POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

A Stackelberg Game Approach for Managing AI Tasks in a Mobile Edge Cloud System with Multiple Applications

Tesi di Laurea Magistrale in
Mathematical Engineering - Ingegneria Matematica

Author: **Ettore Busani**

Student ID: 968064
Advisor: Prof. Danilo Ardagna
Co-advisors: Hamta Sedghani
Academic Year: 2021-22

Abstract

The combination of Mobile and cloud computing is the essence of many applications in the fields of Augmented reality and Internet of Things. The common obstacle faced by the providers is to guarantee Quality of Service requirements (i.e. execution time of applications) while minimizing the cost of cloud resources. A newly popular approach to these difficulties is provided by the edge paradigm, by which the client data is processed by a platform in the periphery of the network, as close to the originating source as possible. This topic is presented in [3], where the platform profit optimization problem is defined as a Stackelberg game, and solved via an heuristic approach. This project aims at generalizing the work presented in [3], by extending the number of AI applications offered by the edge provider to the users. Moreover, we refined the final solution provided by the heuristic approach by adding a clever search algorithm based on Tabu Search. Finally, the model and algorithms presented in this document have been implemented in a C++ program in order to evaluate the performance of our approach and measure the quality of our results by comparing them with the commercial solution of a traditional solver.

Keywords: Edge & Cloud Computing, AI Applications, Stackelberg Game, Optimal Resource Allocation

Contents

Introduction

Artificial Intelligence (AI) and Internet of Things (IoT) are two rapidly evolving technologies that together form intelligent and connected systems. IoT refers to the network of physical devices, vehicles, home appliances, and other items embedded with electronics, sensors, and software, enabling them to connect and exchange data.
The integration of AI and IoT has significant implications for various industries and provides users with intelligent AI applications that can be executed on mobile devices. These applications leverage AI technologies such as NLP, machine learning, and computer vision to provide many different services like voice assistants, image recognition, personalized recommendations, virtual try-on, health monitoring, language translation, and fraud detection.
In most cases, this type of applications generates vast amounts of data that require real-time processing and analysis. These tasks prevent many AI applications from being processed directly on mobile devices, that are restricted by limited storage capacities and computational power. One possible solution is resorting to cloud computing, which however, may originate new problems: indeed, cloud servers are usually located far from mobile devices meaning that the execution of the application will be likely affected by a long latency.
For these reasons, a more reliable solution is the edge paradigm, also known as edge computing, which is a computing model that brings computational power and data storage closer to the devices and sensors that generate data to achieve better performances. By processing data at the edge, near the source of the data, edge computing reduces latency, improves response times, and conserves bandwidth. All these features make it well-suited for applications that require low latency, high performance, and real-time data processing.

In this work, we consider a Mobile Edge Cloud System composed by a large number of mobile users and the edge provider (or platform). Moreover, the edge provider has access to a pool of unlimited cloud VMs, that provides additional computational power whenever the load of users participating in the system is too heavy to be processed just by the edge servers. In order to participate in the system, the users must pay a fee which is set by the platform and used to cover the expenses of edge and cloud resource allocation. The difference of the users' revenue and the cost of edge and cloud resources is the net profit of the platform.
Within this framework, our purpose is to use a Game Theory approach for modelling the interaction between the edge provider and mobile users. As such, each user is considered as a separate and independent agent who, depending on the memory and energy capacities of his/her device, can choose to run his/her AI application locally or offload part of

the computation to the edge platform, so to minimize his/her expenses. Similarly, the edge platform wants to maximize its net profit and does so by fixing the optimal fee and finding the resource allocation which minimize its costs and is compliant to all performance constraints.
Notice how by changing the participation fee, the platform has control over users'decision, making our setting equivalent to a Stackelberg Game where the platform is the leader, and the users are the followers.

In summary, the main contributions of this work are the following:

1. We formulate the interaction among multiple mobile users and edge platform as a one-leader multi-follower Stackelberg's game, where the edge platform is the leader and the users are the followers.
2. We develop an approach by relying on KKT condition to find a closed form for providing the optimal number of edge and cloud resources and solve the mixed integer nonlinear problem quickly.
3. We finally evaluate the performance of the proposed approach by analyzing the time required to reach the optimal solution under different conditions.

The remaining portion of this thesis is structured in the subsequent manner. Chapters 1 introduces all the relevant mathematical definitions for the mobile edge cloud system model, while the formulation of the problem as a Stackelberg game is presented in Chapter 2. In Chapter 3, we explain our proposed approach and reformulate the Stackelberg Game as a convex problem and find a closed form for the optimal solution. The implementation of the model and algorithms in C++ code are presented in Chapter 4. Finally, the experimental analysis of our approach is discussed in Chapter 5, where we measure the quality of our solutions by comparison with the results obtained by running a traditional solver.

1 | System Model and Problem Formulation

The purpose of this Chapter is to introduce all the relevant definitions composing the mathematical framework of our setting.
More specifically, in Section 1.1 we present the Mobile Edge Cloud System model, with all its relevant variables and constraints; as well as the users and platform utility functions. Section 1.2 contains the mathematical formulation of the users' decision problem and platform optimization problem.

1.1. Mobile Edge Cloud model

In this thesis, we consider a mobile edge cloud system as shown in Figure 1.1, with an edge platform that includes edge nodes with limited resources, a pool of unlimited VMs in the cloud side and a large number of mobile users with smart devices who would like to run one among multiple AI applications. We assume that cloud VMs are connected to the edge platform through a fast network with negligible delay, whereas users' connection is affected by a non-negligible delay.

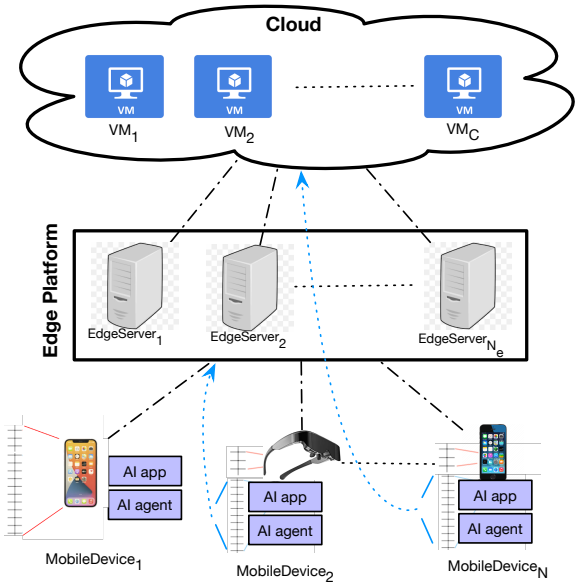


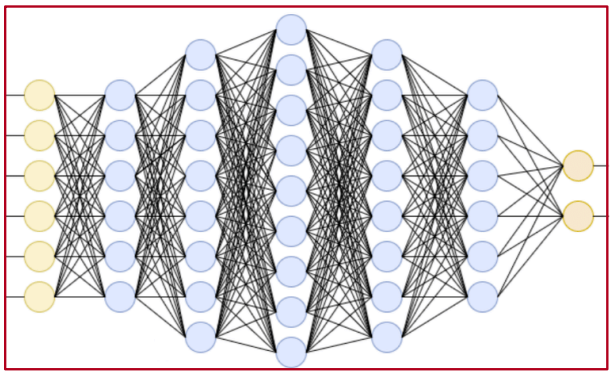
Figure 1.1: Application's resource assignment model.

We denote the set of users as $\mathcal{U}$ and assume that there are N_e homogeneous servers in the edge system and an unlimited pool of homogeneous cloud VMs. On the contrary, the users are heterogeneous, meaning that their devices have different characteristics from one to another.
Moreover, we denote the set of AI applications with $\mathcal{A}$ and assume that each user can run only one application at the time. Users communicate to the platform to inform which application they are willing to run, meaning that the platform knows the load of users for each application.
When referring to different applications we will use the superscript a that takes values in $\mathcal{A} = \{1, 2, \cdots, |\mathcal{A}|\}$. For example, we denote with $\mathcal{U}^{(a)}$ the set of users that want to run application a; meaning that the set of users is partitioned as follows:

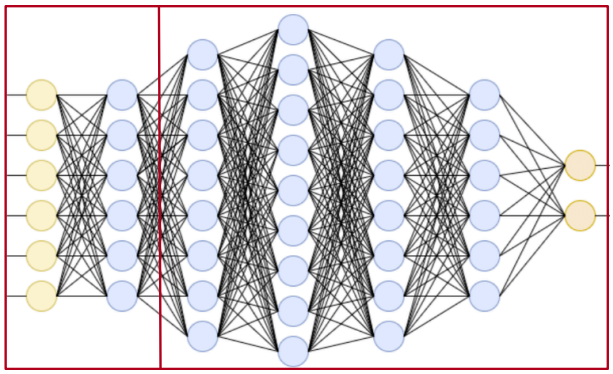
$$\mathcal{U} = \bigcup_{a=1}^{|\mathcal{A}|} \mathcal{U}^{(a)}.$$

We assume that all AI applications provided by the edge platform are based on a single DNN. As mentioned previously, we assume that an AI application can be specified by 2 alternative deployments indexed by k: an example of AI application is shown in Figure 1.2. The first deployment ($k = 1$, see Figure 1.2a) is a full DNN which can be deployed only on the user device, while the other ($k = 2$, see Figure 1.2b) has two partitions: the first partition will be run only by the user device while the second one can be run on edge servers or on cloud VMs.

For the rest of this thesis, whenever we say that user i runs deployment k of application a, we actually mean that user i runs the first partition of deployment k of application a, where the first partition of deployment 1 is the full application.



(a) Candidate deployment 1 (full component)



(b) Candidate deployment 2

Figure 1.2: Example of AI application component with two deployments

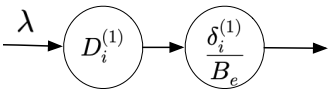
The power consumption of user-i for running deployment k is denoted by $p_i^{(k)}$. Users pay a constant time unit fee equal to $r^{(a1)}$ when running application a locally, while pay $r^{(a2)} = r^{(a1)} + \gamma^{(a)} r$ for the second deployment incurring an extra cost $r > 0$ set by the edge provider which varies in the interval $[r_{min}, r_{max}]$.

We assume each user is interested in running just one specific AI application; and we denote the utility gain for running their desired application with U_i, $\forall i \in \mathcal{U}$. Therefore, users are willing to participate in the system as long as they have enough energy and memory requirements and U_i is larger than the cost of running the application.

