Service Placement with Provable Guarantees in Heterogeneous Edge Computing Systems

Stephen Pasteris*, Shiqiang Wang†, Mark Herbster*, Ting He‡
*University College London, London, UK. Email: {s.pasteris, m.herbster}@cs.ucl.ac.uk
†IBM T. J. Watson Research Center, Yorktown Heights, NY, USA. Email: wangshiq@us.ibm.com
‡Pennsylvania State University, University Park, PA, USA. Email: t.he@cse.psu.edu

***Abstract*—Mobile edge computing (MEC) is a promising technique for providing low-latency access to services at the network edge. The services are hosted at various types of edge nodes with both computation and communication capabilities. Due to the heterogeneity of edge node characteristics and user locations, the performance of MEC varies depending on where the service is hosted. In this paper, we consider such a heterogeneous MEC system, and focus on the problem of placing multiple services in the system to maximize the total reward. We show that the problem is NP-hard via reduction from the set cover problem, and propose a deterministic approximation algorithm to solve the problem, which has an approximation ratio that is not worse than $\left(1-e^{-1}\right)/4$. The proposed algorithm is based on two subroutines that are suitable for small and arbitrarily sized services, respectively. The algorithm is designed using a novel way of partitioning each edge node into multiple slots, where each slot contains one service. The approximation guarantee is obtained via a specialization of the method of conditional expectations, which uses a randomized procedure as an intermediate step. In addition to theoretical guarantees, simulation results also show that the proposed algorithm outperforms other state-of-the-art approaches.**

I. Introduction

Many emerging applications such as the Internet of Things (IoT), virtual/augmented reality, etc. require low-latency access to services at the network edge. Mobile edge computing (MEC) has emerged as a key technology to make this possible [1], [2]. In MEC, services are hosted at edge nodes with communication, computation, and storage capabilities. The edge nodes can include micro servers, IoT gateways, routers, mobile devices, etc. They are connected to the wide-area network (WAN) and provide low-latency service to users that are within a suitable communication distance. An example of an MEC system is shown in Fig. 1.

A major challenge in MEC is to decide which services each edge node should host in order to satisfy the user demand, which we refer to as the *service placement* problem. The service placement has to take into account the heterogeneity of edge nodes, services, and users. For example, the response time of edge services can vary significantly depending on the network interface and hardware configuration of edge nodes [3]. Users can have different communication latencies to different edge nodes. Different services may consume different amount of resource and are compatible with different operating systems and hardware. All these aspects pose significant challenges to solving the service placement problem.

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

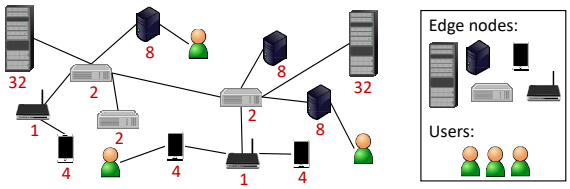

Fig. 1: Example of an MEC system with heterogeneous edge nodes that are arbitrarily interconnected with each other. The numbers below each edge node indicate some capacity notion (such as storage capacity for service programs) of the node. The user can connect to the system via any edge node.

Due to these challenges, existing work on service placement often has limitations in terms of practicality and performance guarantee. Heuristic algorithms without approximation guarantees are proposed in [4]–[7]. Among those approaches that provide approximation/optimality guarantees, [8]–[11] focus on offloading decisions, which do not consider the placement of services onto multiple edge nodes that can host services for other users. The work in [12] focuses on job scheduling for multiple edge nodes, but does not incorporate parallel execution of multiple services at the same edge node. Others consider the trade-off between delay and energy [13], [14], which neglect the heterogeneity of edge node platforms where some service may require hardware that only exists on some specific edge nodes. Elastic services that can be partitioned in arbitrary ways is considered in [15], which can be unrealistic in practice because it is usually impossible to split a computer program arbitrarily. The work in [16]–[18] does not consider concrete capacity limits, which therefore cannot capture the strict resource limitation of edge nodes. The work in [19] proposes a greedy service placement algorithm that can be shown to have a constant approximation ratio when all services have the same size and the reward is homogeneous. No approximation guarantee is shown for the heterogeneous setting. In addition, many of the existing algorithms, other than [19], only guarantee non-constant approximation ratios.

Different from the above existing work, in this paper, we consider an MEC system with 1) non-splittable service entities that are shareable among multiple users, 2) heterogeneous service and edge node sizes, 3) heterogeneous rewards of serving users, and 4) strict capacity limits of edge nodes. There are an arbitrary number of edge nodes, services, and users, where each user requires a service. We propose a

constant factor approximation algorithm to find a feasible service placement that maximizes the total system reward.

The challenge in our problem is that many standard techniques for approximation algorithms, such as those used in [19], are either not applicable or can only provide a bad approximation ratio. For example, as we will show in Section VI-C, the greedy algorithm used in [19] can perform arbitrarily badly for our problem. We design our algorithm based on a novel approach that partitions each node into multiple slots, together with a highly non-trivial way of applying the idea of the method of conditional expectations [20].

Our main contributions in this paper are as follows:

1) We formulate the general service placement (GSP) problem as described above, and convert it to an equivalent problem that we call service placement with set constraints (SPSC) which is easier to approximate. We show that both GSP and SPSC are NP-hard.

2) For the case where all services are small compared to the capacities of edge nodes, we propose an algorithm that solves SPSC with an approximation ratio[1] of $1 - e^{-\left(1-\sqrt{\beta}\right)^2}$, where β is the maximum size of any service divided by the minimum capacity of any node.

3) For the general case with arbitrary service sizes and node capacities, we propose an algorithm that solves SPSC with an approximation ratio of $\left(1 - e^{-1}\right)/4$.

4) We combine the above two algorithms and propose an algorithm that works for the general case and has an approximation ratio of $\max\left\{1 - e^{-\left(1-\sqrt{\beta}\right)^2}; \left(1 - e^{-1}\right)/4\right\}$.

5) We present simulation results that show the effectiveness of our proposed algorithms empirically.

II. Problem Formulation

A. General Service Placement (GSP)

In this paper, we aim at solving the GSP defined as follows.

1) Input: Let S denote the set of services, V denote the set of nodes, and U denote the set of users. For all $i \in S$, we define some $s_i \in \mathbb{R}^+$ as the size of service i. For all $j \in V$, we define some $c_j \in \mathbb{R}^+$ as the capacity of node j. For all $k \in U$, we define some $\xi_k \in S$ as the service required by user k. For all $k \in U$, we define a map $\hat{r}_k : V \to \mathbb{R}^+$, where $\hat{r}_k(j)$ is the reward obtained for serving user k if its service ξ_k is provided by node j. The size and capacity definitions can be related to storage or any other type of resource that can be shared among multiple users. Users that require non-shareable resource can be considered as requiring different services. The reward can be defined as related to the service quality (such as response time) perceived by the user, and its value can differ by users, services, and service placement configurations. Such a reward definition can capture the heterogeneity in the operation system, hardware, and networking aspects. We also note that a user does not need to be a "real" user, it can be any instance of service request.

[1]For the maximization problem we consider in this paper, we define the approximation ratio ρ as $\rho \cdot \mathrm{OPT} \leq R^* \leq \mathrm{OPT}$, where R^* is the solution from the approximation algorithm and OPT is the true optimal solution. A larger approximation ratio indicates a better performance.

2) Service Placement: A *service placement* X is an indexed set $\{X_i : i \in S\}$ where, for every $i \in S$, we have $X_i \subseteq V$, and X_i is the set of nodes that service i is placed on. A service placement X is *feasible* if and only if (iff):

$$\sum_i s_i \cdot \mathcal{I}\left(j \in X_i\right) \leq c_j \tag{1}$$

for all $j \in V$, where we define $\mathcal{I}(\cdot)$ to be the indicator function, i.e., $\mathcal{I}(E) := 1$ if E is true and $\mathcal{I}(E) := 0$ otherwise. Condition (1) states that the total size of services hosted at a node does not exceed its capacity.

3) Objective: In our system, if there are multiple nodes containing service ξ_k, then the service of user k is provided by a node j that gives the maximum reward $\hat{r}_k(j)$. If no node contains ξ_k then we obtain no reward from user k. With this definition, given a service placement X, the reward given to us by a user k is equal to $\max_{j \in X_{\xi_k}} \hat{r}_k(j)$, where $\max_{j \in \emptyset} \hat{r}_k(j) := 0$. The objective of GSP to find a feasible service placement X that maximizes the total reward, i.e.:

$$\max_X \quad \sum_{k \in U} \max_{j \in X_{\xi_k}} \hat{r}_k(j) \tag{2}$$
$$\text{s.t.} \quad \text{Condition (1).}$$

B. Service Placement with Set Constraints (SPSC)

We first introduce the SPSC problem as follows. Then, we will show that both GSP and SPSC are NP-hard. Afterwards, we show how to convert any GSP to an equivalent SPSC and focus on approximation algorithms for SPSC.

