Joint Service Placement and Request Routing in Multi-cell Mobile Edge Computing Networks

Konstantinos Poularakis[1], Jaime Llorca[2], Antonia M. Tulino[2,3], Ian Taylor[4], and Leandros Tassiulas[1]

[1]Department of Electrical Engineering and Institute for Network Science, Yale University, USA
[2]Nokia Bell Labs, USA
[3]Department of Electrical Engineering and Information Technologies, University of Naples Federico II, Italy
[4]School of Computer Science and Informatics, Cardiff University, UK

***Abstract*—The proliferation of innovative mobile services such as augmented reality, networked gaming, and autonomous driving has spurred a growing need for low-latency access to computing resources that cannot be met solely by existing centralized cloud systems. Mobile Edge Computing (MEC) is expected to be an effective solution to meet the demand for low-latency services by enabling the execution of computing tasks at the network-periphery, in proximity to end-users. While a number of recent studies have addressed the problem of determining the execution of service tasks and the routing of user requests to corresponding edge servers, the focus has primarily been on the efficient utilization of computing resources, neglecting the fact that non-trivial amounts of data need to be stored to enable service execution, and that many emerging services exhibit asymmetric bandwidth requirements. To fill this gap, we study the joint optimization of service placement and request routing in MEC-enabled multi-cell networks with multidimensional (storage-computation-communication) constraints. We show that this problem generalizes several problems in literature and propose an algorithm that achieves close-to-optimal performance using randomized rounding. Evaluation results demonstrate that our approach can effectively utilize the available resources to maximize the number of requests served by low-latency edge cloud servers.**

I. Introduction

A. Motivation

Emerging distributed cloud architectures, such as *Fog* and *Mobile Edge Computing (MEC)*, push substantial amounts of computing functionality to the edge of the network, in proximity to end-users, thereby allowing to bypass fundamental latency limitations of today's prominent centralized cloud systems [1]. This trend is expected to continue unabated and play an important role in next-generation 5G networks for supporting both computation-intensive and latency-sensitive services [2].

With MEC, services can be housed in wireless base stations (BSs) endowed with computing capabilities (or edge servers close to BSs) that can be used to accommodate service requests from users lying in their coverage regions. The computation capacity of BSs, however, is much more limited than that of centralized clouds, and may not suffice to satisfy all user requests. This naturally raises the question of *where to execute* each service so as to better reap the benefits of available computation resources to serve as many requests as possible.

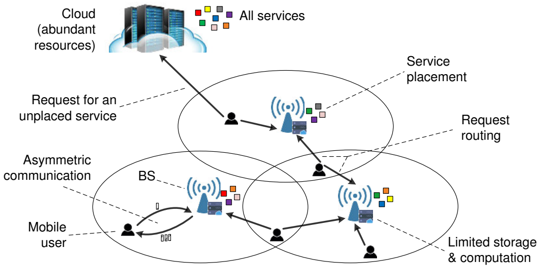


Fig. 1: An example MEC system. Service placement and request routing are constrained by the storage, computation and bandwidth resources of BSs.

While there have been several interesting approaches to determine the execution (or offloading) of services in MEC, e.g., [3] and [4], to cite two of the most recent, an important aspect has been hitherto overlooked. Specifically, many services today require not only the allocation of computation resources, but also a *non-trivial amount of data that needs to be pre-stored* (or pre-placed) at the BS. In an Augmented Reality (AR) service, for example, the placement of the object database and the visual recognition models is needed in order to run classification or object recognition before delivering the augmented information to the user [5]. Yet, the storage capacity of BSs may be not large enough to support all offered services.

The above issue is further complicated by the communication requirements of the services. Many modern services require uploading data from the user to be used as input for service execution, whose output must then be downloaded for consumption by the user. Such *bidirectional communication* may be asymmetric in general, taking up different portions of BSs' uplink and downlink bandwidth capacities [6].

In addition, the density of BSs has been increasing and is expected to reach up to 50 BSs per km^2 in future 5G deployments [7]. This will create a *complex multi-cell environment* where users are concurrently in range of multiple BSs with overlapping coverage regions, and hence the operator can use multiple paths to route services to them. Figure 1 illustrates an example of such a system.

Evidently, in this context, MEC operators have a large repertoire of service placement and request routing alternatives

This publication was supported partly by the National Science Foundation under Grants CNS 1815676 and 1619129, and the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001.

for satisfying the user requests. In order to serve as many requests as possible from the BSs, the operator has to *jointly* optimize these decisions while simultaneously satisfying storage, computation, and communication constraints. Clearly, this is an important problem that differs substantially from previous related studies (e.g., see [3], [4] and the survey in [1]) that did not consider storage-constrained BSs and asymmetric communication requirements. While a few recent works [8], [9], [10] studied the impact of storage in MEC, they neither considered all the features of these systems discussed above nor provided *optimal* or *approximate* solutions for the joint service placement and request routing problem.

Given the above issues, the key open questions are:

- *Which services to place in each BS to better utilize its available storage capacity?*
- *How to route user requests to BSs without overwhelming their available computation and (uplink/downlink) bandwidth capacities?*
- *How the above decisions can be optimized in a joint manner to offload the centralized cloud as much as possible?*

B. Methodology and Contributions

In this paper, we follow a systematic methodology in order to answer the above questions, summarized as follows.

1) We formulate the joint service placement and request routing problem (JSPRR) in multi-cell MEC networks aiming to minimize the load of the centralized cloud. We consider practical features of these systems such as overlapping coverage regions of BSs and multidimensional (storage, computation, and communication) resource constraints.
2) We identify several placement and routing problems in literature that are special cases of JSPRR, gaining insights into the complexity of the original problem.
3) Using randomized rounding techniques [11], we develop a bi-criteria algorithm that provably achieves approximation guarantees while violating the resource constraints in a bounded way. To the best of our knowledge, this is the first approximation algorithm for this problem.
4) We extend the results for dynamic scenarios where the user demand profiles change with time, and show how to adapt the solution accordingly.
5) We carry out evaluations to demonstrate the performance of the proposed algorithm. We show that, in many practical scenarios, our algorithm performs close-to-optimal and far better than a state-of-the-art method which neglects computation and bandwidth constraints.

The rest of the paper is organized as follows. Section II describes the system model and defines the JSPRR problem formally. We analyze the complexity of this problem and present algorithms with approximation guarantees in Section III and IV, respectively. Section V discusses extensions of our results and practical cases, while Section VI presents our evaluation results. We conclude our work in Section VII.

II. Model and Problem Definition

We consider a MEC system consisting of a set $\mathcal{N}$ of N BSs equipped with storage and computation capabilities and a set $\mathcal{U}$ of U mobile users, subscribers of the MEC operator, as depicted in Figure 1. The users can be arbitrarily distributed over the coverage regions of BSs, while the coverage regions may be overlapping in dense BS deployments. We denote by $\mathcal{N}_u \subseteq \mathcal{N}$ the set of BSs covering user u.

We consider multiple types of resources for BSs. First, each BS n has *storage capacity* R_n (hard disk) that can be used to pre-store data associated with services. Second, BS n has a CPU of *computation capacity* (i.e., maximum frequency) C_n that can be used to execute services in an on-demand manner. Third, BS n has uplink (downlink) *bandwidth capacity* $B_n^{\uparrow}$ ($B_n^{\downarrow}$) that can be used to upload (download) data from (to) mobile users requesting services.