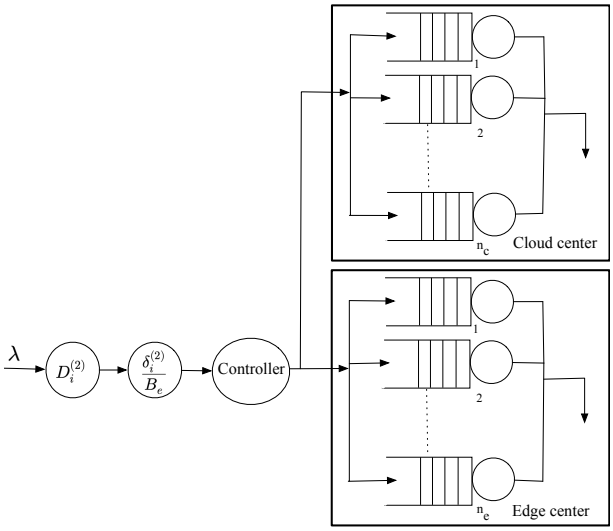
We also define the demanding time (i.e., the time required to serve a single request of the underlying service without resource contention [2]) $D_i^{(k)}$ required to run the first partition[1] of deployment $k = 1, 2$ on user-i's device; while we denote with $D_e^{(a)}$ and $D_c^{(a)}$ the demanding time to serve the second partition of the second deployment of application $a = 1, 2, \ldots, |\mathcal{A}|$ on an edge server and cloud VM, respectively.
Both the edge servers and cloud VMs are modeled as a M/G/1 queue [4] while the local IoT devices are modelled as a delay center (see Figure 1.3).

In our model, the edge provider would like to maximise its net income given by the revenues to support the users' applications minus the energy costs to run the partitions

[1]The first partition of deployment 1 is the full application.



(a) Deployment 1



(b) Deployment 2

Figure 1.3: Queuing model of resources.

of the users who choose the second deployment at the edge servers and the cost of remote cloud resources. Resource planning is performed periodically on a time horizon T and consists in assigning the optimal amount of cloud and edge servers $(n_e^{(a)},\ n_c^{(a)})$ to each application, in order to guarantee all requirements are satisfied. The total number of edge servers that can be allocated is limited, meaning that $\sum_{a=1}^{|\mathcal{A}|} n_e^{(a)} \leq N_e$. Moreover, the platform can also serve users requests on the cloud with no limitations on the number of VMs available. It is assumed that the edge platform owns the edge infrastructure and will try to maximize the usage of edge resources before leasing any additional resources on a public cloud.

On the other hand, each user decides which deployment of the chosen application to run, according to his/her energy consumption for the execution of the deployment and also the cost of running the deployment on the edge which is set by the edge provider. Hence, the mobile users trade-off the cost of their deployment (the more they run locally the lower is their cost) with the energy consumption of their device.

In order to denote the assignment decisions, namely to characterize which deployment is selected by every user for his/her desired application and how the load of the second deployment is assigned to the resources, we introduce the following binary variables:

$$x_i^{(k)} = \begin{cases} 1 & \text{if user } i \text{ selected deployment } k \text{ ,} \\ 0 & \text{otherwise.} \end{cases} \tag{1.1}$$

$$y_i^e = \begin{cases} 1 & \text{if user } i\text{'s deployment is served by edge VMs,} \\ 0 & \text{otherwise} \end{cases} \tag{1.2}$$

$$y_i^c = \begin{cases} 1 & \text{if user } i\text{'s deployment is served by cloud VMs,} \\ 0 & \text{otherwise.} \end{cases} \tag{1.3}$$

Where $x_i^{(k)}$ is a user decision variable, while y_i^e and y_i^c are the edge platform decision variables.

We have the following constraint for $x_i^{(k)}$ (at most, only one deployment can be selected):

$$\sum_{k=1}^{2} x_i^{(k)} \leq 1 \qquad \forall i \in \mathcal{U}, \tag{1.4}$$

and we also have the constraint that forces the platform to run the second partition either on edge or cloud for all the users that opted the second deployment:

$$y_i^e + y_i^c = x_i^{(2)} \qquad \forall i \in \mathcal{U}. \tag{1.5}$$

Each user has a maximum memory and energy capacity denoted by $\bar{M}_i$ and $\bar{E}_i$, meaning they can't run deployments whose requirements exceed these limits. These constraints are formulated as follows:

$$\sum_{k=1}^{2} m^{(ak)} x_i^{(k)} \leq \bar{M}_i \qquad \forall i \in \mathcal{U}^{(a)},\ \forall a \in \mathcal{A}, \tag{1.6}$$

where $m^{(ak)}$ denotes the memory requirement of deployment k of application a.

$$E_i \leq \bar{E}_i \qquad \forall i \in \mathcal{U}, \tag{1.7}$$

where E_i is the energy consumption of user-i, and it is defined as follows:

$$E_i = \lambda^{(a)} T_i^2 \sum_{k=1}^{2} p_i^{(k)} x_i^{(k)} \quad \forall i \in \mathcal{U}^{(a)},\ \forall a \in \mathcal{A}. \tag{1.8}$$

T_i is the time duration user-i uses his/her application; while $\lambda^{(a)}$ denotes the input load of users belonging to $\mathcal{U}^{(a)}$ (expressed in terms of requests/sec), meaning that the total

load corresponding to the second partition of the second deployment of each application is computed as follows:

$$\Lambda^{(a)} = \sum_{i \in \mathcal{U}^{(a)}} \lambda^{(a)} x_i^{(2)} \qquad \forall\ a \in \mathcal{A}. \tag{1.9}$$

In our model, each edge server and cloud VM is modeled as single server multiple class queue system (i.e., as an individual M/G/1 queue). Hence, the load of each application in edge and cloud will be as follows:

$$L_e^{(a)} = D_e^{(a)} \lambda^{(a)} \sum_{i \in \mathcal{U}^{(a)}} x_i^{(2)} y_i^e \qquad \forall\ a \in \mathcal{A}, \tag{1.10}$$

$$L_c^{(a)} = D_c^{(a)} \lambda^{(a)} \sum_{i \in \mathcal{U}^{(a)}} x_i^{(2)} y_i^c \qquad \forall a \in \mathcal{A}. \tag{1.11}$$

In order to avoid resource saturation, if the second partition of the second deployment is deployed onto an edge or cloud VM, the equilibrium conditions for the M/G/1 queue must hold. In particular, this is equivalent to prescribe on the edge and cloud:

$$\frac{L_e^{(a)}}{n_e^{(a)}} < 1 \iff L_e^{(a)} < n_e^{(a)} \qquad \forall a \in \mathcal{A}, \tag{1.12}$$

$$\frac{L_c^{(a)}}{n_c^{(a)}} < 1 \iff L_c^{(a)} < n_c^{(a)} \qquad \forall a \in \mathcal{A}. \tag{1.13}$$

Now, for every $a \in \mathcal{A}$ and for every $i \in \mathcal{U}^{(a)}$ we compute the average response time for user i as follows:

$$R_i = \sum_{k=1}^{2} D_i^{(k)} x_i^{(k)} + \frac{\delta^{(a)} x_i^{(2)}}{B_i} + \frac{D_e^{(a)} x_i^{(2)} y_i^e}{1 - \frac{L_e^{(a)}}{n_e^{(a)}}} + \frac{D_c^{(a)} x_i^{(2)} y_i^c}{1 - \frac{L_c^{(a)}}{n_c^{(a)}}} \tag{1.14}$$

Where the first expression denotes the running time of the selected deployment on users' device. The second expression indicates the latency from users to the edge platform. Note that by running the first deployment, the users do not send any data to the platform. $\delta^{(a)}$ denotes the data transfer size of the first partition of the second deployment of application a and B_i indicates the bandwidth of the network from user i to the platform. In the above equation, only the transmission delay between the end users and the edge platform is considered, which includes the radio interface. The delay between the edge servers and the central cloud is ignored because that network segment has high-speed links and is rarely the bottleneck [1]; therefore the resulting delay is negligible. The third and forth expressions indicate the average running time of the second deployment on edge and cloud, respectively. It is assumed that $D_e^{(a)} \lambda^{(a)} < 1,\ \forall\ a \in \mathcal{A}$, which means that one server in the edge is powerful enough to serve the load of a single user. Moreover, the cloud VMs are more powerful than the edge servers and it holds $D_c^{(a)} < D_e^{(a)},\ \forall\ a \in \mathcal{A}$.

In order to guarantee QoS requirements of the application, we define a threshold $\bar{R}^{(a)}$ for the average response time of a user running application a:

$$R_i \leq \bar{R}^{(a)} \quad \forall i \in \mathcal{U}^{(a)},\ \forall a \in \mathcal{A}. \tag{1.15}$$

1.1.1. Edge platform profit model

According to our model, the platform decision variables are the following:

- r: is the extra cost charged by the platform for the users opting for the second deployment (equal for all applications),
- $y_i^e, y_i^c \ \forall i \in \mathcal{U}$: binary variables that specify whether the users are served in the edge or cloud,
- $n_e^{(a)}, n_c^{(a)} \ \forall a \in \mathcal{A}$: are the number of edge servers and cloud VMs allocated by the platform to process the load of each application.

Given $n_c = \sum_{a \in \mathcal{A}} n_c^{(a)}$ and $n_e = \sum_{a \in \mathcal{A}} n_e^{(a)}$, the profit of the edge platform is computed as follows:

$$P_e = \sum_{a \in \mathcal{A}} \sum_{i \in \mathcal{U}^{(a)}} \sum_{k=1}^{2} T_i r^{(ak)} x_i^{(k)} - T\left(c_e n_e + c_c n_c\right) \tag{1.16}$$

where c_e and c_c are, respectively, the edge and cloud VM cost per second (per second billing option is recently available in AWS[2] and Azure[3]) and T is the set time horizon used by the edge provider to manage resource planning. In particular, the parameter c_e takes into account both the scenarios in which the provider owns the edge platform, so it has to pay the energy consumption, or it has to pay an edge provider, which fixes a price per second. As stated previously, we assume that the cost of running the deployment on edge is always less than the cloud one, i.e., $c_e D_e^{(a)} < c D_c^{(a)} \quad \forall a \in \mathcal{A}$.

