained further in the analysis section.

### 4.3.2 Analysis of Results

We first compare the results of ACP with OCP-Cost, OCP-Latency, and GACP approaches in terms of device coverage. In Figure 4.2, 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. 4.2a, uses the minimum cloudlet resources to obtain the minimum cost. OCP-Latency, shown in Fig. 4.2b, covers devices



(a) OCP-Cost Placement (b) OCP-Latency Placement

(c) ACP Placement (d) GACP Placement

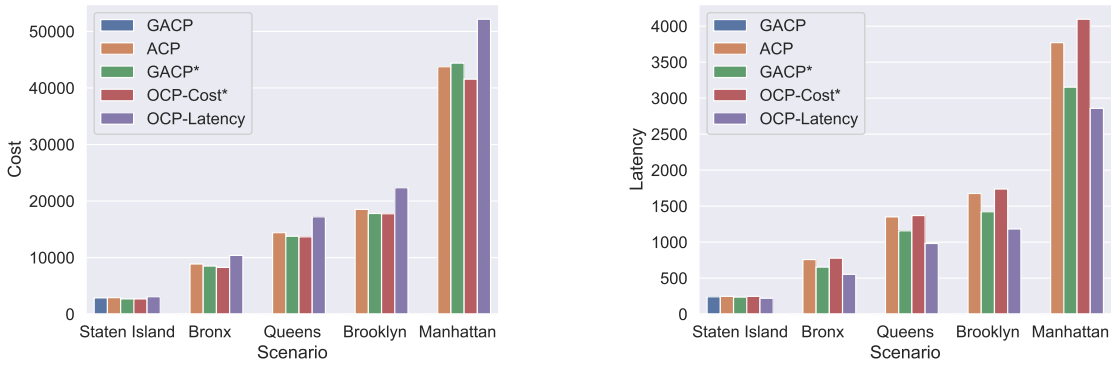
Figure 4.2: Comparative Coverage Visualization of OCP, GACP, and ACP

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. 4.2d, 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. 4.2c, 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.

Figure 4.3a 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 Figure 4.4b. 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 Figure 4.4a.

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

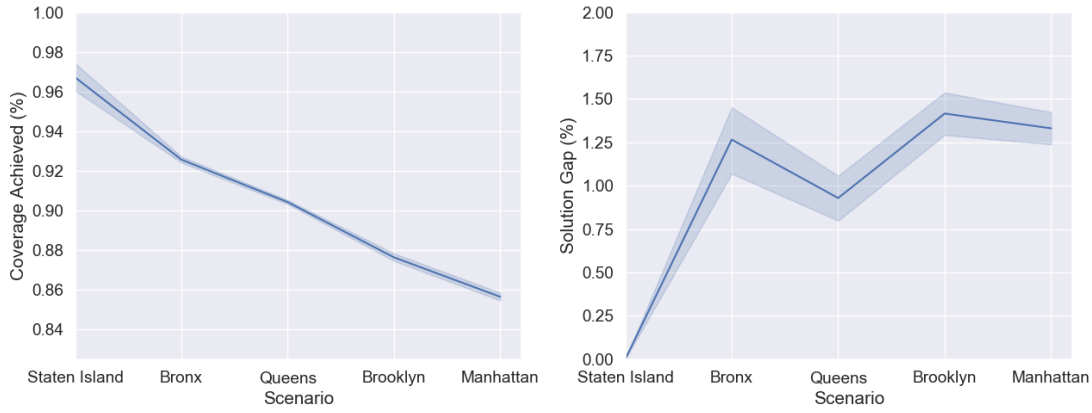



(a) Cost (b) Latency

Figure 4.3: Summary of OCP, GACP, and ACP 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.*

obtained by ACP are similarly much lower than OCP-Latency costs.

The latency values presented in Figure 4.3b 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.




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

Figure 4.4: Partial Solution Scenarios of GACP and OCP

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. Figure 4.5 shows running time as a function of the number of candidate points 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 4.2. 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

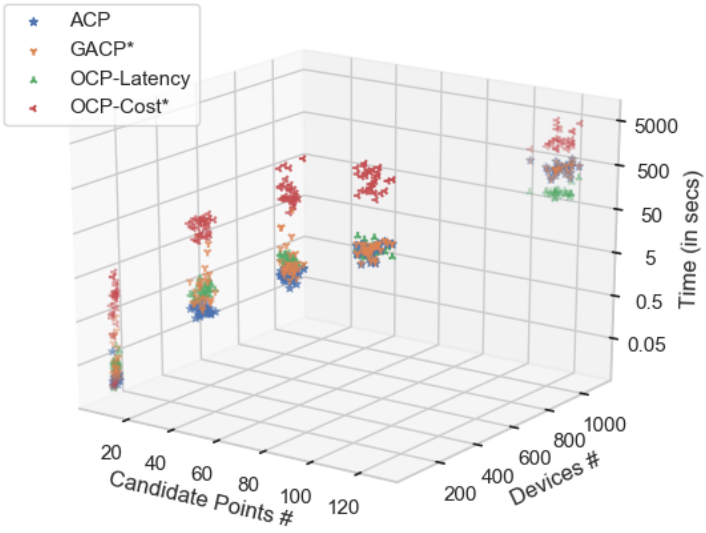



Figure 4.5: Comparison of OCP, GACP, and ACP Running Time

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 Figure 4.4a 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.

## 4.4 Chapter Summary

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 chapter, we addressed 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 placement cost, respectively. The rigorous experimental results show that ACP is scalable and achieves close to the optimal results in experimental scenarios.

The research article based on this chapter has been published in the IEEE Transactions on Parallel and Distributed Systems [13]. The results of this research are fully reproducible and the complete code in the reproducible capsule is available on Code Ocean [12]. We establish the impact of user mobility in cloudlet placement and define the dynamic cloudlet placement problem in the next chapter.

# Chapter 5

# THE DYNAMIC MOBILE CLOUDLET PLACEMENT PROBLEM

## 5.1 Background

New and emerging applications like real-time translation, autonomous driving, and Extended Reality (XR) have a unique combination of requirements including on-demand services with extremely high network speed and low latency [52]. As a result, efforts to further progress and envision future technologies in wireless networks and telecommunications based on increasing demand for data, flexible computing resources, and connectivity are continuously being made. Edge computing can partially address these requirements by allowing users to consume the computing resources in their vicinity via offloading, thereby reducing the access latency and data traffic to the core infrastructure [47]. The convergence of edge computing with next-generation networks (5G and beyond) can complete the requirements puzzle by providing extremely high network speeds [53], thereby creating the notion of next-generation edge networks.

Nonetheless, covering highly mobile users while providing stable low-latency services as they move across the service region is a complex problem. Locations with limited or disrupted connectivity during natural disasters or generally poor infrastructure cannot sufficiently serve mobile users. Even infrastructures with wide coverage may not meet offloading and ultra-low latency requirements. Next-generation wireless networks with multi-gigabit data rates can still have technical challenges such as signal reflection, path loss, and blockages [53]. On the other hand, offloading to static edge servers (or cloudlets) is equally challenging and leads to frequent service and task migrations for mobile users. Accordingly, the long-term, static placement of cloudlets does not meet all the requirements of mobile users in next-generation edge networks.

Thus, we can adequately serve highly mobile users with dynamic demands only when efficient approaches to dynamically place the cloudlets are available.

Dynamic deployment of mobile cloudlets presents an elegant solution to this problem. Mobile cloudlets mounted on Unmanned Aerial Vehicles (UAVs) have been well-studied and already under adoption in the industry [42, 55] to enhance user experience and enable new innovative applications. Federal Aviation Authority (FAA) and Verizon's Skyward are currently collaborating to test cellular-connected UAVs [60]. Some studies have proposed UAVs to complement wireless backhaul networks [65] and even an entire cellular infrastructure based on a hierarchical deployment of UAVs [57]. UAV-mounted cloudlets can be flexibly deployed as the user demands and distributions change over a region. This is especially applicable to support compute-intensive latency-critical computations in disaster scenarios [55], massive demand areas such as stadiums, and emerging applications including drone air-delivery. For instance, dispersed teams of rescue workers can use XR headgears in inaccessible or partially accessible landscapes for cognitive assistance such as identifying objects, people, and threats, and path finding during natural disasters. These rapidly evolving compute-intensive applications (object detection, risk assessment, and navigation) need ultra-low latency and stable offloading services so that well-informed decisions can be made in a matter of milliseconds, which are otherwise not possible under damaged or distant infrastructure, and where emergency support vehicles and equipment cannot reach. Mobile cloudlets meet these critical requirements by directly bringing resources to the rescuers in response to their uncertain movements.

The same UAVs can provide connectivity to emergency services in disaster-disrupted infrastructure by following vehicles of evacuating public. Other use cases include, sudden burst of clustered connectivity demand in tailgating crowds in sporting events and Coming to an example in dense environments, mobile cloudlets allow autonomous processing of a large number of continuously moving inventory through video-based intelligent applications in large warehouses to streamline supply-chain issues.User experience in well-known emerging applications such as Augmented Reality

(AR) and real-time translation can be greatly enhanced deploying such UAVs nearby.

In the next sections, we define the dynamic cloudlet placement model and formulate the optimal mobile cloudlet placement problem as an integer program by capturing quintessential aspects of highly heterogeneous and high mobility next-generation edge networks. We also provide an illustration to show how the mobile cloudlet placement works under user mobility.

## 5.2 Dynamic System Model

Our aim in this chapter is to continuously place and readjust the locations of mobile cloudlets in a region to maximize device coverage and maintain stable services, given the uncertain changes in user demands and movements. As such, we model the region as a finite 3D grid within which the devices and the mobile cloudlets move. The 3D grid is made up of smaller contiguous 3D sub-divisions called cells.

The 3D grid is denoted by a collection of non-overlapping equal-sized cells or cubes denoted by $\mathcal{C} = \{c_1, c_2, \ldots, c_k, \ldots, c_n\}$. Note that the geometric representation of the cells can be other 3D shapes such as a hexagonal prism or a sphere. We consider cubes, each of which denotes 3D coverage of a unit cloudlet, for simplicity. The interconnected cells make the contiguous 3D region. The cells, represented using 3D coordinate axes, are always indexed from top-left to bottom-right with the index denoting the area they cover. The mobile cloudlets and devices may change their locations from cell to cell over time $\tau = \{1, 2, \ldots, t, \ldots, T\}$. The cloudlets move between the cells to follow the changes in the device distribution, as shown in Figure 5.1 (shown in 2D for simplicity).

We denote the set of heterogeneous mobile cloudlets by $\mathcal{M} = \{m_1, m_2, \ldots, m_j, \ldots, m_u\}$. Each mobile cloudlet $m_j \in \mathcal{M}$ is represented by a 3-tuple $m_j = \{\mu_j^t, r_j, \Lambda_j\}$ denoting its attributes: $\mu_j^t$ is the service capacity (in resource units) at time $t$, $r_j$ is the 3D coverage (3D cell dimensions), and $\Lambda_j$ is the maximum cell distance the cloudlet can travel within a time slot $t$. Note that the resources indicated by service capacity $\mu_j^t$ can be processing, memory, or storage individually or as a combination. We assume a

single resource for simplicity and without loss of generality. Similarly, the maximum cell travel $\Lambda_j$ depends on multiple parameters such as the speed of the mobile cloudlet, power consumption/available battery, and the weight carried by the mobile cloudlet. This is used to set a constraint on general mobility of the cloudlets so that they follow a realistic travel path and travel a limited distance in every time slot while avoiding collisions during placement.

Likewise, we denote the set of heterogeneous devices by $\mathcal{D} = \{d_1, d_2, \ldots, d_i, \ldots, d_v\}$. Each device $d_i \in \mathcal{D}$ is represented by a 3-tuple $d_i = \{\delta_i^t, c_i^t, \lambda_i^t\}$ denoting its attributes: $\delta_i^t$ is the demand (in resource units) at time $t$, $c_i^t$ is the 3D cell of the mobile device at time $t$, and $\lambda_i^t$ is an ordered triple denoting the 3D coordinates of the mobile device at time $t$.

For simplicity of modeling, if a cloudlet fails at any instant, the cloudlet capacity becomes 0. If it has recovered, its capacity comes back to non-zero. The same applies to a device when it is disconnected or connected, and the device demand goes to 0 or non-zero, respectively. These can also be treated as a device going out of bounds and returning back to the service region. We assume that these values are set by the underlying sensor networks. Also, all connected devices are assumed to receive ultra-low latency services if they are within 3D coverage regardless of their absolute distance from the mobile cloudlet.

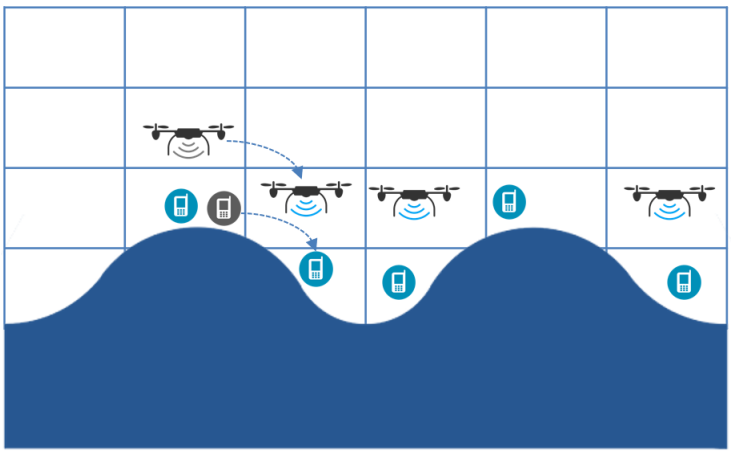


Figure 5.1: Mobile cloudlet placement scenario in 2D

Our goal is to maximize the coverage of the devices in the region (number of connected devices) with ultra-low latency services by moving the cloudlets in response to the device mobility.