The system offers a library $\mathcal{S}$ of S services to the mobile users. Examples include video streaming, augmented reality and networked gaming. Services may have different requirements in terms of the amount of storage, CPU cycles, and uplink/downlink bandwidth. We denote by r_s the storage space occupied by the data associated with service s. The notation c_s indicates the required computation, while $b_s^{\uparrow}$ and $b_s^{\downarrow}$ indicate the uplink and downlink bandwidth required to satisfy a request for service s, respectively.

The system receives service requests from users in a stochastic manner. Without loss of generality, we assume that each user u performs one request for a service denoted by s_u[1]. User requests can be predicted for a certain time period (e.g., a few hours) by using well-studied learning techniques (e.g., auto-regression analysis) [8]. Yet, the user demand can change after that period as the users may gain or lose interest in some services. We provide more details about this issue in Section V.

The request of user u can be routed to a nearby BS in $\mathcal{N}_u$ provided that service s_u is locally stored and the BS has enough computation and bandwidth resources. If there is no such BS, we assume that the user can access the centralized cloud, which serves as a last resort for all users. Accessing the cloud, however, may cause high delay due to its long distance from the users, and therefore should be avoided.

The network operator needs to decide in which BSs to place the services and how to route user requests to them. To model these decisions, we introduce two sets of optimization variables: (i) $x_{ns} \in \{0,1\}$ which indicates whether service s is placed in BS n ($x_{ns} = 1$) or not ($x_{ns} = 0$), and (ii) $y_{nu} \in \{0,1\}$ which indicates whether the request of user u is routed to BS n ($y_{nu} = 1$) or not ($y_{nu} = 0$). Similarly, we denote by y_{lu} the routing decision for the cloud. We refer by *service placement* and *request routing* policies to the respective vectors:

$$\boldsymbol{x} = (x_{ns} \in \{0,1\} \ : n \in \mathcal{N}, s \in \mathcal{S}) \tag{1}$$

$$\boldsymbol{y} = (y_{nu} \in \{0,1\} \ : n \in \mathcal{N} \cup \{l\}, u \in \mathcal{U}) \tag{2}$$

The service placement and request routing policies need to satisfy several constraints. First, each user request needs to be routed to exactly one of the nearby BSs, or the cloud:

$$\sum_{n \in \mathcal{N}_u \cup \{l\}} y_{nu} = 1, \ \forall u \in \mathcal{U} \tag{3}$$

[1] If a user performs multiple requests, we can split it into multiple users.

Second, in order for the request of user u to be routed to a BS n, the service s_u needs to be placed in the latter:

$$y_{nu} \leq x_{ns_u},\ \forall n \in \mathcal{N}, u \in \mathcal{U} \tag{4}$$

Third, the total amount of data of services placed in a BS must not exceed its storage capacity:

$$\sum_{s \in \mathcal{S}} x_{ns} r_s \leq R_n,\ \forall n \in \mathcal{N} \tag{5}$$

Fourth, the total computation load of user requests routed to BS n must not exceed its computation capacity:

$$\sum_{u \in \mathcal{U}} y_{nu} c_{s_u} \leq C_n,\ \forall n \in \mathcal{N} \tag{6}$$

Fifth, the total bandwidth load of user requests routed to BS n must not exceed its uplink and downlink bandwidth capacity:

$$\sum_{u \in \mathcal{U}} y_{nu} b^{\uparrow}_{s_u} \leq B^{\uparrow}_n,\ \forall n \in \mathcal{N} \tag{7}$$

$$\sum_{u \in \mathcal{U}} y_{nu} b^{\downarrow}_{s_u} \leq B^{\downarrow}_n,\ \forall n \in \mathcal{N} \tag{8}$$

The goal of the network operator is to find the joint service placement and request routing policy that maximizes the number of requests served by the BSs, or, equivalently, minimizes the load of the cloud:

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{y}} \quad & \textstyle\sum_{u \in \mathcal{U}} y_{lu} \\ \text{s.t.} \quad & \text{constraints: } (1) - (8) \end{aligned} \tag{9}$$

We refer by *JSPRR* to the above problem. This is an integer optimization problem and such problems are typically challenging to solve. In the next two sections, we analyze the complexity of this problem and propose approximation algorithms.

III. Complexity Analysis

The JSPRR problem is *NP-Hard* since it generalizes the knapsack problem [12] by comprising multiple packing constraints (inequalities (5)-(8)). The problem remains challenging even when simplified by the assumption of *homogeneous (unit-sized) service requirements*, i.e., $r_s = c_s = b^{\uparrow}_s = b^{\downarrow}_s = 1$ $\forall s$. This simplified problem includes as special cases several *well-studied placement and routing problems* in literature, which shows the universality of our model. Subsequently, we describe several such special cases. We begin with the (rather trivial) special case of non-overlapping BS coverage regions before diving into more challenging special cases.

A. Special case 1: Non-overlapping BS coverage regions

In the first special case, we make the simplifying assumption (in addition to the homogeneity of service requirements) that the coverage regions of the BSs do not overlap with each other. This particularly applies to sparse BS deployments where the BSs are located far away one from the other. It follows that the JSPRR problem can be *decomposed into N independent subproblems*, one per BS n. The objective of subproblem n is to maximize the number of requests served by BS n.

It is not difficult to show that there is always an optimal solution to subproblem n that places in BS n the R_n most locally popular services, i.e., the services requested by most users inside the coverage region of BS n. Then, BS n will admit as many requests for the placed services as its computation and bandwidth capacities C_n, $B^{\uparrow}_n$ and $B^{\downarrow}_n$ can handle, i.e., $\min\{C_n, B^{\uparrow}_n, B^{\downarrow}_n\}$ requests at most. Indeed, consider a solution that places in BS n a service s_1 requested by fewer users inside the respective coverage region than another service s_2. Then, one could swap the two services in the placement solution and route the same number of requests to BS n without changing the objective function value. Therefore, the JSPRR problem is trivial to solve in this special case.

B. Special case 2: Data placement/caching problem

In the second special case, we allow the coverage regions of BSs to overlap, but we make the simplifying assumption that the computation and bandwidth resources are non-congestible, i.e., they always suffice to route all user requests to BSs. In other words, we assume that the capacities C_n, $B^{\uparrow}_n$ and $B^{\downarrow}_n$ are greater than or equal to the demand of users, so that we can remove constraints (6)-(8) from the problem formulation without affecting the optimal solution.

Without the computation and bandwidth constraints, the placement and routing problem becomes much simpler. For a given service placement solution $\boldsymbol{x}$, finding the optimal request routing policy $\boldsymbol{y}$ is straightforward; simply route each user request to a nearby BS having stored the requested service, if any, otherwise to the cloud. This special case has been extensively studied in literature under the title *'data placement'* [13] or *'caching' problem* [14]. This problem asks to place data items (services) to caches (BSs) with the objective of maximizing the total number of requests served by the caches.

While the data placement problem is NP-Hard, several approximation algorithms are known in literature. The main method used to derive such approximations is based on showing the submodularity property of the optimization problem. That is, to show that the marginal value of the objective function never increases as more data items are placed in the caches. Having shown the submodularity property, several 'classic' algorithms can be applied, with the most known being greedy, local search, and pipage rounding [14]. Among the three algorithms, the greedy is the simplest and fastest, and, hence, the most practical.

