Generalized Cost-Aware Cloudlet Placement for Vehicular Edge Computing Systems

Dixit Bhatta
Department of Computer and Information Sciences
University of Delaware
Newark, USA
Email: dixit@udel.edu

Lena Mashayekhy
Department of Computer and Information Sciences
University of Delaware
Newark, USA
Email: mlena@udel.edu

***Abstract*—One of the well-known challenges in Edge Computing is strategic placement of cloudlets. The fundamental goals of this challenge are to minimize the deployment cost of cloudlets and to guarantee minimum latency for users of edge services. However, building cloudlet infrastructure may not be feasible in many situations and areas (e.g., disaster situations, unexpected surge in demand, and remote rural areas). Vehicular edge computing, VEC, introduces mobile cloudlets to augment edge computing capacity, enhance its coverage, and reduce latency significantly. However, efficient cloudlet placement is even more critical in VEC as it is not a long-term decision and needs to be repeated over time. In this paper, we address this challenge by designing a generalized cost-aware cloudlet placement approach that places a set of heterogeneous cloudlets in a region and fully maps user applications to appropriate cloudlets while ensuring their latency requirements. We first formulate the problem as a multi-objective integer programming model in a general deployment scenario. This is a computationally NP-hard problem. To tackle its intractability, we then propose a genetic algorithm-based approach, GACP. We investigate the effectiveness of GACP by performing extensive experiments on multiple deployment scenarios based on New York City OpenData. The results show that GACP obtains close to optimal cost placement in significantly reduced time.**

***Index Terms*—edge computing; mobile cloudlets; optimization; genetic algorithm**

I. Introduction

With advances in wireless network technologies and computational capabilities of smart connected devices, it is now possible to use many innovative applications not feasible before. For instance, live streaming on social media apps, games on virtual reality headsets, and more general applications in domains such as healthcare, connected vehicles, and smart cities [1]. Despite improvements in hardware capabilities, it is still a challenge to run computation and data intensive applications on mobile devices as they are restricted by weight, size, battery life, and heat dissipation [2]. These restrictions impose limitations on processing power, memory, and storage capacities of these devices.

Edge computing, sometimes used interchangeably with the term Fog computing, has been introduced as a new paradigm that provides a distributed computing solution at the edge of the network, where mobile users consume the computing resources in their vicinity. These resource-rich components placed closer to the users are called Cloudlets [3], [4]. Therefore, mobile devices can offload their resource-intensive applications to cloudlets, which is a comparatively economical solution with significantly reduced latency compared to the conventional cloud [5], [6]. Since cloudlets are distributed, a challenge lies in strategically placing them in a service area to provide low-latency edge services.

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 [7]. For these situations, an emerging computing paradigm 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 [8], [9]. SV-mounted cloudlets can augment edge computing capacity, enhance its coverage, and 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.

In this paper, we address the placement challenge by designing a generalized cost-aware cloudlet placement approach. We first formulate a general representation of the cloudlet placement problem considering a set of heterogeneous cloudlets and multiple user devices. Our objectives are to minimize the cloudlet placement cost and to ensure the placed cloudlets can cover all the users' requests guaranteeing minimum latency. Furthermore, an essential part of our problem is to find a complete mapping of devices to the placed cloudlets.

The placement problem is known to be NP-hard [10], and an efficient way to solve large-scale placement problems is by using heuristic and meta-heuristic approaches [11]. 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 [12]. 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 [13].

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 [14].

Related Work. To realize the vision of edge computing, efficient approaches for placing cloudlets are needed. In the literature, different approaches have been used to tackle the cloudlet placement problem. Wang *et al.* [10] and Jia *et al.* [15] proposed clustering-based approaches to place cloudlets. The focus of the former study is to balance workload and reduce access delay. The latter study focuses on cloudlet placement using density-based clustering and k-means clustering of users in order to minimize response time. Zeng *et al.* [16] proposed a greedy-based algorithm that minimizes the number of cloudlets to be placed considering latency requirements. A greedy heuristic approach is proposed in [17] to reduce access delay for users being served from access points. Li *et al.* [18] proposed energy-aware placement of cloudlets using swarm optimization, while assuming users are mapped to cloudlets through base stations. All of these studies do not directly consider placement cost of cloudlets, and they also assume all available cloudlets need to be placed.