In SPSC, we reuse the definitions in Section II-A1 except for the reward. The reward for each user in SPSC is defined by a subset of nodes as follows. For all $k \in U$, we define some $W_k \subseteq V$ to represent a set of nodes, one of these nodes must contain service ξ_k in order for user k to be satisfied. For all $k \in U$, we define some $r_k \in \mathbb{R}^+$ to represent the reward received by the system if user k is satisfied.

Given a service placement X, a user k is satisfied (so the system receives reward r_k) iff $W_k \cap X_{\xi_k} \neq \emptyset$, i.e., service ξ_k is placed on some node in W_k. Our objective is to find a feasible service placement that maximizes the total reward:

$$\max_X \quad \sum_{k \in U} r_k \cdot \mathcal{I}\left(W_k \cap X_{\xi_k} \neq \emptyset\right) \tag{3}$$
$$\text{s.t.} \quad \text{Condition (1).}$$

Let R denote the true (but unknown) optimal value of the objective in (3).

Theorem 1. *Both GSP and SPSC are NP-hard.*

Proof. The proof is based on reduction from the decision version of the set cover problem, which is NP-complete [21]. See our technical report [22, Appendix A] for details. □

C. Converting GSP to SPSC

Because GSP is NP-hard, we seek for approximate solutions. It is difficult to approximate GSP directly. Therefore, we transform GSP to an equivalent SPSC problem, and propose approximation algorithms for SPSC.

Suppose we have a GSP instance. We will now construct an equivalent instance of SPSC. For clarity we will, in this subsection, refer to the users we construct for the SPSC

Algorithm 1 Converting GSP to SPSC

1: For all $l \in U$:
2: Order V as $j_{l,1}, j_{l,2}, ..., j_{l,|V|}$ so that $\hat{r}_l(j_{l,b}) \geq \hat{r}_l(j_{l,b+1}), \forall b < |V|$;
3: For all $b \leq |V|$:
4: Create a restricted user $k_{(l,b)}$ with:
5: $\xi_{k_{(l,b)}} \leftarrow \xi_l$;
6: $W_{k_{(l,b)}} \leftarrow \{j_{l,b'} : b' \leq b\}$;
7: If $b < |V|$ then $r_{k_{(l,b)}} \leftarrow \hat{r}_l(j_{l,b}) - \hat{r}_l(j_{l,b+1})$;
8: If $b = |V|$ then $r_{k_{(l,b)}} \leftarrow \hat{r}_l(j_{l,b})$;
9: Output $U' \leftarrow \{k_{(l,b)} : l \in U, b \leq |V|\}$;

instance as "restricted users" and define U' to be the set of restricted users we construct. Note that U is the set of users in the original GSP instance. The sets S and V, as well as the values $\{s_i : i \in S\}$ and $\{c_j : j \in V\}$ in the SPSC instance are the same as in the GSP instance. This implies that a service placement is feasible for the SPSC instance iff it is feasible for the GSP instance.

The set of restricted users, as well as their associated sets, services and rewards, is constructed in Algorithm 1. In the rest of this subsection, we will use the notation introduced in Algorithm 1 to show that the two problems are equivalent.

Theorem 2. *If we have a feasible service placement X, then for all $l \in U$ we have:*

$$\max_{j \in X_{\xi_l}} \hat{r}_l(j) = \sum_{b \leq |V|} r_{k_{(l,b)}} \cdot \mathcal{I}\left(W_{k_{(l,b)}} \cap X_{\xi_{k_{(l,b)}}} \neq \emptyset\right).$$

Proof. Define $\hat{b} := \min\{b : j_{l,b} \in X_{\xi_l}\}$. For all $b' < \hat{b}$ we have $j_{l,b'} \notin X_{\xi_l}$. Also for all $b < \hat{b}$ we have, by Line 6 of Algorithm 1, that $W_{k_{(l,b)}} \subseteq \{j_{b'} : b' < \hat{b}\}$ which hence does not intersect with X_{ξ_l}. On the other hand, if $b \geq \hat{b}$ then, by Line 6 of Algorithm 1, we have $j_{l,\hat{b}} \in W_{k_{(l,b)}}$. By definition $j_{l,\hat{b}}$ is also in X_{ξ_l} so $W_{k_{(l,b)}}$ intersects with X_{ξ_l}.

Since for all $b \leq |V|$ we have $X_{\xi_{k_{(l,b)}}} = X_{\xi_l}$, we have shown that for any $b \leq |V|$ we have $W_{k_{(l,b)}} \cap X_{\xi_{k_{(l,b)}}} \neq \emptyset$ if and only if $b \geq \hat{b}$. This implies:

$$\sum_{b \leq |V|} r_{k_{(l,b)}} \cdot \mathcal{I}\left(W_{k_{(l,b)}} \cap X_{\xi_{k_{(l,b)}}} \neq \emptyset\right) = \sum_{\hat{b} \leq b \leq |V|} r_{k_{(l,b)}}$$
$$= \hat{r}_l(j_{l,|V|}) + \sum_{\hat{b} \leq b < |V|} (\hat{r}_l(j_{l,b}) - \hat{r}_l(j_{l,b+1})) = \hat{r}_l(j_{\hat{b}}).$$

where the second equality follows from Lines 7 and 8 of Algorithm 1. Now suppose there exists some node $j' \in V$ which satisfies $\hat{r}_l(j') > \hat{r}_l(j_{\hat{b}})$. Let b' be such that $j' = j_{l,b'}$. By the ordering in Line 2 we then have $b' < \hat{b}$ and hence $j_{l,b'} \notin X_{\xi_l}$. This shows that $\max_{j \in X_{\xi_l}} \hat{r}_l(j) = \hat{r}_l(j_{\hat{b}})$ (as $j_{\hat{b}} \in X_{\xi_l}$) which, combining with above, proves the theorem. □

By Theorem 2 we have:

$$\sum_{l \in U} \max_{j \in X_{\xi_l}} \hat{r}_l(j) = \sum_{l \in U} \sum_{b \leq |V|} r_{k_{(l,b)}} \cdot \mathcal{I}\left(W_{k_{(l,b)}} \cap X_{\xi_{k_{(l,b)}}} \neq \emptyset\right)$$
$$= \sum_{k \in U'} r_k \cdot \mathcal{I}\left(W_k \cap X_{\xi_k} \neq \emptyset\right).$$

Thus, for any feasible[2] service placement X, the reward of the GSP instance is the same as the reward of the SPSC instance. This shows that with the conversion given in Algorithm 1, the two problems are equivalent.

[2]Note that a service placement X that is feasible for GSP is also feasible for SPSC, and vice versa, because the feasibility of both GSP and SPSC are specified by (1).

Outline: In the following, we present algorithms to solve SPSC with approximation guarantees. The approximation algorithm starts with solving a linear program (LP) presented in Section III. Then, the algorithm is based on a notion of "slot allocation" that will be explained later. We present two slot allocation algorithms, referred to as SA1 and SA2, in Sections IV and V, respectively. Then, in Section VI, we present an algorithm that combines SA1 and SA2 which is the final algorithm for solving SPSC (and thus GSP). Section VII presents simulation results. Section VIII discusses some further related work, and Section IX draws conclusion.

III. Linear Programming Step

Define indexed sets $\{\omega_{i,j} : i \in S, j \in V\} \subseteq \mathbb{R}^+$ and $\{\alpha_k : k \in U\} \subseteq \mathbb{R}^+$. Both SA1 and SA2 for solving SPSC solve the following LP as a first step:

$$\max_{\{\omega_{i,j}\},\{\alpha_k\}} \quad \hat{R} := \sum_{k \in U} \alpha_k r_k \tag{4a}$$
$$\text{s.t.} \quad \alpha_k \leq \sum_{j \in W_k} \omega_{\xi_k,j}, \quad \forall k \in U, \tag{4b}$$
$$\alpha_k \leq 1, \qquad \forall k \in U, \tag{4c}$$
$$\sum_{i \in S} \omega_{i,j} s_i \leq c_j, \quad \forall j \in V, \tag{4d}$$
$$\omega_{i,j} = 0, \qquad \forall i \in S, j \in V : s_i > c_j, \tag{4e}$$
$$0 \leq \omega_{i,j} \leq 1, \qquad \forall i \in S, j \in V. \tag{4f}$$

Theorem 3. *We have $\hat{R} \geq R$.*

Proof. Choose a feasible service placement X that has a total reward of R. If we then define $\omega_{i,j} := \mathcal{I}(j \in X_i)$ and $\alpha_k := \mathcal{I}(W_k \cap X_{\xi_k} \neq \emptyset)$ it is clear that all the above constraints are satisfied and $\sum_{k \in U} \alpha_k r_k = R$. Hence, if we choose $\{\omega_{i,j} : i \in S, j \in V\} \subseteq \mathbb{R}^+$ and $\{\alpha_k : k \in U\} \subseteq \mathbb{R}^+$ that satisfy the constraints and maximize $\sum_{k \in U} \alpha_k r_k$ we must have $\sum_{k \in U} \alpha_k r_k \geq R$. □

IV. First Slot Allocation Algorithm (SA1)

For both SA1 and SA2, we define:

$$\gamma := 1 - \sqrt{\beta}; \qquad \delta := (1 - \sqrt{\beta})^2 \tag{5}$$

for some given $\beta < 1$. We use $\mathbb{N}$ to denote the set of natural numbers (excluding zero) throughout the paper.