C. Special case 3: Middlebox placement problem

In the third special case, we allow the coverage regions of BSs to overlap and the computation and bandwidth resources to be congestible, but we make the simplifying assumption that the storage capacities are unit-sized ($R_n = 1$ $\forall n$). That is, we assume that only one service can be stored per BS.

Under this special case, the JSPRR problem can be reduced to the *'middlebox placement' problem* [15], [16]. While there exist many different variants of the middlebox placement problem in literature, typically, this problem asks to pick l out of m nodes in a network to deploy middleboxes. The goal is to maximize the total number of source-destination flows (out of f flows) that can be routed through network paths containing at least one middlebox, subject to a constraint that limits the number of flows per middlebox.

Although the reduction is not so straightforward, the main idea is to create: (i) a distinct node for each pair of a BS and a service ($m = NS$) and (ii) a distinct flow for each user ($f = U$). Each flow can be routed through any node whose BS-service pair satisfies that the BS covers the respective user and the service is the requested by the user. The question is which $l = N$ out of the $m = NS$ nodes to pick to deploy middleboxes, with an additional constraint per BS that only 1 our of the S nodes corresponding to that BS can be picked. The picked node will determine which of the S services is placed at that BS.

Recent works have shown that the maximum flow objective of the middlebox problem is a submodular function [15], [16]. Therefore, this problem can be solved by using the same approximation algorithms mentioned in special case 2.

D. General case: Non-submodular

Although it would be tempting to conjecture that our JSPRR problem is submodular in its general form (with overlapping coverage regions, congestible bandwidth and computation and large storage capacities), we can construct counter-examples where this property does not hold. First, we introduce the definition of submodular functions.

Definition 1. *Given a finite set of elements $\mathcal{G}$ (ground set), a function $f : 2^{\mathcal{G}} \to \mathbb{R}$ is submodular if for any sets $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{G}$ and every element $g \notin \mathcal{B}$, it holds that:*

$$f(\mathcal{A} \cup \{g\}) - f(\mathcal{A}) \geq f(\mathcal{B} \cup \{g\}) - f(\mathcal{B}) \tag{10}$$

Next, we introduce the element e_{ns} to denote the placement of service s in BS n. The ground set is given by $\{e_{11}, \ldots, e_{NS}\}$. Every possible service placement policy can be expressed by a subset $\mathcal{E} \subseteq \mathcal{G}$ of elements, where the elements included in $\mathcal{E}$ correspond to the service placement. Given a service placement $\mathcal{E}$, we denote by $f(\mathcal{E})$ the maximum number of user requests that can be satisfied by the BSs.

We will construct a counter-example where the function $f(\mathcal{E})$ is not submodular. Specifically, we consider a system of $N = 2$ BSs and $U = 2$ users located in the intersection of the two coverage regions. The users request two different services denoted by s_1 and s_2. We set the computation capacities to $C_1 = C_2 = 1$ (i.e., at most one service request can be satisfied by each BS), while the storage and bandwidth capacities are abundant. The two placement sets we consider are $\mathcal{A} = \{e_{11}\}$ and $\mathcal{B} = \{e_{11}, e_{21}\}$, where $\mathcal{A} \subseteq \mathcal{B}$. We note that $f(\mathcal{A}) = f(\mathcal{B}) = 1$ since in both cases only one of the two services is stored (s_1), and hence only one of the two requests can be served. Besides, $f(\mathcal{A} \cup \{e_{12}\}) = 1$ since the computation constraint prevents the BS 1 from serving both user requests. However, $f(\mathcal{B} \cup \{e_{12}\}) = 2$ since now each BS can serve one user request. Therefore, the marginal performance is larger for the set $\mathcal{B}$ than the $\mathcal{A}$, which means that f is not submodular.

We note that similar counter-examples can be identified when the bandwidth (instead of computation) capacity is congestible, provided that storage capacity is greater than 1.

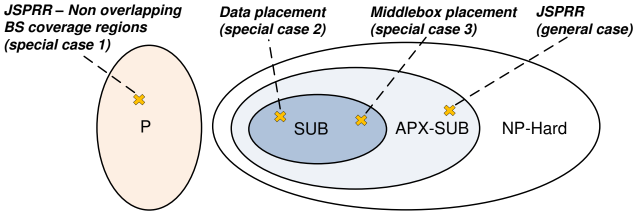

JSPRR – Non overlapping BS coverage regions (special case 1)
Data placement (special case 2)
Middlebox placement (special case 3)
JSPRR (general case)
P
SUB
APX-SUB
NP-Hard

Fig. 2: Complexity of special cases of JSPRR: Polynomial-time solvable (P), Submodular (SUB) and Approximately submodular (APX-SUB) classes.

E. General case: Approximately-submodular

Although our JSPRR problem does not fall into the class of submodular problems, we can show that it belongs to the wider class of *approximately submodular* problems [17]. The complexity of JSPRR for the general and special cases is illustrated in Figure 2.

Definition 2. *A function $f : 2^{\mathcal{G}} \to \mathbb{R}$ is δ-approximately submodular if there exists a submodular function $F : 2^{\mathcal{G}} \to \mathbb{R}$ such that for any $\mathcal{E} \subseteq \mathcal{G}$:*

$$(1 - \delta)F(\mathcal{E}) \leq f(\mathcal{E}) \leq (1 + \delta)F(\mathcal{E}) \tag{11}$$

We define by $F(\mathcal{E})$ the maximum number of user requests that can be satisfied by the BSs given the service placement set $\mathcal{E}$ in the special case that the bandwidth and computation resources are non-congestible (special case 2). Since there are fewer constraints in this special case than in the general case, it holds that $f(\mathcal{E}) \leq F(\mathcal{E})$. Therefore, for any $\delta \in [0, 1]$, we have $f(\mathcal{E}) \leq (1 + \delta)F(\mathcal{E})$. What remains to find is a δ value that satisfies the first inequality in (11), i.e., $(1 - \delta)F(\mathcal{E}) \leq f(\mathcal{E})$.

We note that when computing the value of $F(\mathcal{E})$, the BS n is allowed to satisfy all the requests for stored services generated by users in its coverage region. We denote by Φ_n the number of these requests. In case that it happens $\Phi_n \leq C_n$, $\Phi_n \leq B_n^{\uparrow}$ and $\Phi_n \leq B_n^{\downarrow}$ $\forall n$, then the computation and bandwidth resources are non-congestible and we have $f(\mathcal{E}) = F(\mathcal{E})$. In the other case that, for some n, it happens $\Phi_n > C_n$ or $\Phi_n > B_n^{\uparrow}$ or $\Phi_n > B_n^{\downarrow}$, then the BS n can process up to Φ_n / C_n times more requests, compared to $f(\mathcal{E})$. Similarly, the BS n can receive (deliver) data from (to) up to $\Phi_n / B_n^{\uparrow}$ ($\Phi_n / B_n^{\downarrow}$) times more users. Therefore, the total number of satisfied requests is upper bounded by:

$$F(\mathcal{E}) \leq \max_{n \in \mathcal{N}} \{\frac{\Phi_n}{C_n}, \ \frac{\Phi_n}{B_n^{\uparrow}}, \ \frac{\Phi_n}{B_n^{\downarrow}}, \ 1\} f(\mathcal{E}) \tag{12}$$

where the value 1 inside the max operator ensures that $F(\mathcal{E})$ will never be lower than $f(\mathcal{E})$. We thus can ensure that $(1 - \delta)F(\mathcal{E}) \leq f(\mathcal{E})$ by picking:

$$\delta = 1 - \frac{1}{\max_{n \in \mathcal{N}} \{\frac{\Phi_n}{C_n}, \ \frac{\Phi_n}{B_n^{\uparrow}}, \ \frac{\Phi_n}{B_n^{\downarrow}}, \ 1\}} \tag{13}$$

The problem of maximizing a δ-approximately submodular function has been studied in the past [17]. Based on the results in [17], we can use a simple greedy algorithm to achieve the approximation ratio described in the following proposition.