### 5.3 Optimal Mobile Cloudlet Placement

We now formulate the Optimal Mobile Cloudlet Placement problem (OMCP). We define the following decision variables:

$$\alpha_{jk}^{t} = \begin{cases} 1 & \text{if cloudlet } m_j \text{ is assigned to cell } c_k \text{ at time } t, \\ 0 & \text{otherwise,} \end{cases}$$

$$\gamma_{jkl}^{t} = \begin{cases} 1 & \text{if cloudlet } m_j \text{ moves from } c_k \text{ to } c_l \text{ at time } t, \\ 0 & \text{otherwise,} \end{cases}$$

and

$$\beta_{ij}^{t} = \begin{cases} 1 & \text{if device } d_i \text{ is assigned to cloudlet } m_j \text{ at time } t, \\ 0 & \text{otherwise.} \end{cases}$$

We mathematically formulate OMCP as an Integer Program (IP) as follows:

$$\max \sum_{t=1}^{T} \sum_{i=1}^{v} \sum_{j=1}^{u} \beta_{ij}^{t} \tag{5.1}$$

Subject to:

$$\sum_{j=1}^{u} \sum_{k=1}^{n} \alpha_{jk}^{t} = |\mathcal{M}|, \quad \forall t \in \tau \tag{5.2}$$

$$\sum_{k=1}^{n} \Delta(c_i^t, c_k)(\beta_{ij}^{t} + \alpha_{jk}^{t} - 1) \leq r_j, \forall d_i \in \mathcal{D}, m_j \in \mathcal{M}, t \in \tau \tag{5.3}$$

$$\sum_{i=1}^{v} \delta_i^t \beta_{ij}^t \leq \mu_j^t, \quad \forall m_j \in \mathcal{M}, t \in \tau \tag{5.4}$$

$$\gamma_{jkl}^t \geq \alpha_{jk}^{t-1} + \alpha_{jl}^t - 1, \forall m_j \in \mathcal{M}, c_k, c_l \in \mathcal{C}, t \in \{2, \ldots, T\} \tag{5.5}$$

$$\gamma_{jkl}^t \leq \alpha_{jk}^{t-1}, \quad \forall m_j \in \mathcal{M}, c_k, c_l \in \mathcal{C}, t \in \{2, \ldots, T\} \tag{5.6}$$

$$\gamma_{jkl}^t \leq \alpha_{jl}^t, \quad \forall m_j \in \mathcal{M}, c_k, c_l \in \mathcal{C}, t \in \{2, \ldots, T\} \tag{5.7}$$

$$\Delta(c_k, c_l)\gamma_{jkl}^t \leq \Lambda_j, \forall c_k, c_l \in \mathcal{C}, m_j \in \mathcal{M}, t \in \{2, \ldots, T\} \tag{5.8}$$

$$\sum_{j=1} \alpha_{jk}^t \leq 1, \quad \forall c_k \in \mathcal{C}, t \in \tau \tag{5.9}$$

$$\sum_{k=1}^{n} \alpha_{jk}^t \leq 1, \quad \forall m_j \in \mathcal{M}, t \in \tau \tag{5.10}$$

$$\beta_{ij}^t \leq \sum_{k=1}^{n} \alpha_{jk}^t, \quad \forall d_i \in \mathcal{D}, m_j \in \mathcal{M}, t \in \tau \tag{5.11}$$

$$\sum_{j=1}^{u} \beta_{ij}^t \leq 1, \quad \forall d_i \in \mathcal{D}, t \in \tau \tag{5.12}$$

$$\gamma_{jkl}^t \in \{0, 1\}, \quad \forall m_j \in \mathcal{M}, c_k, c_l \in \mathcal{C}, t \in \tau \tag{5.13}$$

$$\alpha_{jk}^t, \beta_{ij}^t \in \{0, 1\}, \quad \forall m_j \in \mathcal{M}, d_i \in \mathcal{D}, c_k \in \mathcal{C}, t \in \tau \tag{5.14}$$

The objective function shown in Eq (5.1) maximizes the coverage of the devices, i.e., mapping of the devices to the placed cloudlets. Constraints (5.2) ensure that the total number of cloudlets placed in the region is equal to the number of available cloudlets. Constraints (5.3) guarantee that each covered device must be within the coverage radius of some cloudlet. The constraint indicates that a device $d_i$ currently at cell $c_i^t$ if mapped to a cloudlet $m_j$ placed at a cell $c_k$, the distance between their cells must be less than or equal to the 3D coverage $r_j$ of the cloudlet. Constraints (5.4) satisfy supply and demand based on each cloudlet capacity and its covered device demands. Sum of all device demands assigned to a cloudlet must be less than or equal to the resources of that cloudlet. Constraints (5.5) ensure that a cloudlet moves from one cell to another at $t$ only if its cell association changes over time $t-1$ and $t$. Constraints (5.6) and (5.7) ensure that a cloudlet moves from one cell to another at $t$ only if it was placed in

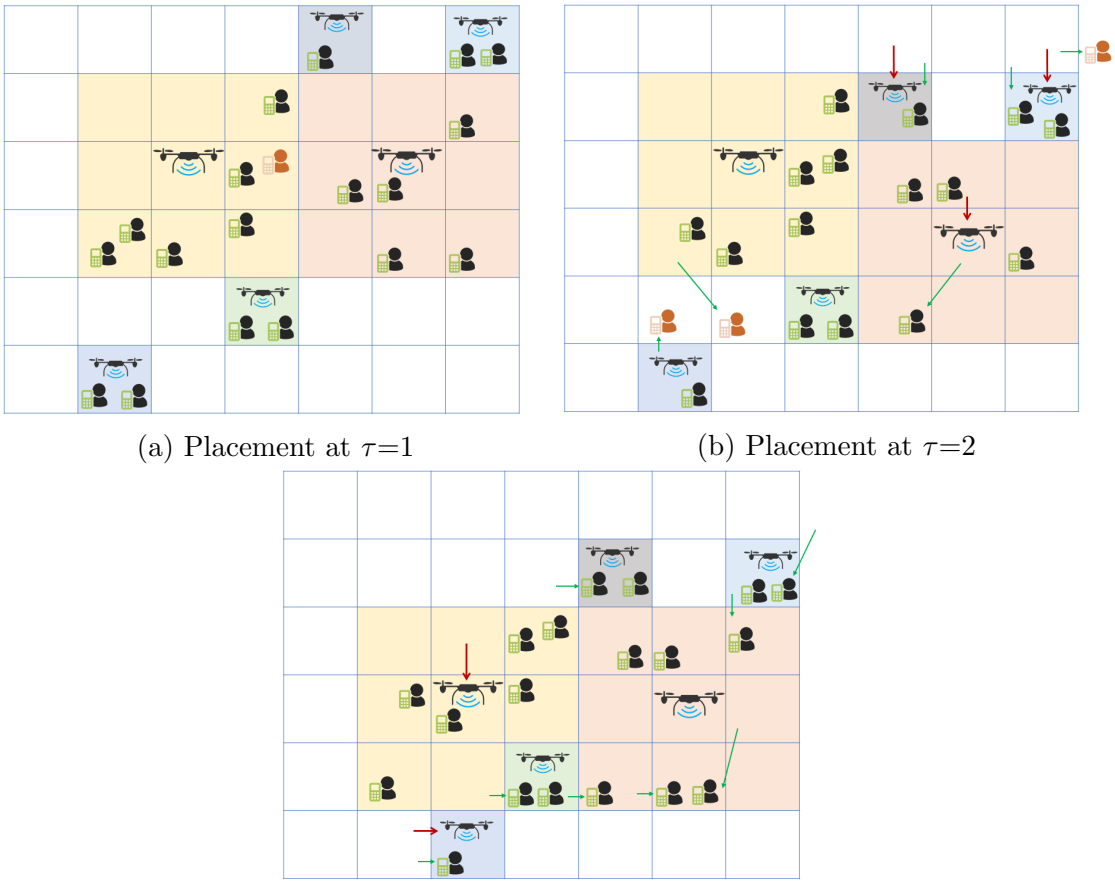
the source cell at $t-1$ and in the destination cell at $t$. Constraints (5.8) ensure that a cloudlet moves no further than the maximum distance it can travel within a time slot $t$. Constraints (5.9) ensure that at most one cloudlet is placed at any cell in the region. Constraints (5.10) ensure that a cloudlet can only be placed at a single cell. Constraints (5.11) guarantee that a device can only be served by a cloudlet that is placed in the grid. Constraints (5.12) guarantee that each device is served by at most one cloudlet. Finally, constraints (5.13) and (5.14) ensure the integrality requirements of the decision variables.

### 5.3.1 Model Illustration

Since a 3D example can be harder to visualize, a 2D illustration of the model and how placements take place over time is shown in Figure 5.2 using 6 mobile cloudlets and 19 mobile users. The figure is reflective of all parameters of the model except that the 3D grid is flattened to two dimensions. The mobile cloudlets are depicted by drones with their size (small and large) representing their capacities. Assuming all mobile users have the same demand, there are two types of mobile cloudlets in the illustration: large, can serve up to 6 users, and small, can serve up to two users.

At time $\tau = 1$, the best possible placement can cover only 18 users with one mobile user highlighted in orange currently out of coverage. Note that the goals are to maximize coverage (full coverage may not be possible) while tracking movements of cloudlets in response to device mobility. Hence, it is not a requirement to cover all the users all the time. In the next time slot $\tau = 2$, three users move to different cells (marked by green arrows) and three cloudlets respond to the changes accordingly by moving to other cells (marked by red arrows). By this optimal readjustment, three users are now out of coverage. It is noteworthy that one user (top right) went outside the grid, and as a result, is not needed to be covered anyway. Although the coverage is worse off, it is still the best scenario given the constraints.

At $\tau = 3$, seven users move to different cells. As we can see, two cloudlets are needed to move in the optimal solution to attain full coverage. This summarizes the



(a) Placement at $\tau$=1

(b) Placement at $\tau$=2

(c) Placement at $\tau$=3

Figure 5.2: Dynamic Cloudlet Placement Illustration

whole idea of maximizing coverage while moving the cloudlets based on device mobility at each time slot.

## 5.4 Chapter Summary

In this chapter, we provided motivating factors behind dynamic mobile cloudlet placement and designed a dynamic system model to place mobile cloudlets in regions with continuously changing user demands and uncertain user mobility. We then formulated and illustrated an optimal mathematical model (OMCP) to solve the dynamic mobile cloudlet placement problem. However, OMCP is NP-hard and an efficient approach is required to solve the problem. As a result, we propose a physics-inspired mobile cloudlet placement approach to solve this problem in the next chapter.

# Chapter 6

# PHYSICS-INSPIRED MOBILE CLOUDLET PLACEMENT

## 6.1 Background

In this chapter, we present an ingenious placement of mobile cloudlets in next-generation edge networks to provide ultra-low latency services for the emerging applications considering realistic constraints. We aim to provide stable services to the users through mobile cloudlets when user distribution and demands change rapidly. Our proposed approach responds to uncertain device mobility in real time. It not only removes the overhead of predicting device locations but also does not require the acquisition of sensitive mobility data to make those predictions.

We design a novel approach to address these issues by borrowing the concepts of "center of gravity" and "force of attraction" from physics. The center of gravity approach has been used in the field of Operations Research for weighted facility location based on demands and is useful in deciding initial placements for adaptive approaches [9]. In the same way, the force of attraction has been the core idea behind gravitational clustering which allows formation of dynamic clusters and establishment of their respective centers [74]. Furthermore, it has been applied in robust adaptive clustering in distributed networks [14]. Although we use the force of attraction in a different way than these works, all of them suggest that our approach is highly suitable for the dynamic placement of mobile cloudlets under uncertainty.

Our approach performs the initial mobile cloudlet placement based on the center of gravity to evenly balance the location of the cloudlets. Such a balanced initial placement allows devices to be placed around weighted centers of device demands, enhancing potential coverage. Subsequently, our approach readjusts the locations of

the mobile cloudlets in real-time by moving them using the force of attraction to follow the sub-regions with high user concentration. Since cloudlets follow the devices, devices are likely to be served by the same cloudlets over time leading to less migration and switching services between the cloudlets, which is significantly important in providing low-latency stable edge services. A cyber-physical system driven by user demands and mobility in next-generation edge networks can hence be dynamically configured and balanced using a combination of these two principles of our approach.

Our idea is to maximize the device coverage while providing a stable service. We initially place the mobile cloudlets using center of gravity to achieve a balanced placement, and then adaptively relocate the cloudlets based on changing user demands and mobility using force of attraction.

As a part of our study, we define an optimal integer programming formulation of the mobile cloudlet placement problem, OMCP, and implement the classic DBSCAN algorithm [21] with path planning to serve as our two performance benchmarks. We perform extensive experiments by creating multiple scenarios, including both 2D and 3D user distribution and mobility, based on real mobility traces obtained from KTH Walkers dataset [36]. Multiple experiment scenarios created from the traces include both 2D and 3D user distribution and mobility. We then compare our results with OMCP and DBSCAN. The results show that our approach achieves high coverage values, superior energy efficiency, and stable services for all scenarios in real time.

### 6.2 Physics-Inspired Mobile Cloudlet Placement

We propose a novel Physics-Inspired Mobile Cloudlet Placement (PMCP) approach to place and dynamically change locations of the mobile cloudlets inspired by the concept of the *center of gravity* and *force of attraction*. PMCP performs the initial mobile cloudlet placement based on the center of gravity to evenly balance the location of the cloudlets. Such a balanced initial placement allows cloudlets to be placed around weighted centers of device demands, enhancing potential coverage. Subsequently, PMCP readjusts the locations of the mobile cloudlets in real-time by

**Algorithm 8** PMCP - Initialization

**Input:** $\mathcal{M}, \mathcal{C}, \mathcal{D}$
A ← AggregateDemandsByCell($\mathcal{C}, \mathcal{D}$)
CoG ← TopCells(A,$u$) ▷ Top $u$ cells in A
CoGMap ← CoGDeviceMappings($\mathcal{D}$,CoG)
CoGMap* = $\emptyset$
**while** CoGMap* != CoGMap **do**
    CoG = $\emptyset$
    **for** $\{c_k : [d_a, d_b, ..]\} \in$ CoGMap **do**
        Calculate $(\widehat{X}, \widehat{Y}, \widehat{Z})$ based on $[d_a, d_b, ..]$
        CoG ← CoG $\cup$ the cell closest to $(\widehat{X}, \widehat{Y}, \widehat{Z})$
    CoGMap* ← CoGMap
    CoGMap ← CoGDeviceMappings($\mathcal{D}$,CoG)
CoGMap ← SortDescByDemands(CoGMap)
MCP ← MapCloudletsCoGs(CoGMap,$\mathcal{M}$)
**Output:** MCP

moving them using the force of attraction to follow the sub-regions with high user concentration. Since cloudlets follow the devices, devices are likely to be served by the same cloudlets over time leading to less migration and switching services between the cloudlets, which is significantly important in providing low-latency stable edge services. As the cloudlets astutely match the devices' mobility, it can extend the battery life of the cloudlets by avoiding unnecessary cloudlet movements. Next, we explain these parts in detail.

### 6.2.1 Initial Placement

The initial placement by PMCP is inspired by the center of gravity method to minimize the weighted distance of the devices from the mobile cloudlets to be placed in the 3D grid. PMCP estimates the weighted centers of gravity for the devices in the grid based on the demands and the cell locations of the devices at time $\tau = 1$. The estimated centers of gravity are iteratively improved until they converge. The cloudlets are then initially placed at the central cell locations of the converged centers of gravity. The initial placement by PMCP is described in Algorithm 8.

In Algorithm 8, individual demands of the devices are first aggregated per cell and stored in list `A`. Thereafter, a list of $u$ cells with the highest demands, i.e., `CoG` is generated from `A` (line 3). Note that `CoG` is the list of temporary centers of gravity cells, and $u \leq n$ is the number of all mobile cloudlets. Function `CoGDeviceMappings(`$\mathcal{D}$`,CoG)` (line 4) returns a mapping between the cells identified in `CoG` and the devices, where each device is assigned to the closest cell in `CoG`. This mapping `CoGMap` $= \{c_k : [d_a, d_b, ..]\}$ represents the temporary mapping of the devices to the centers of gravity `CoG` cells which converges through calculations in next steps of Algorithm 8 (lines 5-12) to provide the initial mobile cloudlet placements.

Once the temporary mapping is done, the algorithm initializes `CoGMap*` as an empty CoG cells-to-devices mapping to keep track of the changes in the mapping between the centers of gravity `CoG` and the devices. The recalculation of centers of gravity happens iteratively (lines 6-12) as long as the new `CoG` and mappings differ from the previous round, i.e., `CoGMap` and `CoGMap*` are not equal. For every CoG cell $c_k$ in `CoGMap` and its corresponding list of devices $[d_a, d_b, ..]$, central coordinates $(\widehat{X}, \widehat{Y}, \widehat{Z})$ are calculated (line 9) using Equation 6.1.

$$\widehat{X} = \frac{\sum \delta_i x_i}{\sum \delta_i}, \widehat{Y} = \frac{\sum \delta_i y_i}{\sum \delta_i}, \widehat{Z} = \frac{\sum \delta_i z_i}{\sum \delta_i} \tag{6.1}$$

The sums, noted by $\sum$, in the equations above are over a set of devices that is supposed to be covered by a single mobile cloudlet. Therefore, $(\widehat{X}, \widehat{Y}, \widehat{Z})$ represents a weighted center of gravity, where a mobile cloudlet will be placed. This implies that the mobile cloudlet will be closer to the devices with higher demand, inadvertently prioritizing cells with more demands, enhancing the overall coverage.