1.1.2. User cost model

We define the cost function of user i as follows ($\forall i \in \mathcal{U}^{(a)}, \ \forall a \in \mathcal{A}$):

$$C_i = \sum_{k=1}^{2} T_i x_i^{(k)} \left(\alpha_i r^{(ak)} + (1-\alpha_i)\beta_i p_i^{(k)} \lambda^{(a)} T_i + x_i^{(2)} \zeta_i \delta^{(a)} \lambda^{(a)}\right), \tag{1.17}$$

where α_i is a trade-off parameter between cost and energy, β_i is a coefficient to convert the energy consumption of user's device to monetary cost and ζ_i is a coefficient to convert the data transfer size to monetary cost.

1.2. Problem Formulation

In this Section we finally set the mathematical formulation of the constrained optimization problem featuring our model, both from a user and platform perspective.
The platform optimization problem is formulated in Subsection 1.2.1 and the users' decision problem in Subsection 1.2.2.

[2] `https://aws.amazon.com/ec2/pricing/`
[3] `https://azure.microsoft.com/en-us/pricing/details/virtual-machines/windows/`

1.2.1. Platform optimization problem

The edge platform optimization problem is formulated as follows:

$$\max_{r,n_e^{(a)},n_c^{(a)},y_i^e,y_i^c} P_e = \sum_{a\in\mathcal{A}}\sum_{i\in\mathcal{U}^{(a)}}\sum_{k=1}^{2} T_i r^{(ak)} x_i^{(k)} - T\left(c_e n_e + c_c n_c\right) \tag{1.18}$$

Subject to: (1.19)

$$\sum_{a\in\mathcal{A}} n_e^{(a)} \leq N_e, \tag{1.20}$$

$$\sum_{k=1}^{2} D_i^{(k)} x_i^{(k)} + \frac{\delta^{(a)} x_i^{(2)}}{B_i} + \frac{D_e^{(a)} x_i^{(2)} y_i^e}{1-\frac{L_e^{(a)}}{n_e^{(a)}}} + \frac{D_c^{(a)} x_i^{(2)} y_i^c}{1-\frac{L_c^{(a)}}{n_c^{(a)}}} \leq \bar{R}^{(a)} \quad \forall i \in \mathcal{U}^{(a)},\ \forall a \in \mathcal{A}, \tag{1.21}$$

$$L_e^{(a)} < n_e^{(a)} \qquad \forall a \in \mathcal{A}, \tag{1.22}$$

$$L_c^{(a)} < n_c^{(a)} \qquad \forall a \in \mathcal{A}, \tag{1.23}$$

$$y_i^e + y_i^c = x_i^{(2)} \qquad \forall i \in \mathcal{U}, \tag{1.24}$$

$$r^{(a2)} = r^{(a1)} + \gamma^{(a)} r \quad \gamma^{(a)} > 1, \qquad \forall a \in \mathcal{A}, \tag{1.25}$$

$$r_{min} \leq r \leq r_{max}, \tag{1.26}$$

$$y_i^e, y_i^c \in \{0,1\} \quad \forall\ i \in \mathcal{U}, \tag{1.27}$$

$$n_e^{(a)}, n_c^{(a)} \in \mathbb{Z}_+ \quad \forall a \in \mathcal{A}. \tag{1.28}$$

Where $r^{(a1)}$ is the constant time unit fee for running application a, while r is the extra cost to be paid by users if they select the second deployment, meaning the platform will have to allocate edge or cloud resources to serve the computational load of the second partition of the DNN. Note that r is one of the decision variables of the edge platform, whereas $r^{(a1)}$ are assumed to be fixed.

1.2.2. User's decision problem

Once the price r has been proposed by the platform, $x_i^{(k)}$ is the only decision variable of user i. The users make a decision about $x_i^{(k)}$ based on their device's memory and energy limit and the cost of running the deployments. The user's optimization problem

is formulated as follows (where a denotes the application of user i):

$$\max_{x_i^{(k)}} \sum_{k=1}^{2} x_i^{(k)} \left(U_i - T_i \left(\alpha_i r^{(ak)} + (1-\alpha_i)\beta_i p_i^{(k)} \lambda^{(a)} T_i + x_i^{(2)} \zeta_i \delta^{(a)} \lambda^{(a)} \right) \right) \tag{1.29}$$

Subject to: (1.30)

$$\sum_{k=1}^{2} x_i^{(k)} \leq 1, \tag{1.31}$$

$$\lambda^{(a)} T_i^2 \sum_{k=1}^{2} p_i^{(k)} x_i^{(k)} \leq \bar{E}_i, \tag{1.32}$$

$$\sum_{k=1}^{2} m^{(ak)} x_i^{(k)} \leq \bar{M}_i, \tag{1.33}$$

$$x_i^{(k)} \in \{0,1\} \qquad \forall k = 1,2. \tag{1.34}$$

We introduced users' utility U_i inspired by an incentive mechanisms like [3], as the reward-value for user i to run his/her desired AI application. In this way, users are motivated to join the system whenever their cost is less than the value U_i. Note that given U_i, the budget constraint of user i is not considered necessary for the platform, because if the price of the deployments is too high (greater than U_i), the user will simply not participate in the system.

In order to guarantee the problem feasibility, we have the following assumption on users' parameters: if the memory and energy of user's device are enough to run the k-th deployment and $U_i \geq C_i^{(k)}$[4], the following condition must hold ($\forall i \in \mathcal{U}^{(a)},\ \forall a \in \mathcal{A}$):

$$D_i^{(1)} < \bar{R}^{(a)},$$
$$D_i^{(2)} + \frac{\delta^{(a)} x_i^{(2)}}{B_i} + \min\{D_e^{(a)}, D_c^{(a)}\} < \bar{R}^{(a)}.$$

For the reader's convenience, the variables and parameters of the model are summarized in Table 1.1 & Table 1.2.

[4] C_i^k denotes the cost of user i for running deployment k of his/her application.

Table 1.1: Decision Variables

Decision Variables	
$x_i^{(k)}$	1 if user i selected deployment k of his/her application.
r	Extra price for running deployment 2.
y_i^e	1 if user i is served on edge.
y_i^c	1 if user i is served on cloud.
$n_e^{(a)}$	The number of edge VMs to serve application a.
$n_c^{(a)}$	The number of cloud VMs to serve application a.

Table 1.2: Problem Parameters

Parameters	
$\mathcal{U}$	Set of users.
$\mathcal{A}$	Set of applications.
$\mathcal{U}^{(a)}$	Set of users of applications a
N_e	Maximum number of available nodes in the edge.
$D_i^{(k)}$	Demanding time to run deployment k on user-i's device.
$D_e^{(a)}$	Demanding time to run the second deployment of application a on edge server.
$D_c^{(a)}$	Demanding time to run the second deployment of application a on cloud VMs.
$m^{(ak)}$	Memory requirement for deployment k of application a.
$p_i^{(k)}$	Power consumption of user-i's device to run deployment k of his/her application.
$\lambda^{(a)}$	Incoming workload of users running the second deployment of application a.
$\delta^{(a)}$	The data transfer size of the first partition of the second deployment of application a.
B_i	Bandwidth from users' device to edge platform.
$\bar{R}^{(a)}$	Upper bound threshold for the average response time of application a.
c_c	Cost of cloud VMs.
c_e	Cost of edge servers.
β_i	Coefficient to convert the energy consumption of user-i's device to monetary cost.
α_i	Parameter to make a trade off between cost and energy consumption of users.
$\bar{E}_i$	Maximum energy of user-i's device to run the applications.
$\bar{M}_i$	Maximum memory of user-i's device.
T_i	Total activation time of user-i for his/her selected application.
$r^{(a1)}$	Constant service fee for running the application a locally.
U_i	Utility of user-i for running his/her application.

2 | Stackelberg Game Formulation

In this Chapter, we present the Stackelberg Game as a mixed-integer nonlinear optimization problem, by embedding the users' problem in the platform problem. In our framework, the leader of the Stackelberg game is the platform, whereas the users are the followers.

In our scenario the AI applications have a controller agent that provides the users' parameters to the platform, such as $T_i, \beta_i, \bar{E}_i, \bar{M}_i$ and $p_i^{(k)}$. Moreover, the platform can influence users' decision by increasing or decreasing the extra-fee r charged for the second deployment. Therefore, the platform can anticipate users' decision without contacting them by solving their problem in a centralized manner. Therefore, it is reasonable to embed the users' decision problem in the platform optimization problem.
This new problem is the Stackelberg Game, and it will be the focus of the rest of this thesis. Its formulation is reported below:

$$\max_{r,n_e^{(a)},n_c^{(a)},y_i^e,y_i^c} P_e = \sum_{a\in\mathcal{A}}\sum_{i\in\mathcal{U}^{(a)}}\sum_{k=1}^{2} T_i r^{(ak)} x_i^{(k)} - T\left(c_e n_e + c_c n_c\right) \tag{2.1}$$

Subject to: (2.2)

$$\mathbf{x}_i^*(\mathbf{r}) = \arg\max_{\mathbf{x}_i}\{\sum_{k=1}^{2} x_i^{(k)}\left(U_i - T_i\alpha_i r^{(ak)} + (1-\alpha_i)\beta_i p_i^{(k)}\lambda^{(a)}T_i + x_i^{(2)}\zeta_i\delta^{(a)}\lambda^{(a)}\right): \tag{2.3}$$

$$\sum_{k=1}^{2} x_i^{(k)} \le 1, \quad \lambda T_i^2 \sum_{k=1}^{2} p_i^{(k)} x_i^{(k)} \le \bar{E}_i, \tag{2.4}$$

$$\sum_{k=1}^{2} m^{(ak)} x_i^{(k)} \le \bar{M}_i, \tag{2.5}$$

$$x_i^{(k)} \in \{0,1\} \quad \text{for } k=1,2\}, \qquad \forall\ \ i \in \mathcal{U}^{(a)},\ \ \forall a \in \mathcal{A}. \tag{2.6}$$

$$\sum_{a\in\mathcal{A}} n_e^{(a)} \le N_e, \tag{2.7}$$

$$\sum_{k=1}^{2} D_i^{(k)} x_i^{(k)} + \frac{\delta^{(a)} x_i^{(2)}}{B_i} + \frac{D_e^{(a)} x_i^{(2)} y_i^e}{1-\frac{L_e^{(a)}}{n_e^{(a)}}} + \frac{D_c^{(a)} x_i^{(2)} y_i^c}{1-\frac{L_c^{(a)}}{n_c^{(a)}}} \le \bar{R}^{(a)} \quad \forall i \in \mathcal{U}^{(a)},\ \forall a \in \mathcal{A}, \tag{2.8}$$

$$L_e^{(a)} < n_e^{(a)} \qquad \forall a \in \mathcal{A}, \tag{2.9}$$

$$L_c^{(a)} < n_c^{(a)} \qquad \forall a \in \mathcal{A}, \tag{2.10}$$

$$y_i^e + y_i^c = x_i^{(2)} \qquad \forall i \in \mathcal{U} \tag{2.11}$$

$$r^{(ak)} = r^{(1)} + \gamma^{(a)} r, \quad \gamma^{(a)} > 1 \quad \forall a \in \mathcal{A} \tag{2.12}$$

$$r_{min} \le r \le r_{max} \tag{2.13}$$

$$y_i^e, y_i^c \in \{0,1\} \qquad \forall\ i \in \mathcal{U}, \tag{2.14}$$

$$n_e^{(a)}, n_c^{(a)} \in \mathbb{Z}_+. \qquad \forall a \in \mathcal{A}. \tag{2.15}$$

