Latency Focused Function Placement and Request Routing in Multi Access Mobile Edge Computing Networks

Mustafa F. Özkoç, Chen Li [*]

Abstract

In this project, we study the problem of network function placement and request routing under node and link constraints. With the deployment of 5G, more and more networks will benefit from mobile edge computing. Networks can provide variety of low latency services by processing the demands at the edge. However, the limited computation power and memory of computing devices at the edge requires an optimal placement of the services. An edge computing node can not provide the resources required for all the services and has to forward or possibly neglect some requests from users. The limited node resources and the ability to forward the requests to neighboring nodes naturally raises the question of where to serve a request and how to route the request, a joint function placement and request routing optimization problem. We formulate the problem as mixed integer programming optimization which is known to be an *NP*-hard problem. We solve the optimization problem using the CPLEX solver. Then, we relax the variables to solve the optimization as a Linear Programming. Moreover, we develop two different heuristic algorithms, one being bottom up approach and the other being top to bottom approach where we trade achieved optimal value with execution time. Finally, we compare the performance and execution times of these 4 different approaches.

1 Summary for the course

1.1 Problem Definition

- What is the problem : Joint service placement, computation and routing in multi-access edge computing networks. An edge computing node have limited resources, where it needs to allocate its resources to many different services requested by large number of users. Most of these services do not have fractional nature hence it results in an integer programming problem. This problem can be seen as an example of the well known knapsack problem, where the capacity of the bag is an analogy for the computational resources of the node. However, in this example the nodes are also limited by their storage capacity which further complicates the problem. Moreover, a user can be served in any of the nodes as long as the target node is close enough to satisfy the latency requirements of the user demand. Allowing a demand to be served in many nodes, also requires an efficient routing algorithm between access node and the server node.
- How it is related to course : We formulate a network design optimization problem where we try to maximize the profit made by the mobile network operator. Moreover, we explicitly state the equality and inequality constraints of the problem. We use node-link based formulation to route the requests between access and server nodes. Our problem has contents related but not limited to, link dimensioning, routing, topological design. We use solving methods such as branch and bound, and develop heuristic methods to optimize the problem. Beyond the class, we not only focus on the achieved optimal value but we also take into account the time required to apply these solution methods and provide insights about the design principles.

[*]ozkoc@nyu.edu, chen.lee@nyu.edu . Individual Contributions : Mostly the problem formulation is done by M. F. Ozkoc. Heuristic 1 is developed by M.F. Ozkoc and Heuristic 2 is developed and proved by Chen Li. The scripts to run Heuristic 2 on NYU HPC is written by Chen Li, as well. The linear programming, greedy scripts for AMPL are written by two students separately and students converged on one script by discussing each step. Finally all the simulations for 100 users are run on NYU HPC by M.F. Ozkoc. Simulations for 500 Users are run on locally by Chen Li. Unless specifically written here students have equal contributions.

1.2 Design Models

- Models covered in class : Node link formulation, branch and bound method.
- Other models from textbook : None
- Variations or new models proposed by us : Non-fractional demand constraints. Flexible selection of origin and destination for each service request with constraints enforced by limited node resources. Routing model with latency limitations imposed by the QoS requirements.

1.3 Solution Techniques

- Optimization software : We used AMPL softwares, Specifically we used CPLEX solver for MIP and LP solutions. We used MATLAB for heuristic 1. We also used AMPL/CPLEX for heuristic 2 for iteration steps, and used Python v3.6 for the overall algorithm steps.
- Customized Heuristics: The solution methods we call as heuristic 1 and heuristic 2, where we use bottom-up fashion building and adding users, or top to bottom approach and removing users, respectively.

1.4 Performance Evaluation

- Instances of problem : We used MATLAB to generate 1000 randomly generated scenarios to run Monte Carlo simulation. We change the node capacities, demands, demand values, user locations and association etc. Details are given in the following sections.
- Performance of our algorithm :
  - For Heuristic 1 : bottom up approach, we have incredibly fast solution using MATLAB. But the correctness of the solution is questionable with performance deficiency up to 50%.
  - For Heuristic 2 : Top to bottom approach, we have solutions that achieve close to optimal objective function obtained by LP and MIP solutions with a shorter running time.
- Comparison Study : We compared LP, MIP as baseline with heuristic 1 and heuristic 2 algorithms developed by us in terms of performance and execution time.

2 Introduction

The massive bandwidth enabled by 5G, creates new opportunities for revenue generating verticals. These verticals include use cases such as autonomous driving, robotic control, immersive AR/VR applications [1]. Use cases like robotic control and autonomous driving require ultra low end to end latency as low as tens of milliseconds [2]. This stringent quality of service (QoS) requirement can be satisfied with Multi-Access Edge Computing (MEC) [3, 4] where application servers are located in network function virtualization (NFV) [5] enabled edge nodes of the network. The flows served at the edge, do not travel long distances deep into network and as a result experience much lower latency.

Edge computing devices may not have large enough storage or computation capability to store and serve all the services offered by the mobile network operator (MNO) [3]. Moreover, an edge node can provide limited number of connection, which limits the number of users that can access the network through that node. A MNO needs to decide which functions to place on which nodes according to user demands at a certain time. This issue is further complicated with the limit of how many users can be served in a node. Moreover, a user can be served at the access node or can be routed to another close-by node as long as the latency requirement of the service demanded by user is satisfied which leads to a complicated joint service placement and request routing problem.