Our first algorithm, SA1, for solving SPSC is used in the case that we have $\max_{i \in S} s_i \leq \beta(\min_{j \in V} c_j)$ for some given $\beta < 1$. SA1 has an approximation ratio of $1 - e^{-(1-\sqrt{\beta})^2}$, which will be shown in Theorem 7 later.

Definition 1. *After solving the LP in (4) we define the following for SA1. $\forall j \in V, q \in \mathbb{N}$:*

$$\Omega_{j,q} := \{i \in S : \gamma^q c_j \beta < s_i \leq \gamma^{q-1} c_j \beta\}; \quad \Delta_{j,q} := \sum_{i \in \Omega_{j,q}} \omega_{i,j};$$
$$\delta'_j := \frac{\delta c_j}{\sum_{i \in S} s_i \omega_{i,j}}; \qquad \eta_{j,q} := \lceil \delta'_j \Delta_{j,q} \rceil.$$

Procedure of SA1: First, we solve the LP in (4) to obtain $\{\omega_{i,j}\}$. Then, Algorithm 2 creates a set, Λ, of slots (see Section IV-A). After we have the set Λ, Algorithm 3 computes

Algorithm 2 Slot Creation of SA1

For all $j \in V$ and $q \in \mathbb{N}$ such that $\Omega_{j,q} \neq \emptyset$:
    Create $\eta_{j,q}$ slots σ with $\mu(\sigma) \leftarrow j$ and $\lambda(\sigma) \leftarrow q$;
Output Λ as the set of all slots created;

Algorithm 3 Service Placement

Receive Λ from the slot creation algorithm;
For every $\sigma \in \Lambda$: set $\tau'(\sigma) \leftarrow \emptyset$;
For every $\sigma \in \Lambda$:
    For every $i \in \Omega_{\mu(\sigma),\lambda(\sigma)}$:
        $\tau_i^*(\sigma) \leftarrow i$;
        For all $\sigma' \setminus \{\sigma\}$: set $\tau_i^*(\sigma') \leftarrow \tau(\sigma')$;
    $i' \leftarrow \operatorname{argmax}_{i \in \Omega_{\mu(\sigma),\lambda(\sigma)}} \mathcal{E}\left(\tau_i^*\right)$;
    $\tau'(\sigma) \leftarrow i'$;
$\psi \leftarrow \tau'$;
For all $i \in S$:
    Output $X_i^{\psi} \leftarrow \{j \in V : \exists \sigma \text{ with } \mu(\sigma) = j, \tau(\sigma) = i\}$;

the service placement X^{ψ}. In Algorithm 3, the objects τ' and ψ are maps from Λ into $\mathbb{N} \cup \{\emptyset\}$. The algorithm has a function $\mathcal{E}(\cdot)$ which takes, as input, a map from Λ into $\mathbb{N} \cup \{\emptyset\}$. This function $\mathcal{E}(\cdot)$ is computed in Algorithm 4.

In the rest of this section, we give a description of the mechanics of the algorithm and a proof of the approximation ratio and feasibility of the computed service placement X^{ψ}.

A. Slots Allocations

A *slot* σ is an object that has two associated values: $\mu(\sigma) \in V$ and $\lambda(\sigma) \in \mathbb{N}$. Intuitively, a slot σ is a space, with capacity $\gamma^{\lambda(\sigma)-1} c_j \beta$, on node $\mu(\sigma)$. A slot σ will hold a single service in $\Omega_{\mu(\sigma),\lambda(\sigma)}$. For all $j \in V$ and $q \in \mathbb{N}$, Algorithm 2 creates $\eta_{j,q}$ slots σ with $\mu(\sigma) := j$ and $\lambda(\sigma) := q$. Λ is the set of all slots created.

A *slot allocation* τ is a function from Λ into S such that, given a slot $\sigma \in \Lambda$, we have $\tau(\sigma) \in \Omega_{\mu(\sigma),\lambda(\sigma)}$. A slot allocation τ is an assignment of services to slots such that given a slot σ, the service $\tau(\sigma)$ assigned to it is in $\Omega_{\mu(\sigma),\lambda(\sigma)}$, implying that $\tau(\sigma)$ does not exceed the capacity of σ.

A *partial slot allocation* τ' is a function from Λ into $S \cup \{\emptyset\}$ such that, given a slot $\sigma \in \Lambda$, we have $\tau'(\sigma) \in \Omega_{\mu(\sigma),\lambda(\sigma)} \cup \{\emptyset\}$. A partial slot allocation τ' is a partial assignment of services to slots: given a slot σ, $\tau'(\sigma) = \emptyset$ means that no service has been assigned to σ (i.e. the slot is empty), and $\tau'(\sigma) \neq \emptyset$ means that service $\tau'(\sigma)$ has been assigned to σ (and, as for slot allocations, we have $\tau'(\sigma) \in \Omega_{\mu(\sigma),\lambda(\sigma)}$).

Given a partial slot allocation τ' we define $\mathcal{T}_{\tau'}$ as the set of all slot allocations τ, where $\tau(\sigma) = \tau'(\sigma)$ for all $\sigma \in \Lambda$ with $\tau'(\sigma) \neq \emptyset$. $\mathcal{T}_{\tau'}$ is the set of all slot allocations that can be obtained by assigning services to all the empty slots of τ'.

Given any slot allocation τ we define its *associated service placement*, X^{τ}, by:

$$X_i^{\tau} := \{j : \exists\ \sigma \in \Lambda \text{ with } \mu(\sigma) = j \text{ and } \tau(\sigma) = i\}$$

for all $i \in S$. This means that service i is placed on node j iff there exists a slot on node j which contains service i.

Given a slot allocation τ and a user $k \in U$, we define $f_{\tau,k} := 1$ if there exists a node j in W_k and a slot $\sigma \in \Lambda$ with $\mu(\sigma) = j$ and $\tau(\sigma) = \xi_k$, and $f_{\tau,k} := 0$ otherwise. Note that:

$$f_{\tau,k} = \mathcal{I}\left(W_k \cap X_{\xi_k}^{\tau} \neq \emptyset\right). \tag{6}$$

Theorem 4. *For any slot allocation τ (with set of slots Λ), its associated service placement X^{τ} is feasible.*

Algorithm 4 Computing $\mathcal{E}(\tau^*)$

Receive Λ from the slot creation algorithm;
For all $k \in U$ such that $\exists \sigma \in \Lambda$ with $\mu(\sigma) \in W_k, \tau^*(\sigma) = \xi_k$:
    $\theta_k \leftarrow 1$;
For all $k \in U$ such that $\nexists \sigma \in \Lambda$ with $\mu(\sigma) \in W_k, \tau^*(\sigma) = \xi_k$:
    $p \leftarrow 1$;
    For all $\sigma \in \Lambda$ with $\mu(\sigma) \in W_k$, $\tau^*(\sigma) = \emptyset$, $\xi_k \in \Omega_{\mu(\sigma),\lambda(\sigma)}$:
        $p \leftarrow \left(1 - \omega_{\xi_k,\mu(\sigma)} / \Delta_{\mu(\sigma),\lambda(\sigma)}\right) p$;
    $\theta_k = 1 - p$;
Output $\mathcal{E}(\tau^*) \leftarrow \sum_{k \in U} \theta_k r_k$;