We remark that the optimality constraint (2.3)-(2.6) can be replaced by a set of linear constraints by introducing few additional binary variables and parameters, whose definition is reported below:

$$s_i^{(k)} = \begin{cases} 1 & \text{If user } i \text{ has enough energy and memory to run deployment } k, \\ 0 & \text{otherwise.} \end{cases}$$

$$t_i^{(k)} = \begin{cases} 1 & \text{If it is convenient for user } i \text{ to run deployment } k, \\ 0 & \text{otherwise.} \end{cases}$$

$$z_i^{(12)} = \begin{cases} 1 & \text{If user } i \text{ prefers to run deployment 1 to deployment 2,} \\ 0 & \text{otherwise..} \end{cases}$$

The linear constraints equivalent to (2.3)-(2.6) are shown in the following lines (where we

denote the cost of running deployment k for user i as $C_i^{(k)}$):

$$-(1-t_i^{(k)})M \leq U_i - C_i^{(k)} \leq t_i^{(k)} M \qquad \text{for } k=1,2 \tag{2.16}$$

$$-(1-z_i^{(12)})M \leq C_i^{(2)} - C_i^{(1)} \leq z_i^{(12)} M \tag{2.17}$$

$$x_i^{(k)} \leq s_i^{(k)} t_i^{(k)} \qquad \text{for } k=1,2 \tag{2.18}$$

$$x_i^{(1)} \geq s_i^{(1)} t_i^{(1)} + s_i^{(2)}(z_i^{(12)} - 1) \tag{2.19}$$

$$x_i^{(2)} \geq s_i^{(2)} t_i^{(2)} - s_i^{(1)} z_i^{(12)} \tag{2.20}$$

$$\sum_{k=1}^{2} x_i^{(k)} \leq 1 \tag{2.21}$$

$$x_i^{(k)}, t_i^{(k)}, z_i^{(12)} \in \{0,1\} \quad \text{for } k=1,2 \tag{2.22}$$

where M is a large enough positive real number.

The Stackelberg game (2.1)–(2.22) is a mixed-integer nonlinear program (MINLP) and it can be solved by a global solver. However, not only there is no guarantee to find the optimal solution because of bilinear non-convex term $r^{(ak)}x_i^{(k)}$ in the objective function, but also computing the solution is very slow because of the large number of constraints in (2.8). Therefore, in the next Section we propose an heuristic approach to solve the problem faster.
This method has already been presented in [3] for a similar problem. This document shows its adaptation to the framework of our model, where multiple applications are considered.

3 | Solution

Since finding an optimal solution for the Stackelberg Game by a state of the art tool is very slow because of the very large number of variables and constraints, in this section we propose an efficient heuristic approach to solve problem instances of practical interest. The heuristic approach is an adaptation to our current framework of the method presented in [3], which is used to solve a similar problem.
The heuristic approach is based on three main ingredients. First, Section 3.1 shows that, under the assumption of a fixed platform price, a solution to the edge platform resource allocation problem (i.e., the decisions for $n_e^{(a)}$ and $n_c^{(a)}$) close to the optimum can be identified by solving a convex optimization problem. Secondly, in Section 3.2 it is proved that the optimal price can be identified by inspection, considering $O(N)$ relevant price values (where N is the number of users).
The third step is to cluster the users opting for the second deployment in two groups, one served by the edge and one served by the cloud, starting from the approximate solutions found at the previous steps. This is done by an heuristic algorithm presented in Section 3.3.
Finally, as the original contribution of this thesis, section 3.4 presents a Tabu search algorithm for exploring more elements of the solutions space, starting from the solution of the heuristic approach, in the attempt to further optimize the final platform profit.
Figure 3.1 highlights the main steps we have just presented, that together frame our proposed approach.

3.1. Relaxed Problem

In this Section we show that the edge platform resource allocation problem, which consists in finding the optimal values for $n_e^{(a)}$ and $n_c^{(a)}$, can be approximately reformulated as a convex optimization problem by assuming fixed prices for the deployments of each application and by considering new simplified time-response constraints. This solution will then allow to solve the overall Stackelberg game quickly by means of a heuristic approach.
As we already mentioned, thanks to the controller agent, the platform can obtain the users' parameters and find their desired deployment for a fixed price by solving the users optimization problem following algorithm 3.1. This operation is trivial, as it only requires to check whether the user's device has enough energy and memory constraint to run the first or second deployment, and if that's the case, the corresponding cost is updated (lines 4-9). Then, the user's binary variables $x_i^{(1)}$ and $x_i^{(2)}$ are set to 0 or 1, by comparing the costs of the 2 deployments with the utility of the user for running the application (lines

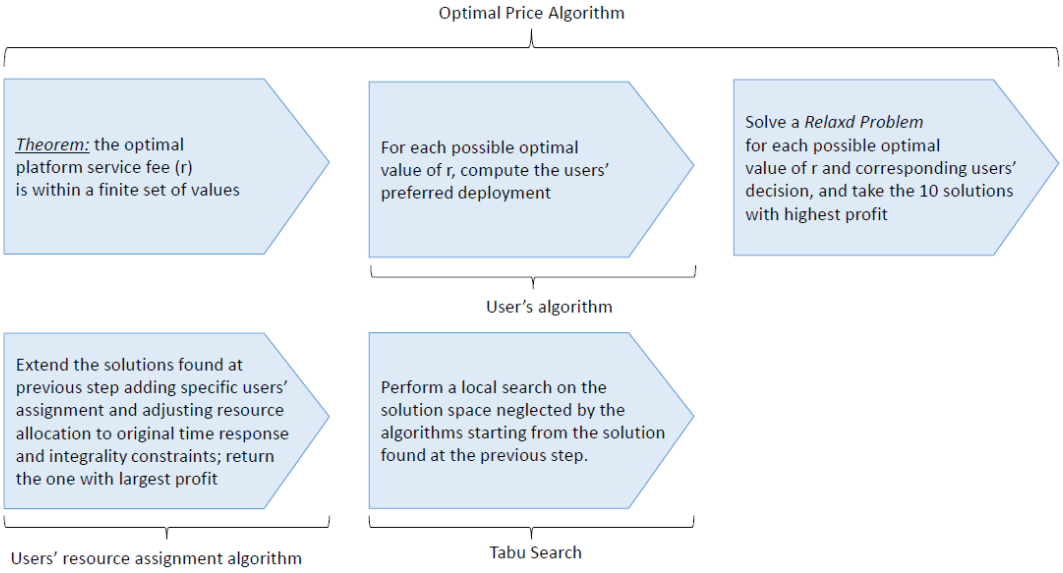


Figure 3.1: Flow chart of our proposed approach.

10-17).

Algorithm 3.1 User's algorithm

Input: i, a, α_i, $p_i^{(k)}$, $\bar{E}_i$, $\bar{M}_i$, β_i, $r^{(ak)}$
$Cost^{(1)} \leftarrow \infty$
$Cost^{(2)} \leftarrow \infty$
if $\lambda^{(a)} T_i^2 p_i^{(1)} \leq \bar{E}_i$ **and** $m^{(a1)} \leq \bar{M}_i$ **then**
    $Cost^{(1)} \leftarrow T_i \left(\alpha_i r^{(a1)} + (1-\alpha_i)\beta_i p_i^{(1)} \Lambda^{(a)} T_i \right)$
end if
if $\lambda^{(a)} T_i^2 p_i^{(2)} \leq \bar{E}_i$ **and** $m^{(a2)} \leq \bar{M}_i$ **then**
    $Cost^{(2)} \leftarrow T_i \left(\alpha_i r^{(a2)} + (1-\alpha_i)\beta_i p_i^{(2)} \Lambda^{(a)} T_i + \zeta_i \lambda^{(a)} \delta^{(a)} \right)$
end if
if $U_i \geq Cost^{(1)}$ **and** $Cost^{(1)} < Cost^{(2)}$ **then**
    $x_i^{(1)} = 1,\ x_i^{(2)} = 0$
else if $U_i \geq Cost^{(2)}$ **and** $Cost^{(1)} \geq Cost^{(2)}$ **then**
    $x_i^{(1)} = 0,\ x_i^{(2)} = 1$
else
    $x_i^{(1)} = 0,\ x_i^{(2)} = 0$
end if
return $x_i^{(1)}, x_i^{(2)}$.