Most of the proposed services will be especially beneficial for mobile user equipment which can be a drone, an autonomous vehicle or an AR/VR headset. Hence, our main focus is on millimeter wave enabled 5G new radio (5G-NR) where the links between users and access nodes can be easily blocked [6, 7]. The sporadic nature of the access links requires multiple paths as well as a resilient design of the routing and function placement. Moreover, the hard to satisfy QoS requirements of these services in terms of reliability combined with intermittent nature of mmWave frequencies, motivates multi-connectivity [8] and dense deployment of access points (AP)s. Hence, a user can be in the coverage range of multiple APs, and can be connected to the network over any of these APs as long as the target AP is not limited by bandwidth, or by maximum user connection limit. Given bandwidth and connection availability, an AP can prefer to serve the user request if it has the memory and computational resources for the requested service, or can prefer to route the demand to another edge node or a centralized cloud as long as the latency requirement of the service is satisfied.

2.1 Related Work

The resource limited nature of the edge computing devices arises the question of deciding which users to serve at the edge and which users to route to the centralized cloud or even decline in some scenarios. Most of the attention is given to the computational and communication limitations of this problem. Function placement problem in MEC networks are studied extensively and a large number of solutions are developed for this constrained optimization problem. Most of the existing work is focused on optimally allocating the computation, memory resources on the nodes and the available bandwidth on the links. The mixed integer programming (MIP) nature of the problem requires an efficient algorithm to approximately solve the problem. An optimal algorithm in the asymptotic regime is given in [9] where they focus on static resource allocation of the network. Another approach is to dynamically allocate the resources. Many metric are used to define the optimality of the problem. Cost in terms of energy as an optimization function is given in [10]. The resources such as computation and bandwidth are not shareable. However, the storage resource can be shared amongst different applications to store common information such as image dataset or a common query database. Service placement with shareable resources problem is examined in [11]. The authors prove the problem is *NP*-hard even after removing the computation and communication constraints.

The current traffic is switching from stationary devices to handheld devices. A user connected to 5G network, can/should maintain connection to reduce the out of service instances caused by blockages. This limitation requires a dense deployment of APs and a user can be in the coverage of more than one AP. To that end the request routing aspect of the problem should take into account that an user can switch between nodes and connect the network through any of its available APs. To the best of our knwoledge, we are the first to study the problem with multiple access points except [9]. The authors in [9] allow a user to be served in multiple APs however, they do not allow routing between edges nodes which we believe is crucial for an efficient use of resources. They consider two options for a user, either to be served at the AP that the user connects or to be routed to the centralized cloud. However, a user can be served in another edge node where the resources are available while using a different node as the main AP.

3 System Model

We consider a MEC network consisting of a set $\mathcal{N}$ of N nodes (base stations) equipped with storage and computation capabilities, a set of $\mathcal{U}$ of U mobile users, and a set of $\mathcal{E}$ of E links. An example of the network is illustrated in Fig. 1.

The centralized cloud can be represented as another node l where combined with other nodes form the set $\mathcal{N} \cup \{l\}$. Each user can access to the network through a set of access nodes $\mathcal{N}_u$. If the request of user u determined to be served in the network then, one of the connections between user u and nodes $\mathcal{N}_u$ should be active. The MEC network receives service requests from its subscribers in a stochastic manner. Without the loss of generality, we assume each user request one service $s_u \in \mathcal{S}$ where the set $\mathcal{S}$ contains the library of services offered in this particular MEC network. If a user performs multiple requests, we can split it into multiple users. Each of these services can require different computation power, storage space on the computation nodes. Moreover a service can require to transfer data between the user and serving computing node hence consumes bandwidth. Generally the services offered by 5G-NR, such as autonomous driving, AR/VR and tactile internet, can tolerate up to a certain latency. These requirements of a service s on computation power, storage space (memory), bandwidth (traffic volume), latency are given as $\bar{c}_s$, $\bar{m}_s$, $\bar{h}_s$, and $\bar{t}_s$, respectively. A mobile network operator can price this various services as w_s, which can model how much a user can be charged for service s. Notice that this price is independent of user, however, without increasing the problem complexity, an MNO can selectively price its users.

The request of user u can be routed to any node on the edge network. The memory and computation resources required by the service will be reserved on the node. The request can, also, be served at the centralized cloud where the memory and computational constrains are no longer a system limitation. However, accessing the centralized cloud may cause high end to end latency due to its long distance. The latency requirement of the service should be satisfied when the request is decided to be served at the cloud. Notice that the service will still use the available bandwidths on the network.