Proposition 1. *The Greedy algorithm returns a solution set $\mathcal{E}^*$ such that:*

$$f(\mathcal{E}^*) \geq \frac{1}{2}\Big(\frac{1-\delta}{1+\delta}\Big)\frac{1}{1+\frac{\sum_{n\in\mathcal{N}} R_n \delta}{1-\delta}} \max_{\mathcal{E}} f(\mathcal{E}) \tag{14}$$

Consider for example the case that the demand exceeds the available resources by up to 50%, i.e., there exists a BS n for which $\Phi_n = 1.5C_n$ or $\Phi_n = 1.5B_n^{\uparrow}$ or $\Phi_n = 1.5B_n^{\downarrow}$. Then, $\delta = 1/3$, and the approximation factor becomes:

$$f(\mathcal{E}^*) \geq \frac{1}{4}\frac{1}{1+\frac{\sum_{n\in\mathcal{N}} R_n}{2}} \max_{\mathcal{E}} f(\mathcal{E}) \tag{15}$$

The above approximation ratio worsens as the network becomes more congested (δ increases) and the storage capacities increase (R_n). This suggests that the JSPRR problem is substantially harder than the placement and routing problems described above, and thus a method of different philosophy is needed to find a tight approximation ratio. In the next section, we present such a method that goes beyond submodularity and achieves a solution with better approximation guarantees.

IV. Approximation Algorithm

In this section, we present one of the main contributions of this work; a novel approximation algorithm for the JSPRR problem. Our algorithm leverages a Randomized Rounding technique, from which it takes its name. The algorithm is described in detail below and summarized in Algorithm 1.

The Randomized Rounding algorithm starts by solving the linear relaxation of the JSPRR problem (line 1). That is, it relaxes the variables $\{x_{ns}\}$ and $\{y_{nu}\}$ to be fractional, rather than integer. The Linear Relaxation of JSPRR problem, *LR-JSPRR* for short, can be expressed as follows:

$$\min_{\boldsymbol{x},\boldsymbol{y}} \quad \textstyle\sum_{u\in\mathcal{U}} y_{lu} \tag{16}$$

$$\text{s.t.} \quad \text{constraints: } (3) - (8)$$

$$x_{ns} \in [0,1], \ \forall n \in \mathcal{N}, s \in \mathcal{S} \tag{17}$$

$$y_{nu} \in [0,1], \ \forall n \in \mathcal{N} \cup \{l\}, u \in \mathcal{U} \tag{18}$$

where we have replaced equations (1)-(2) with (17)-(18). Since the objective and the constraints of the above problem are linear, it can be optimally solved in polynomial time using a linear program solver. We denote by $\{x_{ns}^{\dagger}\}$ and $\{y_{nu}^{\dagger}\}$ the optimal solution values. The next step is to round these values to obtain an integer solution, denoted by $\{\widehat{x}_{ns}\}$ and $\{\widehat{y}_{nu}\}$. For each pair of node n and service s, the algorithm rounds variable $\widehat{x}_{ns}$ to 1 with probability $x_{ns}^{\dagger}$ (lines 2-3). Each rounding decision is taken independently from each other.

Finally, the algorithm uses the rounded placement variables $\{\widehat{x}_{ns}\}$ to decide the rounding of the routing variables (lines 4-9). For each user u, it defines the set of nearby BSs that have stored the requested service s_u by $\mathcal{N}_u' = (n \in \mathcal{N}_u \ : \ \widehat{x}_{ns_u} > 0)$ and uses this to distinguish between two cases: (i) if user u cannot find service s_u in any of the nearby BSs ($\mathcal{N}_u' = \emptyset$), then the user is served by the cloud (lines 6-7), (ii) otherwise, the user is randomly routed to one of the BSs in $\mathcal{N}_u'$ or the cloud (lines 8-9). The routing probabilities depend on the fractional values $\{x_{ns}^{\dagger}\}$ and $\{y_{nu}^{\dagger}\}$. Higher probability is given to BSs with larger $y_{nu}^{\dagger}$ values.

Algorithm 1: Randomized Rounding algorithm

```
1  Solve the linear relaxation of JSPRR problem to obtain
     $(\boldsymbol{x}^{\dagger}, \boldsymbol{y}^{\dagger})$ optimal solution
2  for $n \in \mathcal{N}$, $s \in \mathcal{S}$ do
3      Set $\widehat{x}_{ns} = 1$ with probability $x_{ns}^{\dagger}$
   end
4  for $u \in \mathcal{U}$ do
5      Define $\mathcal{N}_u' = (n \in \mathcal{N}_u \ : \ \widehat{x}_{ns_u} > 0)$ and
6      if $\mathcal{N}_u' = \emptyset$ then
7          set $\widehat{y}_{lu} = 1$ and $\widehat{y}_{nu} = 0 \ \forall n \in \mathcal{N}$
       else
8          set $\widehat{y}_{nu} = 1$, $n \in \mathcal{N}_u'$, with probability $\frac{y_{nu}^{\dagger}}{x_{ns_u}^{\dagger}}$, or
9          set $\widehat{y}_{lu} = 1$ with probability
             $\Big[\frac{y_{lu}^{\dagger} - \prod_{n\in\mathcal{N}_u'}(1-x_{ns_u}^{\dagger})}{1-\prod_{n\in\mathcal{N}_u'}(1-x_{ns_u}^{\dagger})}\Big]_+$
       end
   end
10 Output $\widehat{\boldsymbol{x}}, \widehat{\boldsymbol{y}}$
```

Subsequently, we provide guarantees on the quality of the solution returned by the Randomized Rounding algorithm. We begin with the following lemma.

Lemma 1. *The Randomized Rounding algorithm routes all user requests with high probability.*

Proof. For a given user u, there are two cases when rounding the fractional variable $y_{nu}^{\dagger}$ to $\widehat{y}_{nu}$: (i) there is no nearby BS having stored the requested service ($\mathcal{N}_u' = \emptyset$) and (ii) there is at least one such BS ($\mathcal{N}_u' \neq \emptyset$). The probability that the request of user u is routed to the cloud is given by:

$$\begin{aligned}
\Pr[\widehat{y}_{lu} = 1] &= \Pr\Big[\widehat{y}_{lu} = 1 \mid \mathcal{N}_u' = \emptyset\Big] \Pr\Big[\mathcal{N}_u' = \emptyset\Big] \\
&\quad + \Pr\Big[\widehat{y}_{lu} = 1 \mid \mathcal{N}_u' \neq \emptyset\Big] \Pr\Big[\mathcal{N}_u' \neq \emptyset\Big] \\
&= 1 \prod_{n\in\mathcal{N}_u'} (1 - x_{ns_u}^{\dagger}) \\
&\quad + \frac{y_{lu}^{\dagger} - \prod_{n\in\mathcal{N}_u'}(1-x_{ns_u}^{\dagger})}{1-\prod_{n\in\mathcal{N}_u'}(1-x_{ns_u}^{\dagger})}(1 - \prod_{n\in\mathcal{N}_u'}(1 - x_{ns_u}^{\dagger})) \\
&= y_{lu}^{\dagger}
\end{aligned} \tag{19}$$