A few studies concentrate on placing edge computing resources for one specific application such as big data processing [19] or virtual reality [20]. Fan and Ansari [19] considered the placement for big data processing to reduce cost and delay, and they proposed an integer programming model while relying on CPLEX to solve it. Veith *et al.* [21] presented a study on application placement for data stream analytics at edge, and proposed placement configurations to improve the response time of such applications. Bastug *et al.* [20] discussed edge server placement for augmented and virtual reality applications. None of these studies investigates a general heterogeneous deployment scenario. It is also important to emphasize that they consider cloudlet placement as a long term decision, i.e., the placed cloudlets stay at their locations permanently. To the best of our knowledge, this is the first study to design a genetic algorithm approach for the *mobile cloudlet placement* problem.

Another point to consider is that these studies ran their experiments on randomly generated networks. Only [10], [18], and [15] have used scenarios based on real datasets in their experiments. Wang *et al.* [10] and Li *et al.* [18] used base station dataset to validate their respective approaches. Jia *et al.* [15] used transportation network data under the assumption that a WiFi network would be similarly distributed. Finally, none of these studies considers the mapping of individual users or devices to cloudlets.

Organization. The rest of the paper is organized as follows. In Section II, we introduce the cloudlet placement problem and provide a mathematical optimization model. In Section III, we present our proposed GA-based approach, GACP, in detail. In Section IV, we evaluate the performance of GACP by extensive experiments. In Section V, we summarize our results and present possible directions for future research.

II. Cloudlet Placement Problem

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 $P = \{\rho_1, \rho_2, \ldots, \rho_n\}$, where 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 (including processor capacity Π_j (in GHz), RAM M_j (in GB), and 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 π_i is the processing demand (in GHz), m_i is its memory demand (in GB), s_i is its storage demand (in GB), and λ_i represents the location of the device (in coordinate axes). In addition, we define a 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, ϕ_{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 [22]. To capture that, we define a latency function $L(e_i, \rho_k)$ (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 the latency in accessing edge services, while provideing the services to all

end devices (full coverage). This is a bicriteria optimization problem and computationally NP-hard [10]. We next provide the mathematical optimization model of this problem.

A. Optimal Cloudlet Placement

We now formulate the cloudlet placement problem as a bi-criteria 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:

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

Subject to:

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

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

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

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

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

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

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

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

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

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

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

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

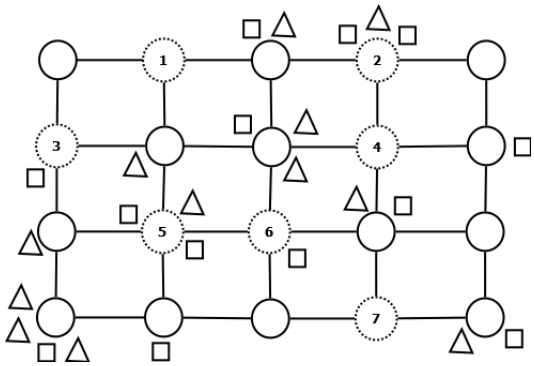


Fig. 1: Optimal Cloudlet Placement Scenario

OCP finds the optimal placement of the cloudlets minimizing both deployment cost and service latency. Our goal is to solve OCP in the presence of trade-offs between the two conflicting objectives. Next, using an example we show the relaxation approach for solving our multi-objective problem. This approach relaxes one of the objectives each time and solves the corresponding IP in order 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.

B. Optimization Example

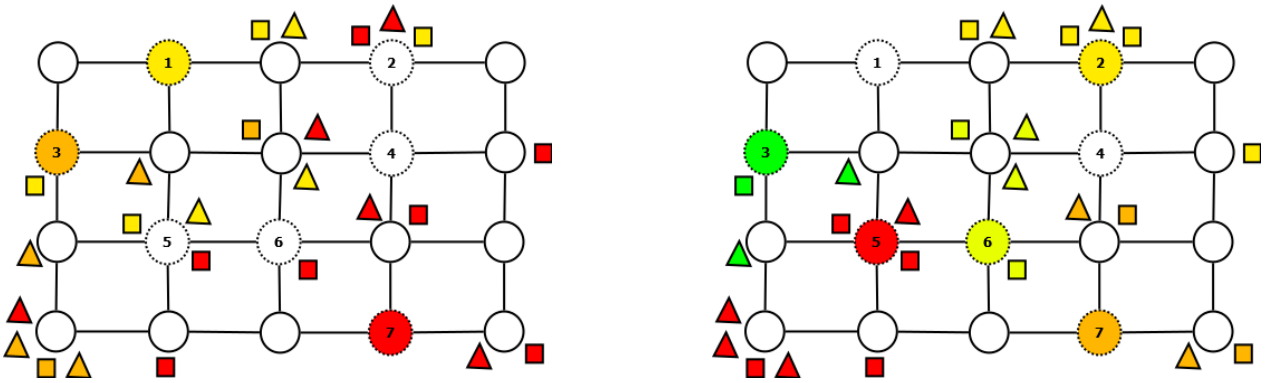
Figure 1 represents a scenario with 20 grid points, 7 candidate points, 25 end devices, and 5 cloudlets to be placed such that the optimization criteria of OCP are met. In this figure, the