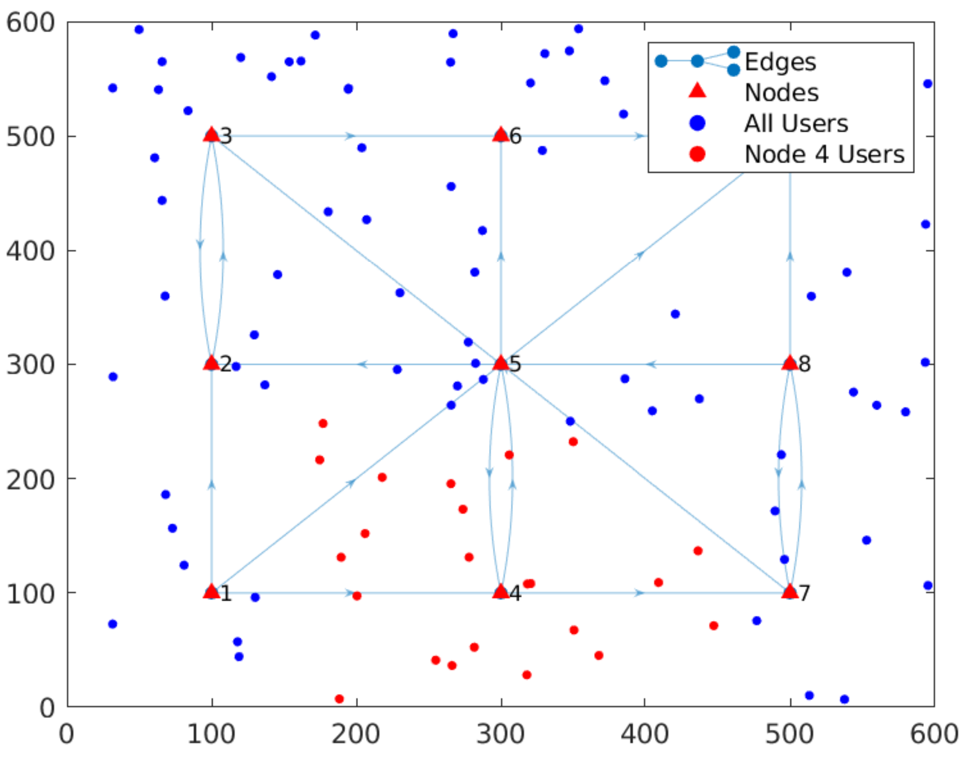

Figure 1: An example scenario. User positions are generated randomly, the graph consisting of nodes and edges is fixed over all the simulations. Blue and red points together represents all the users, where the latter is the users covered by access point 4 are given as an example coverage region.

The network operator needs to decide whether to serve a user at the edge network, at the centralized cloud or ignore the service request if it can not satisfy the requirements of the service. If the network operator decides to serve the user, it needs to allocate the service on a node, and also use a path through one of the access node of that user. To model these decisions we introduce optimization variables

- x_{nu} : binary variable. 1 if the request of user u is served at node n, 0 otherwise.
- β_u : binary variable ($\beta_u = \sum_{n \in \mathcal{N} \cup \{l\}} x_{nu}$). 1 if the request of user u is served, 0 if the request from the user is dropped.
- $y_{n'u}$: binary variable. 1 if the user u access the network through node n' where $n' \in \mathcal{N}_u$, 0 otherwise.
- u_{ue} : binary variable. 1 if the link e is used while serving the user u, 0 otherwise.
- y_e: traffic on link e.
- ξ_e: unit cost on link e.

3.1 Formulation

The indices $u = \{1, 2, \ldots, U\}$, $n = \{1, 2, \ldots, N, l\}$, $e = \{1, 2, \ldots, E\}$ are used to represent users, nodes and edges in our network. Constants c_e, l_e represents the capacity and delay of link e, respectively. Let's define $a_{en}(b_{en})$ as another constant where it is equal to 1 if the link e originates(terminates) at node n, 0 otherwise. Recall that earlier, we introduced the decision variables x_{nu}, $y_{n'u}$, u_{ue}.

The objective of MNO is to generate the largest possible revenue from a centralized cloud server and a MEC network topology pair. The total revenue of the MNO can be expressed as

$$F = \sum_{u \in \mathcal{U}} w_{s_u} \sum_{n \in \mathcal{N}} x_{nu} - \sum_{u \in \mathcal{U}} \sum_{e \in \mathcal{E}} u_{ue} \xi_e y_e \tag{1}$$

where w_{s_u} is the revenue of serving the service requested by user u. The first term is profits and the second term is link cost. Node cost is ignore because we believe forward cost is low and computational cost can be absorbed in w_{s_u}. The service placement and request routing problem need to satisfy several constraints.

A user can be served at most one location or neglected. :

$$\beta_u \leq 1, \forall u \in \mathcal{U} \tag{2}$$

If a user is to be served the user needs to access the network through any of its access nodes, otherwise it shouldn't access to the network and waste resources at access nodes:

$$\sum_{n' \in \mathcal{N}_u} y_{n'u} = \beta_u, \forall u \in \mathcal{U} \tag{3}$$

The flow generated by user u should be allocated to links if and only if that user is to be served:

$$\sum_{e \in \mathcal{E}} u_{ue} \leq E\beta_u, \forall u \in \mathcal{U} \tag{4}$$

Each link has a capacity bound. The total flow realized on a link should be less than this upper bound:

$$\sum_{u \in \mathcal{U}} u_{ue}\bar{h}_{s_u} \leq c_e, \forall e \in \mathcal{E} \tag{5}$$

Each node has a memory bound. The total served memory on a node should be less than its upper bound:

$$\sum_{u \in \mathcal{U}} x_{nu}\bar{m}_{s_u} \leq M_n, \forall n \in \mathcal{N} \tag{6}$$

Each node has a computational power bound. The total served computational power on a node should be less than its upper bound :

$$\sum_{u \in \mathcal{U}} x_{nu}\bar{c}_{s_u} \leq C_n, \forall n \in \mathcal{N} \tag{7}$$

Each node has a maximum number of sites that can be handled, denotes K_n:

$$\sum_{u \in \mathcal{U}} y_{nu} \leq K_n, \forall n \in \mathcal{N} \tag{8}$$

The total delay experienced over the network by a service flow should be less than end to end latency requirement of the service:

$$\sum_{e \in \mathcal{E}} u_{ue} l_e \leq \bar{t}_{s_u}, \forall u \in \mathcal{U} \tag{9}$$