The first equation is by the definition of conditional probability. The second equation is because the request of user u will be routed with probability 1 to the cloud if $\mathcal{N}_u' = \emptyset$, and the $\{x_{ns}^{\dagger}\}$ variables are rounded independently of one another (hence, $\Pr[\mathcal{N}_u' = \emptyset] = \prod_{n\in\mathcal{N}_u'}(1 - x_{ns_u}^{\dagger})$). To simplify the analysis, we have assumed that $y_{lu}^{\dagger} \geq \prod_{n\in\mathcal{N}_u'}(1 - x_{ns_u}^{\dagger})$ and, thus, removed the $[.]_+$ operator from the respective probability. This condition is expected to hold in practice as the BS deployment becomes more and more dense (larger $\mathcal{N}_u'$ sets) and the BS storage capacities increase (larger $x_{ns_u}^{\dagger}$ values).

Similarly, the probability that user u is routed to BS n is:

$$\begin{aligned}\Pr[\widehat{y}_{nu}=1] &= \Pr\left[\widehat{y}_{nu}=1 \mid \widehat{x}_{ns_u}=1\right]\Pr\left[\widehat{x}_{ns_u}=1\right] \\ &\quad + \Pr\left[\widehat{y}_{nu}=1 \mid \widehat{x}_{ns_u}=0\right]\Pr\left[\widehat{x}_{ns_u}=0\right] \\ &= \frac{y^{\dagger}_{nu}}{x^{\dagger}_{ns_u}}x^{\dagger}_{ns_u} = y^{\dagger}_{nu} \end{aligned} \tag{20}$$

The sum of probabilities of routing the request of user u to the cloud or the BSs is:

$$\sum_{n\in\mathcal{N}_u\cup\{l\}}\Pr[\widehat{y}_{nu}=1] = \sum_{n\in\mathcal{N}_u\cup\{l\}} y^{\dagger}_{nu} = 1 \tag{21}$$

where the last equation holds due to (3). The above is very close to the probability of routing the request of user u except for an additive gap that converges to zero as the number of BSs covering user u increases. □

By construction, Randomized Rounding routes requests only to BSs that have stored the respective service (in $\mathcal{N}'_u$ set in line 5) or to the cloud. Therefore, constraint (4) is also satisfied. Next, we study whether the remaining constraints in (5), (6), (7) and (8) are satisfied.

Lemma 2. *The solution returned by the Randomized Rounding algorithm satisfies in expectation the storage, computation and bandwidth capacity constraints in (5), (6), (7), and (8).*

Proof. We begin with the storage capacity constraint. The expected amount of data placed in BS n is given by:

$$\mathbb{E}[\sum_{s\in\mathcal{S}}\widehat{x}_{ns}r_s] = \sum_{s\in\mathcal{S}}\Pr[\widehat{x}_{ns}=1]r_s = \sum_{s\in\mathcal{S}}x^{\dagger}_{ns}r_s = R_n \tag{22}$$

where the second equation is because the $\{\widehat{x}_{ns}\}$ variables are binary, with success probabilities the fractional values $\{x^{\dagger}_{ns}\}$. The last equation is due to constraint (5) and the fact that it would be wasteful to not use all the storage space.

Next, we consider the computation capacity constraint. The expected computation load of BS n is given by:

$$\mathbb{E}[\sum_{u\in\mathcal{U}}\widehat{y}_{nu}c_{s_u}] = \sum_{u\in\mathcal{U}}\Pr[\widehat{y}_{nu}=1]c_{s_u} = \sum_{u\in\mathcal{U}}y^{\dagger}_{nu}c_{s_u} \leq C_n \tag{23}$$

where the second equation holds due to equation (20). The inequality is by constraint (6). Similar inequalities can be shown for the uplink/downlink bandwidth constraints:

$$\mathbb{E}[\sum_{u\in\mathcal{U}}\widehat{y}_{nu}b^{\uparrow}_{s_u}] = \sum_{u\in\mathcal{U}}\Pr[\widehat{y}_{nu}=1]b^{\uparrow}_{s_u} = \sum_{u\in\mathcal{U}}y^{\dagger}_{nu}b^{\uparrow}_{s_u} \leq B^{\uparrow}_n \tag{24}$$

$$\mathbb{E}[\sum_{u\in\mathcal{U}}\widehat{y}_{nu}b^{\downarrow}_{s_u}] = \sum_{u\in\mathcal{U}}\Pr[\widehat{y}_{nu}=1]b^{\downarrow}_{s_u} = \sum_{u\in\mathcal{U}}y^{\dagger}_{nu}b^{\downarrow}_{s_u} \leq B^{\downarrow}_n \tag{25}$$

where we have used equations (20), (7), and (8). □

A similar result holds for the objective function value.

Lemma 3. *The objective value returned by the Randomized Rounding algorithm is in expectation equal to that of the optimal fractional solution.*

Proof. The expected number of user requests routed to the cloud by Randomized Rounding is given by:

$$\mathbb{E}[\sum_{u\in\mathcal{U}}\widehat{y}_{lu}] = \sum_{u\in\mathcal{U}}\Pr[\widehat{y}_{lu}=1] = \sum_{u\in\mathcal{U}}y^{\dagger}_{lu} \tag{26}$$

where the second equation holds due to equation (19). □

The above lemmas have shown that the Randomized Rounding algorithm satisfies *in expectation* all the constraints and achieves the optimal objective function value. However, *in practice*, the constraints may be violated. Therefore, it is important to bound the factor by which this happens.

Theorem 1. *The amount of data placed by the Randomized Rounding algorithm in BS n will not exceed its storage capacity by a factor larger than $\frac{3\ln(S)}{R_n}+4$ with high probability.*

Proof. For a given BS n, the products $\widehat{x}_{ns}r_s$ $\forall s\in\mathcal{S}$ are independent random variables with expected total value $\mathbb{E}[\sum_{s\in\mathcal{S}}\widehat{x}_{ns}r_s] = R_n$ (cf. equation (22)). Moreover, by appropriately normalizing the r_s and R_n values, we can ensure that the $\widehat{x}_{ns}r_s$ variables take values within $[0,1]$. Therefore, we can apply the Chernoff Bound theorem [18] to show that for any $\epsilon>0$:

$$\Pr[\sum_{s\in\mathcal{S}}\widehat{x}_{ns}r_s \geq (1+\epsilon)R_n] \leq \exp^{\frac{-\epsilon^2 R_n}{2+\epsilon}} \tag{27}$$

Next, we find an ϵ value for which the probability upper bound above becomes very small. Specifically, we require that:

$$\exp^{\frac{-\epsilon^2 R_n}{2+\epsilon}} \leq \frac{1}{S^3} \tag{28}$$

which means that the probability bound goes quickly (at a cubic rate) to zero as the number of services increases. In order for this to be true, the ϵ value must satisfy:

$$\epsilon \geq \frac{3\ln(S)}{2R_n} + \sqrt{\frac{9\ln^2(S)}{4R_n^2}+\frac{6\ln(S)}{R_n}} \tag{29}$$