Back to Algorithm 8, the cell closest to $(\widehat{X}, \widehat{Y}, \widehat{Z})$ is added to `CoG` list as a new center of gravity (line 10). Note that `CoG` is cleared in line 7 to enlist newly calculated centers of gravity. In line 11, the previous `CoGMap` becomes `CoGMap*`, and the new `CoGMap` is calculated by reassigning devices to the closest cells in the updated `CoG` (line 12). At the end of each iteration, `CoGMap` represents temporary `CoG` cells with

their corresponding devices. These iterations continue until the updated `CoG` and their device mappings, i.e, `CoGMap` do not change any further.

After convergence, `CoGMap` is then sorted in descending order of the total demand of the devices assigned to each cell in `CoG` using `SortDescByDemands(CoGMap)` function (line 13). The mobile cloudlets are mapped one-to-one to the converged `CoG` based on their size by `MapCloudletsCoGs(CoGMap,`$\mathcal{M}$`)` function to provide the mobile cloudlet placements `MCP`, which is the mapping of mobile cloudlets to the CoG cells $\{(m_a, c_b), \ldots, (m_j, c_k), \ldots\}$ and the output of Algorithm 8.

### 6.2.2 Adaptive Placement

In physics, the force of attraction between two bodies can be formulated in multiple ways based on the type of force. Attractive forces can be the magnetic force, electric force, or the most well-known gravitational force. In our approach, we formulate the force of attraction as a *vector pull force* from the devices acting on the mobile cloudlets which attracts them to their new locations.

Our proposed approach, PMCP, uses this physics concept by considering the devices as moving masses (which need to be served) since they continuously change positions between time slots. In addition, PMCP considers the mobile cloudlets as fixed masses (that are deployed to serve devices) since they are assumed to be at rest in their placed cells until pulled by the changing device demands and mobility into new cells.

Essentially, mobile cloudlets (fixed masses) are initially placed around the devices (moving masses) using the center of gravity method i.e., Algorithm 8. The mobile cloudlets (fixed masses) then move as they are attracted (pulled) by the devices (moving masses), as illustrated in Figure 6.1. We use the capacities of mobile cloudlets and the demands of the mobile devices to represent their physical masses. The mobile cloudlets thus respond to the changes in device locations and demands, and they move to a cell in the direction of the aggregate vector pull forces of the devices. It is

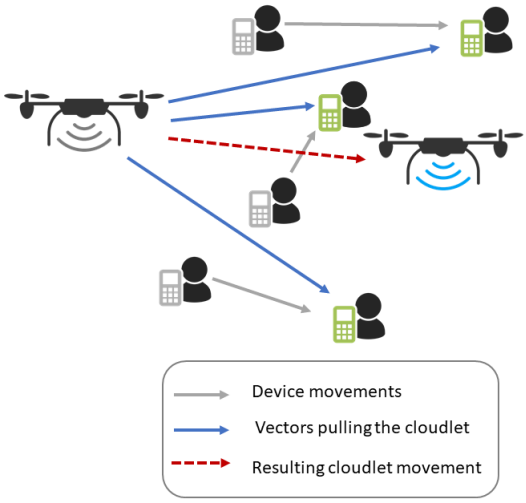



Figure 6.1: A mobile cloudlet responding to device mobility

important to note that when calculating the forces, mobile cloudlets do not physically interact with each other, neither do the devices.

As we formulate the adaptive part of PMCP, we aim to relocate the mobile cloudlets by estimating their new cell locations at each time slot, denoted by $w_j(t)$ for cloudlet $m_j$, based on changes in the device specifications at each time slot. Algorithm 9 determines and outputs the new positions of all mobile cloudlets at every time slot based on the overall vector pull force acting on them. The force acting on mobile cloudlet $m_j$ is the superposition of all individual forces emerging from each device assigned to it at the end of the previous time slot, obtained by calling `DeviceAssignments()` function, presented in Algorithm 10. Further details of Algorithm 9 are provided later in this subsection.

As we can observe in Algorithm 10, each device is assigned to its closest feasible mobile cloudlet. The device assignment to the cloudlet is performed as long as the device is within the coverage radius and the mobile cloudlet has enough capacity to meet its demand (lines 1 to 15). For every device, the algorithm searches for the closest cloudlet that can cover the device and has sufficient remaining capacity to meet its demand (lines 6-12). If such a cloudlet exists, the device is assigned to the cloudlet and the cloudlet capacity is adjusted accordingly (lines 13-15). Otherwise, the

**Algorithm 9** PMCP - Adaptive Placement

1: **Input:** `MCP`
2: `DCA` = `DeviceAssignments(MCP)`
3: **for** $t = 2, 3, \ldots, T$ **do**
4: **for all** $m_j \in \mathcal{M}$ **do**
5: $\widehat{\mathcal{D}_j}(t)$ = `DCA`$[m_j]$
6: Calculate force $\overrightarrow{F_j}(t)$ using Eqn (6.2)
7: Assign constrained force $\overrightarrow{F_j}(t)$ to $m_j$ using Eqn (6.3)
8: Find estimated location $\widehat{w_j}(t)$ of $m_j$ using Eqn (6.4)
9: $w_j(t)$ = `ClosestCell`$(\widehat{w_j}(t))$
10: `MCP`$(t) \leftarrow w_j(t) \quad \forall m_j \in \mathcal{M}$
11: `DCA` = `DeviceAssignments(MCP(`$t$`))`
12: **Output at** $t$: `MCP(`$t$`)`

devices are simply included in the *uncovered devices* list (line 17). This part of the algorithm provides the actual assignment of the devices which are connected to the mobile cloudlets and are used to calculate the coverage values.

Next, Algorithm 10 performs an extended assignment to assist the cloudlets' mobility (lines 18-22). Here, all devices that could not be covered due to the radius and capacity constraints are assigned to the cloudlets if the devices are within twice the coverage radius of the cloudlet. This is a key feature of the algorithm to make cloudlets more aware of the devices in their vicinity and move better so that even uncovered devices are covered in later time slots. This also prevents mobile cloudlets with a unit coverage radius from getting stuck in a cell with no device since they will be aware of neighboring cells and can move to serve those uncovered devices in later time slots instead of waiting until some device moves into their coverage radius. The output of Algorithm 10 contains a mapping of all mobile cloudlets to their extended device assignments denoted by `DCA`.

Algorithm 9 uses the output of Algorithm 10 to obtain a list of devices assigned to cloudlet $m_j$ at the start of time slot $t$, denoted by $\widehat{\mathcal{D}_j}(t)$ (lines 2-5). Then, Algorithm 9 calculates the total vector pull force acting on mobile cloudlet $m_j$ using the formula given by Equation (6.2).

**Algorithm 10** `DeviceAssignments()`

**Input:** `MCP`
`DCA` $\leftarrow \emptyset$, *uncovered_devices* $\leftarrow \emptyset$
**for** $d_i \in \mathcal{D}$ **do** ▷ Assignments for Coverage
    $min\_distance = \infty$
    $best\_cloudlet = \emptyset$
    **for** $m_j \in \mathcal{M}$ **do**
        $c_k =$ `CloudletCell(MCP,`$m_j$`)`
        $distance = \Delta(c_i^t, c_k)$
        **if** $\mu_j^t \geq \delta_i^t$ and $distance \leq r_j$ **then**
            **if** $distance < min\_distance$ **then**
                $best\_cloudlet = m_j$
                $min\_distance = distance$
    **if** $best\_cloudlet \neq \emptyset$ **then**
        $\mu_j^t = \mu_j^t - \delta_i^t$
        `DCA` $\leftarrow \{m_j, d_i\}$
    **else**
        $uncovered\_devices \leftarrow d_i$
**for** $d_i \in uncovered\_devices$ **do** ▷ Assignments for Mobility
    **for** $m_j \in \mathcal{M}$ **do**
        $distance = \Delta(c_i^t, c_k)$
        **if** $distance \leq 2 * r_j$ **then**
            `DCA` $\leftarrow \{m_j, d_i\}$
**Output:** `DCA`

$$\overrightarrow{F_j}(t) = \frac{\sum_{\forall d_i(t) \in \widehat{\mathcal{D}_j}(t)} \overrightarrow{\eta}\, \delta_i^t}{\sum_{\forall d_i(t) \in \widehat{\mathcal{D}_j}(t)} \delta_i^t} \tag{6.2}$$

Note that force $\overrightarrow{F_j}(t)$ is a vector quantity with component forces in the direction of each 3D axis based on the displacement vector $\overrightarrow{\eta}$. We define $\overrightarrow{\eta} = \Delta(\lambda_i^t, w_j(t-1))$ as the displacement between the position of mobile cloudlet $m_j$ at $t-1$ to the current device position $\lambda_i^t$. Hence, force $\overrightarrow{F_j}(t)$ denotes the expected weighted positional change of mobile cloudlet $m_j$.

Since the mobility of the cloudlets is constrained by the maximum travel parameter $\Lambda_j$, we update the force value. The actual value of the force used in deducing

the new position is given by Equation (6.3).

$$\overrightarrow{F_j}(t) = \begin{cases} \Lambda_j & \text{if } \overrightarrow{F_j}(t) > \Lambda_j \\ \\ \overrightarrow{F_j}(t) & \text{otherwise.} \end{cases} \tag{6.3}$$

Then, the estimated position $\widehat{w_j}(t)$ of mobile cloudlet $m_j$ is given by Equation (6.4).

$$\widehat{w_j}(t) = \overrightarrow{F_j}(t) + w_j(t-1), \tag{6.4}$$

where the vector components of the force are added to the corresponding 3D coordinates of the position of the cloudlet in the previous time slot $w_j(t-1)$. Note that `MCP` from the initial placement (Algorithm 8) is utilized as $w_j(1)$.

After these calculations (lines 6-8), the final position $w_j(t)$ of the cloudlet is determined by finding the closest cell $c_k$ to the estimated position $\widehat{w_j}(t)$ (line 9). The history of cloudlet locations is maintained in `MCP(`$t$`)`, and the device assignments `DCA` are updated for using in the next round (lines 10-11). Every cloudlet moves to its new final position $w_j(t)$, i.e., the center of the cell $c_k$ using `MCP(`$t$`)`.

To summarize, the mobile cloudlets move towards the devices with the highest attractive force and finally relocate to the centers of the cells, closest to their estimated positions in the grid. Algorithm 9 outlines the entire process of the adaptive placement of PMCP. After each adaptive placement iteration, the devices are individually mapped to the relocated mobile cloudlets using Algorithm 10 and used to make the placement decision in the next round.

## 6.3 Experimental Results

### 6.3.1 Experimental Setup

For an extensive evaluation, the experiments on the PMCP approach need to be performed under a realistic user mobility model and appropriate distribution of

changing user demands. We utilize the KTH Walkers dataset [36] to establish mobility scenarios for our experiments. The dataset represents traces of pedestrian mobility from a part of downtown Stockholm. Moreover, the dataset was specifically designed to study mobility models for opportunistic communications, meaning that it is a highly suitable dataset for evaluating our approach. There are four sub-traces (3 Ostermalm traces and 1 Subway trace) in the KTH Walkers dataset, each of which we use to generate one 2D and one 3D mobility dataset for our experiments. We likewise assume a uniform distribution for mobile cloudlet capacities and individual user demands. We experiment with a different number of devices to test the scalability of our approach and deduce the number of mobile cloudlets for each scenario based on this number of devices.

We compare the performance of the PMCP with the optimal solutions by OMCP and the results of DBSCAN with path planning (DBSCAN-PP). DBSCAN-PP is chosen as a benchmark since it is a well-known density-based clustering method and behaves similar to existing mobile cloudlet placement approaches in the literature (e.g., [76]) when we add the path planning component. It is hence a strong benchmark given the unavailability of implementation of the existing studies. We compare our approach with these benchmarks in terms of user coverage attained at each time slot, the rate of cloudlet switches made by the devices (to test the stability of each approach), cumulative distance traveled by the mobile cloudlets, and the running time across different time slots. We also visualize the cloudlet placements to demonstrate the movement of the mobile cloudlets in response to dynamic changes in device specifications.

Table 6.1: Experiment Scenarios - PMCP

| **Scenario** | $\lvert\mathcal{M}\rvert$ | $\lvert\mathcal{D}\rvert$ | $\lvert T\rvert$ | $\mu_j^t$ | $\delta_i^t$ |
|---|---|---|---|---|---|
| Scenario3 (S3-2D/3D) | 16 | 300 | 10 | $U[140, 170]$ | $U[2, 10]$ |
| Scenario2 (S2-2D/3D) | 12 | 180 | 10 | $U[120, 160]$ | $U[2, 10]$ |
| Scenario1 (S1-2D/3D) | 8 | 100 | 10 | $U[80, 100]$ | $U[2, 10]$ |
| Subway (SW-2D/3D) | 8 | 100 | 6 | $U[80, 120]$ | $U[2, 10]$ |

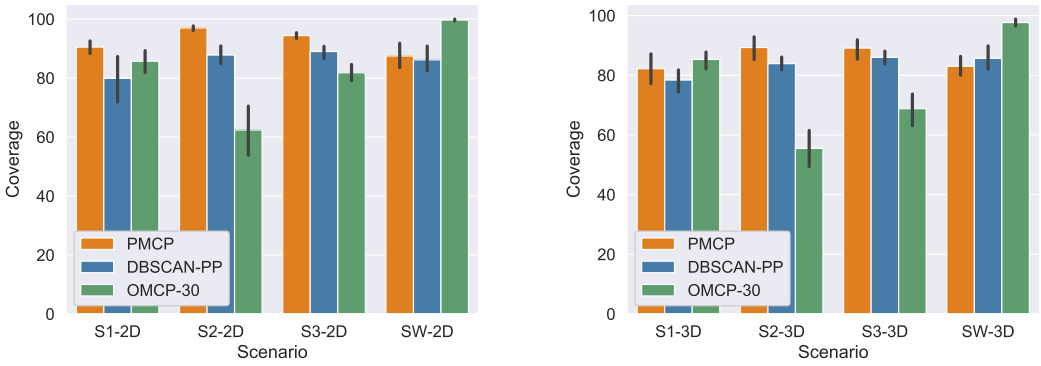
As we can see, Table 6.1 presents the different experiment scenarios in terms of the number of mobile cloudlets available, the number of devices, the number of time slots in the experiment, distribution of cloudlet capacities, and distribution of device demands. For each scenario, both 2D and 3D mobility datasets have been created. 2D datasets are direct representations of available traces, while 3D datasets were generated by assuming a 3D landscape over which mobility happens. This is done by adding non-negative z-coordinates (to denote elevations of the assumed landscape) to existing trace data. In addition, the maximum travel parameter $\Lambda_j$ for all of these experiment scenarios are based on a uniform distribution $U[2, 3]$.