TABLE I: Specifications of Cloudlets

Type	Processing	Memory	Storage	Coverage radius
Large	200	200	200	3
Medium	100	100	100	2
Small	50	50	50	1

TABLE II: Specifications of Devices

Type	Processing	Memory	Storage
High-Demand	20	20	20
Low-Demand	10	10	10



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

Fig. 2: Optimal Cloudlet Placement Scenarios

candidate points are denoted by dotted circles with their respective index. Likewise, low-demand devices are represented by *squares* and high-demand devices are shown as *triangles*. For this example, the specifications of cloudlets and devices are provided in Table I and Table II, respectively.

In this example, among the 5 available cloudlets, 1 is large (indexed by a), 3 are medium (b), and 1 is of small size (c), according to their resources and coverage. The cost associated with placing each cloudlet at a specific candidate point is based on its size. In this regard, the small cloudlet has the cost value (η_a) = 1, the medium (η_b) = 2, and the large (η_c) = 5. We assume the locations ρ_2, ρ_4, and ρ_5 are premium candidate points and would cost more. We calculate the placement cost of cloudlet c_j of type x (i.e., a, b, or c) using the following equation:

$$\Phi(c_j, \rho_k) = \begin{cases} \eta_x + 1 & \text{if } \rho_k \text{ is premium,} \\ \eta_x & \text{otherwise.} \end{cases} \tag{13}$$

The latency on the other hand is related with distance and bandwidth. In this example, we assume a constant bandwidth across the grid. Therefore, just the distance determines the latency. Equation (14) provides the latency values for each possible device location when served from a cloudlet at a specific candidate point:

$$L(e_i, \rho_k) = \lceil d(\lambda_i, \rho_k) * 10 \rceil \tag{14}$$

In the next two sub-sections, we discuss the optimal cloudlet placement for this example based on cost and latency, respectively.

1) Minimizing Cost (OCP-Cost): Figure 2a shows the perspective of only minimizing the cost (i.e., latency is relaxed in making placement decisions). In the figure, the large cloudlet is shown as *red*, medium as different shades of *yellow*, and the small as *green*. The end devices assigned to each cloudlet are denoted by the respective colors. The figure shows that only three (2 medium, 1 large) cloudlets are placed to serve all the devices, and the minimum cost is 9. However, this solution does not lead to the minimum latency for the users (the obtained latency is 340).

2) Minimizing Latency (OCP-Latency): Figure 2b depicts the perspective of only minimizing the latency (i.e., cost is relaxed). The color codes in the example are the same as described in the previous subsection. It is noteworthy that unlike the solution of OCP-Cost, the optimal solution of OCP-Latency consists of all available cloudlets. The minimum latency is 190.

3) Sensitivity Analysis: We now investigate how the solutions obtained using the relaxation method will change when specifying a hard constraint (a threshold) on the relaxed objective. Note that this threshold becomes an additional constraint in OCP.

For our analysis, we start with the optimal solution of a relaxed version (e.g., OCP-Cost) and find the value of the relaxed objective (e.g., latency). We use this value as the initial threshold, that 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, that means we reach the optimal solution of the relaxed objective (latency). This is shown in Figure 3. When the minimum cost obtained by OCP-Cost is 9, the best value of latency is 340 (see red "x"). By adding a new hard constraint on latency (e.g., 300), the best achievable cost increases to 10. If we further decrease

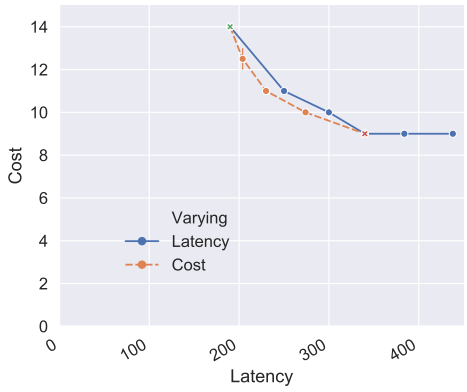


Fig. 3: Sensitivity Analysis

the latency constraint to 190 (optimal latency), the best cost becomes 14 (see green "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 larger. We discuss our proposed approach to address this challenge in the upcoming sections.