Proof. First note that for all $j \in V$ and $i \in S$ we have:

$$\mathcal{I}\left(j \in X_i^{\tau}\right) = \mathcal{I}\left(\exists\ \sigma \in \Lambda \text{ with } \mu(\sigma) = j \text{ and } \tau(\sigma) = i\right)$$

so for all $j \in V$ we have:

$$\sum_{i \in S} s_i \cdot \mathcal{I}\left(j \in X_i^{\tau}\right) \leq \sum_{i \in S} \sum_{\sigma \in \Lambda : \mu(\sigma) = j, \tau(\sigma) = i} s_i = \sum_{\sigma \in \Lambda : \mu(\sigma) = j} s_{\tau(\sigma)}$$

$$= \sum_{q \in \mathbb{N}} \sum_{\sigma \in \Lambda : \mu(\sigma) = j, \lambda(\sigma) = q} s_{\tau(\sigma)} \overset{\text{ⓐ}}{\leq} \sum_{q \in \mathbb{N}} \sum_{\sigma \in \Lambda : \mu(\sigma) = j, \lambda(\sigma) = q} \gamma^{q-1} c_j \beta$$

$$\overset{\text{ⓑ}}{\leq} \sum_{q \in \mathbb{N}} \left(\delta_j' \sum_{i \in \Omega_{j,q}} \omega_{i,j} + 1\right) \gamma^{q-1} c_j \beta$$

$$= c_j \beta \left(\sum_{q \in \mathbb{N}} \gamma^{q-1}\right) + \frac{\delta_j'}{\gamma} \left(\sum_{q \in \mathbb{N}} \sum_{i \in \Omega_{j,q}} \omega_{i,j} \gamma^q \beta c_j\right)$$

$$= \frac{c_j \beta}{1-\gamma} + \frac{\delta_j'}{\gamma} \left(\sum_{q \in \mathbb{N}} \sum_{i \in \Omega_{j,q}} \omega_{i,j} \gamma^q \beta c_j\right) \leq \frac{c_j \beta}{1-\gamma} + \frac{\delta_j'}{\gamma} \left(\sum_{q \in \mathbb{N}} \sum_{i \in \Omega_{j,q}} \omega_{i,j} s_i\right)$$

$$= \frac{c_j \beta}{1-\gamma} + \frac{\delta_j'}{\gamma} \left(\sum_{i \in S} \omega_{i,j} s_i\right) \leq \frac{c_j \beta}{1-\gamma} + \frac{\delta c_j}{\gamma} = c_j$$

which proves the feasibility of X^{τ}. In the above, step ⓐ is because for all $\sigma \in \Lambda$ with $\mu(\sigma) = j$ and $\lambda(\sigma) = q$, we have $\tau(\sigma) \in \Omega_{j,q}$; step ⓑ is because $|\{\sigma \in \Lambda : \mu(\sigma) = j, \lambda(\sigma) = q\}| = \eta_{j,q} \leq \delta_j' \sum_{i \in \Omega_{j,q}} \omega_{i,j} + 1$. The other steps are mainly from the definitions in (5) and Definition 1. □

B. Probability Distribution for SA1

For the analysis of SA1, we define a probability distribution on the set of possible slot allocations. This will guide, via the method of conditional expectations [20], the construction of the slot allocation ψ in Algorithm 3. The probability distribution on slot allocations τ is defined as follows: for every slot $\sigma \in \Lambda$ independently, draw a service i from $\Omega_{\mu(\sigma),\lambda(\sigma)}$ with probability $\omega_{i,\mu(\sigma)} / \Delta_{\mu(\sigma),\lambda(\sigma)}$ and set $\tau(\sigma) \leftarrow i$.

Definition 2. *Given a partial slot allocation τ', define:*

$$\mathcal{E}(\tau') := \mathbb{E}\left(\sum_{k \in U} f_{\tau,k} r_k \middle| \tau \in \mathcal{T}_{\tau'}\right)$$

where $\mathbb{E}$ denotes the expectation. $\mathcal{E}(\cdot)$ appears in Algorithm 3 and is computed in Algorithm 4 (see Theorem 8).

The next theorem shows that when we draw τ from the above probability distribution, the expected total reward of its associated service placement is bounded below by $(1 - e^{-\delta})R$.

Theorem 5. *Under our probability distribution for SA1, the expected total reward $\mathbb{E}\left(\sum_{k \in U} f_{\tau,k} r_k\right) \geq (1 - e^{-\delta})R$.*

Proof. First note that:

$$\mathbb{E}\left(\sum_{k \in U} f_{\tau,k} r_k\right) = \sum_{k \in U} r_k \mathbb{E}(f_{\tau,k}) = \sum_{k \in U} r_k \mathbb{P}\left(f_{\tau,k} = 1\right)$$

where the last equality is since $f_{\tau,k}$ is boolean. We shall now bound $\mathbb{P}\left(f_{\tau,k}=1\right)$. Note first that $f_{\tau,k}=1$ if there exists a slot $\sigma\in\Lambda$ with $\mu(\sigma)\in W_k$ and $\tau(\sigma)=\xi_k$ so, since $\tau(\sigma)$ is drawn independently for every slot σ, we have, by Lemma 2 in [22, Appendix B], that:

$$\mathbb{P}\left(f_{\tau,k}=1\right)\geq\left(1-\exp\left(-\sum_{\sigma\in\Lambda:\mu(\sigma)\in W_k}\mathbb{P}\left(\tau(\sigma)=\xi_k\right)\right)\right).$$

Given a node $j\in V$ define $q_{j,k}$ such that $\xi_k\in\Omega_{j,(q_{j,k})}$. We then have

$$\begin{aligned}
&\sum_{\sigma\in\Lambda:\mu(\sigma)\in W_k}\mathbb{P}\left(\tau(\sigma)=\xi_k\right)=\sum_{j\in W_k}\sum_{\sigma\in\Lambda:\mu(\sigma)=j}\mathbb{P}\left(\tau(\sigma)=\xi_k\right)\\
&=\sum_{j\in W_k}\sum_{\sigma\in\Lambda:\mu(\sigma)=j,\lambda(\sigma)=q_{j,k}}\mathbb{P}\left(\tau(\sigma)=\xi_k\right)\\
&=\sum_{j\in W_k}\sum_{\sigma\in\Lambda:\mu(\sigma)=j,\lambda(\sigma)=q_{j,k}}\omega_{\xi_k,\mu(\sigma)}/\Delta_{\mu(\sigma),\lambda(\sigma)}\\
&=\sum_{j\in W_k}\sum_{\sigma\in\Lambda:\mu(\sigma)=j,\lambda(\sigma)=q_{j,k}}\omega_{\xi_k,j}/\Delta_{j,(q_{j,k})}\\
&=\sum_{j\in W_k}\frac{\omega_{\xi_k,j}}{\Delta_{j,(q_{j,k})}}\sum_{\sigma\in\Lambda:\mu(\sigma)=j,\lambda(\sigma)=q_{j,k}}1\\
&=\sum_{j\in W_k}\frac{\omega_{\xi_k,j}}{\Delta_{j,(q_{j,k})}}\eta_{j,(q_{j,k})}\\
&=\sum_{j\in W_k}\frac{\omega_{\xi_k,j}}{\Delta_{j,(q_{j,k})}}\left\lceil\delta\Delta_{j,(q_{j,k})}\right\rceil\geq\delta\sum_{j\in W_k}\omega_{\xi_k,j}\geq\delta\alpha_k.
\end{aligned}$$

Plugging into the above, we have:

$$\mathbb{P}\left(f_{\tau,k}=1\right)\geq 1-\exp(-\delta\alpha_k).$$

Since $\alpha_k\leq 1$, by Lemma 3 in [22, Appendix B], we have $1-\exp(-\delta\alpha_k)\geq\alpha_k(1-e^{-\delta})$ which gives us an expected total reward of:

$$\sum_{k\in U}r_k\alpha_k(1-e^{-\delta})=(1-e^{-\delta})\hat{R}\geq(1-e^{-\delta})R$$

where the last equality is from Theorem 3. □

Remark: The randomized step is only needed for theoretical analysis. It *does not* exist in the algorithm. Our algorithm (See Algorithms 2, 3, and 4) only needs to compute the expected value given in Definition 2 and *does not* include any randomized step. Hence, *our algorithm SA1 is deterministic*. The same applies to SA2 and other algorithms presented later.

C. Placing Services

We now describe and analyze Algorithm 3, which uses the idea of the method of conditional expectations [20].

In Algorithm 3, we maintain a partial slot allocation τ' where, initially, all slots are empty. Every time we go around the loop in Lines 4-8 of Algorithm 3, we do the following:

1) Pick an empty slot σ.
2) For all $i\in\Omega_{\mu(\sigma),\lambda(\sigma)}$ define τ^*_i to be the partial slot allocation formed from τ' by assigning the service i to the slot σ.
3) Choose $i'\in\Omega_{\mu(\sigma),\lambda(\sigma)}$ that maximizes $\mathcal{E}\left(\tau^*_{i'}\right)$.
4) Update τ' by assigning service i' to the slot σ.