Notice that if the service will not be served this constraint is trivially satisfied by the constraint (4). For any node in the network the flow conservation should be satisfied, i.e the summation of total incoming flows and flows generated at that node should be the summation of total outgoing flows and flows destined to that node and this constraint will guarantee only one flow for one user/service :

$$\sum_{e \in \mathcal{E}} a_{en}u_{ue} - \sum_{e \in \mathcal{E}} b_{en}u_{ue} = (y_{vu} - x_{vu}), \forall n \in \mathcal{N} \cup \{l\} \tag{10}$$

4 Experiment Setup

We conduct experiments to compare the performance and execution times of four different approaches. As a baseline we relax the problem to a linear problem by allowing fractional function placement and routing. Second, we solve the exact problem by using branch and bound methods. Third, we develop a bottom up approach heuristic solution where we start from the most profitable user and place the service requested by that user, move on to the next most profitable user until we run out of resources. Fourth, we iteratively solve the mixed integer programming by using semi-relaxed integer programming where we allow fractional service placement but strictly require integer routing. We use the output of this file, according to fractional solution we set some of the variables as one or zero and add this as an additional constraint to formulation. The details of the heuristic algorithms are given in their respective subsections.

4.1 Linear Programming

Linear programming (LP, also called linear optimization) is a method to achieve the best outcome in a mathematical model whose requirements are represented by linear relationships. Linear programming is a special case of mathematical programming (also known as mathematical optimization).

More formally, linear programming is a technique for the optimization of a linear objective function, subject to linear equality and linear inequality constraints. Its feasible region is a convex polytope, which is a set defined as the intersection of finitely many half spaces, each of which is defined by a linear inequality. Its objective function is a real-valued affine (linear) function defined on this polyhedron. A linear programming algorithm finds a point in the polytope where this function has the smallest (or largest) value if such a point exists.

A covering LP is a linear program of the form:

Minimize:

$$b^T y$$

subject to:

$$A^T y \geq c$$

$$y \geq 0$$

where the matrix A and the vectors b and c are non-negative.

The dual of a covering LP is a packing LP, a linear program of the form:

Maximize:

$$c^T x$$

subject to:

$$Ax \geq b$$

$$x \geq 0$$

where the matrix A and the vectors b and c are non-negative.

In the cplex solver, it uses simplex iteration and dual simplex iteration.

- Simplex algorithm of Dantzig The simplex algorithm, developed by George Dantzig in 1947, solves LP problems by constructing a feasible solution at a vertex of the polytope and then walking along a path on the edges of the polytope to vertices with non-decreasing values of the objective function until an optimum is reached for sure. In many practical problems, "stalling" occurs: many pivots are made with no increase in the objective function[12][13].

In practice, the simplex algorithm is quite efficient and can be guaranteed to find the global optimum.

4.2 Mixed Integer Programming

An integer programming problem is a mathematical optimization or feasibility program in which some or all of the variables are restricted to be integers. In many settings the term refers to integer linear programming (ILP), in which the objective function and the constraints (other than the integer constraints) are linear.

Integer programming is NP-complete. In particular, the special case of 0-1 integer linear programming, in which unknowns are binary, and only the restrictions must be satisfied, is one of Karp's 21 NP-complete problems.

If some decision variables are not discrete the problem is known as a mixed-integer programming problem. Mixed-integer linear programming (MILP) involves problems in which only some of the variables are constrained to be integers, while other variables are allowed to be non-integers.

Mixed Integer Linear Programming problems are generally solved using a linear-programming based branch-and-bound algorithm.

4.2.1 Branch and Bound

Basic LP-based branch-and-bound can be described as follows. We begin with the original MIP. Not knowing how to solve this problem directly, we remove all of the integrality restrictions. The resulting LP is called the linear-programming relaxation of the original MIP. We can then solve this LP. If the result happens to satisfy all of the integrality restrictions, even though these were not explicitly imposed, then we have been quite lucky. This solution is an optimal solution of the original MIP, and we can stop. If not, as is usually the case, then the normal procedure is to pick some variable that is restricted

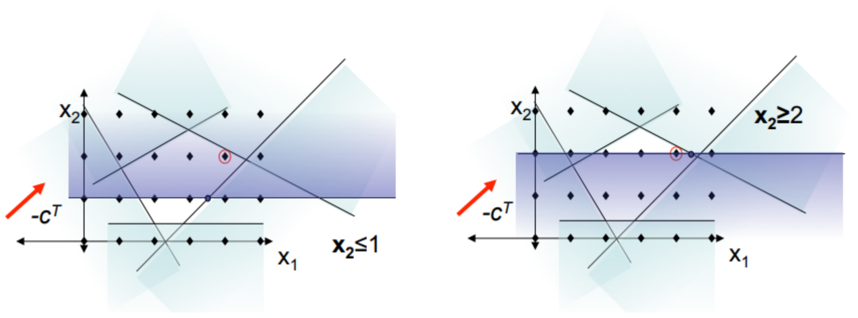

Figure 2: Two sub-branches. New constraint leads to two sub-branches

to be integer, but whose value in the LP relaxation is fractional. Create two sub-branches by adding constraints. Branch and Bound Algorithm shows as follows:

1. Solve LP relaxation for lower bound on cost for current branch
   - If solution exceeds upper bound, branch is terminated
   - If solution is lower than current upper bound, and solution is integer, replace current upper bound with solution cost, branch is terminated.
2. Create two branched problems by adding constraints to original problem
   - Select integer variable with fractional LP solution
   - Add integer constraints to the original LP
3. Repeat until no branches remain, return optimal solution.