The above condition holds if we pick:

$$\epsilon = \frac{3\ln(S)}{R_n}+3 \tag{30}$$

since, in practice, $R_n \geq \ln(S)$. Finally, we upper bound the probability that *any* of the BS storage capacities is violated:

$$\begin{aligned}&\Pr[\bigcup_{n\in\mathcal{N}}\sum_{s\in\mathcal{S}}\widehat{x}_{ns}r_s \geq (1+\epsilon)R_n] \\ &\leq \sum_{n\in\mathcal{N}}\Pr[\sum_{s\in\mathcal{S}}\widehat{x}_{ns}r_s \geq (1+\epsilon)R_n] \\ &\leq N\frac{1}{S^3} \leq \frac{1}{S^2}\end{aligned} \tag{31}$$

where the first inequality is due to the Union Bound theorem. The second inequality is due to inequality (28) and because the number of BSs is N. The last inequality is because, in practice, the service library size is larger than the number of BSs ($S>N$). Therefore, with high probability, the storage capacity of any BS n will not be exceeded by more than a factor of $1+\epsilon = \frac{3\ln(S)}{R_n}+4$. □

Theorem 2. *The computation load of BS n returned by the Randomized Rounding algorithm will not exceed its capacity by a factor of $\frac{3\ln(S)}{\lambda^\dagger} + 4$ with high probability, where $\lambda^\dagger$ is the minimum computation load among BSs in the optimal fractional solution.*

Proof. The proof is similar to Theorem 1. For a given BS n, the variables $\widehat{y}_{nu}c_{s_u}$ $\forall u \in \mathcal{U}$ are independent with expected total value $\mathbb{E}[\sum_{u\in\mathcal{U}} \widehat{y}_{nu}c_{s_u}] = \sum_{u\in\mathcal{U}} y^\dagger_{nu}c_{s_u}$ (cf. inequality (23)). Moreover, they can be normalized to take values within $[0, 1]$. Therefore, we can apply the Chernoff Bound theorem:

$$\Pr[\sum_{u\in\mathcal{U}} \widehat{y}_{nu}c_{s_u} \geq (1+\epsilon) \sum_{u\in\mathcal{U}} y^\dagger_{nu}c_{s_u}] \leq \exp^{\frac{-\epsilon^2 \sum_{u\in\mathcal{U}} y^\dagger_{nu}c_{s_u}}{2+\epsilon}} \quad (32)$$

Unlike storage, however, the expected computation load may not be equal to the capacity, i.e., $\sum_{u\in\mathcal{U}} y^\dagger_{nu}c_{s_u} \neq C_n$. Therefore, we cannot replace it in the above inequality. To overcome this obstacle, we use the fact that $\sum_{u\in\mathcal{U}} y^\dagger_{nu}c_{s_u} \leq C_n$ (by constraint (6)) and $\lambda^\dagger \leq \sum_{u\in\mathcal{U}} y^\dagger_{nu}c_{s_u}$ (by definition of $\lambda^\dagger$) to show the following two inequalities:

$$\Pr[\sum_{u\in\mathcal{U}} \widehat{y}_{nu}c_{s_u} \geq (1+\epsilon)C_n] \leq \Pr[\sum_{u\in\mathcal{U}} \widehat{y}_{nu}c_{s_u} \geq (1+\epsilon) \sum_{u\in\mathcal{U}} y^\dagger_{nu}c_{s_u}] \quad (33)$$

$$\exp^{\frac{-\epsilon^2 \sum_{u\in\mathcal{U}} y^\dagger_{nu}c_{s_u}}{2+\epsilon}} \leq \exp^{\frac{-\epsilon^2\lambda^\dagger}{2+\epsilon}} \quad (34)$$

By combining inequalities (32), (33), and (34), we obtain:

$$\Pr[\sum_{u\in\mathcal{U}} \widehat{y}_{nu}c_{s_u} \geq (1+\epsilon)C_n] \leq \exp^{\frac{-\epsilon^2\lambda^\dagger}{2+\epsilon}} \quad (35)$$

To complete the proof, we will find an ϵ value for which the probability upper bound above becomes very small, i.e., at most $1/S^3$. Similarly to Theorem 1, we can set $\epsilon = \frac{3\ln(S)}{\lambda^\dagger} + 3$. Then, we can upper bound the probability that *any* of the computation capacities is violated by:

$$\begin{aligned} &\Pr[\bigcup_{n\in\mathcal{N}} \sum_{u\in\mathcal{U}} \widehat{y}_{nu}c_{s_u} \geq (1+\epsilon)C_n] \\ &\leq \sum_{n\in\mathcal{N}} \Pr[\sum_{u\in\mathcal{U}} \widehat{y}_{nu}c_{s_u} \geq (1+\epsilon)C_n] \\ &\leq N\frac{1}{S^3} \leq \frac{1}{S^2} \end{aligned} \quad (36)$$

This means that, with high probability, the computation capacity of any BS n will not be exceeded by more than a factor of $1 + \epsilon = \frac{3\ln(S)}{\lambda^\dagger} + 4$. □

Using similar arguments, the following two theorems can be proved for the uplink and downlink bandwidth capacities.

Theorem 3. *The uplink bandwidth load of BS n returned by the Randomized Rounding algorithm will not exceed its capacity by a factor of $\frac{3\ln(S)}{\mu^\dagger} + 4$ with high probability, where $\mu^\dagger$ is the minimum uplink bandwidth load among BSs in the optimal fractional solution.*

Theorem 4. *The downlink bandwidth load of BS n returned by the Randomized Rounding algorithm will not exceed its capacity by a factor of $\frac{3\ln(S)}{\nu^\dagger} + 4$ with high probability, where $\nu^\dagger$ is the minimum downlink bandwidth load among BSs in the optimal fractional solution.*

What remains it to describe the worst case performance of the (in expectation optimal) Randomized Rounding algorithm.

Theorem 5. *The objective value returned by Randomized Rounding algorithm is at most $\frac{2\ln(S)}{\xi^\dagger} + 3$ times worse than the optimal with high probability, where $\xi^\dagger$ is the optimal objective value in the linear relaxed problem.*

Proof. The proof is similar to the previous theorems, yet the bound is tighter since we do not need to apply the Union Bound theorem. We begin by showing that:

$$\Pr[\sum_{u\in\mathcal{U}} \widehat{y}_{lu} \geq (1+\epsilon)\xi^\dagger] \leq \exp^{\frac{-\epsilon^2\xi^\dagger}{2+\epsilon}} \quad (37)$$

Since $\xi^\dagger \leq \widehat{\xi}$ where $\widehat{\xi}$ is the optimal integer solution value, it also holds that:

$$\Pr[\sum_{u\in\mathcal{U}} \widehat{y}_{lu} \geq (1+\epsilon)\widehat{\xi}] \leq \exp^{\frac{-\epsilon^2\xi^\dagger}{2+\epsilon}} \quad (38)$$

Next, we upper bound the right hand side of the above inequality by $1/S^2$. In order for this to be true, the ϵ value must satisfy the following condition:

$$\epsilon \geq \frac{\ln(S)}{\xi^\dagger} + \sqrt{\frac{\ln^2(S)}{\xi_n^{\dagger 2}} + \frac{4\ln(S)}{\xi^\dagger}} \quad (39)$$

The above condition holds if we pick:

$$\epsilon = \frac{2\ln(S)}{\xi^\dagger} + 2 \quad (40)$$

since, in practice, the requests will be more than the services ($\xi^\dagger \geq \ln(S)$). Thus, with high probability, performance will be at most $1 + \epsilon = \frac{2\ln(S)}{\xi^\dagger} + 3$ times worse than optimal. □

The factors in Theorems 1-5 are *bi-criteria approximations* with respect to both the objective value and capacity constraints. In many practical scenarios, these factors are constant. For example, consider a system with thousands of users generating requests for services in a library of size $S = 1,000$. Each BS can process up to a thousand requests ($C_n = 1,000$) and the minimum computation capacity utilization is 40% ($\lambda^\dagger = 400$). Then, the bi-criteria approximation factor becomes $\frac{3\ln(1000)}{400} + 4 \approx 4.05$.