We loop through the above until all slots have been assigned services (i.e., τ' is a slot allocation). We let ψ be the slot allocation that has now been constructed and output its associated service placement X^ψ. Theorem 4 shows that X^ψ is feasible. We now prove the approximation ratio.

Theorem 6. *The service placement X^ψ, computed by Algorithm 3 with slot creation computed by Algorithm 2, has a total reward of at least $\mathbb{E}\left(\sum_{k\in U}f_{\tau,k}r_k\right)$.*

Proof. We maintain the inductive hypothesis that $\mathcal{E}\left(\tau'\right)\geq\mathbb{E}\left(\sum_{k\in U}f_{\tau,k}r_k\right)$ throughout the algorithm.

Initially we have $\tau'(\sigma)=\emptyset$ for every $\sigma\in\Lambda$, thus $\mathcal{T}_{\tau'}$ is the set of all possible slot allocations and hence $\mathcal{E}\left(\tau'\right)=\mathbb{E}\left(\sum_{k\in U}f_{\tau,k}r_k\right)$, so the inductive hypothesis holds.

Now suppose that the inductive hypothesis holds at some point during the algorithm. After Line 7 and before Line 8 of Algorithm 3, we have:

$$\begin{aligned}
\mathcal{E}\left(\tau^*_{i'}\right)&=\max_{i\in\Omega_{\mu(\sigma),\lambda(\sigma)}}\mathcal{E}\left(\tau^*_i\right)=\max_{i\in\Omega_{\mu(\sigma),\lambda(\sigma)}}\mathbb{E}\left(\sum_{k\in U}f_{\tau,k}r_k\middle|\tau\in\mathcal{T}_{\tau^*_i}\right)\\
&\geq\mathbb{E}\left(\sum_{k\in U}f_{\tau,k}r_k\middle|\tau\in\mathcal{T}_{\tau'}\right)=\mathcal{E}\left(\tau'\right)
\end{aligned}$$

where the inequality is because $\{\mathcal{T}_{\tau^*_i}:i\in\Omega_{\mu(\sigma),\lambda(\sigma)}\}$ is a partition of $\mathcal{T}_{\tau'}$. By the inductive hypothesis, this is bounded below by $\mathbb{E}\left(\sum_{k\in U}f_{\tau,k}r_k\right)$. The fact that τ' is then updated by $\tau^*_{i'}$ in Line 8 of Algorithm 3 proves the inductive hypothesis.

The inductive hypothesis hence holds always which means, as $\mathcal{T}_\psi=\{\psi\}$, we have $\sum_{k\in U}f_{\psi,k}r_k=\mathcal{E}\left(\psi\right)\geq\mathbb{E}\left(\sum_{k\in U}f_{\tau,k}r_k\right)$, which proves the result. □

Theorem 7. *The service placement X^ψ, computed by Algorithm 3 with slot creation computed by Algorithm 2, has a total reward of at least $(1-e^{-\delta})R$.*

Proof. The result is direct from Theorems 5 and 6 □

We now prove the correctness of Algorithm 4.

Theorem 8. *Algorithm 4 computes $\mathcal{E}\left(\tau^*\right)$ correctly.*

Proof. Algorithm 4 first computes the value:

$$\theta_k:=\mathbb{P}\left(f_{\tau,k}=1|\tau\in\mathcal{T}_{\tau^*}\right).$$

The fact that the algorithm computes the correct value of θ_k can be seen as follows:

If there exists a node $j\in W_k$ and a slot σ with $\mu(\sigma)=j$ and $\tau^*(\sigma)=\xi_k$, then by definition of $\mathcal{T}_{\tau^*}$, every service placement τ in $\mathcal{T}_{\tau^*}$ has $\tau(\sigma)=\xi_k$. Therefore, by definition of $f_{\tau,k}$, we have $f_{\tau,k}=1$ for all τ in $\mathcal{T}_{\tau^*}$. This implies that $\theta_k:=\mathbb{P}\left(f_{\tau,k}=1|\tau\in\mathcal{T}_{\tau^*}\right)=1$.

We now consider the situation where there does not exist a node $j\in W_k$ and a slot σ with $\mu(\sigma)=j$ and $\tau^*(\sigma)=\xi_k$. For all $\tau\in\mathcal{T}_{\tau^*}$, we have $f_{\tau,k}=1$ iff there exists some $j\in W_k$ and slot σ with $\tau(\sigma)=\xi_k$. Thus, since each slot is filled independently (even under the condition $\tau\in\mathcal{T}_{\tau^*}$) we have, by Lemma 1 in [22, Appendix B], that:

$$\theta_k=\mathbb{P}\left(f_{\tau,k}=1|\tau\in\mathcal{T}_{\tau^*}\right)=1-\prod_{\sigma\in\Lambda:\mu(\sigma)\in W_k}(1-\mathbb{P}\left(\tau(\sigma)=\xi_k|\tau\in\mathcal{T}_{\tau^*}\right)).$$

For any slot $\sigma\in\Lambda$ with $\tau'(\sigma)\neq\emptyset$ or $\xi_k\notin\Omega_{j,\lambda(\sigma)}$, we have $1-\mathbb{P}\left(\tau(\sigma)=\xi_k|\tau\in\mathcal{T}_{\tau^*}\right)=1$, thus we can remove it from the product above. Noting then that for all other σ we have $1-\mathbb{P}\left(\tau(\sigma)=\xi_k|\tau\in\mathcal{T}_{\tau^*}\right)=1-\omega_{i,j}/\Delta_{j,\lambda(\sigma)}$, we have shown the correctness of the computation of θ_k by Algorithm 4.

We now have:

$$\mathcal{E}\left(\tau^*\right)=\mathbb{E}\left(\sum_{k\in U}f_{\tau,k}r_k\middle|\tau\in\mathcal{T}_{\tau^*}\right)=\sum_{k\in U}r_k\mathbb{E}(f_{\tau,k}|\tau\in\mathcal{T}_{\tau^*})$$

$$= \sum_{k \in U} r_k \mathbb{P}\left(f_{\tau,k} = 1 \middle| \tau \in \mathcal{T}_{\tau^*}\right) = \sum_{k \in U} r_k \theta_k$$

which proves the correctness of Algorithm 4. □

V. Second Slot Allocation Algorithm (SA2)

The second algorithm, SA2, for solving SPSC is for the case where services can be arbitrarily large. We fix β equal to 1/4. From (5), we immediately get $\gamma = 1/2$ and $\delta = 1/4$. SA2 has an approximation ratio of $\left(1 - e^{-1}\right)/4$, which will be given by Theorem 13 later.

Definition 3. *After solving the LP in (4)we define the following for SA2, for all $j \in V$:*

$$\Omega_{j,\oplus} := \{i \in S : \frac{1}{2} c_j < s_i \leq c_j\};$$

$$\Omega_{j,\ominus} := \{i \in S : c_j \beta < s_i \leq \frac{1}{2} c_j\};$$

$$\Omega_{j,q} := \{i \in S : \gamma^q c_j \beta < s_i \leq \gamma^{q-1} c_j \beta\}, \qquad \forall q \in \mathbb{N};$$

$$\Delta_{j,q} := \sum_{i \in \Omega_{j,q}} \omega_{i,j}, \qquad \forall q \in \mathbb{N} \cup \{\oplus, \ominus\};$$

$$\delta'_j := \frac{\delta c_j}{\sum_{i \in S : s_i \leq c_j \beta} s_i \omega_{i,j}};$$

$$\eta_{j,q} := \left\lceil \delta'_j \Delta_{j,q} \right\rceil, \qquad \forall q \in \mathbb{N}.$$

We also define, for all $j \in V$, the quantity Q_j as follows:

- *If $\Delta_{j,\ominus} < 2$, then $Q_j := \Delta_{j,\ominus}$.*
- *If $\Delta_{j,\ominus} \geq 2$, then $Q_j := \Delta_{j,\ominus}/2$.*

The only difference between SA1 and SA2 is in the slots created by the respective slot creation algorithms. In SA2, a slot σ is again an object with two associated values: $\mu(\sigma)$ and $\lambda(\sigma)$, except that in SA2, we now have $\lambda(\sigma) \in \mathbb{N} \cup \{\oplus, \ominus\}$.

Procedure of SA2: First, we solve the LP in (4) to obtain $\{\omega_{i,j}\}$. Then, Algorithm 5 creates a set, Λ, of slots. After we have the set Λ, Algorithm 3 (in Section IV) computes the service placement X^ψ. In Algorithm 5, the objects ζ' and λ are maps from V into $\{1, 2, 3, \emptyset\}$. The algorithm has a function $\mathcal{D}(\cdot)$ which takes, as input, a map from V into $\{1, 2, 3, \emptyset\}$. This function $\mathcal{D}(\cdot)$ is computed in Algorithm 6.