4.3 Heuristic 1 : Bottom up approach

We develop a simple algorithm where the decision is made per user basis starting from the most profitable user and moving down to the least profitable user. First we check if the access nodes for a user have the resources to satisfy that user's request. If there are enough resources, the user is served in one of these nodes without any routing. If the resources at the access nodes are not enough to serve the request, then the algorithm checks if any other node has enough resources to that user's request. Then the algorithm computes the routes between any of the access nodes to any of the node with enough resources. In order to limit the latency and the usage of the links we select the shortest path in terms of hops. Then, we check if the latency requirement of the requested service can be satisfied with that path. If the latency requirement is satisfied as well, we route the demand on that path and move onto the next user.

We also change the ordering parameter as a variation to this algorithm. We order the users according with respect to user_profit per memory requirement and user_profit per computation requirement in two separate examples. Since this algorithm quite fast and scales well one can try all three options to see which one performs better. In a system where the memory limitation is more stringent, the ordering with respect to user_profit per memory requirement would perform significantly better.

4.4 Heuristic 2: Top to bottom approach

We proposed a top to bottom approach to solve large-scale MIP problem. By reducing the number of integer variables, we can solve large-scale MIP problem iteratively. We know that more integer variables, more time the algorithm will take. We can make iterative procedures to optimize the objective function with less integer variables. In our model, we have binary variable: $access\_indicator(y_{n'u})$,$link\_indicator(u_{ue})$ and $server\_indicator(x_{nu})$. But when we fix $link\_indicator$ and $access_indicator$ to binary variables, the $server\_indicator$ will be binary automatically.

So we fix $link\_indicator$ to be binary, and get 'binary' $access\_indicator$ iteratively. During the iteration, we add more constraints to the model according to the output of each iteration to gradually get the mixed integer solution.

Algorithm 2 shows the details for this approach. First, we fix $access\_indicator = 1$ as soon as they become 1 in the iterative procedure. Then for $access\_indicator$, we want to try to make higher value

Algorithm 1 Heuristic 1

```
1: Load Scenario : service, node, link, user parameters.
2: user_profit $\leftarrow$ service_value(user_service)
3: sort(user_profit,descending)
4: for user $\in$ user_profit do
5:     target_nodes $\leftarrow$ access_nodes with enough resources
6:     if target_nodes $\neq \emptyset$ then
7:         Assign the demand to a random target_node
8:         profit $\leftarrow$ profit + user_profit
9:     else
10:        candidate_nodes $\leftarrow$ any of the remaining nodes with enough resources
11:        source_nodes $\leftarrow$ access_nodes
12:        Find paths between all source nodes and candidate nodes
13:        Start from shortest path,
14:        if path_latency $\leq$ service_delay_tolerance then
15:            Assign flow to that path
16:            Place the function to corresponding candidate node.
17:        else
18:            try the next path
19:        end if
20:    end if
21: end for
```

to 1. (Like make 0.95 to 1). In other words, we try to increase these majority part of flow to one by reducing the minority to zero, which will guarantee the feasible solution. So, in each iteration, we add more constraints which fix one *access_indicator* to 0 (Line16-17), but the position is according to the maximum in *access_indicator*(For the same user). We will drop off this user if the summation over all flows of the users is less than 1(Line12-14). Because it means the network can not server this user in this budget. At the same time, the *successive_drop* will increases and it decides when we terminate the loop. We find if we have two successive drop off, then almost all of the users in the future iteration will be drop off too. So setting it to 2 makes sense.

Let's assume that each user choose one service and each service has a different service value. In our model, each flow only has one routing path and one user may have multiple flow. For each user, there will be several available paths to route, and let's note the shortest path(SP) has highest priority, the second SP has 2^{nd} highest priority and so on. To get the global optimal solution, each user tends to choose their SP.

By doing this policy, we have two great properties:

- A. We just drop off user x if and only if it has the lowest service value on the access node of the user's majority flow.[1]
- B. For these user who is not drop off. If the shortest path can not accommodate this user, it will choose the second shortest path. If the second shortest path can not do either, then the third . . . [2]

4.4.1 Theoretical Proof on Properties

There are several properties on our network(**Network Property**):

Let's assume that each user choose one service and each service has a different service value. In our model, each flow only has one routing path and one user may have multiple flow. For each user, there will be several available paths to route, and let's note the shortest path(SP) has highest priority, the second SP has 2^{nd} highest priority and so on. To get the global optimal solution, each user tends to choose their SP.

[1]If x has k splits. The flow volume is sorted like $x_1 \geq x_2 \geq ... \geq x_{k-1} \geq x_k$. Majority split is x_1. If flow x_1 access $node_1$, and there are other users y,z,t access the same node, $node_1$. x has the lowest service value means that x.value should be min{x.value,y.value,z.value,t.value,...}

[2]In other words, if we do not drop off this user x, x will choose the highest priority SP who can serve it alone