V. Extension and Practical Cases

In this section, we discuss how to handle changes in the user demand. Besides, we describe how to make the solution of the Randomized Rounding algorithm satisfy the capacity constraints, thereby making the algorithm more practical.

A. Handling user demand changes

The service placement and request routing decisions are taken for a certain time period during which the demand is fixed and predicted. The demand, however, may change over time, e.g., after a few hours or even at a faster timescale depending on the scenario. The MEC operator will have to repeatedly predict the new demand for the next time period and *adapt* the service placement and request routing decisions accordingly. For example, the MEC operator should replace services that are no longer popular with other services that recently gained popularity.

The adaptation of the service placement is not without cost. In fact, replacing previously placed services with new ones would require from the BSs to download non-trivial amounts of data from the cloud through their backhaul links. This operation creates overheads which, depending on the timescale, can be significant and therefore should be avoided.

The Randomized Rounding algorithm can be extended to become aware of the service placement adaptation costs. To carry this out, we add a new constraint into the JSPRR problem. This constraint upper bounds by a constant D the total amount of data associated with the replaced services:

$$\sum_{n\in\mathcal{N}}\sum_{s\in\mathcal{S}} x_{ns}(1-x_{ns}^{p})r_s \leq D \tag{41}$$

where x_{ns}^{p} is the placement solution in the previous time period. Here, placing a service s at a BS n ($x_{ns}=1$) adds r_s to the adaptation cost if and only if that service was not placed in the previous time period ($x_{ns}^{p}=0$).

We note that all the presented lemmas and theorems still hold as they do not depend on the presence of constraint (41). What remains to analyze is how likely is for the rounded solution $\widehat{\boldsymbol{x}}$ returned by the algorithm to violate constraint (41). This is described in the following theorem.

Theorem 6. *The total amount of data associated with service placement adaptation will not exceed the upper bound D by a factor of $\frac{2\ln(S)}{D}+3$ with high probability.*

Proof. The proof is similar to the previous theorems. The Chernoff Bound is applied for the sum of random variables $\{x_{ns}(1-x_{ns}^{p})r_s\}$ the expected total value of which is D. □

B. Constructing a feasible solution

As the Randomized Rounding algorith may violate the storage capacities of the BSs by a factor of $3\ln(S)/R_n+4$, the MEC operator may not be able to store all the services required to ensure the performance guarantee of the algorithm. Similarly, the service placement may violate the limit of allowable adaptations D, while the request routing may overwhelm the computation and bandwidth capacities. To respond to such cases, the operator needs to convert the bi-criteria solution into a *feasible* solution, i.e., a solution that satisfies constraints (5), (6), (7), (8) and (41).

To obtain such a solution, we start with the service placement $\widehat{\boldsymbol{x}}$ outputted by the Randomized Rounding algorithm. Then, we iteratively perform the removal of a service from a BS that yields the minimum cloud load increment. When a service is removed from a BS, the user requests for that service previously routed to that BS are now re-directed to other nearby BSs with available bandwidth and computation (if any), otherwise to the cloud. The procedure ends when constraints (5) and (41) are satisfied. To satisfy the remaining (computation and bandwidth) constraints we perform one more step. That is, we iteratively re-direct a user request from an overloaded BS to another BS with available resources (if any) or the cloud, until there are not any overloaded BSs.

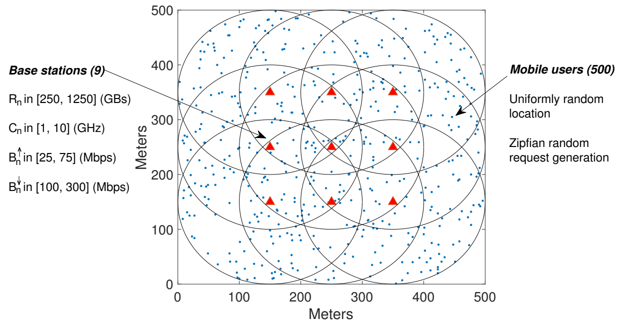

Base stations (9)
R_n in [250, 1250] (GBs)
C_n in [1, 10] (GHz)
$B_n^{\uparrow}$ in [25, 75] (Mbps)
$B_n^{\downarrow}$ in [100, 300] (Mbps)
Mobile users (500)
Uniformly random location
Zipfian random request generation
Meters
Meters
0
100
200
300
400
500
0
100
200
300
400
500

Fig. 3: Evaluation setup.

We need to emphasize that the above process may deteriorate the quality of the solution. However, as we show numerically in the next section, the obtained solution operates very close to the optimal in realistic settings.

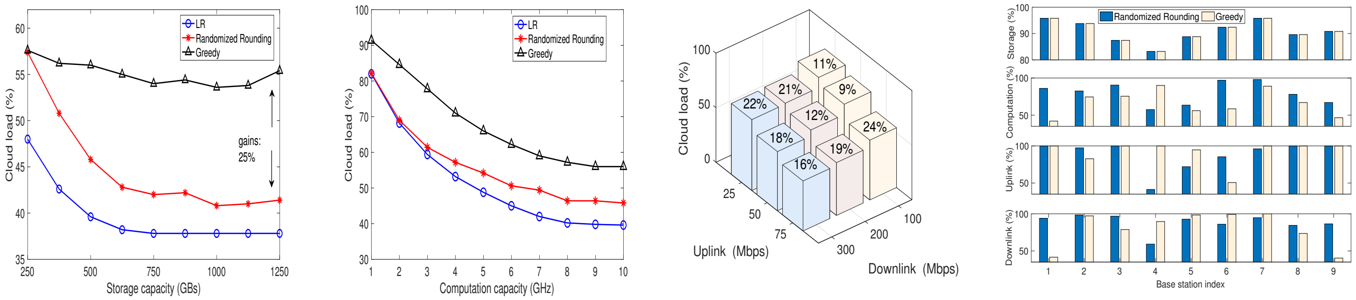
VI. Evaluation Results

In this section, we carry out evaluations to show the performance of the proposed Randomized Rounding algorithm. We consider a similar setup as in the previous work [8], depicted in Figure 3. Here, $N=9$ base stations (BSs) are regularly deployed on a grid network inside a 500m×500m area. $U=500$ mobile users are distributed uniformly at random over the BS coverage regions (each of 150m radius). Each user requests one service drawn from a library of $S=100$ services. The service popularity follows the Zipf distribution with shape parameter 0.8, which is a common assumption for several types of services such as video streaming.