Next, we give a description of the mechanics of the algorithm and present theoretical results on the approximation ratio and feasibility of the computed service placement X^ψ.

A. Construction Maps

A *construction map*, ζ, is a map from V to $\{1, 2, 3\}$. Intuitively, a construction map ζ is a labelling of all nodes by the label 1, 2 or 3.

A *partial construction map*, ζ', is a map from V to $\{1, 2, 3, \emptyset\}$. Intuitively a partial construction map ζ' is a partial labelling of the nodes by labels 1, 2, 3. If, for node $j \in V$, $\zeta'(j) = \emptyset$, then j has no label. Otherwise, j has label $\zeta'(j)$.

Given a partial construction map ζ', we define $\mathcal{Y}_{\zeta'}$ to be the set of all construction maps ζ such that $\zeta(j) = \zeta'(j)$ for all $j \in V$ with $\zeta'(j) \neq \emptyset$. Intuitively, $\mathcal{Y}_{\zeta'}$ is the set of all construction maps that can be obtained from ζ' by assigning labels to the unlabelled nodes.

Every construction map has an associated set of slots defined as follows, for all $j \in V$:

- If $\zeta(j) = 1$, we have a single slot σ with $\mu(\sigma) := j$ and $\lambda(\sigma) := \oplus$. There are no other slots σ' with $\mu(\sigma') = j$.
- If $\zeta(j) = 2$, we have two slots σ with $\mu(\sigma) := j$ and $\lambda(\sigma) := \ominus$. There are no other slots σ' with $\mu(\sigma') = j$.
- If $\zeta(j) = 3$, then for all $q \in \mathbb{N}$, we have $\eta_{j,q}$ slots σ with $\mu(\sigma) := j$ and $\lambda(\sigma) := q$. There are no other slots σ' with $\mu(\sigma') = j$.

Algorithm 5 Slot Creation of SA2

```
For every j ∈ V: set ζ'(j) ← ∅;
For every j ∈ V:
    For all a ∈ {1,2,3}:
        ζ*_a(j) ← a;
        For all j' ∈ V \ {j}: set ζ*_a(j') ← ζ'(j');
    a' ← argmax_{a∈{1,2,3}} D(ζ*_a);
    ζ'(j) ← a' ;
λ ← ζ';
For all j ∈ V:
    If λ(j) = 1: create a single slot σ with μ(σ) ← j, λ(σ) ← ⊕;
    If λ(j) = 2: create two slots σ with μ(σ) ← j, λ(σ) ← ⊖;
    If λ(j) = 3 then: for all q ∈ ℕ with Ω_{j,q} ≠ ∅:
        Create η_{j,q} slots σ with μ(σ) ← j and λ(σ) ← q;
Output Λ as the set of all slots created;
```

Algorithm 6 Computing $\mathcal{D}(\zeta^*)$

```
For all i ∈ S, j ∈ V:
    If s_i > c_j then: ε_{i,j,a} ← 0 for all a ∈ {1,2,3};
    If i ∈ Ω_{j,⊕} then:
        ε_{i,j,1} ← ω_{i,j}/Δ_{j,⊕};
        For all a ∈ {2,3}: set ε_{i,j,a} ← 0;
    If i ∈ Ω_{j,⊖} then:
        ε_{i,j,2} ← (1 − (1 − ω_{i,j}/Δ_{j,⊖})^2);
        For all a ∈ {1,3}: set ε_{i,j,a} ← 0;
    If ∃q ∈ ℕ with i ∈ Ω_{j,q} then:
        ε_{i,j,3} ← (1 − (1 − ω_{i,j}/Δ_{j,q})^{η_{j,q}});
        For all a ∈ {1,2}: set ε_{i,j,a} ← 0;
    ε_{i,j,∅} ← δΔ_{j,⊕} ε_{i,j,1} + δQ_j ε_{i,j,2} + (1 − δΔ_{j,⊕} − δQ_j) ε_{i,j,3};
For all k ∈ U:
    p ← 1;
    For all j ∈ W_k:
        p ← p(1 − ε_{ξ_k,j,ζ*(j)});
    θ'_k = 1 − p;
Output D(ζ*) ← Σ_{k∈U} θ'_k r_k;
```

B. Probability Distribution for SA2

For the theoretical analysis of SA2, we define a probability distribution over the set of slots Λ as well as the slot allocation. Similar to the analysis of SA1, this probability distribution will guide the construction of the set, Λ, of slots in Algorithm 5.

We first define a probability distribution over a map $\zeta : V \to \{1, 2, 3\}$ as follows: for each $j \in V$ independently, set $\zeta(j) \leftarrow 1$ with probability $\delta\Delta_{j,\oplus}$, set $\zeta(j) \leftarrow 2$ with probability δQ_j, and set $\zeta(j) \leftarrow 3$ with probability $1 - \delta\Delta_{j,\oplus} - \delta Q_j$. Once ζ has been sampled from this probability distribution, we let Λ be its associated set of slots (defined in Section V-A). For the selected Λ, we define the probability distribution over slot allocations, τ, in the same way as for SA1 in Section IV-B.

Definition 4. *Define:*

$$\mathcal{D}(\zeta') := \mathbb{E}\left(\sum_{k \in U} f_{\tau,k} r_k \middle| \zeta \in \mathcal{Y}_{\zeta'}\right)$$

where $f_{\tau,k}$ is defined as in (6). $\mathcal{D}(\cdot)$ appears in Algorithm 5 and is computed in Algorithm 6 (see Theorem 14).

The proofs of some of the theorems presented next are included in our online technical report [22].

Theorem 9. *Under our probability distribution for SA2, for any slot allocation τ of non-zero probability, its associated service placement, X^τ, is feasible.*

Theorem 10. *Under our probability distribution for SA2, the expected total reward $\mathbb{E}\left(\sum_{k\in U} f_{\tau,k} r_k\right) \geq \left(1-e^{-1}\right)\delta R$.*

C. Creating Slots

In Algorithm 5, we maintain a partial construction map ζ' where, initially, all nodes are unlabelled. Every time we run the loop in Lines 3-7 of Algorithm 5, we do the following:

1) Pick an unlabelled node j.
2) For all $a \in \Omega_{\mu(\sigma),\lambda(\sigma)}$, define ζ_a^* as the partial construction map formed from ζ' by labelling j with a.
3) Choose $a' \in \{1,2,3\}$ that maximizes $\mathcal{D}(\zeta_a^*)$.
4) Update ζ' by labelling j with a'.

We loop through the above until all nodes have been labelled. Then, let λ be the construction map we have made. Lines 9-13 of Algorithm 5 then create its associated set of slots, Λ.

Theorem 11. $\mathbb{E}\left(\sum_{k\in U} f_{\tau,k} r_k | \zeta = \lambda\right) \geq \mathbb{E}\left(\sum_{k\in U} f_{\tau,k} r_k\right)$.

Theorem 12. *The service placement X^ψ, computed by Algorithm 3 with slot creation computed by Algorithm 5, has a total reward of at least $\mathbb{E}\left(\sum_{k\in U} f_{\tau,k} r_k | \zeta = \lambda\right)$.*

Proof. As in the proof of Theorem 6, noting that the service placement is computed, from Λ, by Algorithm 3. □

Theorem 13. *The service placement X^ψ, computed by Algorithm 3 with slot creation computed by Algorithm 5, has a total reward of at least $(1-e^{-1})\delta R$.*

Proof. Directly from Theorems 10, 11, and 12. □

Theorem 14. *Algorithm 6 computes $\mathcal{D}(\zeta^*)$ correctly.*

VI. Overall Algorithm

A. Combining the Algorithms and Repeating

We now present the overall algorithm for SPSC, which has better empirical performance than SA1 and SA2 alone. First, we note that SA1 and SA2 can be combined as follows. Find the minimum β such that $\max_{i\in S} s_i \leq \beta(\min_{j\in V} c_j)$. If $\beta \geq 1$ or $\left(1-e^{-1}\right)/4 > 1-e^{-\left(1-\sqrt{\beta}\right)^2}$, run SA2; else, run SA1. We call this the combined slot allocation algorithm (CSA).

We found empirically that after running CSA, many nodes have an excessive amount of capacity remaining, i.e., $\sum_{i:j\in X_i} s_i$ is significantly smaller than c_j. This is because SA1 and SA2, thus CSA, both guarantee feasibility and may under-utilize the nodes. We now give a repeated slot allocation algorithm (RSA) which utilizes the remaining capacity by repeating CSA, as shown in Algorithm 7.