III. Genetic Algorithm-based Cloudlet Placement

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, Φ, L, δ, R, X_p, M_p, where δ 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 ($\hat{w}$) 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 R random cloudlet placements to the candidate points, called *Cloudlet Candidate Placement (CCP)*, where each has $\hat{w}$ cloudlets (line 3). The goal of GACP is to improve this set iteratively through 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. 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, that is $\min d(e_i, \rho_k), \forall \rho_k \in P$. This approach helps in minimizing latency and also in providing a consistent benchmark for calculating device coverage for a cloudlet placement.

In each iteration, GACP selects two least cost cloudlet placements from the priority queue $\mathcal{Q}$ (formed of placements in CCP) as parent placements (lines 9-10). GACP then crossovers these placements with probability X_p by 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 not 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 a lower-cost placement, our fitness function is defined to select a placement that its cost is closer to the optimal placement cost. Since the optimal cost is not known due to NP-hardness, GACP uses $\widehat{OPT}$ to reduce the optimality gap. The `Fitness()` function for a cloudlet placement y is calculated as (detailed in Algorithm 4):

$$\texttt{Fitness}(y) = |\widehat{OPT} - \textsc{placement\_cost}(y)| \qquad (15)$$

Once the fitness values are calculated (lines 17-18), GACP finds the device coverage of the parent placements and offspring placements (lines 19-20). The `coverage()` function (given in Algorithm 5) uses the device assignments and the current cloudlet placement considering memory, processing, storage, and coverage radius constraints (by calling `InRange&Capacity()` function) 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:

$$\texttt{Coverage}(y) = \frac{\textsc{devices\_covered}(y)}{|E|} \qquad (16)$$

If the offspring have strictly better fitness or equal fitness with better device coverage than their parents, GACP adds

Algorithm 1 GACP

1: **Input:** C, E, P, Φ, L, δ, R, X_p, M_p
2: $(\widehat{OPT}, \hat{w}) \leftarrow$ `LP-OCP-Cost()`
3: $CCP \leftarrow$ `RandPlace(`$R, \hat{w}$`)`
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}

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

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: $c_j = y[k]$
4: f_y += ϕ_{jk}
5: $f_y = |f_y - \widehat{OPT}|$
6: **return** f_y

Algorithm 5 Coverage(y)

1: $covered = 0$
2: **for** $i = 1$ **to** $|E|$ **do**
3: ρ_k = $DCA[i]$
4: c_j = $y[\rho_k]$
5: **if** (c_j = 0) **then** /*no cloudlet is placed at ρ_k*/
6: $min\_dist = \infty$
7: **for** $k = 1$ **to** $|P|$ **do**
8: c_k = $y[\rho_k]$
9: **if** (c_k != 0 &&
10: `InRange&Capacity`(e_i, c_k)) **then**
11: **if** ($d(e_i, \rho_k) < min\_dist$) **then**
12: $min\_dist \leftarrow d(e_i, \rho_k)$
13: $index \leftarrow k$
14: **if** $min\_dist \neq \infty$ **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** $covered/|\mathrm{E}|$

the offspring to B (lines 21-24). Otherwise, the parents are added (lines 26-28). Note that the offspring and parents are only added to B if they satisfy a certain coverage (at least the specified threshold, δ); otherwise, GACP simply moves to the next iteration.

At the end of a generation, GACP replaces CCP with the new generation of the best placements (line 29). 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.