We obtain the optimal results from OMCP by solving the integer program using IBM ILOG CPLEX Concert Technology API for Java [29]. All approaches are implemented in the same version of Java, and all experiments are run on the same JVM on the Nautilus HyperCluster [5] with 16 CPU cores and 64 GB RAM. This justifies a direct comparison between the approaches. It is also noteworthy that OMCP is an NP-hard problem. As such, CPLEX may never converge to a provably optimal result for some experiment scenarios. Hence, the best results obtained within a specified time limit are presented for consistency reasons. Since the optimization time limit is set to 30 minutes in our experiments, we refer to the OMCP results as OMCP-30 hereafter. This means if OMCP could not solve the problem optimally in 30 min, the best obtained result in this duration has been used for comparison.

### 6.3.2 Analysis of Results

#### 6.3.2.1 Coverage

We first compare the coverage values of PMCP, DBSCAN-PP and OMCP-30. Figure 6.2a shows the mean and standard deviations of the 2D scenarios. The mean value of coverage obtained by PMCP across different 2D scenarios and time slots is 92.89% with standard deviation of 4.63. In all scenarios, PMCP outperforms DBSACN-PP in terms of coverage. DBSCAN-PP obtains lower mean coverage of 85.67% with standard deviation of 8.3. As an NP-hard problem, OMCP does not




(a) Coverage for 2D Scenarios

(b) Coverage for 3D Scenarios

Figure 6.2: PMCP: Comparison of average coverage values.

guarantee termination with an optimal solution in a finite time. As the problem size increases in terms of the number of devices and the number of time slots, it is more difficult to obtain the optimal solutions or even high-coverage time-limited solutions using OMCP-30. This is most noticeable in Scenario 2 (S2-2D and S2-3D).

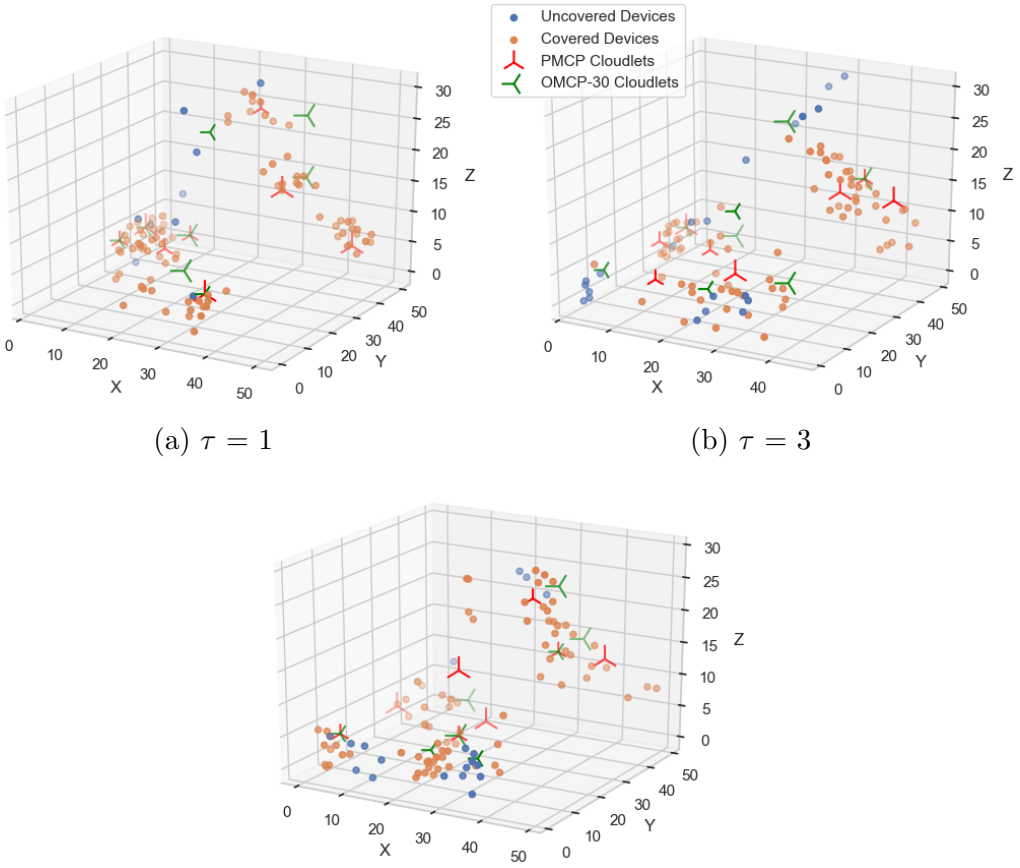
Figure 6.2b shows the coverage results of the 3D scenarios. The mean coverage obtained by PMCP is lower than the 2D scenarios but still high at 86.22% with standard deviation of 7.14. DBSCAN-PP is more consistent with a lower standard deviation of 5.59, however, it again achieves lower mean coverage of 83.25%. The coverage values obtained by PMCP are consistently more than DBSCAN-PP and OMCP-30 with lower standard deviations except for a few cases as seen in Figure 6.2. The Subway scenarios (SW-2D and SW-3D) and Scenario1-3D (S1-3D) are exceptions to this observation where OMCP-30 has higher coverage with lower standard deviation. This is because the size of these scenarios is much smaller. They either have a smaller number of devices or fewer time slots or both. Hence, it is easier for an exhaustive approach like OMCP-30 to find values closer to the optimal solution. DBSCAN-PP performs better in terms of coverage in Subway-3D (SW-3D) scenario since the scenario has distinct dense clusters, but PMCP is still comparable given a large overlap in the error bars of both approaches. Moreover, cloudlets can move more freely in DBSCAN-PP since new cluster centers can be far (larger than $\Lambda_j$) from current cloudlet cells, which may

lead to infeasible placements.

#### 6.3.2.2 Adaptive Nature

We visualize the cloudlet placements to illustrate smooth adaptive movement of cloudlets in PMCP in response to the devices' movements. In Figure 6.3, we present the comparative cloudlet placements according to PMCP (red) and OMCP-30 (green) along with the devices (blue and orange based on PMCP coverage) across different time slots of the Subway 3D (SW-3D) scenario. The device distributions across time slots first show that the Subway 3D scenario realistically depicts the visible 3D features in subway stations like stairs and escalators at the entrance and the exit, and the area around and outside the station. In addition, Figure 6.3a shows the initial placement at $\tau = 1$, which looks better for PMCP since it places cloudlets (red) close to the demand centers (with coverage of 91%). Hence, the placement of cloudlets by PMCP appear closer on average to the devices than the placement by OMCP-30 (green). There are some exact overlaps between PMCP cloudlets and OMCP-30 cloudlets too. In comparison, OMCP-30 cloudlets are placed more strategically to cover a larger number devices (about 95%) but are not close to the devices in some of their locations.

As the devices move, both approaches readjust the placements of the cloudlets. At $\tau = 3$ (see 6.3b), the placements by OMCP-30 look better based on the distribution of devices. The coverage of OMCP-30 here is 96% compared to only 78% for PMCP. However, PMCP has significantly readjusted cloudlet placements based on the new user distribution. As the devices have scattered more, so have the PMCP cloudlets. Moving into the final time slot $\tau = 6$, Figure 6.3c shows that PMCP has caught up significantly in terms of following the devices and has placed cloudlets close to device clusters at the bottom-left and the top-right (notice the wider coverage in the figure), which did not have nearby PMCP cloudlets in $\tau = 3$. The visualizations here validate that PMCP places the mobile cloudlets by truly responding to dynamic and uncertain changes in device distribution at every time slot and achieves consistent coverage in comparison to the optimal coverage. They also demonstrate that PMCP does not favor




(a) $\tau = 1$ (b) $\tau = 3$

(c) $\tau = 6$

Figure 6.3: Visualization of comparative cloudlet placements and PMCP coverage at different time slots.

the devices in dense regions only and adapts to even sparse regions over time. It fairly serves all devices in long-term since the devices far away from the dense regions are eventually covered too.

#### 6.3.2.3 Stability

Providing stable services is also critical as it reduces excessive migration that will lead to high latency. Both high coverage and stable service are the motivations behind the design of our approach. As a result, our approach compensates for the coverage gap by improving the stability of the services. We measure the stability in terms of the *switching rate* for devices. The switching rate is the total number of

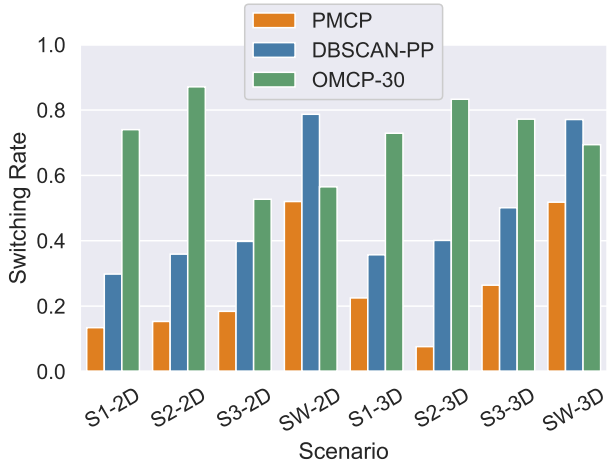



Figure 6.4: PMCP: Comparison of cloudlet switching rates.

cloudlets switched by the devices between consecutive time slots divided by the total number of connections established by the devices across all time slots. Thus, a lower switching rate means the devices are likely to be served by the same cloudlet across consecutive time slots which leads to fewer migrations between cloudlets and ultimately to smoother, more stable edge services. Figure 6.4 shows the switching rate of the three approaches across all experiment scenarios.

The average switching rate of PMCP, DBSCAN-PP, and OMCP-30 are 0.259, 0.484, and 0.716, respectively. This means only 25.9% of the connected devices change cloudlets between the two time slots in PMCP while a more significant number of devices 48.4% and 71.6% do that in DBSCAN-PP and OMCP-30, respectively. The high rate obtained by OMCP-30 is because it is agnostic to switching and objectively focuses on maximizing coverage, while DBSCAN-PP has path planning to minimize movement of cloudlets between clusters which offers some advantage. Hence, even in the higher coverage scenarios, users using DBSCAN-PP and OMCP-30 experience more delay for their services. Though we can make OMCP-30 more adept at handling switching, introducing hard constraints to limit switching in OMCP-30 may even render some scenarios theoretically infeasible to solve. This further supports the performance of PMCP in terms of coverage since OMCP-30's coverage values are extremely high

benchmarks to compare against. Among all scenarios, the only comparable switching rate between the PMCP and OMCP-30 can be noticed for SW-2D scenario. This indicates that OMCP-30 probably found a solution with less switching by chance. Otherwise, PMCP always has a significantly lower switching rate, consequently, ultra-low latency services, compared to both DBSCAN-PP and OMCP-30.

#### 6.3.2.4 Energy Efficiency

Given that the mobile cloudlets are deployed for the same distribution of devices and assuming all other factors constant, the energy consumption of the UAV-mounted mobile cloudlets largely depends on the total distance traveled by them. Hence, we assess the energy efficiency using the cumulative distance traveled by the cloudlets in terms of unit cells, i.e., the distance between centroids of the unit cells switched by the cloudlets across all time slots. As shown in Figure 6.5, PMCP is extremely efficient in terms of cloudlet movements for all scenarios. The mean cumulative distance traveled by the cloudlets are 110.84, 525.69, and 887.95 for PMCP, DBSCAN-PP, and OMCP-30, respectively across all scenarios. For some scenarios, the cloudlets have negligible movement between cells in PMCP, notably S1-2D and S3-2D. This is because in PMCP, cloudlets do not move to a different cell unless there is a significant change in the user

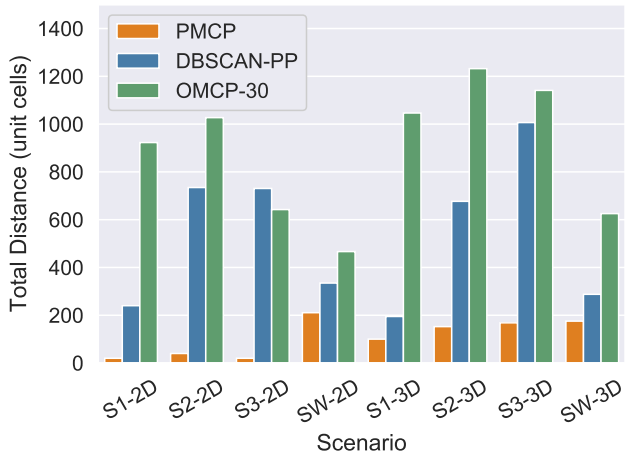



Figure 6.5: PMCP: Comparison of cloudlet distance traveled (cell units).

distribution in and around its covered cells. DBSCAN-PP is significantly worse since it re-clusters the devices in each time slot and moves cloudlets with no maximum travel constraint to newly established cluster centers based on matching. Unrestricted movement with poorer energy efficiency makes DBSCAN-PP impractical. These are obvious limitations of the existing approaches in the literature that are based on similar principles. OMCP-30 is again agnostic to minimizing the distance traveled and purely focuses on maximizing coverage.

#### 6.3.2.5 Running Time

We compare the running time of PMCP against DBSCAN-PP and OMCP-30 (i.e., 30 minute time limit shown by the flat line). Illustrated by Figure 6.6, PMCP has exceptionally low running time compared to OMCP-30. DBSCAN-PP has a similar running time, however, it should be noted that DBSCAN needs to be tuned using two hyperparamters ($\epsilon$ and $minPoints$) and the running time shown here is for a tuned version of DBSCAN. If there are $n$ combinations of hyperparameters considered for tuning DBSCAN, then obtaining the presented results would take $n$ times the presented running time. On the contrary, PMCP runs in a few milliseconds with total running time for any scenario never exceeding 56 milliseconds. It should again be

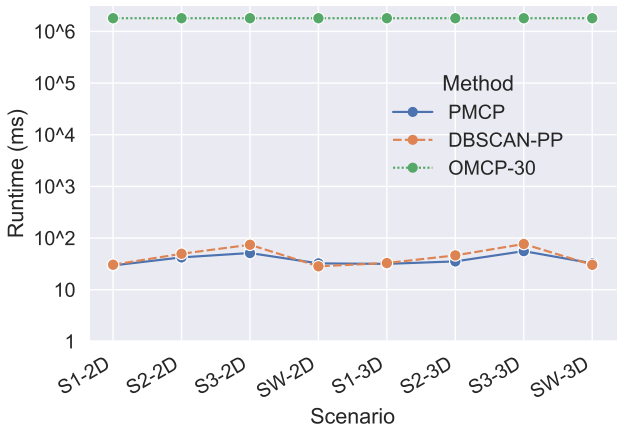



Figure 6.6: PMCP: Comparison of total running time (ms).

noted that most of this time is spent on the initial placement of the mobile cloudlets, where mobility decisions are not yet being made. Also, this figure shows the sum of the running times across all time slots (sum of 6-10 time slots). Calculations in the adaptive part of the approach only take close to 1-3 milliseconds for each time slot, even for the largest experimental scenario.

In summary, PMCP offers consistent high device coverage, stable services through low switching rate, and markedly superior energy efficiency through reduced cumulative distance traveled by the cloudlets. PMCP is well designed to run in real time and scales well across scenarios.

## 6.4 Chapter Summary

Deployment of mobile cloudlets is an elegant solution to provide better services in next-generation edge networks with high user mobility and dynamic, uncertain demands. To address the goals of enhancing user coverage and providing stable edge services, we designed a novel dynamic mobile cloudlet placement approach, PMCP, inspired by the concepts in physics. The results show that PMCP not only achieves high coverage and more stable services but is also energy-efficient and runs in real time. This means our approach is practical for dynamic cloudlet placements and can also be extended to solve dynamic assignment and resource placement problems. Since the approach includes iterative calculations, it can also be parallelized to run faster. The scalability can be further enhanced using a distributed approach for dynamic cloudlet placements which we study in the next chapter.