In essence, RSA runs CSA and then, with the remaining users and remaining space on the nodes, runs CSA again to place new services in addition to those placed originally. It keeps repeating this with the remaining users and remaining space on the nodes until the combined service placement does not change. Because SA1 and SA2 respectively guarantee feasibility, CSA and RSA also guarantee feasibility.

We can easily see that the approximation ratio of CSA and RSA is $\max\left\{1-e^{-\left(1-\sqrt{\beta}\right)^2}; \left(1-e^{-1}\right)/4\right\}$, because CSA adaptively chooses between SA1 and SA2 according to the one that gives the better approximation ratio, and the repeating step in RSA can only increase the reward which does not make the approximation ratio worse.

Algorithm 7 Repeated slot allocation (RSA)

```
For all i ∈ S: set Y_i ← ∅;
Set U' ← U;
For all j ∈ V: set c'_j ← c_j;
Repeat the following until Y_i does not change for all i ∈ S:
    Run CSA with user set U' and capacities {c'_j : j ∈ V};
    Let X be the output service placement from CSA;
    Set U' ← {k ∈ U' : X_{ξ_k} ∩ W_k = ∅};
    For all j ∈ V: set c'_j ← c'_j − Σ_{i∈S:j∈X_i} s_i;
    For all i ∈ S: set Y'_i ← Y'_i ∪ X_i;
Output service placement Y;
```

B. Computational Complexity

We first derive the time complexity of SA1/SA2/CSA.

The construction of the sets $\Omega_{j,q}$ (for all $q \in \mathbb{N} \cup \{\oplus, \ominus\}$) takes a total time of $\mathcal{O}(|S|\cdot|V|)$ since, for every pair $(i,j) \in S \times V$, determining q such that $i \in \Omega_{j,q}$ takes constant time.

Algorithm 4 takes a time of $\mathcal{O}(|\Lambda|\cdot|U|))$ which means Algorithm 3 takes a time of $\mathcal{O}(|\Lambda|\cdot|S|\cdot|(|\Lambda|\cdot|U|)$ which, since $|\Lambda| \in \mathcal{O}(|S|\cdot|V|)$, is equal to $\mathcal{O}(|S|^3\cdot|V|^2\cdot|U|)$.

Algorithm 6 takes a time of $\mathcal{O}(|S|\cdot|V|+|U|\cdot|V|) = \mathcal{O}(|U|\cdot|V|)$, which means Algorithm 5 takes a time of $\mathcal{O}(|U|\cdot|V|^2)$.

For the LP step, we have an input size of $\mathcal{O}(|U|\cdot|V| + |S|\cdot|V|) = \mathcal{O}(|U|\cdot|V|)$ and $\mathcal{O}(|S|\cdot|V| + |U|)$ variables. Karmarkar's algorithm [23] takes a time of $\tilde{\mathcal{O}}\left((|S|\cdot|V|+|U|)^{3.5}(|U|\cdot|V|)^2\right)$, which is equal to:

$$\tilde{\mathcal{O}}\left(\left(|S|^{3.5}\cdot|V|^{5.5}\cdot|U|\right) + \left(|V|^2\cdot|U|^{4.5}\right)\right). \tag{7}$$

Based on the above analysis, we can see that the bottleneck is the LP step. Hence the time complexity of SA1/SA2/CSA is given in (7). On each call to CSA in RSA, the number of served users increases by at least one. Hence, there are at most $|U|$ iterations. This gives the following complexity of solving SPSC with RSA:

$$\tilde{\mathcal{O}}\left(\left(|S|^{3.5}\cdot|V|^{5.5}\cdot|U|^2\right) + \left(|V|^2\cdot|U|^{5.5}\right)\right). \tag{8}$$

The conversion from GSP to SPSC multiplies the number of users by $|V|$. Hence, the overall time complexity for converting GSP to SPSC and then solving the resulting SPSC (thus also solving the original GSP) using RSA has a complexity of:

$$\tilde{\mathcal{O}}\left(\left(|S|^{3.5}\cdot|V|^{7.5}\cdot|U|^2\right) + \left(|V|^{7.5}\cdot|U|^{5.5}\right)\right). \tag{9}$$

C. Bad Example of Greedy Algorithm

We consider a greedy algorithm that greedily places services starting with the highest reward. Such an algorithm has been shown to have a constant approximation ratio for the homogeneous setting with identical service sizes [19]. In the heterogeneous setting we consider in this paper, the following example shows that it can have an arbitrarily bad approximation ratio for the GSP and SPSC problems.

Consider an example where we have $n+1$ services, $n+1$ users, and a single node with capacity $c_1 := 1$. Service $i = 1$ has size $s_1 := 1$, all the other services $i > 1$ have size $s_i := \frac{1}{n}$. Each user k requires service $\xi_k := k$. The reward of user $k = 1$ is $\hat{r}_1(1) := 2$, and the reward of all the other users $k > 1$ is $\hat{r}_k(1) := 1$. The greedy algorithm will simply place

service $i = 1$ to the node, giving a reward of 2. The optimal solution will place all the other n services with $i > 1$ on the node, giving a reward of n. This is true for all n. Hence, the approximation ratio of the greedy algorithm can be arbitrarily bad (close to zero) as $n \to \infty$ in this example.

VII. Simulation Results

We evaluate the performance of the proposed final algorithm, RSA (Algorithm 7), via simulations. Inspired by practical measurements of mobile app popularity [24] as well as related work [19], we assume that the service required by each user follows a Zipf distribution with parameter κ and $|S| = 1000$. The size of each service is $\phi \cdot (1 + Z_s/14.13)$, where Z_s is an exponentially distributed random variable with rate parameter 0.12 and ϕ is a scaling parameter. These values are estimated from the results given in [24] for the 33% largest apps and normalized by the lower bound, because we consider that only large enough services need to be offloaded to other edge nodes, and the size of edge services can be on a different scale compared to mobile apps thus we do not directly use the mobile app sizes. The size of each node is randomly chosen from $\{4, 8, 16, 32\}$, to take into account that capacities of computational devices are usually integer powers of 2.

We randomly partition all nodes into two subsets, each service can either run on the first subset of nodes, the second subset of nodes, or both, with equal probability, to represent platform dependency of services. If user k requires a service that cannot be run on node j, then we set the reward $\hat{r}_k(j)$ to zero. Otherwise, we set $\hat{r}_k(j)$ as $Z_u + Z_n$, where Z_u follows a uniform distribution within $[0.01, 1]$ to represent the importance of the service to the user, Z_n follows a uniform distribution within $[-d, d]$ to represent the difference in rewards at different nodes. We also enforce that $Z_u + Z_n \in [0.01, 1]$ and re-sample the values of Z_u and Z_n until this is satisfied. We convert the GSP problem to its equivalent SPSC problem and present results for the SPSC problem.

We compare the proposed RSA algorithm (Algorithm 7) with the following baselines: 1) the optimal solution that has exponential time complexity; 2) the greedy algorithm proposed in [19]; 3) an LP rounding mechanism that randomly rounds the solution $\{\omega_{i,j}\}$ in (4) to integers (with $\{\omega_{i,j}\}$ as probabilities) until the node capacity has reached.

By default, we choose $|U| = 1000$, $|V| = 10$, $\phi = 1$, $\kappa = 1.3$, $d = 0$. We vary one parameter while keeping the others fixed at the default value. The average results of 10 different simulation runs are shown in Fig. 2. In addition to the total reward, which is the objective we consider in this paper, we also plot the percentage of satisfied users (i.e., users that provide non-zero reward to the system) for comparison. We see that in all cases, both the total reward and the percentage of satisfied users of our proposed algorithm perform very close to the optimal, and better than the greedy and LP rounding baselines. This aligns with our theoretical result that shows our algorithm gives a good approximation ratio.

VIII. Related Work

In addition to the MEC-related work mentioned in Section I, our work is also related to virtual network embedding (VNE) [25], which studies the scheduling of distributed tasks to multiple machines. However, service components that are shareable among multiple users is usually not considered in VNE, which is an important characteristic in MEC [19].

Our problem bears some similarities with the data caching problem. However, a fundamental difference between data files and service programs is that data files can be partitioned in arbitrary ways without affecting the cache efficiency. Therefore, existing work on data caching usually considers identically sized files/contents [26], [27], which is inadequate for the placement of service programs. Mathematically, the data caching problem has been extended to consider non-partitionable files of different sizes in [28], which requires that the cost (opposite of the reward) is related to distances between nodes defined on a metric space. Such a distance metric is also a common assumption in facility location problems [20]. As discussed in [29], network delays often do not satisfy the triangle inequality, thus approaches based on metric assumptions [20], [28] are not practical. The generalized assignment assignment problem [30] does not have metric assumptions, but it does not allow the replication of a service (item) into multiple copies that are placed in multiple edge nodes (bins).