Note that, the incoming load of each application $\Lambda^{(a)}$ (defined in 1.9) depend only on x_i which in turn depend only on r. Hence, if r is fixed, x_i will be fixed, meaning that all application loads are also going to be fixed. Thus, for each price r, the platform knows the corresponding $\Lambda^{(a)}$.
We assume that the total incoming load of application a is partitioned between cloud ($\Lambda_c^{(a)}$) and edge ($\Lambda_e^{(a)}$) such that we have:

$$\Lambda_e^{(a)} + \Lambda_c^{(a)} = \Lambda^{(a)} \tag{3.1}$$

Hence, we introduce the new platform decision variables $\Lambda_e^{(a)}$ and $\Lambda_c^{(a)}$ which represents the rate of users for each application that will be served at the edge and cloud, respectively. Moreover, we consider a new and simplified time response constraint for each application defined as:

$$\bar{\bar{R}}^{(a)} = \bar{R}^{(a)} - \mathbb{E}_a\left[D_i^{(2)} x_i^{(2)}\right] - \mathbb{E}_a\left[\frac{\delta^{(a)} x_i^{(2)}}{B_i}\right] \qquad \forall a \in \mathcal{A}, \tag{3.2}$$

where $\mathbb{E}_a\left[D_i^{(2)} x_i^{(2)}\right]$ denotes the expected demand time of running the first partition of deployment 2 on the local device of users who selected the second deployment of application a and $\mathbb{E}_a\left[\frac{\delta^{(a)} x_i^{(2)}}{B_i}\right]$ indicates the expected transmission delay to transfer the output data to the platform of the first partition of deployment 2 of users who selected the second deployment of application a. The rationale behind (3.2) is that, in order to guarantee the server side response time, it is necessary to provide a margin to account for the user side processing time and data transmission delay.
Hence, the response time constraints (1.21) is redefined on the edge platform side as follows:

$$\frac{\Lambda_e^{(a)}}{\Lambda^{(a)}} \cdot \frac{D_e^{(a)} n_e^{(a)}}{n_e^{(a)} - D_e^{(a)} \Lambda_e^{(a)}} + \frac{\Lambda_c^{(a)}}{\Lambda^{(a)}} \cdot \frac{D_c^{(a)} n_c^{(a)}}{n_c^{(a)} - D_c^{(a)} \Lambda_c^{(a)}} \leq \bar{\bar{R}}^{(a)} \quad \forall a \in \mathcal{A}. \tag{3.3}$$

The following result can be demonstrated.

Theorem 3.1. *The response time constraints in* (3.3) *are convex.*

Proof. The proof is given in Appendix A.1. □

In order to identify an estimate for $n_e^{(a)}$ and $n_c^{(a)}$, a simplified edge platform problem can be written by neglecting y_i^e and y_i^c, assuming a fixed price for the deployment, and relaxing the integrality constraint on $n_e^{(a)}$ and $n_c^{(a)}$, which are now considered as continuous

variables:

$$\min_{n_e^{(a)}, n_c^{(a)}, \Lambda_e^{(a)}, \Lambda_c^{(a)}} \quad c_e n_e + c_c n_c \tag{3.4}$$

$$\text{Subject to:} \tag{3.5}$$

$$\sum_{a \in \mathcal{A}} n_e^{(a)} \leq N_e \tag{3.6}$$

$$\forall a \in \mathcal{A}: \tag{3.7}$$

$$\frac{\Lambda_e^{(a)}}{\Lambda^{(a)}} \frac{D_e^{(a)} n_e^{(a)}}{n_e^{(a)} - D_e^{(a)} \Lambda_e^{(a)}} + \frac{\Lambda_c^{(a)}}{\Lambda^{(a)}} \frac{D_c^{(a)} n_c^{(a)}}{n_c^{(a)} - D_c^{(a)} \Lambda_c^{(a)}} \leq \bar{\bar{R}}^{(a)} \tag{3.8}$$

$$D_e^{(a)} \Lambda_e^{(a)} < n_e^{(a)} \tag{3.9}$$

$$D_c^{(a)} \Lambda_c^{(a)} < n_c^{(a)} \tag{3.10}$$

$$\Lambda_e^{(a)} + \Lambda_c^{(a)} = \Lambda^{(a)}, \tag{3.11}$$

$$n_e^{(a)}, n_c^{(a)} \geq 0 \tag{3.12}$$

$$\Lambda_e^{(a)}, \Lambda_c^{(a)} \geq 0. \tag{3.13}$$

Thanks to the linear objective function and the convex constraints, the Karush-Kuhn-Tucker (KKT) optimality conditions can be exploited to obtain the optimal values of decision variable $n_e^{(a)}$ and $n_c^{(a)}$. However, the resulting system of KKT conditions is still too complex to be solved analytically, hence we propose a method that simplifies the problem even further.

We first estimate the resources needed in the scenarios in which only edge servers (OnlyEdge scenario) or only cloud VMs (OnlyCloud scenario) are available, and then decide which applications to assign completely to the edge, which to the cloud, and which one to divide between edge and cloud.
This method is based on the assumption of cheaper edge servers, meaning that it's in the platform interest to allocate the users' load in the edge resources as far as it is available, rather than using cloud VMs. The problem is that the available edge resources may not be enough to serve the load from all users of all applications, making it necessary resorting to cloud VMs. Therefore, our proposed approach consists in assigning one by one the load of every application to the edge, until the edge resources are saturated; then, the remaining applications, and fraction of one application load, that did not fit into the edge are going to be served in the cloud. Notice that the order, or priority, in which applications are first assigned to the edge is an important component of our approach, as choosing different orders will result in having different solutions. Since we don't know a priori which is the best order, we should test our method on all the possible permutations of the set of applications. Given the *order* specifying the priority of applications, the necessary steps for solving the relaxed problem 3.4 according to the proposed approach presented above, are outlined in algorithm 3.2.

We first compute the optimal number of edge resources ($n_e^{(a)}$) for all applications under OnlyEdge scenario given by Theorem 3.2 (line 2). If the maximum number of edge nodes available is enough to cover all applications, we simply assign all applications to the edge and the optimal solution is found. Otherwise, following the given order of applications

Algorithm 3.2 Heuristic solution to relaxed problem

1: **procedure** ComputeOptimalVMs(order)
2: $n_e^{(a)} \leftarrow$ Solution through Theorem 3.2
3: **if** $\sum_{a \in \mathcal{A}} n_e^{(a)} \leq N_e$ **then**
4: $\Lambda_e^{(a)} \leftarrow \Lambda^{(a)}, \quad \Lambda_c^{(a)} \leftarrow 0, \quad n_c^{(a)} \leftarrow 0 \quad \forall a \in \mathcal{A}$
5: **else**
6: $n_c^{(a)} \leftarrow$ Solution through Theorem 3.4
7: $UsedEdgeVMs \leftarrow \sum_{a \in \mathcal{A}} n_e^{(a)}$, $idx \leftarrow |\mathcal{A}|$
8: **while** $UsedEdgeVMs > N_e$ **do**
9: $idx \leftarrow idx - 1$, $\tilde{a} \leftarrow order[idx]$
10: $UsedEdgeVMs \leftarrow UsedEdgeVMs - n_e^{(\tilde{a})}$
11: $\Lambda_c^{(\tilde{a})} \leftarrow \Lambda^{(\tilde{a})}$, $n_e^{(\tilde{a})} \leftarrow 0$
12: **end while**
13: $\bar{N}_e \leftarrow N_e - \sum_{a \in \mathcal{A}} n_e^{(a)}$
14: $n_e^{(\tilde{a})}, \Lambda_e^{(\tilde{a})}, n_c^{(app)}, \Lambda_c^{(\tilde{a})} \leftarrow$ Solution through Theorem 3.3, given $r, x, \bar{N}_e$
15: **for** $j \leftarrow 1$ **to** $idx - 1$ **do**
16: $a \leftarrow order[j]$
17: $\Lambda_e^{(a)} \leftarrow \Lambda^{(a)}$, $n_c^{(a)} \leftarrow 0$
18: **end for**
19: **end if**
20: **end procedure**

($order \in Sym(\mathcal{A})$[1]), we assign $n_e^{(order_1)}, n_e^{(order_2)}, \ldots, n_e^{(order_m)}$ number of edge nodes to the corresponding applications until the remaining edge resources are not enough to cover the load of the m^{th} application (lines 3-12). At this point, we divide the load of the m^{th} application in two parts, one will be served in the edge using the remaining edge nodes, and the other one in the cloud. To compute the load fraction that will be served in the cloud, as well as the cloud VMs needed, we use the results of Theorem 3.3 (lines 13-14). Now that the edge resources are saturated, we assign all the remaining applications in the cloud, by allocating the necessary VMs, that are computed accordingly to Theorem 3.4 (lines 15-20).

In the following Subsections, we present the mathematical results that will be used to compute the number of edge and cloud as a closed form under OnlyEdge and OnlyCloud scenarios (Section 3.1.1 and Section 3.1.3), and how to assign the load of a single application partially to the edge and cloud due to the lack of edge nodes (Section 3.1.2).

3.1.1. Only Edge Scenario

In this section we compute the optimal number of servers for each application under the assumption of *only edge scenario*, meaning that all application loads are served with edge nodes; namely: $\Lambda^{(a)} = \Lambda_e^{(a)} \ \forall a \in \mathcal{A}$. With this additional assumption, problem 3.4

[1] $Sym(\mathcal{A})$ is the permutations set of set $\mathcal{A}$

becomes:

$$\min_{n_e^{(a)}} c_e n_e \tag{3.14}$$

$$\text{s.t.} \quad \sum_{a \in \mathcal{A}} n_e^{(a)} \leq N_e \tag{3.15}$$

$$\frac{D_e^{(a)} n_e^{(a)}}{n_e^{(a)} - D_e^{(a)} \Lambda^{(a)}} \leq \bar{\bar{R}}_a \qquad \forall a \in \mathcal{A}, \tag{3.16}$$

$$n_e^{(a)} - D_e^{(a)} \Lambda^{(a)} > 0 \qquad \forall a \in \mathcal{A} \tag{3.17}$$

Theorem 3.2. *If*

$$\sum_{a \in \mathcal{A}} \frac{\bar{\bar{R}}^{(a)} D_e^{(a)} \Lambda^{(a)}}{\bar{\bar{R}}^{(a)} - D_e^{(a)}} \leq N_e, \tag{3.18}$$

the solution of problem 3.14 is:

$$n_e^{(a)} = \frac{\bar{\bar{R}}_a D_e^{(a)} \Lambda^{(a)}}{\bar{\bar{R}}^{(a)} - D_e^{(a)}} \quad \forall a \in \mathcal{A}$$

Proof. The proof is given in Appendix A.2. □

3.1.2. Single Application Scenario

In this section we cover the problem of finding the optimal resource allocation in the case of a single application. Let us consider the simplified edge platform problem 3.4 in the case with only one application. For simplicity, we omit superscript a:

Theorem 3.3. *If the total load satisfies the following condition*

$$\Lambda \leq \frac{N_e(\bar{\bar{R}} - D_e)}{\bar{\bar{R}} D_e},$$

then the edge platform does not use cloud resources and the optimal number of edge servers is

$$n_e = \frac{\bar{\bar{R}} D_e \Lambda}{\bar{\bar{R}} - D_e}.$$

Otherwise, if

$$\Lambda > \frac{N_e(\bar{\bar{R}} - D_e)}{\bar{\bar{R}} D_e},$$

then edge resources are saturated ($n_e = N_e$), the optimal edge load is

$$\Lambda_e = \frac{N_e \Lambda (\bar{\bar{R}} - \sqrt{D_c D_e})}{N_e D_e + \bar{\bar{R}} \Lambda D_e - N_e \sqrt{D_c D_e}}$$

and the optimal number of cloud resources is

$$n_c = \frac{D_c \Lambda [\bar{\bar{R}} D_e \Lambda - N_e (\bar{\bar{R}} - D_e)]}{N_e (\sqrt{D_e} - \sqrt{D_c})^2 + D_e \Lambda (\bar{\bar{R}} - D_c)}.$$

Proof. The proof is given in Appendix A.3. □

3.1.3. Only Cloud Scenario

In this section we compute the optimal number of servers for each application under the assumption of *only cloud scenario*, meaning that all application loads are served with cloud VMs; namely: $\Lambda^{(a)} = \Lambda_c^{(a)}\ \forall a \in \mathcal{A}$. With this additional assumption, problem 3.4 becomes:

$$\min_{n_c^{(a)}} c_c n_c \tag{3.19}$$

$$\text{s.t.} \quad \frac{D_c^{(a)} n_c^{(a)}}{n_c^{(a)} - D_c^{(a)} \Lambda^{(a)}} \leq \bar{\bar{R}}_a \tag{3.20}$$

$$n_c^{(a)} - D_c^{(a)} \Lambda^{(a)} > 0 \qquad \forall a \in \mathcal{A} \tag{3.21}$$

The solution is given by:

Theorem 3.4. *The optimal cloud resources are*

$$n_c^{(a)} = \frac{\bar{\bar{R}}_a D_c^{(a)} \Lambda^{(a)}}{\bar{\bar{R}}_a - D_c^{(a)}} \quad \forall a \in \mathcal{A}$$

Proof. The proof is given in Appendix A.4. □

3.2. Optimal price

In the previous section we had the fundamental assumption of fixed prices for the deployments. This is quite important as it meant that users' decisions and, consequentially, the computational load of all applications were also fixed. This allowed us to formulate a relaxed problem that we solved using a method based on KKT conditions and permutations. In reality, the extra cost for the second deployment belongs to the set of the platform decision variables, meaning that the prices of deployments, as well as the computational load of applications are not fixed parameters. However, in this section we prove that the optimal price belongs to a finite set of special points, meaning it can be found by inspection: by applying our heuristic approach to every point in the set, the platform will then choose the price that ultimately leads to the solution with the greatest profit.

To describe the behavior of the platform profit, the profit function related to 10 users, 1 application scenario is shown in Figure 3.2. This plot shows that, by varying the price, the platform profits exhibits points of discontinuity, which are located when a small change in the price forces users to drop from the system or change their deployment decisions. Hence, two types of points of discontinuity are defined:

- **Dropping points:** The values for the extra cost that makes the user's net gain of running second deployment equal to zero. To obtain these points, for each user i,

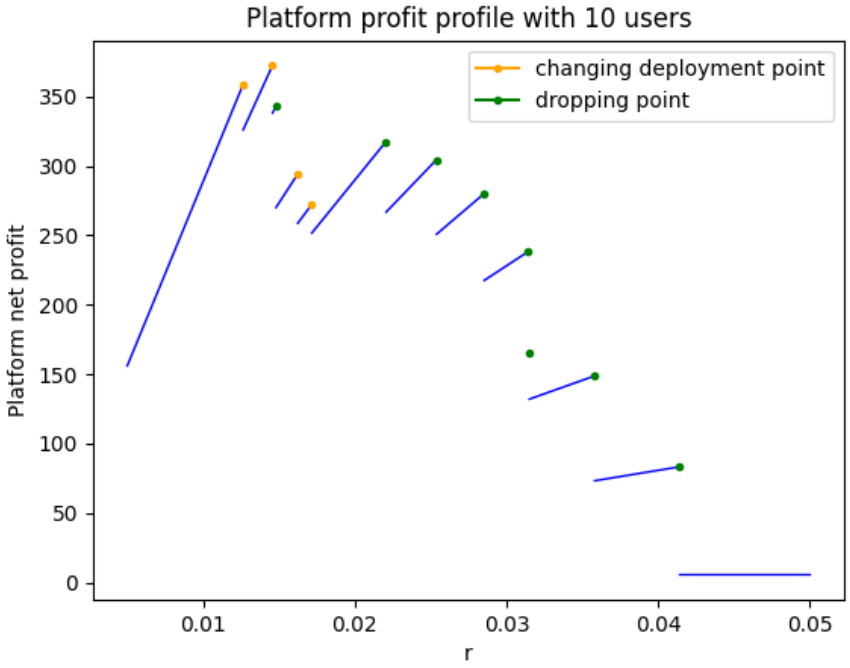


Figure 3.2: Platform net profit function with 10 users.

the cost of running the second deployment on his/her device is calculated:

$$\forall i \in \mathcal{U}^{(a)},\ \forall a \in \mathcal{A}:$$
$$C_i^{(2)} = T_i \left(\alpha_i r^{(a2)} + (1-\alpha_i)\beta_i p_i^{(2)} \lambda^{(a)} T_i + \zeta_i \delta^{(a)} \lambda^{(a)} \right).$$

If the cost above exceeds user's utility U_i, he/she does not want to run the second deployment, meaning that the platform won't receive the revenue $r^{(a2)}T_i$ from that user. We call *dropping points* the values of r that cause users to choose to not participate in the system (by running the second deployment). More specifically, we denote with drop_i the value for the extra-cost r that makes the profit of user i for running the second deployment of his/her application equal to zero. Therefore:

$$\forall i \in \mathcal{U}^{(a)},\ \forall a \in \mathcal{A}:$$
$$\text{drop}_i = \frac{U_i - T_i[\alpha_i r^{(a1)} + (1-\alpha_i)\beta_i p_i^{(2)} \lambda^{(a)} T_i + \zeta_i \delta^{(a)} \lambda^{(a)}]}{T_i \alpha_i \gamma^{(a)}}$$

- **Changing deployment points:** These points specify the prices that make the benefit of running the first and second deployments equal. Since the utility is the same for both deployments, the user's profits are equal when the costs are the same. We denote with change_i the extra-cost r that makes the cost (hence the profit) of user-i of running deployment 1 and 2 of his/her application equal. Therefore:

$$\forall i \in \mathcal{U}^{(a)},\ \forall a \in \mathcal{A}:$$
$$\text{change}_i = \frac{(1-\alpha_i)\beta_i \lambda^{(a)} T_i (p_i^{(1)} - p_i^{(2)}) - \zeta_i \lambda^{(a)} \delta^{(a)}}{\alpha_i \gamma^{(a)}}.$$

We are then able to state the core theorem for our solution as follows:

Theorem 3.5. *The optimal price solution lays at one of the points of discontinuity.*

Proof. Assuming the users' decisions $x_i^{(k)}$ are fixed, the only part of the platform profit function that changes with increasing r is $r^{(k)}x_i^{(k)}$, which causes an increase of the platform profit. Therefore, in each interval in which users' decisions are fixed, the maximum point is at the end of the interval. A discontinuity in the platform profit function happens when the extra cost r reaches the point where one user either wants to change deployment (from 2° to 1°) or stop participating in the system. Therefore, the discontinuity points of the platform profit function coincide with the changing deployment points and dropping points that we defined previously. Moreover, as it can be seen in figure 3.2, both these cases cause a drop in the total profit of the platform; this is to be expected since serving users should be a profitable business for the platform.
We can conclude that the optimal value for the extra cost of second deployment must coincide with one of the discontinuity points of the platform profit function. Hence, the set of r-values candidates is determined by:

$$\mathcal{R} = \{\text{change}_i,\ \text{drop}_i,\ \forall i \in \mathcal{U}\} \tag{3.22}$$

□

At this point we know that the optimal price r is within a finite set of values given by Theorem 3.5. In order to identify the optimal price r we follow the approach outlined in algorithm 3.3. Namely, for each value of $r \in \mathcal{R}$, we solve each user's problem through algorithm 3.1; once the users' decision are known, we compute the load for each application according to 1.9 (lines 8-12), meaning that the relaxed problem is now well defined. Next, for each possible ordering of the applications we solve the relaxed problem, relative to the current value of r, through algorithm 3.2 and we append the corresponding solution to a previously initialized list (lines 14-20). Finally, we sort the solution-list by profit and we return the 10 elements with the highest profit (lines 21-22).
The reason why we do not consider just the partial solution with the highest profit and its corresponding r, is that it's not guaranteed to be optimal in the sense of the complete problem 2.1. Therefore, by selecting more partial-solutions, we have an higher chance that one of them will approximate better the optimal solution of the complete problem.