Algorithm 2 Heuristic 2

```
1: $C_1$={ } constraints to fix variable to 1
2: $C_0$={ } constraints to fix variable to 0
3: terminate_threshold = 2
4: successive_drop = 0
5: solve sub-mip.run
6: $C_1 = C_1 \cup \{(n_0,u_0)\}$ if access_indicator[$n_0$,$u_0$]=1
7: while successive_drop < terminate_threshold do
8:   solve sub_mip.run
9:   $C_1 = C_1 \cup \{(n_i,u_i)\}$ if access_indicator[$n_i$,$u_i$]=1
10:  successive_drop = 0
11:  findmax_exceptOne{access_indicator}
12:  if $\sum_{n \in N}$access_indicator[$n$,$u_{max}$] < 1 then
13:    $C_0 = C_0 \cup \{(u_{max})\}$
14:    successive_drop++
15:  else
16:    min_pos = min_pos corresponding to $u_{max}$
17:    $C_0 = C_0 \cup \{(\text{min\_pos})\}$
18:    successive_drop = 0
19:  end if
20: end while
21: $C_0 = C_0 \cup$ {all of the pos whose value $\in$ (0,1)}
```

1. Each node can serve/access at least one user.
2. Each node can serve all services, so service type does not matter for node. The value of each service matters.
3. Split happens when the available shortest path(SP) can not accommodate/serve one user alone.
4. At most one of the access nodes which splits access is not saturated. The path associated with this node should be the lowest priority SP among the these splits, but its priority is higher than any other path without flow because users tend to select high priority path. (If two nodes are not saturated, the lower level SP can move some flow to the higher level SP, and then objective value will increase and it is not optimal.)
5. If user x with split flow access a saturated nodes, then x should has the lowest value among the users who access this nodes. (If y's value is lower than x's, then we replace some y flow with x flow, we can have a larger objective value.)

Let's prove property A and B at the same time:

case 1:

If $\sum_{n \in N}$access_indicator[n,u_{max}] < 1, then all of the access nodes are saturated. With **network property 5**, we can say the majority of flow has the lowest value on associated access node. It will be dropped off, which is **Property A**

case 2:

If $\sum_{n \in N}$access_indicator[n,u_{max}] $= 1$, without loss of generality, let's assume user x has k splits. The flow volume is sorted like $x_1 \geq x_2 \geq ... \geq x_{k-1} \geq x_k$. According to **network property 4**, at most one of these splits is not saturated. Let's see what will happen if there is one or zero splits is not saturated:

case 2.1:

If all of the splits are saturated, after fix $x_k = 0$, then it will go back to **case 1**.

case 2.2:

If one of these splits is not saturated, then **we have two cases**:

case 3.1:

If x_k is not saturated, after we fix $x_k = 0$, if where are other paths $k+1, k+2...$ for user x, the flow x_k will shift to path $k+1, k+2, ...$ If it can shift all of the flow to these paths , then $x_{k+1} + x_{k+2} + ... = x_{k_{old}}$, $x_{k_{new}} = 0$. It will go back to **case 2.2**. If it can only shift part of flow to these paths, then $x_{k+1} + x_{k+2} + ... < x_{k_{old}}$, $x_{k_{new}} = 0$, all of the paths are saturated. It will go back to **case 1**.

case 3.2:

If x_i is not saturated, where $i \leq k-1$. According to **network property 4**, the i^{th} path has the lowest priority among $1, 2, .., k$, but highest priority among $i, k+1, k+2, ..$ if any. If we fix x_k to 0, flow x_k will shift to $i, k+1, k+2, ...$ But first, flow x_k will shift to x_i. **we have two cases**:

case 3.2.1:

If x_i path is saturated after that, then the rest of flow will be shift to $k+1, k+2...$ according to the priority of these path if with enough capacity, and then it will go back to **case 2.2**. If there is not enough capacity on $k+1, k+2, ...$, it will go back to **case 1**.

case 3.2.2:

If x_i path has enough capacity to absorb x_k, $x_{i_{new}} = x_{i_{old}} + x_{k_{old}}$, $x_{k_{new}} = 0$, then it will go back to **case 2.2** until k=2. When k=2, it means x_i(or x_1) will be final path for user x. According to the **network property 4**, we can find that x_i path has the highest priority among all paths who can serve user x alone, which is **Property B**. Because any path with higher priority can not serve user x alone and some other paths maybe serve user x alone but have lower priority.

5 Numerical Results

Table 1: SIMULATION SETUP

Parameters	Value(uniformly selected)
Total service types,S	100
#Users,U	100
#Nodes,N	9
#Links,E	18
Profit value	[0,1]
Service memory requirements	[100 500] GB
Service computational requirements	[0.5 2.5] GHz
Service volume of the traffic	[5 25] Mbps
Service latency requirement	[100 500]ms
Node computation capacity	[1, 10] GHz
Node memory capacity	[250, 1250] GB
Connection number limited	[30 100]
Link delay	[4 10] ms
Link capaacity	[125 250]Mbps

We run Monte Carlo simulations over 1000 randomly generated parameter values of, node, link capacities, user-service matching, service requirements, link latency. Node computation and memory capacity are uniformly selected in ranges [1, 10] GHz and [250, 1250] GB, respectively. The connection number limited to an integer selected randomly in range [30 100]. The link delay, capacity are uniformly selected in range [4 10] ms and in range [125 250] Mbps respectively. The node positions, and links between nodes are fixed over all the iterations where it represents a already existing network over which we build the service network. The fixed network structure is illustrated in Fig. 1.

The users are deployed in 600m by 600m area where each user, without loss of generality, demands a service from a library of services with size 100. Each service has a profit value, memory requirement, computational requirement, bandwidth requirement, and latency requirement. The profit value is uniformly selected in range [0 1]. The memory and computational requirements are uniform random variables in range [100 500] GB and [0.5 2.5] GHz, respectively. The volume of the traffic is another uniform random variable in range [5 25] Mbps. Finally the latency requirement or in other words the delay tolerance of the services are uniform random variables in range [100 500] ms.