Our problem is a special case of the k-column sparse packing problem studied in [31], where a randomized approximation algorithm is proposed. Applying the approach in [31] to our case, one can get an algorithm with an approximation ratio of $(1 - e^{-1})/7.25$ (see [22, Appendix G] for details). We improve on this result in two ways. First, we have a deterministic algorithm, which is more preferred due to its predictable behavior. Second, we have an approximation ratio that is at least $(1 - e^{-1})/4$, which is better.

Our problem is also a special case of the separable assignment problem (SAP) [32], by considering users as items and nodes as bins in SAP. A subset of items (users) is feasible for a bin if the sum size of all services requested by these users is within the capacity of the node, where there is only one service instance for multiple users sharing the same service. Two approximation algorithms for SAP are proposed in [32]. The first one has an approximation ratio of $1 - e^{-1}$ and uses the ellipsoid method which has, when the number of users is much larger than the number of nodes, a higher complexity bound than our method (when solved with an efficient LP-solver) [33]. The ellipsoid method has also shown to be much slower than other LP-solvers in practice. Note that the approximation ratio of our approach also tends to $1 - e^{-1}$ as β tends to zero. The second algorithm in [32] gives an approximation ratio (upper bound) of $\frac{1}{2}$ with lower complexity than the ellipsoid method. Compared to this, our approach has a better approximation ratio when β is small enough such that $1 - e^{-\left(1-\sqrt{\beta}\right)^2} > \frac{1}{2}$, whereas we do not perform better when β is large. Nevertheless, our work is based on a completely different algorithmic technique compared to [32]. Detailed investigation of the advantages and disadvantages of both approaches is left for future work.

IX. Conclusion

We have proposed an approximation algorithm for solving the service placement problem in MEC systems with hetero-

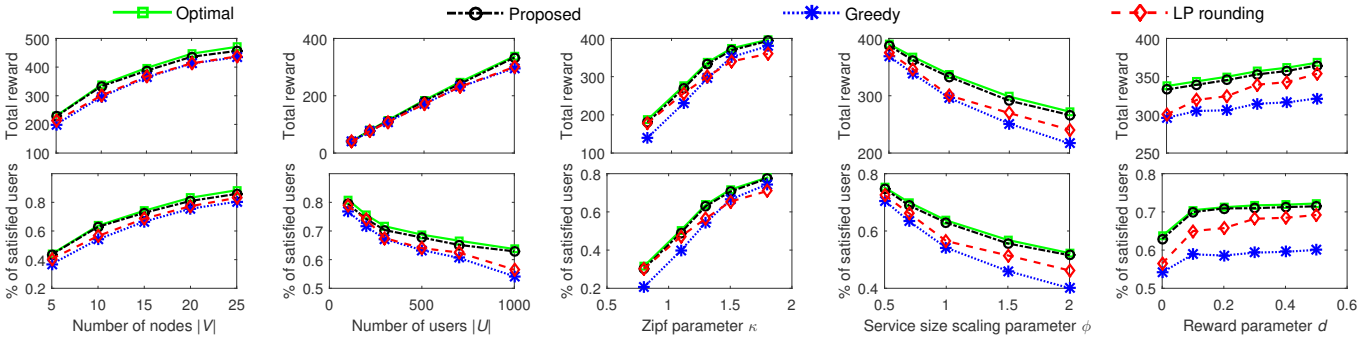

Fig. 2: Simulation results with total reward and percentage of satisfied users (in SPSC) under different parameter settings.

geneous service/node sizes and rewards. The algorithm has a constant approximation ratio, and has been shown to perform very close to optimum and better than baseline approaches in simulations. The algorithm includes a novel construction of slots on nodes. The design of the algorithm and the proof of the approximation guarantee is based on a highly non-trivial application of the method of conditional expectations.

REFERENCES

[1] P. Mach and 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. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[3] Z. Chen, W. Hu, J. Wang, S. Zhao, B. Amos, G. Wu, K. Ha, K. Elgazzar, P. Pillai, R. Klatzky *et al.*, "An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 2017, p. 14.

[4] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *IEEE INFOCOM*, April 2016, pp. 1–9.

[5] A. Ceselli, M. Premoli, and S. Secci, "Mobile edge cloud network design optimization," *IEEE/ACM Transactions on Networking*, vol. 25, no. 3, pp. 1818–1831, June 2017.

[6] 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, no. 10, pp. 2866–2880, Oct. 2016.

[7] 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, no. 4, pp. 725–737, Oct. 2017.

[8] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.

[9] L. Tong and W. Gao, "Application-aware traffic scheduling for workload offloading in mobile clouds," in *IEEE INFOCOM*, April 2016, pp. 1–9.

[10] Y. Xiao and M. Krunz, "Qoe and power efficiency tradeoff for fog computing networks with fog node cooperation," in *IEEE INFOCOM*, May 2017, pp. 1–9.

[11] T. Zhao, I. H. Hou, S. Wang, and K. Chan, "Red/led: An asymptotically optimal and scalable online algorithm for service caching at the edge," *IEEE Journal on Selected Areas in Communications*, 2018, in press.

[12] H. Tan, Z. Han, X.-Y. Li, and F. Lau, "Online job dispatching and scheduling in edge-clouds," in *IEEE INFOCOM*, May 2017.

[13] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling in edge-clouds," *Performance Evaluation*, vol. 91, pp. 205–228, Sept. 2015.

[14] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *IEEE INFOCOM*, 2018.

[15] L. Wang, L. Jiao, J. Li, and M. Muhlhauser, "Online resource allocation for arbitrary user mobility in distributed edge clouds," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 1281–1290.

[16] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1002–1016, Apr. 2017.

[17] S. Wang, M. Zafer, and K. K. Leung, "Online placement of multi-component applications in edge computing environments," *IEEE Access*, vol. 5, pp. 2514–2533, 2017.

[18] L. Wang, L. Jiao, T. He, J. Li, and M. Mühlhäuser, "Service entity placement for social virtual reality applications in edge computing," in *IEEE INFOCOM*, 2018.

[19] T. He, H. Khamfroush, S. Wang, T. L. Porta, and S. Stein, "It's hard to share: Joint service placement and request scheduling in edge clouds with sharable and non-sharable resources," in *IEEE ICDCS*, Jul. 2018, pp. 365–375.

[20] V. V. Vazirani, *Approximation algorithms*. Springer Science & Business Media, 2013.

[21] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[22] S. Pasteris, S. Wang, M. Herbster, and T. He, "Service placement with provable guarantees in heterogeneous edge computing systems," Tech. Rep., 2019, to be uploaded to arXiv.

[23] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Combinatorica*, vol. 4, no. 4, pp. 373–395, Dec 1984.

[24] W. Liu, G. Zhang, J. Chen, Y. Zou, and W. Ding, "A measurement-based study on application popularity in android and ios app stores," in *Workshop on Mobile Big Data*, 2015, pp. 13–18.

[25] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[26] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femtocaching: Wireless content delivery through distributed caching helpers," *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 8402–8413, Dec. 2013.

[27] S. Shukla, O. Bhardwaj, A. Abouzeid, T. Salonidis, and T. He, "Proactive retention-aware caching with multi-path routing for wireless edge networks," *IEEE Journal on Selected Areas in Communications*, 2018, in press.

[28] I. Baev, R. Rajaraman, and C. Swamy, "Approximation algorithms for data placement problems," *SIAM Journal on Computing*, vol. 38, no. 4, pp. 1411–1429, 2008.

[29] G. Wang, B. Zhang, and T. Ng, "Towards network triangle inequality violation aware distributed systems," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. ACM, 2007, pp. 175–188.

[30] D. B. Shmoys and É. Tardos, "An approximation algorithm for the generalized assignment problem," *Mathematical programming*, vol. 62, no. 1-3, pp. 461–474, 1993.

[31] N. Bansal, N. Korula, V. Nagarajan, and A. Srinivasan, "Solving packing integer programs via randomized rounding with alterations," *Theory of Computing*, vol. 8, no. 1, pp. 533–565, 2012.

[32] L. Fleischer, M. X. Goemans, V. S. Mirrokni, and M. Sviridenko, "Tight approximation algorithms for maximum separable assignment problems," *Mathematics of Operations Research*, vol. 36, no. 3, pp. 416–431, 2011.

[33] C. C. Gonzaga, "On the complexity of linear programming," *Resenhas IME-USP*, vol. 2, no. 2, pp. 197–207, 1995.