For each BS n, we set the storage capacity to $R_n=500$ GBs, the computation capacity to $C_n=10$ GHz and the uplink (downlink) bandwidth capacity to $B_n^{\uparrow}=75$ ($B_n^{\downarrow}=250$) Mbps. Yet, all these values are varied during the evaluations. For each service s, we set the occupied storage r_s randomly within $[20,100]$ GBs. The required computation per request c_s takes value within $[0.1,0.5]$ GHz. The required uplink (downlink) bandwidth per request $b_s^{\uparrow}$ ($b_s^{\downarrow}$) takes value within $[1,5]$ ($[1,20]$) Mbps. We compare our algorithm with two baseline methods.

1) *Linear-Relaxation (LR):* The optimal (fractional) solution to the linear relaxation of JSPRR problem. This solution is found by running a linear solver and provides a lower bound to the optimal integer solution value.
2) *Greedy [14]:* Iteratively, places a service to a BS cache that reduces cloud load the most, until all caches are filled. Each request is routed to the nearest BS with the service, neglecting computation and bandwidth.

On the one hand, LR can be used as a benchmark to gauge the performance gap of our algorithm from optimal. On the other hand, it is well-known that Greedy algorithm achieves near-optimal performance for the traditional data placement (or caching) problem, leveraging its submodular property [14]. Therefore, a natural question to ask is whether the efficiency of Greedy is maintained or novel algorithms are



(a) Impact of storage capacity. (b) Impact of computation capacity. (c) Impact of bandwidth capacities. (d) Resource utilization per type and per BS.

Fig. 4: Cloud load for different (a) storage, (b) computation and (c) bandwidth capacities of BSs. (d) Utilization of resources.

needed when the placement of services with multidimensional resource requirements is considered.

We first explore the impact of storage capacity R_n $\forall n$ on the load of the centralized cloud. In Figure 4a, R_n spans a wide range of values, starting from 250GBs to 1250GBs. As expected, increasing storage capacities reduces cloud load for all the algorithms as more requests can be satisfied locally (offloaded) by the BSs. *The proposed Randomized Rounding algorithm performs significantly better than Greedy* with the gains increasing as the storage capacity increases (up to 25% for $R_n = 1250$GBs). At the same time, the gap from LR, and hence optimal, vanishes as R_n increases (below 10% for $R_n \geq 1000$GBs), which is consistent with the approximation factor expression in Theorem 1.

Next, we show the impact of computation capacity C_n in Figure 4b. While the cloud load reduces with C_n for all the algorithms, Randomized Rounding performs consistently better than Greedy and very close to LR. Especially when C_n is lower or equal to 3GHz, the gap from LR is less than 3%. Similarly, Figure 4c depicts the cloud load for different combinations of uplink ($B_n^{\uparrow}$) and downlink ($B_n^{\downarrow}$) bandwidth capacities. While the cloud load reduces with each of the $B_n^{\uparrow}$ and $B_n^{\downarrow}$ values, gains between 9% and 24% over Greedy are achieved (as shown in the bar labels).

Finally, we take a closer look into the utilization of BS resources when the Randomized Rounding and Greedy algorithms are used. The four subplots in Figure 4d show the resource utilization for each of the four resource types (storage, computation, uplink and downlink bandwidth). We observe that both algorithms utilize most of the available storage resources (90% or more for most BSs). Yet, Randomized Rounding manages to utilize more computation resources for 8 out of the 9 BSs. This in turn will facilitate the offloading of more requests to the BSs while spending roughly the same or even slightly less bandwidth resources than Greedy.

VII. Conclusion

In this paper, we studied joint service placement and request routing in MEC-enabled multi-cell networks with multidimensional (storage, computation and communication) constraints. Using a randomized rounding technique, we proposed an algorithm that achieves provably close-to-optimal performance, which, to the best of our knowledge, is the first approximation for this problem. This result can be of value in other research areas (e.g., data and middlebox placement). Interesting directions for future work include studying the coordination between BSs through backhaul links as well as the generalization of our model to services with multiple (chained) functions [19].

References

[1] P. Mach, Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading'", *IEEE Communications Surveys & Tutorials., vol. 19, no. 3, pp. 1628-1656*, 2017.

[2] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, V. Young, "Mobile Edge Computing - A Key Technology Towards 5G", *ETSI White Paper*, 2015.

[3] M. Chen, Y. Hao, "Task Offloading for Mobile Edge Computing in Software Defined Ultra-Dense Network", *IEEE Journal on Selected Areas in Communications, vol 36, no. 3, pp. 587-597*, 2018.

[4] H. Guo, J. Liu, J. Zhang, W. Sun, N. Kato, "Mobile-Edge Computation Offloading for Ultra-Dense IoT Networks", *IEEE Internet of Things Journal*, 2018.

[5] P. Jain, J. Manweiler, R. R. Choudhury, "Low Bandwidth Offload for Mobile AR", *ACM CoNEXT*, 2016.

[6] M.S. Elbamby, C. Perfecto, M. Bennis, K. Doppler,"Towards Low-Latency and Ultra-Reliable Virtual Reality", *IEEE Network*, 2018.

[7] X. Ge, S. Tu, G. Mao, C. X. Wang, T. Han, "5G Ultra-Dense Cellular Networks", *IEEE Wireless Communications, vol. 23, no. 1*, 2016.

[8] J. Xu, L. Chen, P. Zhou, "Joint Service Caching and Task Offloading for Mobile Edge Computing in Dense Networks", *IEEE Infocom*, 2018.

[9] T. He, H. Khamfroush, S. Wang, T.L. Porta, S. Stein, "It's Hard to Share: Joint Service Placement and Request Scheduling in Edge Clouds with Sharable and Non-sharable Resources", *IEEE ICDCS*, 2018.

[10] M. Chen, Y. Hao, L. Hu, M.S. Hossain, A. Ghoneim, "Edge-CoCaCo: Toward Joint Optimization of Computation, Caching, and Communication on Edge Cloud", *IEEE Wireless Communications, vol. 25, no. 3*, 2018.

[11] A. Srinivasan, "Approximation Algorithms Via Randomized Rounding: A Survey", *Advanced Topics in Mathematics, PWN, pp. 9-71*, 1999.

[12] H. Kellerer, U. Pferschy, D. Pisinger, "Knapsack Problems", *Springer*, 2004.

[13] I. Baev, R. Rajaraman, C. Swamy, "Approximation Algorithms for Data Placement Problems", *SIAM Journal on Comp., vol. 38*, 2008.

[14] K. Shanmugam, N. Golrezaei, A. Dimakis, A. Molisch and G. Caire, "FemtoCaching: Wireless Content Delivery Through Distributed Caching Helpers", *IEEE Transactions on Information Theory, vol. 59, no. 12*, 2013.

[15] T. Lukovszki, M. Rost, S. Schmid, "It's a Match! Near-Optimal and Incremental Middlebox Deployment", *ACM SIGCOMM Comput. Commun. Rev., vol. 46, no. 1, pp. 30-36*, 2016.

[16] T Lukovszki, M Rost, S Schmid , "Approximate and Incremental Network Function Placement", *arXiv:1706.06496v1*, 2017.

[17] T. Horel, Y. Singer, "Maximization of Approximately Submodular Functions", *NIPS*, 2016.

[18] M. Mitzenmacher, E. Upfal, "Probability and Computing : Randomized Algorithms and Probabilistic Analysis", *Cambridge Univ. Press*, 2005.

[19] H. Feng, J. Llorca, A.M. Tulino, D. Raz, A.F. Molisch, "Approximation Algorithms for the NFV Service Distribution Problem", *IEEE Infocom*, 2017.