The completed manuscript of this chapter has been accepted for publication in proceedings of the IEEE International Conference on Edge Computing and Communications (EDGE) 2022.

# Chapter 7

# A DISTRIBUTED MOBILE CLOUDLET PLACEMENT

## 7.1 Background

As an efficient approach, PMCP offers many advantages such as smooth adaptivity to device mobility, stability, energy efficiency, and high coverage. Centralized approaches like PMCP gather all information from all system entities and use them to compute the decisions that are globally better overall. However, as the frequency of the sensor data increases and the environment gets larger, dynamic, and more complex, quality decisions are more difficult to produce [38]. As a centralized approach, PMCP suffers from single point of failure if the link to the central decision entity is disrupted or if the central entity fails itself. Furthermore, complete information about the system may not be available if one or more sensors fail. Centralized decisions likewise take more time as the problem size grows which makes it difficult to ensure that all decisions are taken within a bounded time threshold.

Recent literature in distributed systems support that distributed decision-making algorithm designs can replace centralized decision making to achieve faster response and higher reliability [38]. Furthermore, a distributed approach maintains consistent performance as the problem size scales up. As a distributed computing paradigm, distributed approaches perfectly align with edge computing, and especially to mobile cloudlet placement problem since the UAV-mounted mobile cloudlets are fully distributed and make movements with local awareness. Hence, in this chapter, we present a distributed mobile cloudlet placement approach by extending the PMCP approach presented in Chapter 6. The distributed approach hence maintains the key advantages of the PMCP approach while addressing the concerns of single point of failure, slower decision time, and scalability.

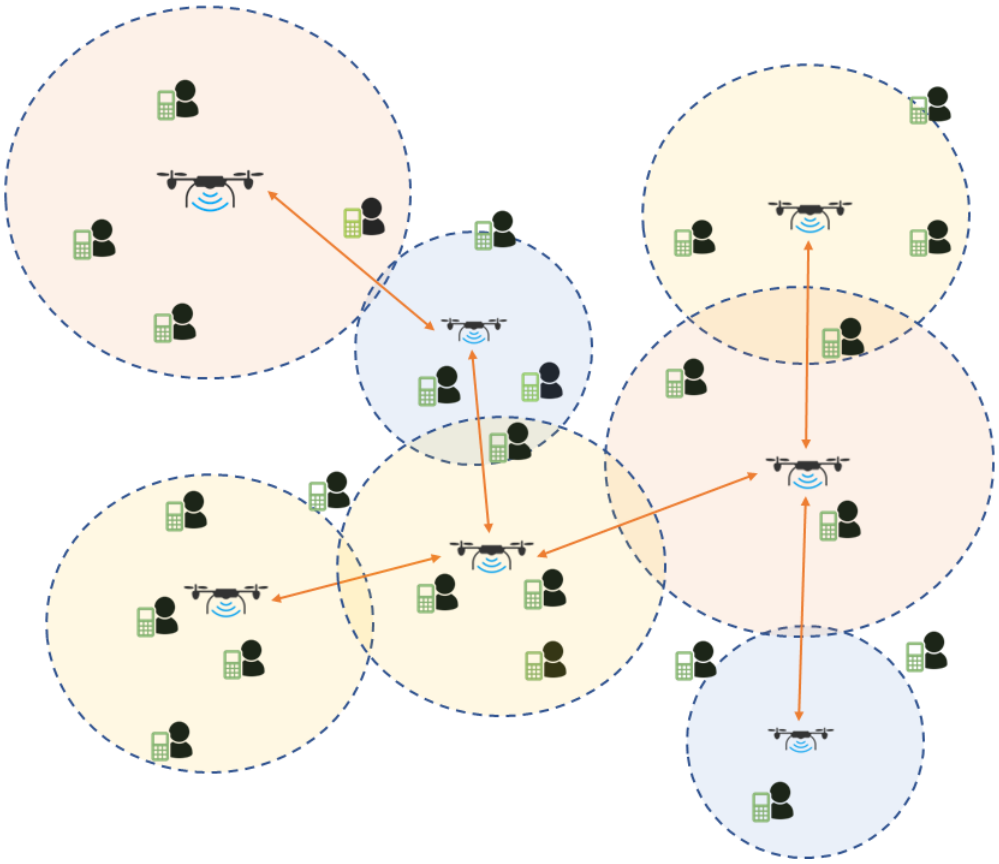


Figure 7.1: Communication between neighboring mobile cloudlets

## 7.2 Distributed Mobile Cloudlet Placement

We propose a novel distributed mobile cloudlet placement (DMCP) algorithm to perform adaptive placement of mobile cloudlets based on the changes in user distribution and demands. As shown in Figure 7.1, mobile cloudlets are distributed across a service area (shown in 2D for simplicity). Each mobile cloudlet can directly communicate with its neighboring cloudlets if they have an overlapping communication region. The communication radius is assumed to be larger than their 3D coverage since neighborhood communication does not require ultra-low latency. Each mobile cloudlet is able to multicast the list of devices within its 3D coverage. Therefore, other cloudlets in its neighborhood can receive this information. The neighbors can thus exchange information to have a better picture of the devices in their extended surroundings. The cloudlets can then make local decisions based on the information available to them.

### 7.2.1 Initial Placement

The initial placement in DMCP is performed based on an observational basis. The mobile cloudlets start sensing devices as they are deployed. When they sense a few devices in an empty cell, they confirm that cell as their initial position. It is important to note that the cloudlets avoid moving into cells where other cloudlets are already initialized since they can sense other cloudlets too. The overall initial placement is done in a random order, and cloudlets are deployed from different locations to allow a uniform placement across perceived clusters of devices. This step should be taken more as deploying the mobile cloudlets to prepare for service rather than actual placement to serve the devices. Nonetheless, the cloudlets can start serving the devices straightway and provide significant starting coverage when initialized in this manner. Also note that, unlike PMCP, the cloudlets in DMCP are assigned indexes from a reserved index pool (for each specific range of capacities) in an increasing order of their size whenever they are deployed. Once all mobile cloudlets are ready, the distributed adaptive placement of DMCP begins.

### 7.2.2 Distributed Adaptive Placement

The adaptive part of DMCP allows individual decision makings for each mobile cloudlet as shown in Algorithm 11. Every mobile cloudlet $m_j$ follows a sequence of actions outlined in the algorithm as long as it is *active* (line 1), meaning it is functioning at the expected capacity.

**Sense:**

Cloudlet $m_j$ discovers the devices within its 3D coverage $r_j$ and populates the list of its locally discovered devices $\widehat{\mathcal{D}_j^{out}}(t)$ (line 2). This is the outgoing information from the cloudlet.

**Exchange:**

The exchange of messages between the cloudlets happens (lines 3-4). The cloudlet multicasts the list of devices $\widehat{\mathcal{D}_j^{out}}(t)$ to their neighbors. Note that the neighbors

**Algorithm 11** DMCP - Distributed Placement

Repeat the following for mobile cloudlet $m_j \in \mathcal{M}$

1: **while** $m_j$ is *active* **do**

▷ **Sense:**

2: $\widehat{\mathcal{D}_j^{out}}(t) \leftarrow$ Devices within $r_j$

▷ **Exchange:**

3: Multicast $\widehat{\mathcal{D}_j^{out}}(t)$ to neighbors

4: Receive $\widehat{\mathcal{D}_j^{in}}(t)$ from neighbors

5: $\widehat{\mathcal{D}_j^{temp}}(t) = \widehat{\mathcal{D}_j^{out}}(t) \cup \widehat{\mathcal{D}_j^{in}}(t)$

▷ **Select:**

6: **for** $d_i \in \widehat{\mathcal{D}_j^{temp}}(t)$ **do**

7: **if** $d_i$ within $\theta * r_j$ **then**

8: Add $d_i$ to $\widehat{\mathcal{D}_j}(t)$

▷ **Decision:**

9: Calculate force $\overrightarrow{F_j}(t)$ using Eqn (7.1)

10: Assign constrained force $\overrightarrow{F_j}(t)$ to $m_j$ using Eqn (7.2)

11: Find estimated location $\widehat{w_j}(t)$ of $m_j$ using Eqn (7.3)

12: $w_j(t) =$ `ClosestCell(`$\widehat{w_j}(t)$`)`

▷ **Conflict Resolution:**

13: Multicast $w_j(t)$ to neighbors

14: Receive $w_j^{in}(t)$ from neighbors

15: Add $w_j^{in}(t)$ conflicting with neighbors to $W_j^{in}(t)$

16: **if** $W_j^{in}(t) == \emptyset$ **then**

17: $w_j^{final}(t) \leftarrow w_j(t)$

18: **else**

19: $w_j^{final}(t) \leftarrow$ `ResolveConflict`$(W_j^{in}(t))$

▷ **Move:**

20: Move to $w_j^{final}(t)$

are the mobile cloudlets with an overlapping communication radius with cloudlet $m_j$. The lists of all devices received from all the neighbors are maintained in a singular list $\widehat{\mathcal{D}_j^{in}}(t)$. Once the exchange is complete, a temporary list of devices $\widehat{\mathcal{D}_j^{temp}}(t)$ is created (line 5) to guide the selection step. It is important to note that duplicates are discarded when building $\widehat{\mathcal{D}_j^{temp}}(t)$ so that every sensed device is evaluated only once.

**Select:**

The process of selecting devices for decision making is represented in lines 6-8. This process uses the temporary list of devices $\widehat{\mathcal{D}_j^{temp}}(t)$ as a reference since these are all devices known to the cloudlet that are available for selection. The cloudlet selects all devices $d_i$ within $\theta * r_j$ (line 7). Here, $\theta$ is the selection factor which can be determined based on the environment and the implementation requirements. We assume that $theta \geq 1$ since the cloudlets should at least consider all the devices within its coverage $r_j$. The devices eventually selected for decision making are stored in $\widehat{\mathcal{D}_j}(t)$.

**Decision:**

Now that the device selections are finalized, all cloudlets can proceed to making their local decisions using the PMCP approach presented in Chapter 6. To revise the idea, there is a *force of attraction* between two bodies and we formulate the force of attraction as a *vector pull force* from the devices acting on the mobile cloudlets which attracts them to their new locations.

We use this physics concept in our approach by considering the devices as moving masses (which need to be served) since they continuously change positions between time slots. In addition, we consider the mobile cloudlets as fixed masses (that are deployed to serve devices) since they are assumed to be at rest in their placed cells until pulled by the changing device demands and mobility into new cells.

Note that the equations were already presented in Chapter 6. However, we present them here again for readability of the DMCP algorithm. The cloudlet calculates the total vector pull force (line 9) using the formula given by Equation (7.1).

$$\overrightarrow{F_j}(t) = \frac{\sum_{\forall d_i(t) \in \widehat{\mathcal{D}_j}(t)} \overrightarrow{\eta}\, \delta_i^t}{\sum_{\forall d_i(t) \in \widehat{\mathcal{D}_j}(t)} \delta_i^t} \tag{7.1}$$

We define $\overrightarrow{\eta} = \Delta(\lambda_i^t, w_j(t-1))$ as the displacement between the position of the mobile cloudlet at $t-1$ to the current device position $\lambda_i^t$. Hence, force $\overrightarrow{F_j}(t)$ denotes the expected weighted positional change of mobile cloudlet $m_j$. Note that force $\overrightarrow{F_j}(t)$ is a

vector quantity with component forces in the direction of each 3D axis based on the displacement vector $\overrightarrow{\eta}$.

Since the mobility of the cloudlet is constrained by the maximum travel parameter $\Lambda_j$, we update the force value (line 10). The actual value of the force used in deducing the new position is given by Equation (7.2).

$$\overrightarrow{F_j}(t) = \begin{cases} \Lambda_j & \text{if } \overrightarrow{F_j}(t) > \Lambda_j \\ \\ \overrightarrow{F_j}(t) & \text{otherwise.} \end{cases} \tag{7.2}$$

Then, the estimated position $\widehat{w_j}(t)$ (line 11) of mobile cloudlet $m_j$ is given by Equation (6.4).

$$\widehat{w_j}(t) = \overrightarrow{F_j}(t) + w_j(t-1), \tag{7.3}$$

where the vector components of the force are added to the corresponding 3D coordinates of the position of the cloudlet in the previous time slot $w_j(t-1)$. Note that the initial placement is utilized as $w_j(1)$.

After these calculations (lines 9-11), the destination position $w_j(t)$ of the cloudlet is determined by finding the closest cell $c_k$ to the estimated position $\widehat{w_j}(t)$ (line 12). However, it is possible that other cloudlets might have calculated the same cell as their destination position. Hence, the cloudlet needs to resolve any conflicts before moving.

**Conflict Resolution:**

To resolve any conflict, the cloudlet sends $w_j(t)$ and receives $w_j^{in}(t)$ from their neighbors (lines 13 and 14). The received conflicting positions (having the same destination) and related cloudlets are maintained in a hash table $W_j^{in}(t)$ which contains the cloudlet mapped to its current position and the conflicting destination position (line 15).

**Algorithm 12** `ResolveConflict()`

1: **Input:** $W_j^{in}(t)$
2: $m_{top} =$ `maxIDCloudlet`$(W_j^{in}(t))$
3: **if** $m_j = m_{top}$ **then**
4: $\quad w_j^{final}(t) = w_j(t)$
5: **else**
6: $\quad$ **if** $w_{top} = w_j(t-1)$ **then**
7: $\quad\quad w_j^{final}(t) = w_{top}(t-1)$
8: $\quad$ **else**
9: $\quad\quad w_j^{final}(t) = w_j(t-1)$
10: **Output**: $w_j^{final}(t)$

If the cloudlet does not have any conflicts, it can readily move to its calculated destination position (lines 16-17).

To resolve the conflicts based on the accumulated hash table of conflicts positions $W_j^{in}(t)$, the cloudlet calls `ResolveConflict` function, presented in Algorithm 12 (line 19). using this function, cloudlet $m_j$ first finds a cloudlet from the hash table with the highest index $m_{top}$, including itself (the calculating cloudlet) (line 2). The cloudlet indexes are ordered based on the size. This is to move larger cloudlets with the expectation of providing more coverage and stability. If the calculating cloudlet is the one with the highest index, it can move to the its destination position since it has the highest priority to move (lines 3-4). Otherwise, the cloudlet will check if the top cloudlet $m_{top}$ wants to move to its current position $w_j(t-1)$. If that is the case, cloudlet $m_j$ has to vacate the current cell by exchanging its position with $m_{top}$. Therefore, $w_{top}(t-1)$ indicates the new position of the cloudlet (lines 6-7). Finally, if no previous conditions occur, the cloudlet can avoid conflict by simply staying in its current position (line 9) to make way for the larger cloudlet to move. The function returns the final position $w_j^{final}(t)$.

**Move:**

Now that the conflicts are resolved, cloudlet $m_j$ finally moves to its new position $w_j^{final}(t)$.

To summarize, the mobile cloudlets move towards the devices with the highest attractive force and finally relocate to the centers of the cells, closest to their estimated positions in the grid. Algorithm 11 outlines the entire process of the adaptive placement of DMCP. After each adaptive placement iteration, we assume the devices within 3D coverage of a mobile cloudlet automatically connect to the cloudlet as long as sufficient capacity is available on an opportunistic basis. The placement decisions are made again after a certain time window based on the environment.