We allocate 1 hour time window to solve each iteration on NYU-HPC. If cplex solver fails to find a mixed integer solution until the end of the session, we neglect that scenario. Moreover, we allocate 3 GB of ram to each solver, if during the solution the memory overflows, we neglect that scenario. For every script that involves MIP, we call cplex solver with option MIPgap=1e-2 threads=1.

In some of the scenarios, we observe a MIP solution achieving better objective function than LP solution, which indicates something is not correct. We run those scenarios in our local computer with the same script, and we got different results. However, when we run in our local computer some other scenarios had the same issue, where the MIP solution performs better than LP solution. Our guess is that discrepancy caused by cplex solver going deep into an infeasible branch but running out of memory or time and outputting some incorrect result. We will further investigate this issue. In order to have theoretically sane data, we exclude the scenarios where we have better MIP solution than LP solution.

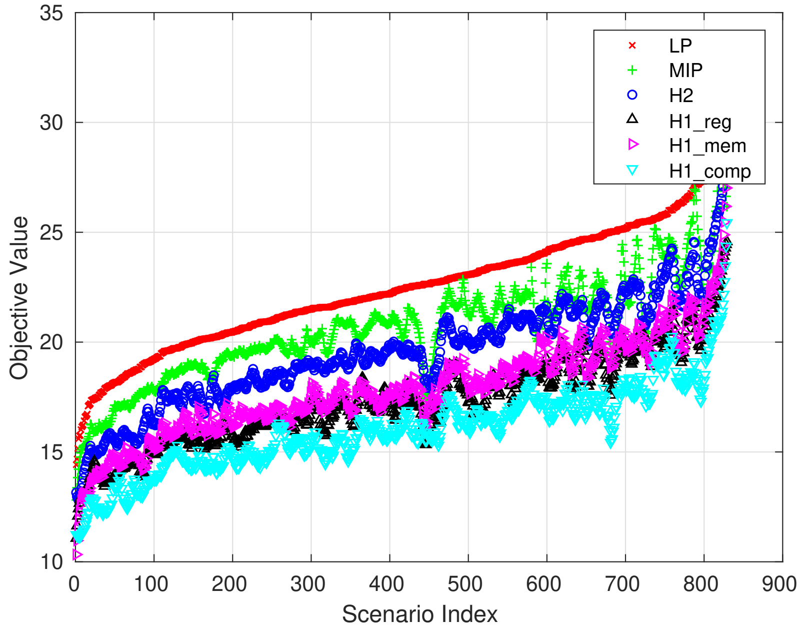

Figure 3: Maximal objective function achieved by the method. As expected the LP is the best performing algorithm. We observe about 10-20% reduction in performance for both MIP and H2 methods, which provides integer solution. The performances of MIP and H2 are close to each other. Our heuristic 1 performance is almost 30% less than the LP solution, and about 10% less than both MIP and H2. Recall that this is also an integer programming solution.

Our first heuristic algorithm, heuristic 1, finds the solution significantly faster than any other examined method. The execution time of the algorithm is about an order of magnitude less than that of LP. The execution time difference becomes even more drastic between our algorithm and the MIP solution where we observe more than 1000 times longer duration for the MIP solution. However in terms of optimal solution, it performs poor and there is about 25% performance gap between heuristic 1 and LP solutions and about 20% gap between the MIP solution. We propose another heuristic method where we relax service placement and access node variables to linear variables while keeping routing variables binary. This is also a MIP problem, however with branch and bound algorithm it converges to a solution significantly faster than standard MIP. Using this relaxed smaller MIP programming we iteratively solve the problem until all of our variables are integer. The performance of the heuristic 2 algorithm is about 10% below the LP solution and about 5% of the standard MIP solution. The advantage of our heuristic 2 algorithm shows itself in execution time. It not only takes less time, but achieve a more stable duration. We observe about an order of magnitude reduced run-time compared to the standard-MIP solution.

Furthermore, we also estimate the objective value on 500 users on our model. Due to the MIP problem may take too much time, we just applied LP, heuristic 2 and heuristic 1 on 500 users and run it on our local machine (not HPC). In order to be consistent with previous configuration, we apply the same setting IPgap=1e-2 threads=1. Figure 5 shows the comparison on the objective value. We can see heuristic 2 still has a great performance and gets a very close objective value with LP.

6 Conclusion

In this paper we examined the problem of placing application services on resource limited edge network devices and routing the demands between access and server nodes. We propose two heuristic algorithm one focused on short execution time with significant degradation on optimal solution value and the other focused on a balanced execution time and ”good” optimal solution. For our algorithm