3.3. Users' Resource Assignment

At this point we have a set of *partial solutions* that were found by solving the Relaxed problem of section 3.1. These solutions however, are not admissible solutions in the sense of the complete Stackelberg problem 2.1. Indeed, they lack the specific users' assignment variables y_i^e, y_i^c and the amount of resources allocated was computed considering simplified time-response constraints and neglecting the integrality of $n_e^{(a)}$ and $n_c^{(a)}$.
Our purpose is now to extend these partial solutions so that they will become admissible solutions for the original platform optimization problem. In order to do so, we have to assign each user that chose 2° deployment either to edge or cloud, namely, fixing y_i^e and

Algorithm 3.3 Optimal prices algorithm

Input: N, users' parameters, $r_{\min}$, $r_{\max}$, $\gamma^{(a)}$, $\delta^{(a)}$, $\lambda^{(a)}$, $r^{(a1)}$ $\forall a \in \mathcal{A}$, ordering
Initialization: *Solutions* $\leftarrow$ []
for $i \leftarrow 1$ **to** N **do**
    $\text{change}_i = \frac{(1-\alpha_i)\beta_i\lambda^{(a)}T_i(p_i^{(1)}-p_i^{(2)})-\zeta_i\lambda^{(a)}\delta^{(a)}}{\alpha_i\gamma^{(a)}}$.
    $\text{drop}_i = \frac{U_i-T_i[\alpha_i r^{(a1)}+(1-\alpha_i)\beta_i p_i^{(2)}\lambda^{(a)}T_i+\zeta_i\delta^{(a)}\lambda^{(a)}]}{T_i\alpha_i\gamma^{(a)}}$
    Add drop_i, change_i to $\mathcal{R}$
end for
for r **in** $\mathcal{R}$ **do**
    if $r_{min} \leq r \leq r_{max}$ **then**
        $r^{(a2)} \leftarrow r^{(a1)} + \gamma^{(a2)}r \quad \forall a \in \mathcal{A}$
        $x_i \leftarrow$ Solve user-i problem given r, $\forall i \in \mathcal{U}$
        $\Lambda^{(a)} \leftarrow \sum_{i \in \mathcal{U}^{(a)}} \lambda^{(a)} x_i^{(2)} \quad \forall a \in \mathcal{A}$
        orders $\leftarrow$ Permutations of applications
        for order **in** orders **do**
            ComputeOptimalVMs(order)
            $P \leftarrow$ Compute platform profit given $r, x, n_e^{(a)}, n_c^{(a)}, \Lambda_e^{(a)}, \Lambda_c^{(a)}$
            append $(P, r, x, n_e^{(a)}, n_c^{(a)}, \Lambda_e^{(a)}, \Lambda_c^{(a)}, order)$ to *Solutions*
        end for
    end if
end for
sort *Solutions* by P decreasingly
return *EliteSolutions* = first n elements of *Solutions*

y_i^c. Moreover, we have to modify the number of edge and cloud resources assigned to each application, so that all the original time-response and integrality constraints are satisfied. Once we have a set of admissible solutions the platform will choose the one with the the greatest profit. This solution will be the final output of the heuristic approach.
The process of extending the partial solutions is rather complex, as such, it is exposed directly in algorithm 3.4, where we use the following notation:

- We define LD_i as the maximum delay that user i can tolerate to avoid response time violation:

$$LD_i = \bar{R}^{(a)} - D_i^{(2)} x_i^{(2)} - \frac{\delta^{(a)} x_i^{(a)}}{B_i} \qquad \forall i \in \mathcal{U}^{(a)},\ \forall a \in \mathcal{A}. \tag{3.23}$$

- $\text{List}_e^{(a)}$, $\text{List}_c^{(a)}$ are the lists of users running application a assigned to the edge and cloud, respectively, and sorted by LD_i.
- $\mathcal{A}_e$, $\mathcal{A}_c$ are the sets of applications whose load is served on edge and cloud, respectively.

Algorithm 3.4 Users' resource assignment

Input: *EliteSols*, $\bar{R}^{(a)}$, $D_i^{(k)}$
Initialization: *BestProfit* $\leftarrow 0$, *BestSol* $\leftarrow 0$, y_i^e, $y_i^c \leftarrow 0$
$\mathcal{U}_2^{(a)} = \{i \in \mathcal{U}^{(a)},\ x_i^{(2)} = 1\ \}$
for all *Sol* **in** *EliteSols* **do**
    r, x, $n_e^{(a)}$, $n_c^{(a)}$, $\Lambda_e^{(a)}$, $\Lambda_c^{(a)}$, $order \leftarrow Sol$
    for $a \in \mathcal{A}$ **do**
        $\bar{n}_e^{(a)} \leftarrow$ CountEdgeVMs(a)
    end for
    if $\sum_{a \in \mathcal{A}} \bar{n}_e^{(a)} > N_e$ **then**
        $idx \leftarrow 0$, $UsedVMs \leftarrow 0$
        while $UsedVMs \leq N_e$ **do**
            $idx \leftarrow idx + 1$, $\tilde{a} \leftarrow order[idx]$
            $UsedVMs \leftarrow UsedVMs + \bar{n}_e^{(\tilde{a})}$
        end while
        $\bar{N}_e \leftarrow N_e - \sum_{j=1}^{idx-1} \bar{n}_e^{(order[j])}$
        while $\bar{n}_e^{(\tilde{a})} > \bar{N}_e$ **do**
            Assign user $i \in \mathcal{U}_2^{(\tilde{a})}$ with highest LD to cloud $(y_i^e \leftarrow 0, y_i^c \leftarrow 1)$
            MoveUser$(i, \tilde{a})$
        end while
        for $j \leftarrow idx + 1$ **to** $|\mathcal{A}|$ **do**
            $a \leftarrow order[j]$
            MoveApplication(a)
        end for
    end if
    for $a \in \mathcal{A}$ **do**
        ComputeCloudVMs(a)
    end for
    $P \leftarrow$ Compute platform profit given $\bar{n}_e^{(a)}, \bar{n}_c^{(a)}$
    if $P > BestProfit$ **then**
        BestProfit $\leftarrow P$
        BestSol $\leftarrow (P, r, \bar{n}_e^{(a)}, \bar{n}_c^{(a)}, y^e, y^c)$
    end if
end for
return *BestSol*

procedure MoveUser(i, a)
  $L_c^{(a)} \leftarrow L_c^{(a)} + D_c^{(a)}\lambda^{(a)}$, $\Lambda_c^{(a)} \leftarrow \Lambda_c^{(a)} + \lambda^{(a)}$
  $L_e^{(a)} \leftarrow L_e^{(a)} - D_e^{(a)}\lambda^{(a)}$, $\Lambda_e^{(a)} \leftarrow \Lambda_e^{(a)} - \lambda^{(a)}$
  $j \leftarrow \text{List}_e^{(a)}[0]$
  $\bar{n}_e^{(a)} \leftarrow \lceil L_e^{(a)} \frac{LD_j}{LD_j - D_e^{(a)}} \rceil$
end procedure

procedure MoveApplication(a)
  $y_i^e \leftarrow 0, y_i^c \leftarrow 1, \forall i \in \mathcal{U}_2^{(a)}$
  $L_c^{(a)} \leftarrow L_c^{(a)} + D_c^{(a)}\Lambda_e^{(a)}$, $\Lambda_c^{(a)} \leftarrow \Lambda_c^{(a)} + \Lambda_e^{(a)}$
  $L_e^{(a)} \leftarrow 0$, $\Lambda_e^{(a)} \leftarrow 0$
  $\bar{n}_e^{(a)} \leftarrow 0$
end procedure

procedure CountEdgeVMs(a)
  if $n_c^{(a)} == 0$ **then**
    $y_i^e \leftarrow 1, \quad \forall i \in \mathcal{U}_2^{(a)}$
    $V[i] = D_i^{(2)} + \frac{\delta^{(a)}}{B_i} + \frac{D_e^{(2)}}{1 - L_e^{(a)}/n_e^{(a)}} - \bar{R}^{(a)}, \quad \forall i \in \mathcal{U}_2^{(a)}$
    if $V[i] \leq 0, \forall i \in \mathcal{U}_2^{(a)}$ **then**
      $\bar{n}_e^{(a)} \leftarrow \lceil n_e^{(a)} \rceil$, $\bar{n}_c^{(a)} \leftarrow \lceil n_c^{(a)} \rceil$
    else
      $j \leftarrow$ user in $\mathcal{U}_2^{(a)}$ with maximum V
      $\bar{n}_e^{(a)} \leftarrow \lceil L_e^{(a)} \frac{LD_j}{LD_j - D_e^{(a)}} \rceil$
    end if
  end if
  if $n_c^{(a)} > 0$ **then**
    $V[i] = D_i^{(a)} + \frac{\delta^{(a)}}{B_i}, \forall i \in \mathcal{U}_a^{(2)}$
    sort the user by $LD, i \in \mathcal{U}_2^{(a)}$ increasingly
    $j \leftarrow \frac{\Lambda_c^{(a)}}{\lambda^a} + 1$
    $y_i^c \leftarrow 1, \qquad \forall i \in \mathcal{U}_2^{(a)} : i < j$
    $y_i^e \leftarrow 1, \qquad \forall i \in \mathcal{U}_2^{(a)} : i \geq j$
    $\bar{n}_e^{(a)} \leftarrow \lceil L_e^{(a)} \frac{LD_j}{LD_j - D_e^{(a)}} \rceil$
  end if
end procedure

procedure ComputeCloudVMs(a)
  if $\Lambda_c^{(a)} > 0$ **then**
    $j \leftarrow \text{List}_c^{(a)}[0]$
    $\bar{n}_c^{(a)} \leftarrow \lceil L_c^{(a)} \frac{LD_j}{LD_j - D_c^{(a)}} \rceil$
  end if
end procedure

3.4. Tabu Search

After applying the proposed heuristic approach we have an approximate optimal solution based on the resource allocation of the relaxed problem 3.4. We recall that the relaxed problem considers a convex and simplified time-response constraint based on the average delay of users. A more *accurate* solution could have been found if the real delay of users had been considered. Moreover, we imposed the load of some application to be entirely served on edge or cloud, therefore excluding those solutions where the load of multiple applications was split between edge and cloud.
Hence, we developed a Tabu Search with the aim of improving our current optimal solution by exploring the solution space not covered by our heuristic approach. Some features of this algorithm are allowing worsening of the solutions and keeping memory of the previous-visited states to influence future search.
The detailed explanation of the tabu search algorithm we implemented can be found in the following sections.

Algorithm

The first step of *Tabu Search* is to generate some feasible neighbors (elements of the solutions space) starting from an initial solution. In order to so, we change the current edge and cloud resource allocation of some applications, forcing the platform to modify the applications loads by moving users from edge to cloud or viceversa. Then, we select one of the neighbors belonging to the solutions space not yet visited, and we set it as new initial solution. The corresponding platform profit is computed and if exceeds the current best profit, the optimal solutions is updated. Then, starting from this new initial solution we repeat the process, until one stop criteria is met.
In sections 3.4.1 and 3.4.2 we provide a detailed explanation about the process of generation and selection of neighbors.