## 7.3 Experimental Results

### 7.3.1 Experimental Setup

We require a realistic user mobility model and appropriate distribution of changing user demands for an extensive evaluation of the DMCP approach. Hence, just like PMCP approach, we use the KTH Walkers dataset [36] to generate multiple mobility scenarios for our experiments since it is a highly suitable dataset for evaluating mobility dependent approaches like DMCP. The scenarios used for experiments are the same scenarios we used to evaluate PMCP, based on four sub-traces (3 Ostermalm traces and 1 Subway trace) in the KTH Walkers dataset, each of which we use to generate one 2D and one 3D mobility dataset for our experiments. We likewise assume a uniform distribution for mobile cloudlet capacities and individual user demands. We experiment with a different number of devices to test the scalability of our approach and deduce the number of available mobile cloudlets for each scenario based on this number of devices.

Table 7.1: Experiment Scenarios - DMCP

| **Scenario** | $\lvert\mathcal{M}\rvert$ | $\lvert\mathcal{D}\rvert$ | $\lvert T\rvert$ | $\mu_j^t$ | $\delta_i^t$ |
|---|---|---|---|---|---|
| Scenario3 (S3-2D/3D) | 16 | 300 | 10 | $U[140, 170]$ | $U[2, 10]$ |
| Scenario2 (S2-2D/3D) | 12 | 180 | 10 | $U[120, 160]$ | $U[2, 10]$ |
| Scenario1 (S1-2D/3D) | 8 | 100 | 10 | $U[80, 100]$ | $U[2, 10]$ |
| Subway (SW-2D/3D) | 8 | 100 | 6 | $U[80, 120]$ | $U[2, 10]$ |

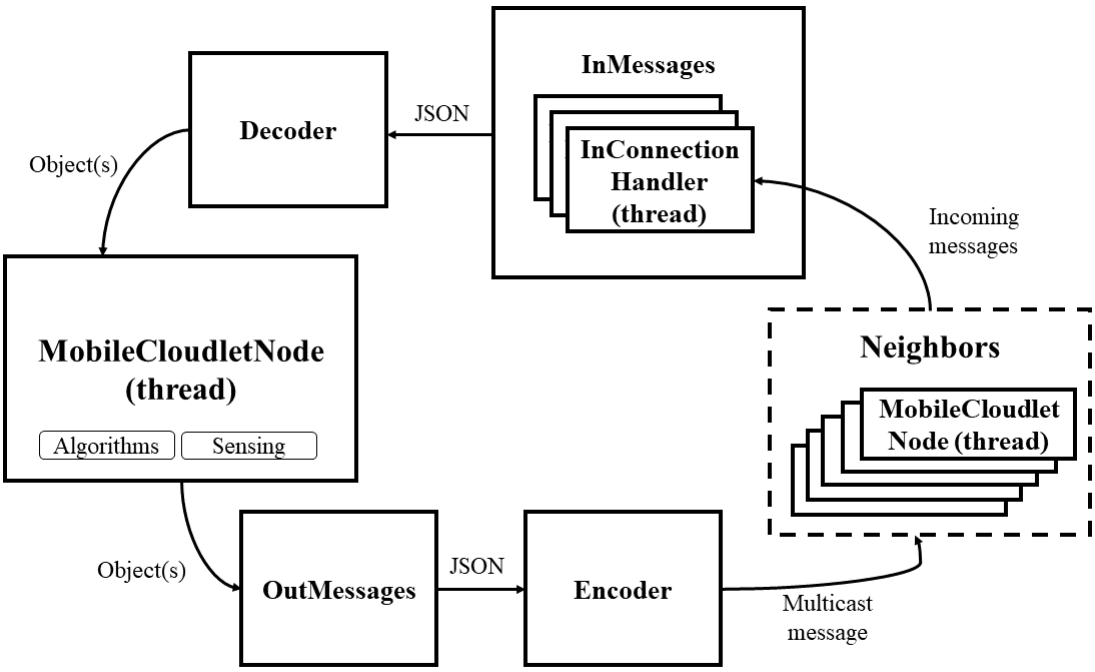



Figure 7.2: DMCP Simulation Architecture

We compare the performance of the DMCP with the solutions obtained by OMCP and PMCP–the centralized version of DMCP. We further include the K-Means approach as an additional benchmark since it is one of the most popular clustering approaches and can easily be embedded alongside a general sequence of actions in DMCP by performing 1-Mean in the *Decision* step. This provides a distributed K-Means approach (D-KMEANS) to compare with our distributed DMCP approach. All of these are strong benchmarks given the unavailability of other suitable distributed algorithmic benchmarks for the mobile cloudlet placement problem. We compare our approach with these benchmarks in terms of user coverage attained at each time slot, the rate of cloudlet switches made by the devices (to test the stability of each approach), cumulative distance traveled by the mobile cloudlets, and the running time across different time slots. We also visualize the service fairness to demonstrate how fairly each device is covered as it moves, connects, disconnects, and reconnects across multiple time slots.

As we can see, Table 6.1 presents the different experiment scenarios in terms of the number of mobile cloudlets available, the number of devices, the number of time slots in the experiment, distribution of cloudlet capacities, and distribution of device demands. For each scenario, both 2D and 3D mobility datasets have been created. 2D datasets are direct representations of available traces, while 3D datasets were generated by assuming a 3D landscape over which mobility happens. This is done by adding non-negative z-coordinates (to denote elevations of the assumed landscape) to existing trace data. In addition, the maximum travel parameter $\Lambda_j$ for all of these experiment scenarios are based on a uniform distribution $U[2, 3]$, the communication radius is the same as the coverage radius, and the selection factor $\theta = 1.5$.

### 7.3.2 Simulation Details

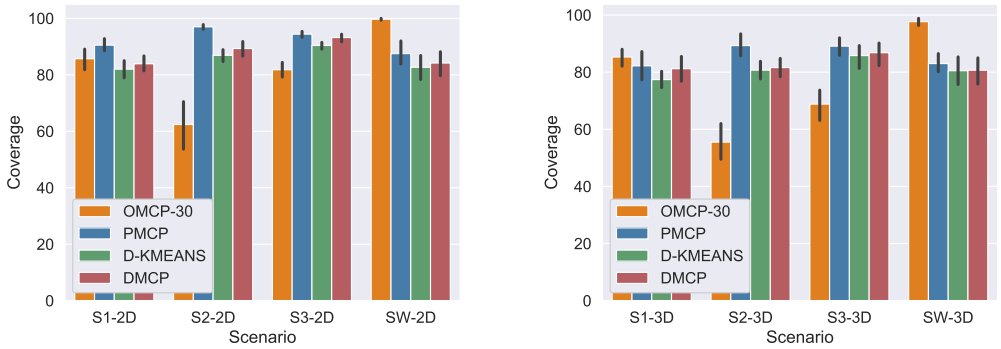
Since DMCP and D-KMEANS are distributed approaches, we built a simulator in Java to accurately evaluate the approaches. The overall architecture of the simulator is depicted in Figure 7.2. As shown in the figure, each mobile cloudlet is a thread that communicates with its neighbors via TCP sockets. Note that DMCP and D-KMEANS algorithms and the sensing module are embedded in each *MobileCloudletNode*. Since the nodes exchange information to run the embedded algorithms consisting of different data structures, we standardize the outgoing messages and incoming messages as JSON objects. This is due to the fact that JSON is lightweight and can be easily encoded and decoded when communicating using TCP sockets. Outgoing messages, once encoded, are multicasted to the neighboring nodes. On the other hand, incoming messages need a connection handler to receive messages coming from multiple neighbors to maintain concurrency. The simulator represents a semi-synchronous environment, where the nodes wait to receive messages from all neighbors before performing their calculations. However, they also have a timeout implemented to start the calculations if messages are not received by that time. Hence, the simulator provides a smooth communication and the ability to scale the experiments to a large number of nodes.

We obtain the optimal results from OMCP by solving the integer program using IBM ILOG CPLEX Concert Technology API for Java [29]. All approaches are implemented in the same version of Java, and all experiments are run on the same JVM on a standalone workstation with Intel Core i7 processor @2.70 GHz and 16GB of RAM. This justifies a direct comparison between the approaches. Since, DMCP and D-KMEANS rely on randomized initialization, the experiments for DMCP and D-KMEANS are repeated numerous times to aggregate the result values and provide a better representation of their performance. It is also noteworthy that OMCP is an NP-hard problem. As such, CPLEX may never converge to a provably optimal result for some experiment scenarios. Hence, the best results obtained within a specified time limit are presented for consistency reasons. Since the optimization time limit is set to 30 minutes in our experiments, we refer to the OMCP results as OMCP-30 hereafter. This means if OMCP could not solve the problem optimally in 30 min, the best obtained result in this duration has been used for comparison.

### 7.3.3 Analysis of Results

#### 7.3.3.1 Coverage

We first compare the coverage values of DMCP, PMCP, D-KMEANS, and OMCP-30. Figure 6.2a shows the mean and standard deviations of the 2D scenarios. The mean value of coverage obtained by DMCP across different 2D scenarios and time slots is 88.03% with standard deviation of 5.71. In all scenarios, PMCP outperforms DMCP in terms of coverage since as a centralized approach it has complete access to all the values and decision parameters to make decisions. D-KMEANS, however, obtains lower mean coverage of 85.80% with standard deviation of 5.39. As an NP-hard problem, OMCP does not guarantee termination with an optimal solution in a finite time. We can notice in Scenario 2 (S2-2D and S2-3D) that OMCP-30 is unable to attain good coverage as the problem size increases in terms of the number of devices and the number of time slots.



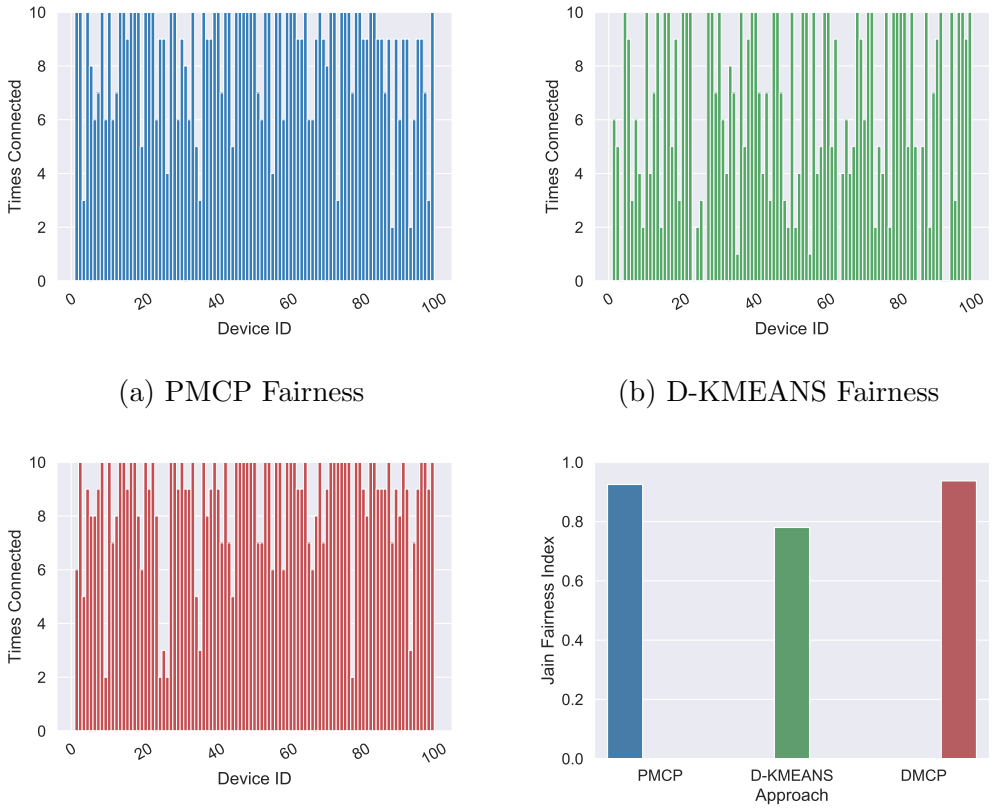

(a) Coverage for 2D Scenarios (b) Coverage for 3D Scenarios

Figure 7.3: DMCP: Comparison of average coverage values.

Figure 6.2b shows the coverage results of the 3D scenarios. The mean coverage obtained by DMCP is lower than the 2D scenarios but still above 80% at 82.77% with standard deviation of 6.78. DMCP is actually more consistent than PMCP which has a higher coverage at 86.22% but also higher standard deviation of 7.14. D-KMEANS again achieves lower mean coverage of 81.16%, although only marginally lower. The coverage values obtained by DMCP are lower but consistently closer to PMCP and outperform D-KMEANS as seen in Figure 6.2. The S2 scenarios (S2-2D and S2-3D) are exceptions to this observation where PMCP has much higher coverage values. This is because of a large number of conflicts occurring during the distributed placement in DMCP (and D-KMEANS). It should be noted that cloudlets can move more freely in D-KMEANS since new cluster centers can be far (larger than $\Lambda_j$) from current cloudlet cells, which may lead to infeasible placements.

#### 7.3.3.2 Service Fairness

One of the major concerns of the placement approaches in dense and dynamic environments is that many devices may not be able to access the resources due to intermittent connectivity, and many devices might be inadvertently favored. This leads to an uneven user experience. We visualize the service fairness of DMCP along with the compared approaches to show that all devices are able to fairly connect to




(a) PMCP Fairness

(b) D-KMEANS Fairness

(c) DMCP Fairness

(d) Jain Fairness Index

Figure 7.4: Service fairness: Visualization and Jain Fairness Index

the mobile cloudlets as they move, connect, disconnect, and reconnect to the mobile cloudlets over different time slots.

Figure 7.4 shows the number of times each device is able to connect to a placed cloudlet over 10 time slots of Scenario1-3D (S1-3D). DMCP and PMCP, both physics-inspired adaptive approaches, provide connectivity to all devices at least twice, whereas in D-KMEANS as much as 7% of the devices are never covered by any cloudlet. In both DMCP and PMCP, easily more than 80% of the devices are connected for 6 or more time slots. Moreover, in DMCP, more than 45% of the devices are covered in all 10 time slots (Figure 7.4c) which is again comparable to the centralized PMCP approach (Figure 7.4a). This kind of complete coverage is only about 38% in D-KMEANS (Figure 7.4b). Similarly, we calculated the Jain Fairness Index [30] of the visualized values

to quantify fairness in connectivity, which are shown in Figure 7.4d. The Jain Fairness Index for PMCP, D-KMEANS, and DMCP are 0.925, 0.780, and 0.937 respectively. It means 93.7% of devices are connected fairly in DMCP whereas only 78% devices experience fair connectivity in D-KMEANS. The visualizations and Jain Fairness Index values validate that DMCP, similar to PMCP, places the mobile cloudlets by truly responding to dynamic and uncertain changes in device distribution at every time slot. The results also demonstrate that DMCP does not favor the devices in dense regions only, and it adapts to even sparse regions over time. DMCP fairly serves all devices in the long-term because the devices far away from the dense regions are eventually covered too.

#### 7.3.3.3 Stability

Providing stable services is also critical as it reduces excessive migration that will lead to high latency. Both high coverage and stable service are the motivations behind the design of our approach. As a result, our approach compensates for the coverage gap by improving the stability of the services. We measure the stability in terms of the *switching rate* for devices. The switching rate is the total number of cloudlets switched by the devices between consecutive time slots divided by the total number of connections established by the devices across all time slots. Thus, a lower switching rate means the devices are likely to be served by the same cloudlet across consecutive time slots which leads to fewer migrations between cloudlets and ultimately to smoother, more stable edge services. Figure 6.4 shows the switching rate of the three approaches across all experiment scenarios.