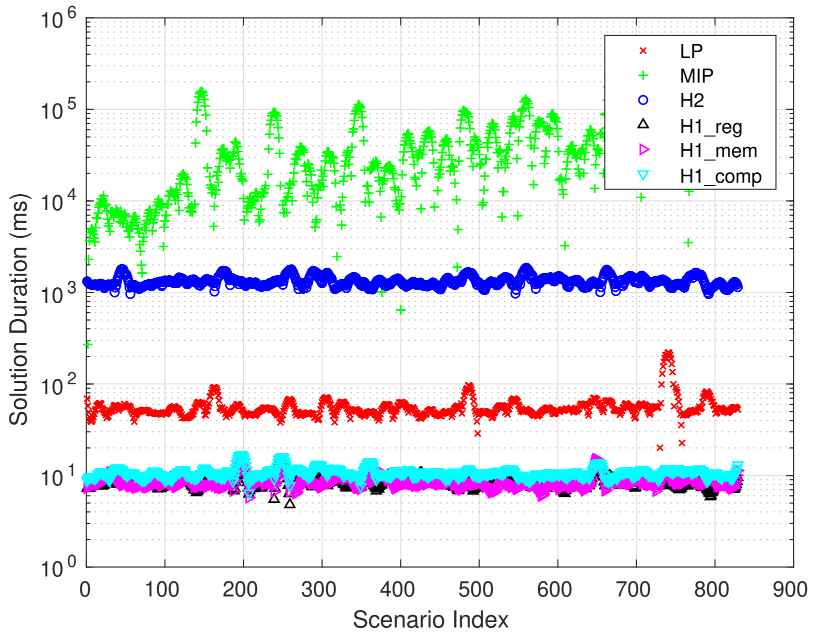

Figure 4: The computation time comparison. H2's execution time is 10 times shorter than MIP's. Besides, heuristic 2 can achieve more stale solution duration than original MIP problem. H1 finds the solution significantly faster than any other method.

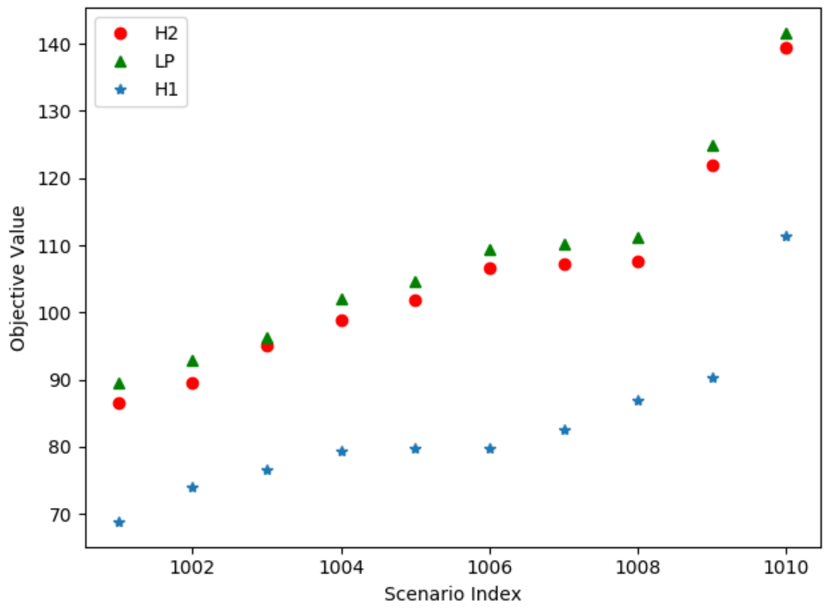

Figure 5: User=500. For H2, it takes around 10 seconds on our local machine. LP is the best performing algorithm and upper bound. We observe about 3% reduction in performance for H2 methods, which provides approximated integer solution.

one, we observe more than 1000 times reduction in execution time compared to MIP solution with about 25% performance degradation. For our algorithm two, we observe about 5% performance degradation compared to MIP solution where the execution time is 10 times shorter. Besides, heuristic 2 can achieve more stale solution duration than original MIP problem. Furthermore, heuristic 2 has hyper-parameter, which gives it more adjustable performance. On a larger set with 500 users, heuristic 2 still have a great approximation.

7 Future work

In Heuristic 2, it is still mixed integer programming in each iteration, which may take very long time on large-scale data set.We will examine more scenarios to compare our algorithms in terms of scalibility. Moreover, wireless links can fail at any time and this requires an online update to our algorithms. We will include failure probabilities, time dependent demands to our future model.

References

[1] "5g empowering vertical industries," Feb 2016.

[2] "Deliverable d2.1 5g car scenarios, use cases, requirements and kpis," Tech. Rep. 5GCAR/D2.1 V2.0, 5GCAR, February 2019.

[3] 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.

[4] ETSI, "Mobile edge computing (mec); framework and reference architecture v1.1.1," *Technical Report, ETSI MEC ISG*, 2016.

[5] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[6] J. Lu, D. Steinbach, P. Cabrol, and P. Pietraski, "Modeling human blockers in millimeter wave radio links," *ZTE Communications Magazine*, vol. 2012, pp. 23–28, 12 2012.

[7] Y. Azar, G. N. Wong, K. Wang, R. Mayzus, J. K. Schulz, H. Zhao, F. Gutierrez, D. Hwang, and T. S. Rappaport, "28 ghz propagation measurements for outdoor cellular communications using steerable beam antennas in new york city," in *2013 IEEE International Conference on Communications (ICC)*, pp. 5143–5147, 2013.

[8] M. Giordani, M. Mezzavilla, S. Rangan, and M. Zorzi, "Multi-connectivity in 5g mmwave cellular networks," in *2016 Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, pp. 1–7, 2016.

[9] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pp. 10–18, April 2019.

[10] B. Yang, W. K. Chai, Z. Xu, K. V. Katsaros, and G. Pavlou, "Cost-efficient nfv-enabled mobile edge-cloud for low latency mobile applications," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 475–488, 2018.

[11] T. He, H. Khamfroush, S. Wang, T. La 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 *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 365–375, 2018.

[12] G. B. Dantzig and M. N. Thapa, *Linear programming 1: Introduction.* Springer-Verlag New York, 1997.

[13] G. B. Dantzig and M. N. Thapa, *Linear Programming 2: Theory and Extensions.* Springer-Verlag New York, 2003.