IV. Experimental Results

A. Experimental Setup

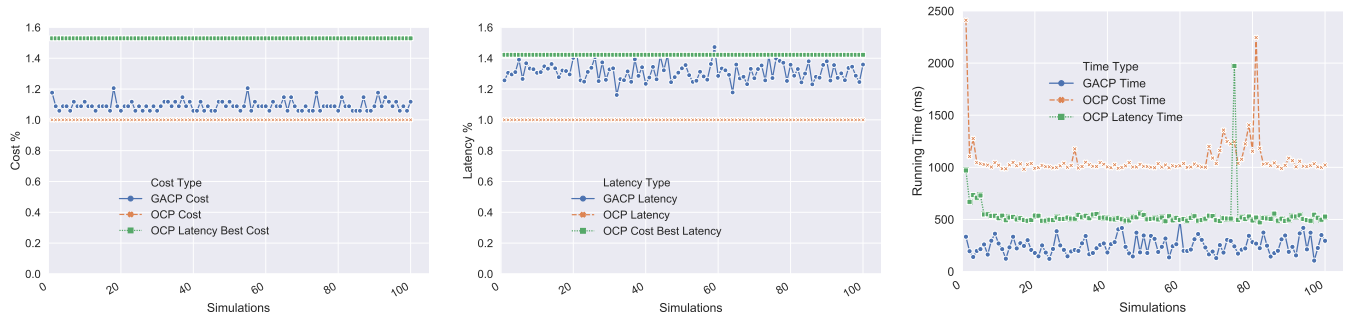
We perform a set of experiments to investigate the effectiveness of our proposed approach. For our deployment scenario, we use parts of New York City as it has been selected by National Science Foundation (NSF) as a testbed for the new wave of mobile technology [23]. Also, the presence of NYC Open Data [14] 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.

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 hotspot locations, we select only a certain percentage as the suitable candidate points. As pointed out in Section II, 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 remain the same during the sensitivity analysis.

The optimal results from OCP are found using IBM ILOG CPLEX Concert Technology API for Java [24]. 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



(a) Cost (b) Latency (c) Running Time

Fig. 4: Experimental Results for Central Harlem Data (10% Candidates)

RAM. 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 used as a proper benchmark. However, it is not able to obtain any results as the problem size increases due to NP-hardness of the problem.

B. Analysis of Results

We first show the results of GACP, OCP-Latency, and 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 cost and latency. The results are summarized in Figure 4.

Figure 4a shows that the costs obtained by GACP are very close to the optimal costs (OCP-Cost). GACP costs on average only 8.8% more than OCP-Cost. Figure 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 also capture the running times for 100 runs of GACP, OCP-Cost, and OCP-Latency. The results are shown in Figure 4c. The GACP running time is significantly less than both optimal approaches. 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 5a shows the effect on cost 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 5b shows that spreading the candidate points makes GACP suffer in terms of latency. This is because a larger distribution of candidate points makes it harder to achieve low latency at lower cost. However, GACP has a slight improvement in the 20% case, which provides an important insight that designating better candidate points is more important than having more of them. Figure 5c shows the running time. While GACP looks linear, OCP-Cost suffers from a slow running time. For instance, for the 15% candidate points case, solving OCP-Cost 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 in the first experiment. However, at 20% candidate point selection, we have just 16 candidate points. Nonetheless, the results (Figure 6) 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 is 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.

V. Conclusion

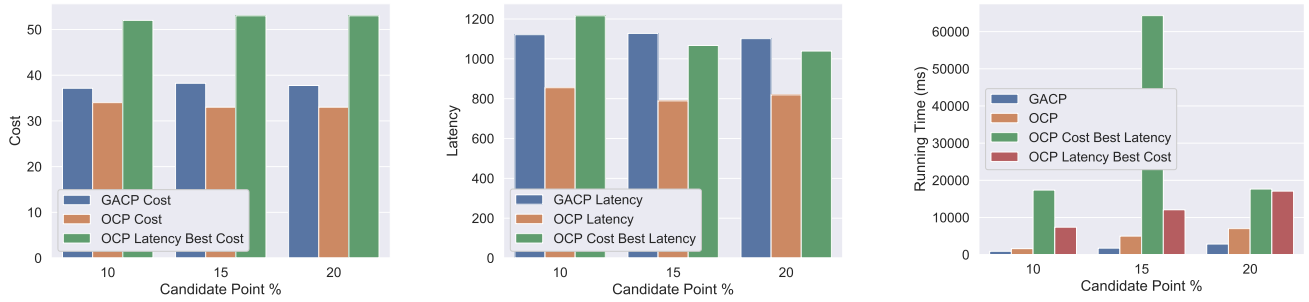
Latency suffered by the users can be mitigated by placing resource-rich mobile cloudlets closer to the network edge. However, strategically placing the cloudlets to both reduce cost and ensure all users are covered within their latency requirements is a major challenge. In this paper, we mathematically formulated a general system model considering heterogeneous cloudlets and user devices. However, it is not possible to optimize both objectives simultaneously. As a result, 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. In future work, we plan to consider the impact of device mobility on the cloudlet placement.