The average switching rate of DMCP, PMCP, D-KMEANS, and OMCP-30 are 0.368, 0.259, 0.345, and 0.716, respectively. This means about 36.8% of the connected devices change cloudlets between the two time slots in DMCP. In comparison, only 25.9% of the connected devices change cloudlets between the two time slots in PMCP and slightly lower 34.5% in D-KMEANS. A more significant number of devices 71.6% do that in OMCP-30. The high rate obtained by OMCP-30 is because it

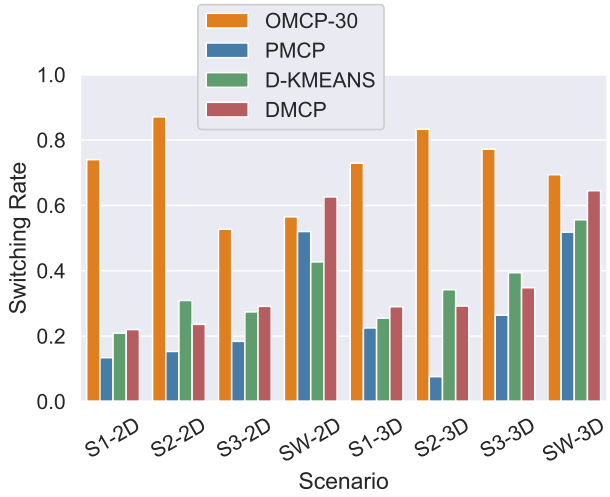



Figure 7.5: DMCP: Comparison of cloudlet switching rates.

is agnostic to switching and objectively focuses on maximizing coverage. D-KMEANS places cloudlets at the center of device clusters and is likely to end up closer to the same devices, which we noticed as a worse service fairness earlier. Hence, even in the higher coverage scenarios, users under OMCP-30 placement experience more delays for their services. PMCP again outperforms DMCP as a centralized approach. However, it should be noticed that DMCP has a higher standard deviation, and one or two instances have pushed its average upwards. It still closely matches PMCP, and more than 60% of the devices experience stable service under DMCP placements.

#### 7.3.3.4 Energy Efficiency

Given that the mobile cloudlets are deployed for the same distribution of devices and assuming all other factors constant, the energy consumption of the UAV-mounted mobile cloudlets largely depends on the total distance traveled by them. Hence, we assess the energy efficiency using the cumulative distance traveled by the cloudlets in terms of unit cells, i.e., the distance between centroids of the unit cells switched by the cloudlets across all time slots. As shown in Figure 6.5, PMCP is extremely efficient in terms of cloudlet movements for all scenarios. However, DMCP is not too far. The mean cumulative distance traveled by the cloudlets are 142.89, 110.84, 164.31,

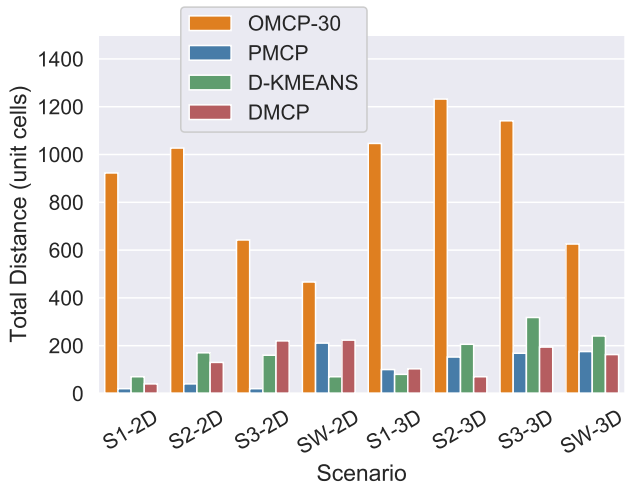



Figure 7.6: DMCP: Comparison of cloudlet distance traveled (cell units).

and 887.95 for DMCP, PMCP, D-KMEANS, and OMCP-30, respectively across all scenarios. For some scenarios, the cloudlets have negligible movement between cells in PMCP, notably S1-2D and S3-2D. This is also comparatively true for DMCP in S1-2D and S2-3D. It is because in DMCP and PMCP, cloudlets do not move to a different cell unless there is a significant change in the user distribution in and around its covered cells. D-KMEANS is close but worse overall than DMCP, especially in more complex 3D scenarios. D-KMEANS thus far has unrestricted movement, poorer service fairness, and poorer energy efficiency which makes it comparatively inferior to DMCP as a distributed approach. OMCP-30 is again agnostic to minimizing the distance traveled and purely focuses on maximizing coverage.

#### 7.3.3.5 Scalability

We now compare the decision time of DMCP against PMCP and D-KMEANS to demonstrate its scalability. The decision time is the average time taken (in milliseconds) by each mobile cloudlet to reach the final decision on making its next move at each time slot. We used 5 different sparse to dense variations (ordered from left to right in Figure 6.6) of the SW-3D scenario to observe the decision time as the number of mobile cloudlets and devices increases under the same mobility pattern. We do not consider

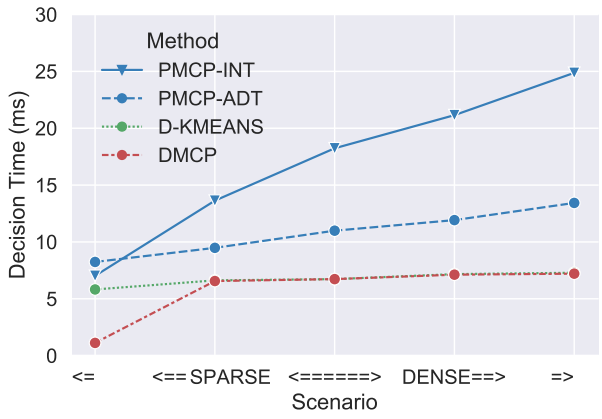



Figure 7.7: DMCP: Comparison of decision time (ms).

OMCP-30 in this comparison since it is originally an exhaustive approach and has a constant time allocated for optimization (i.e., 30 minutes time limit).

Illustrated by Figure 6.6, DMCP has the lowest decision time for all scenarios (1.12-7.21 ms) compared to all approaches. D-KMEANS has a similar decision time (5.82-7.29 ms), however, it should be noted that D-KMEANS is simplified to 1-MEAN and embedded into the same sequence of actions as DMCP. The reduction is hence due to the overall design of DMCP. Another difference is shown by anomalously low decision time for DMCP in the sparsest scenario. This is because DMCP calculations are simple while D-KMEANS still needs to calculate cluster centroids till convergence for each cloudlet, and hence, has much larger decision time for the sparsest scenario. Note that these mean values for DMCP and D-KMEANS do not include the initial random deployment step which is significantly faster and would further lower this mean decision time.

The adaptive part of PMCP (PMCP-ADT) is not much worse than DMCP at 8.24-13.43 ms. However, DMCP remains extremely steady as the scenarios become dense while the decision time increases for PMCP-ADT. In the same way, the initial placement of the mobile cloudlets in PMCP (PMCP-INT) is much more inefficient, and the decision time highly increases as scenarios become denser. The overall higher

coverage and stability in PMCP are partly due to this computationally heavy step with consistently better initial placements. Another point to consider is that DMCP decision time includes time taken to communicate with the neighbors during each exchange and conflict resolution steps. In comparison, the centralized sensing of devices in PMCP is simulated to arrive from a single node that has complete device information, and PMCP's decisions are known by all mobile cloudlets at the same time (once received by any one mobile cloudlet). These suggest that PMCP decision time could take more time depending on the distribution of the sensors and the mobile cloudlets. Based on these scenarios, the average decision time of DMCP is 47% less than PMCP-ADT. Therefore, these results show that DMCP has significantly faster decision time and better scalability.

In short, DMCP offers comparatively high device coverage, stable services through a low switching rate, and marked energy efficiency through reduced cumulative distance traveled by the cloudlets. DMCP is well designed to run in a distributed manner and has no initialization overhead. It is likewise more scalable as the size of the scenarios grows.

## 7.4 Chapter Summary

As the user mobility increases together with dynamic and uncertain demands, deploying mobile cloudlets offers a neat solution to provide better services in next-generation edge networks. We designed a novel distributed mobile cloudlet placement approach, DMCP, to address the goals of enhancing user coverage and providing stable edge services while maintaining scalability. The results show that DMCP achieves high coverage, allows stable services, and is also energy-efficient. It further offers good service fairness, runs in real-time, and maintains a steady decision time even as the problem-size scales up. Our distributed approach is therefore practical for mobile cloudlet placements and can be extended to solve distributed assignment and resource placement problems in a unique way.

# Chapter 8

# CONCLUSIONS AND FUTURE WORK

We tackle several challenges associated with the cloudlet placement problem in this dissertation. First, we propose a mathematical model of the static cloudlet placement problem (OCP) as an integer linear program considering multiple objectives and comprehensively capturing real life constraints associated with the problem. The mathematical model serves as a benchmark for subsequently proposed efficient cloudlet placement approaches. Second, we propose a fast Genetic Algorithm-based Placement Approach (GACP). The results show that GACP obtains close to optimal cost cloudlet placements by smoothly trading-off between the cost and the latency objectives. Then, we propose a bifactor approximation approach (ACP) for the cloudlet placement problem. We perform rigorous theoretical analysis and experiments to show the theoretical bounds for the objectives of the placement problem and efficient performance in real life placement scenarios. Thereafter, we formulate a mathematical model for dynamic mobile cloudlet placement problem (OMCP) for placement in highly dynamic and mobile scenarios. Likewise, we propose a novel approach (PMCP) for dynamically placing mobile cloudlets based on changing user demands and mobility inspired from physics. It provides an elegant way of adaptively placing mobile cloudlets over discrete time slots in dense next generation networks. Finally, we propose a distributed mobile cloudlet placement approach (DMCP) to provide even faster decisions and address scalability issues. These approaches altogether represent a collection of efficient and scalable approaches to place cloudlets at the edge of the next generation networks.

We believe that our research will encourage new research work in the area of cloudlet placement in edge computing. The following present several important and promising directions of possible extensions to the approaches we have discussed in the

dissertation. It is possible to design various multi-objective functions to make the optimal benchmarks more inclusive of other objectives (e.g., minimizing travel path of a mobile cloudlet) of the respective placement problems. This will allow investigating new approaches to solve multi-objective optimizations suitable for short-term or long-term cloudlet placement strategies for edge computing. In addition, since ours is an early work on dynamic cloudlet placement problem, there is an opportunity to capture exact physics of cloudlet movement from cell to cell in the OMCP system model. Even though we considered the mobility limitations of cloudlets, more specifics of that (e.g., impacts of battery level, weight, workload) can be investigated and captured in modeling. Furthermore, there is a research opportunity to study theoretical bounds of PMCP or an extension of it to theoretically guarantee its (worst-case) performance across challenging instances. These performance guarantees can be investigated individually or together for coverage, stability, fairness, and energy efficiency.

Another line of research can focus on detailing the nuances of grid and cell design to complement our approaches since the design of the 3D grid (e.g., discontinuous cells, physical obstructions, fading signal) and shapes of the cells (e.g., hexagonal prism, sphere) have influence on the implementation of these approaches. Similarly, applications with hard real-time requirements (e.g., autonomous vehicles, safety systems, rescue operations) are critically affected and need to be covered differently compared to general applications (e.g., live streaming, real-time translation), which adds another layer of heterogeneity. Hence, priority-based coverage of critical applications and higher application granularity (e.g. sub-applications, dependencies) are some more avenues available to explore. When it comes to DMCP, we can further enhance the robustness of the approach by reducing the communication overhead (e.g., reducing the size and the number of exchanged messages), making the approach fully asynchronous, and guaranteeing properties of distributed consensus (e.g. liveness, fault-tolerance). All of these will make it easier to deploy mobile cloudlets in real environments and lead to better adoption of mobile cloudlet-based services as we transition into future generations of the communication and computation networks.

# BIBLIOGRAPHY


[1] Cloud Enhanced Open Software Defined Mobile Wireless Testbed for City-Scale Deployment (COSMOS). https://www.cosmos-lab.org/. Accessed: 2020-12-29.

[2] First in the U.S. to Mobile 5G – What's Next? https://about.att.com/story/2019/2019_and_beyond.html. Accessed: 2020-12-25.

[3] Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016. https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016. Accessed: 2020-12-25.

[4] Harness the power of 5G Edge. https://enterprise.verizon.com/business/learn/edge-computing/. Accessed: 2020-12-31.

[5] Nautilus documentation. https://pacificresearchplatform.org/nautilus/. Accessed: 2022-05-16.

[6] Karen Aardal, Pieter L van den Berg, Dion Gijswijt, and Shanfei Li. Approximation algorithms for hard capacitated k-facility location problems. *European J. of Operational Research*, 242(2):358–368, 2015.

[7] P. Ahokangas, M. Matinmikko-Blue, S. Yrjölä, V. Seppänen, H. Hämmäinen, R. Jurva, and M. Latva-aho. Business models for local 5g micro operators. *IEEE Transactions on Cognitive Communications and Networking*, 5(3):730–740, 2019.

[8] Maria Fiorina Balcan, Steven Ehrlich, and Yingyu Liang. Distributed k-means and k-median clustering on general topologies. In *Proc. of the 26th International Conference on Neural Information Processing Systems-Volume 2*, pages 1995–2003, 2013.

[9] Ronald H Ballou. Potential error in the center of gravity approach to facility location. *Transportation Journal*, pages 44–50, 1973.

[10] E. Bastug, M. Bennis, M. Medard, and M. Debbah. Toward interconnected virtual reality: Opportunities, challenges, and enablers. *IEEE Communications Magazine*, 55(6):110–117, 2017.

[11] Dixit Bhatta and Lena Mashayekhy. Generalized cost-aware cloudlet placement for vehicular edge computing systems. In *Proc. of the 11th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 159–166, 2019.

[12] Dixit Bhatta and Lena Mashayekhy. Bifactor Approx Cloudlet Placement. https://doi.org/10.24433/CO.4921212.v1, 2021.

[13] Dixit Bhatta and Lena Mashayekhy. A bifactor approximation algorithm for cloudlet placement in edge computing. *IEEE Transactions on Parallel and Distributed Systems*, 33(8):1787–1798, 2021.

[14] Patricia Binder, Michael Muma, and Abdelhak M Zoubir. Gravitational clustering: a simple, robust and adaptive approach for distributed networks. *Signal Processing*, 149:36–48, 2018.

[15] Jaroslaw Byrka and Karen Aardal. An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. *SIAM Journal on Computing*, 39(6):2212–2231, 2010.

[16] Fabián A. Chudak and David B. Shmoys. Improved approximation algorithms for the uncapacitated facility location problem. *SIAM J. Comput.*, 33(1):1–25, 2004.

[17] City of New York. NYC Open Data. https://opendata.cityofnewyork.us/data/. Accessed: 2019-02-22.

[18] Columbia Engineering. NSF Announces New York City as Testbed for New Wave of Mobile Technology. https://engineering.columbia.edu/news/nsf-cosmos-testbed, 2018. Accessed: 2019-02-20.

[19] A. Elgazar, M. Aazam, and K. Harras. Edgestore: Leveraging edge devices for mobile storage offloading. In *Proc. of the IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 56–61, Dec 2018.

[20] Chinwendu Enyioha, Ali Jadbabaie, Victor Preciado, and George Pappas. Distributed resource allocation for control of spreading processes. In *Proc. of the European Control Conference (ECC)*, pages 2216–2221. IEEE, 2015.

[21] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD)*, volume 96, pages 226–231, 1996.

[22] Q. Fan and N. Ansari. Cost aware cloudlet placement for big data processing at the edge. In *Proc. of the IEEE International Conference on Communications*, pages 1–6, May 2017.

[23] Erfan Farhangi Maleki and Lena Mashayekhy. Mobility-aware computation offloading in edge computing using prediction. In *Proc. of the 4th IEEE Intl. Conf. on Fog and Edge Computing*, pages 69–74, 2020.

[24] Gordon Fraser and Andrea Arcuri. Evosuite: Automatic test suite generation for object-oriented software. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, pages 416–419, New York, NY, USA, 2011. ACM.

[25] Teofilo F. Gonzalez. *Handbook of Approximation Algorithms and Metaheuristics*, chapter 79, pages 1–13. Taylor & Francis Group, 2007.

[26] Rohitha Goonatilake and Rafic Bachnak. Modeling latency in a network distribution. *Network and Communication Technologies*, 1(2), 2012.

[27] Yan Guo, Shangguang Wang, Ao Zhou, Jinliang Xu, Jie Yuan, and Ching-Hsien Hsu. User allocation-aware edge cloud placement in mobile edge computing. *Software: Practice and Experience*, 50(5):489–502, 2020.

[28] Anqi Huang, Yingyu Li, Yong Xiao, Xiaohu Ge, Sumei Sun, and Han-Chieh Chao. Distributed resource allocation for network slicing of bandwidth and computational resource. In *Proc of. the IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2020.

[29] IBM. Overview (CPLEX Java API Reference Manual). https://www.ibm.com/support/knowledgecenter/en/SSSA5P_12.8.0/ilog.odms.cplex.help/refjavacplex/html/index.html. Accessed: 2019-02-25.

[30] Raj Jain, Arjan Durresi, and Gojko Babic. Throughput fairness index: An explanation. In *ATM Forum contribution*, volume 99, 1999.

[31] M. Jia, J. Cao, and W. Liang. Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks. *IEEE Transactions on Cloud Computing*, 5(4):725–737, 2017.

[32] J Jiao, L Chen, X Hong, and J Shi. A heuristic algorithm for optimal facility placement in mobile edge networks. *KSII Transactions on Internet and Information Systems*, 11(7):3329–3350, 2017.

[33] S. Kang, L. Ruan, S. Guo, W. Li, and X. Qiu. Geographic clustering based mobile edge computing resource allocation optimization mechanism. In *Proc. of the 15th International Conference on Network and Service Management*, pages 1–5, 2019.

[34] Richard Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, 40:85–103, 01 1972.

[35] Dinesh Kumar Kotary and Satyasai Jagannath Nanda. A distributed neighbourhood DBSCAN algorithm for effective data clustering in wireless sensor networks. *Wireless Personal Communications*, 121(4):2545–2568, 2021.

[36] Sylvia Todorova Kouyoumdjieva, Ólafur Ragnar Helgason, and Gunnar Karlsson. Micro-simulation of pedestrian mobility, CRAWDAD dataset. https://crawdad.org/kth/walkers/20140505, 2014.

[37] Phu Lai, Qiang He, Guangming Cui, Xiaoyu Xia, Mohamed Abdelrazek, Feifei Chen, John Hosking, John Grundy, and Yun Yang. Edge user allocation with dynamic quality of service. In *International Conference on Service-Oriented Computing*, pages 86–101. Springer, 2019.

[38] Tony S Lee, Sumit Ghosh, and Anil Neerode. Asynchronous, distributed, decision-making systems with semi-autonomous entities: A mathematical framework. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 30(1):229–239, 2000.

[39] Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Information and Computation*, 222:45–58, 2013.

[40] Y. Li and S. Wang. An energy-aware edge server placement algorithm in mobile edge computing. In *Proc. of the IEEE International Conference on Edge Computing*, pages 66–73, 2018.

[41] Jiamo Liu, Udita Paul, Olabisi Falowo, Sebastian Troia, and Guido Maier. K-means based spatial base station clustering for facility location problem in 5g. In *Proc. of the Southern Africa Telecommunication Networks and Applications Conference (SATNAC)*, pages 406–409, 2018.

[42] Qian Liu, Long Shi, Linlin Sun, Jun Li, Ming Ding, and Feng Shu. Path planning for UAV-mounted mobile edge computing with deep reinforcement learning. *IEEE Transactions on Vehicular Technology*, 69(5):5723–5728, 2020.

[43] L. Ma, J. Wu, L. Chen, and Z. Liu. Fast algorithms for capacitated cloudlet placements. In *Proc. of the IEEE 21st International Conference on Computer Supported Cooperative Work in Design*, pages 439–444, 2017.

[44] Weibin Ma, Xuanzhang Liu, and Lena Mashayekhy. A strategic game for task offloading among capacitated UAV-mounted cloudlets. In *Proc. of the IEEE International Congress on Internet of Things (ICIOT)*, pages 61–68, 2019.

[45] Weibin Ma and Lena Mashayekhy. Truthful computation offloading mechanisms for edge computing. In *Proc. of the 6th IEEE Intl. Conf. on Edge Computing and Scalable Cloud*, pages 1–8, 2020.

[46] P. Mach and Z. Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys Tutorials*, 19(3):1628–1656, 2017.

[47] P. Mach and Z. Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys Tutorials*, 19(3):1628–1656, thirdquarter 2017.

[48] Mohammad Mahdian, Yinyu Ye, and Jiawei Zhang. Approximation algorithms for metric facility location problems. *SIAM Journal on Computing*, 36(2):411–432, 2006.

[49] Erfan Farhangi Maleki, Weibin Ma, Lena Mashayekhy, and Humberto La Roche. QoS-aware 5G component selection for content delivery in multi-access edge computing. In *Proc. of IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*, 2021.

[50] George Mavrotas. Effective implementation of the $\epsilon$-constraint method in multi-objective mathematical programming problems. *Applied Mathematics and Computation*, 213(2):455 – 465, 2009.

[51] J. Meng, W. Shi, H. Tan, and X. Li. Cloudlet placement and minimum-delay routing in cloudlet computing. In *Proc. of the 3rd International Conference on Big Data Computing and Communications*, pages 297–304, 2017.

[52] Peter Merz and Mikael Rylander. How 5G-Advanced will help quench the world's growing thirst for capacity. Accessed on: November 23, 2021. [Online] Available: https://www.nokia.com/blog/how-5g-advanced-will-help-quench-the-worlds-growing-thirst-for-capacity/, June 2021.

[53] Larry Miller and Jessy Cavazos. *5G & Beyond For Dummies*. Wiley, 2022.

[54] Melanie Mitchell. *Introduction to Genetic Algorithms*, chapter 1,4, pages 4–20, 116–118. MIT Press, 1999.

[55] Mehrdad Moradi, Karthikeyan Sundaresan, Eugene Chai, Sampath Rangarajan, and Z Morley Mao. SkyCore: Moving core to the edge for untethered and reliable UAV-based LTE networks. In *Proc. of the 24th Annual International Conference on Mobile Computing and Networking*, pages 35–49, 2018.

[56] 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 Communications Surveys Tutorials*, 20(1):416–464, Firstquarter 2018.

[57] Mohammad Mozaffari, Ali Taleb Zadeh Kasgari, Walid Saad, Mehdi Bennis, and Mérouane Debbah. Beyond 5G with UAVs: Foundations of a 3D wireless cellular network. *IEEE Transactions on Wireless Communications*, 18(1):357–372, 2018.

[58] Omar Sami Oubbati, Mohammed Atiquzzaman, Pascal Lorenz, Md Hasan Tareque, and Md Shohrab Hossain. Routing in flying ad hoc networks: survey, constraints, and future challenge perspectives. *IEEE Access*, 7:81057–81105, 2019.

[59] Subrat Prasad Panda, Kaustabha Ray, and Ansuman Banerjee. Dynamic edge user allocation with user specified qos preferences. In *International Conference on Service-Oriented Computing*, pages 187–197. Springer, 2020.

[60] Dev Patel. FAA teams up with Verizon to test drones connected to mobile phones. Accessed on: August 5, 2021 . [Online] Available: https://industry-update.com/faa-teams-up-with-verizon-to-test-drones-connected-to-mobile-phones-general-aviation-news/74001/, 2021.

[61] Qinglan Peng, Yunni Xia, Yan Wang, Chunrong Wu, Wanbo Zheng, Xin Luo, Shanchen Panz, Yong Ma, and Chunxu Jiang. A decentralized collaborative approach to online edge user allocation in edge computing environments. In *2020 IEEE International Conference on Web Services (ICWS)*, pages 294–301. IEEE, 2020.

[62] S. Raghavan, Mustafa Sahin, and F. Sibel Salman. The capacitated mobile facility location problem. *European Journal of Operational Research*, 277(2):507 – 520, 2019.

[63] Mahadev Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.

[64] Nafiseh Sharghivand, Farnaz Derakhshan, Lena Mashayekhy, and Leila Mohammadkhanli. An edge computing matching framework with guaranteed quality of service. *IEEE Transactions on Cloud Computing (in press)*, pages 1–14, 2020.

[65] Muhammad Karam Shehzad, Syed Ali Hassan, MA Luque-Nieto, Javier Poncela, and Haejoon Jung. Energy efficient placement of UAVs in wireless backhaul networks. In *Proc. of the 2nd ACM MobiCom Workshop on Drone Assisted Wireless Communications for 5G and Beyond*, pages 1–6, 2020.

[66] David Shmoys. Approximation algorithms for facility location problems. In *Proc. of the Intl. Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 27–33, 2000.

[67] Sihai Tang, Bruce Chen, Harold Iwen, Jason Hirsch, Song Fu, Qing Yang, Paparao Palacharla, Nannan Wang, Xi Wang, and Weisong Shi. VECFrame: A vehicular edge computing framework for connected autonomous vehicles. In *Proc. of the IEEE International Conference on Edge Computing (EDGE)*, pages 68–77, 2021.

[68] Alexandre Veith, Marcos Assuncao, and Laurent Lefèvre. Latency-aware placement of data stream analytics on edge computing. [Online]. Available: https://hal.inria.fr/hal-01875936/document, 11 2018.

[69] Futian Wang, Xingxiang Huang, Hongfang Nian, Qiang He, Yun Yang, and Cheng Zhang. Cost-effective edge server placement in edge computing. In *Proc. of the 5th International Conference on Systems, Control and Communications*, pages 6–10, 2019.

[70] J. Wang, K. Liu, and J. Pan. Online uav-mounted edge server dispatching for mobile-to-mobile edge computing. *IEEE Internet of Things Journal*, 7(2):1375–1386, 2020.

[71] Shangguang Wang, Yali Zhao, Jinlinag Xu, Jie Yuan, and Ching-Hsien Hsu. Edge server placement in mobile edge computing. *Journal of Parallel and Distributed Computing*, 127:160–168, 2019.

[72] Zhongmin Wang, Feng Gao, and Xiaomin Jin. Optimal deployment of cloudlets based on cost and latency in internet of things networks. *Wireless Networks*, 26(8):6077–6093, 2020.

[73] David P. Williamson and David Shmoys. *The Design of Approximation Algorithms*, chapter 1, pages 10–20. Cambridge University Press, 2011.

[74] William E Wright. Gravitational clustering. *Pattern recognition*, 9(3):151–166, 1977.

[75] Chunrong Wu, Qinglan Peng, Yunni Xia, Yong Ma, Wangbo Zheng, Hong Xie, Shanchen Pang, Fan Li, Xiaodong Fu, Xiaobo Li, et al. Online user allocation in mobile edge computing environments: A decentralized reactive approach. *Journal of Systems Architecture*, 113:101904, 2021.

[76] Haolong Xiang, Xiaolong Xu, Haoquan Zheng, Shu Li, Taotao Wu, Wanchun Dou, and Shui Yu. An adaptive cloudlet placement method for mobile applications over gps big data. In *Proc. of the IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2016.

[77] Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo. Efficient algorithms for capacitated cloudlet placements. *IEEE Transactions on Parallel and Distributed Systems*, 27(10):2866–2880, 2016.

[78] Hong Yao, Changmin Bai, Muzhou Xiong, Deze Zeng, and Zhangjie Fu. Heterogeneous cloudlet deployment and user-cloudlet association toward cost effective fog computing. *Concurrency and Computation: Practice and Experience*, 29(16):1–9, 2017.

[79] Mohamed Younis and Kemal Akkaya. Strategies and techniques for node placement in wireless sensor networks: A survey. *Ad Hoc Networks*, 6(4):621–655, June 2008.

[80] Xiaoqun Yuan, Mengting Sun, and Wenjing Lou. A dynamic deep-learning-based virtual edge node placement scheme for edge cloud systems in mobile environment. *IEEE Transactions on Cloud Computing*, 2020.

[81] Feng Zeng, Yongzheng Ren, Xiaoheng Deng, and Wenjia Li. Cost-effective edge server placement in wireless metropolitan area networks. *Sensors*, 19:32, 2018.

[82] Yiwen Zhang, Kaibin Wang, Yuanyuan Zhou, and Qiang He. Enhanced adaptive cloudlet placement approach for mobile application on spark. *Security and Communication Networks*, 2018:1–12, 2018.

[83] Yiwen Zhang, Kaibin Wang, Yuanyuan Zhou, and Qiang He. Enhanced adaptive cloudlet placement approach for mobile application on spark. *Security and Communication Networks*, 2018, 2018.

[84] Z. Zhou, H. Yu, C. Xu, Z. Chang, S. Mumtaz, and J. Rodriguez. Begin: Big data enabled energy-efficient vehicular edge computing. *IEEE Communications Magazine*, 56(12):82–89, December 2018.

# Appendix A

# ADDITIONAL RESOURCES

This appendix provides additional resources related with the approaches discussed in the dissertation. The complete source code of the OCP, GACP, and ACP approaches can be found on GitHub at Cloudlet Placement Final repository. Fully reproducible capsule for ACP is available for run on Code Ocean [12]. A 10-minute video presentation on ACP is available for public viewing on UD Capture.

# Appendix B

# PERMISSIONS

Multiple chapters of the dissertation are based on several academic papers published or accepted for publication by the IEEE. Listed below are the associated publications and the chapters with published content:

- Chapters 2 and 3: Dixit Bhatta and Lena Mashayekhy. Generalized cost-aware cloudlet placement for vehicular edge computing systems. In *Proc. of the 11th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 159–166, 2019.
- Chapter 4: Dixit Bhatta and Lena Mashayekhy. A bifactor approximation algorithm for cloudlet placement in edge computing. *IEEE Transactions on Parallel and Distributed Systems*, 33(8):1787–1798, 2021.
- Chapters 5 and 6: Dixit Bhatta and Lena Mashayekhy, Physics-inspired mobile cloudlet placement in next-generation edge networks. In *Proc. of the IEEE International Conference on Edge Computing (IEEE EDGE)*, pp. 1–10, 2022

IEEE allows authors to reuse their own work as a part of their dissertation. Their policies are as follows:

- "The IEEE shall grant authors and their employers permission to make copies and otherwise reuse the material."[1]
- "You may reuse your published article in your thesis or dissertation without requesting permission."[2]

[1] https://www.ieee.org/publications/rights/copyright-policy.html

[2] https://journals.ieeeauthorcenter.ieee.org/choose-a-publishing-agreement/avoid-infringement-upon-ieee-copyright/