Acknowledgment

This research was supported in part by NSF grant CNS-1755913.

References

[1] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A comprehensive survey on fog computing: State-of-the-art and research challenges," *IEEE Communications Surveys Tutorials*, vol. 20, pp. 416–464, Firstquarter 2018.



(a) Cost (b) Latency (c) Running Time

Fig. 5: Sensitivity Analysis on the Percentage of Candidate Points

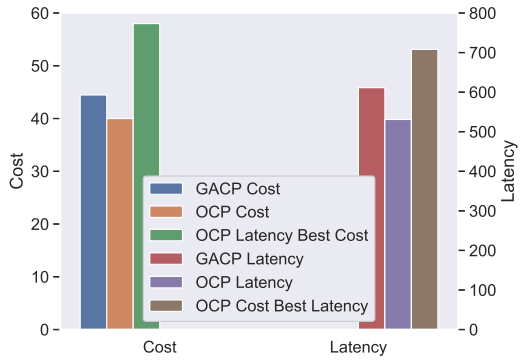


Fig. 6: East Harlem: Cost and Latency Comparisons

[2] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *Pervasive Computing*, December 2009.

[3] K. Li and J. Nabrzyski, "Traffic-aware virtual machine placement in cloudlet mesh with adaptive bandwidth," in *Proceedings of the IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 49–56, Dec 2017.

[4] H. Hong, "From cloud computing to fog computing: Unleash the power of edge and end devices," in *Proc. of the IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 331–334, 2017.

[5] M. Satyanarayanan, "The Emergence of Edge Computing," *Computer*, January 2017.

[6] N. Sharghivand, F. Derakhshan, and L. Mashayekhy, "QoS-aware matching of edge computing services to internet of things," in *Proceedings of the 37th IEEE International Performance Computing and Communications Conference*, pp. 1–8, 2018.

[7] 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)*, pp. 56–61, Dec 2018.

[8] 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*, vol. 56, pp. 82–89, December 2018.

[9] W. Ma, X. Liu, and L. Mashayekhy, "A strategic game for task offloading among capacitated UAV-mounted cloudlets," in *Proceedings of the IEEE International Congress on Internet of Things (ICIOT)*, pp. 61–68, 2019.

[10] S. Wang, Y. Zhao, J. Xu, J. Yuan, and C.-H. Hsu, "Edge server placement in mobile edge computing," *Journal of Parallel and Distributed Computing*, 06 2018.

[11] T. F. Gonzalez, *Handbook of Approximation Algorithms and Metaheuristics*, ch. 79, pp. 1–13. Taylor & Francis Group, 2007.

[12] M. Mitchell, *Introduction to Genetic Algorithms*, ch. 1,4, pp. 4–20, 116–118. MIT Press, 1999.

[13] G. Fraser and A. 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, (New York, NY, USA), pp. 416–419, ACM, 2011.

[14] City of New York, "NYC Open Data." [Online]. Available: https://opendata.cityofnewyork.us/data/. Accessed: 2019-02-22.

[15] 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*, vol. 5, pp. 725–737, Oct 2017.

[16] F. Zeng, Y. Ren, X. Deng, and W. Li, "Cost-effective edge server placement in wireless metropolitan area networks," *Sensors*, vol. 19, p. 32, 12 2018.

[17] Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo, "Efficient algorithms for capacitated cloudlet placements," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, pp. 2866–2880, Oct 2016.

[18] Y. Li and S. Wang, "An energy-aware edge server placement algorithm in mobile edge computing," in *Proceedings of the IEEE International Conference on Edge Computing (EDGE)*, pp. 66–73, July 2018.

[19] 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*, pp. 1–6, May 2017.

[20] E. Bastug, M. Bennis, M. Medard, and M. Debbah, "Toward interconnected virtual reality: Opportunities, challenges, and enablers," *IEEE Communications Magazine*, vol. 55, pp. 110–117, June 2017.

[21] A. da Silva Veith, M. D. de Assunçao, and L. Lefevre, "Latency-aware placement of data stream analytics on edge computing," in *Proceedings of the International Conference on Service-Oriented Computing*, pp. 215–229, Springer, 2018.

[22] R. Goonatilake and R. Bachnak, "Modeling latency in a network distribution," *Network and Communication Technologies*, vol. 1, no. 2, 2012.

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

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