3.4.1. Generating the Neighbors

Now we illustrates how to generate a list of neighbors starting from an initial, feasible solution. Specifically, since our goal is to reduce the number of cloud VMs and consequently improve the cost, we define a move for swapping users from cloud to edge. Algorithm 3.5 presents this move. It receives the initial solution which includes all parameters and variables of the problem solved by our proposed heuristic approach (line 1). Then it picks application a with the lowest load per VMs ratio, to switch off one cloud VM allocated to a (line 2), and compute the number of users that are running application a on cloud (line 3). The rational behind selecting the application with the lowest load per VMs ratio is that we have some hope that the cloud load that is now in excess can be served by the active nodes in the edge. In order to do that, we compute the number of users with highest local delay to be moved in the edge, check if they can be served by the active nodes or if we need to allocate more edge resources, all while guaranteeing the response time constraint of users with lowest local delay is satisfied (lines 4-18). If the extra load is small enough that no extra nodes are needed or the number of idle edge nodes are enough to serve the extra load, no more adjustments need to me made, and the new neighbor is

generated (lines 17-23). On the contrary, if the edge nodes are not enough to serve the extra load, we must free edge resources from other applications, by moving users from edge to cloud.
We present this move in Algorithm 3.6. Note that based on which application(s) is (are) selected for freeing edge resources, we have different neighbors. Algorithm 3.6 receive the solution related to current assignment and the required number of edge servers to be released ($\text{diff\_}n_e^{(a)}$). For each application $a' \in \mathcal{A}_e$, the algorithm checks whether the edge nodes allocated are enough to cover $\text{diff\_}n_e$. If this is the case, the algorithm computes how many users must be moved to cloud in order to release the required amount of edge nodes; then the cloud VMs of application a' are updated given the load of the new users. If instead there are not enough edge nodes, application a' is moved entirely in the cloud, the released edge servers are used to lower the require amount of edge nodes: $\text{diff\_}n_e = \text{diff\_}n_e - n_e^{(a')}$; and a new application $a'' \in \mathcal{A}_e$ is considered.
In the end, the list of generated neighbors is returned.

Algorithm 3.5 Move cloud to edge

Input: $n_c^{(a)}, n_e^{(a)}, y^e, y^c, x_i^{(a)}, LD_i^{(a)}, D_e^{(a)}, D_c^{(a)}, L_e^{(a)}, L_c^{(a)}$ $List_e^{(a)}$, $List_c^{(a)}$
Pick $a \in \mathcal{A}_c$ with lowest ratio $\frac{L_c^{(a)}}{n_c^{(a)}}$
$N_c^{(a)} = \sum_{i \in \mathcal{U}^{(a)}} y_i^c$
$n_c'^{(a)} = n_c^{(a)} - 1$ ▷ Switch off one VM
$j \leftarrow List_c^{(a)}[0]$
$\bar{N}_c^{(a)} \leftarrow \lfloor \frac{\left(LD_j^{(a)} - D_c^{(a)}\right) n_c'^{(a)}}{\lambda^{(a)} LD_j^{(a)} D_c^{(a)}} \rfloor$
$\bar{\bar{N}}_c^{(a)} \leftarrow N_c^{(a)} - \bar{N}_c^{(a)}$
Select $\bar{\bar{N}}_c^{(a)}$ users with lowest local delay in $List_c^{(a)}$ to move to edge and update $List_e^{(a)}, List_c^{(a)}, y^c, y^e$
$L_e'^{(a)} \leftarrow L_e^{(a)} + \bar{\bar{N}}_c^{(a)} \lambda^{(a)}$
$i \leftarrow List_e^{(a)}[0]$
$n_e'^{(a)} \leftarrow \lceil L_e'^{(a)} \frac{LD_i^{(a)}}{LD_i^{(a)} - D_e^{(a)}} \rceil$
$\text{diff\_}n_e^{(a)} \leftarrow n_e'^{(a)} - n_e^{(a)}$
Let Idle_e be the number of idle nodes in the edge
if $\text{diff\_}n_e^{(a)} \leq 0$ **then**
    Move $\bar{\bar{N}}_c^{(a)}$ users to the current active nodes in edge
    $neighbor \leftarrow (n_e'^{(a)}, n_c'^{(a)}, y^e, y^c)$
else if $\text{Idle}_e \geq \text{diff\_}n_e^{(a)}$ **then**
    Switch on $\text{diff\_}n_e^{(a)}$ nodes in edge and allocate $\bar{\bar{N}}_c^{(a)}$ users to them
    $\text{Idle}_e = \text{Idle}_e - \text{diff\_}n_e^{(a)}$
    $neighbor \leftarrow (n_e'^{(a)}, n_c'^{(a)}, y^e, y^c)$
else
    $\text{diff\_}n_e^{(a)} = \text{diff\_}n_e^{(a)} - \text{Idle}_e$
    $\text{Idle}_e = 0$
    $neighbor \leftarrow MoveEdgeToCloud(n_e'^{(a)}, n_c'^{(a)}, y^e, y^c, \text{diff\_}n_e^{(a)})$
end if
return *neighbor*

Algorithm 3.6 MoveEdgeToCloud

Input: $n_c^{(a)}, n_e^{(a)}, y^e, y^c, x_i^{(a)}, LD_i^{(a)}, D_e^{(a)}, D_c^{(a)}, L_e^{(a)}, L_c^{(a)}\ List_e^{(a)}, List_c^{(a)}, \text{diff}\_n_e^{(a)}$
Initialization: $neighbor\_list \leftarrow []$
for all $a' \in \mathcal{A}_e$ **and** $a' \neq a$ **do**
  $N_e^{(a')} = \sum_{i \in \mathcal{U}^{(a')}} y_i^e$
  if $n_e^{(a')} \geq \text{diff}\_n_e^{(a)}$ **then**
    $j \leftarrow List_e^{(a')}[0]$
    $\bar{N}_e^{(a')} \leftarrow \lfloor \frac{\left(LD_j^{(a')} - D_e^{(a')}\right)\left(n_e^{(a')} - diff\_n_e^{(a')}\right)}{\lambda^{(a')} LD_j^{(a')} D_e^{(a')}} \rfloor$
    $\bar{\bar{N}}_e^{(a')} \leftarrow N_e^{(a')} - \bar{N}_e^{(a')}$
    Select $\bar{\bar{N}}_e^{(a')}$ users with highest LD in $List_e^{(a')}$ to move to cloud
    Update $List_e^{(a')}, List_c^{(a')}, y^c, y^e$
    $L_c'^{(a')} \leftarrow L_c^{(a')} + \bar{\bar{N}}_e^{(a')} \lambda^{(a')}$
    $i \leftarrow List_c^{(a')}[0]$
    $n_c^{(a')} \leftarrow \lceil L_c'^{(a')} \frac{LD_i^{(a')}}{LD_i^{(a')} - D_c^{(a')}} \rceil$
    $neighbor\_list \leftarrow neighbor\_list \cup (n_e, n_c, y^e, y^c)$
  else
    Move all users running a' to cloud
    $L_c^{(a')} \leftarrow L_c^{(a')} + N_e^{(a')} \lambda^{(a')}$
    $i \leftarrow List_c^{(a')}[0]$
    $n_c^{(a')} \leftarrow \lceil L_c^{(a')} \frac{LD_i^{(a')}}{LD_i^{(a')} - D_c^{(a')}} \rceil$
    Select $a'' \in \mathcal{A}_e$ randomly such that $n_e^{(a'')} \geq diff\_n_e^{(a)} - n_e^{(a')}$
    $N_e^{(a'')} = \sum_{i \in \mathcal{U}^{(a'')}} y_i^e$
    $j \leftarrow List_e^{(a'')}[0]$
    $\bar{N}_e^{(a'')} \leftarrow \lfloor \frac{\left(LD_j^{(a'')} - D_e^{(a'')}\right)\left(n_e^{(a'')} - diff\_n_e^{(a)} + n_e^{(a')}\right)}{\lambda^{(a'')} LD_j^{(a'')} D_e^{(a'')}} \rfloor$
    $\bar{\bar{N}}_e^{(a')} \leftarrow N_e^{(a')} - \bar{N}_e^{(a')}$
    Select $\bar{\bar{N}}_e^{(a'')}$ users with lowest local delay in $List_e^{(a'')}$ to move to cloud
    Update $List_e^{(a'')}, List_c^{(a'')}, y^c, y^e$
    $L_c^{(a'')} \leftarrow L_c^{(a'')} + \bar{\bar{N}}_e^{(a'')} \lambda^{(a'')}$
    $i \leftarrow List_c^{(a'')}[0]$
    $n_c^{(a'')} \leftarrow \lceil L_c^{(a'')} \frac{LD_i^{(a'')}}{LD_i^{(a'')} - D_c^{(a'')}} \rceil$
    $neighbor\_list \leftarrow neighbor\_list \cup (n_e, n_c, y^e, y^c)$
  end if
end for
return $neighbor\_list$

3.4.2. Tabu Search Algorithm

Algorithm 3.7 shows the implementation of Tabu search. The Inputs are $InitialSolution$ that is the best solution found by our heuristic approach, $TabuSize$ as the memory capacity and $MaxIter$ that is the maximum number of iterations performed by the algorithm. At the start, current solution and best solution are set equal to the initial solution. The tabu list is initialized as an empty list, and will be used to store the solutions visited by the algorithm. To make the algorithm more efficient, not all attributes of a solution object will be stored in the tabu list, but rather only partial information about the resource allocation and profit. The algorithm starts by generating all feasible neighbors of current solution ($N(CurrentSolution)$) by applying the procedures mentioned in subsection 3.4.1. If all neighbors are elements of the Tabu List, the algorithm stops and returns the best solution found (lines 6-8). Otherwise, it selects one of the neighbors according to the specified method, but it uses a tabu list to avoid falling back into a local optimum from which it previously emerged (lines 9-13). We implemented two methods for selection among neighbors:

- RANDOM: Randomly selects one of the neighbors,
- BEST PROFIT: Picks the neighbor with the highest profit.

Then, the selected neighbor becomes the current solution and it is added to the Tabu List; if Tabu List reaches its max capacity, the first element is deleted (lines 14-18). If the profit of the new current solution is greater than the one of the best solution, the best solutions is updated (lines 19-21). Finally, a new iteration starts.

Algorithm 3.7 Tabu Search Algorithm

Input: $InitialSolution$, $TabuSize$, $MaxIter$
Initialization: $BestSolution, CurrentSolution \leftarrow InitialSolution$, $TabuList \leftarrow \emptyset$
for $i \leftarrow 1$**to**$MaxIter$ **do**
    $Neighbors \leftarrow$ Generate all feasible neighbors of $CurrentSolution$ (Section 3.4.1)
    while True **do**
        if all $Neighbors \in TabuList$ **then**
            return $BestSolution$
        end if
        $neighbor \leftarrow SelectNeighbor(Neighbors, method)$
        if $neighbor \in TabuList$ **then**
            Remove $neighbor$ from $Neighbors$
            $neighbor \leftarrow SelectNeighbor(Neighbors, method)$
        else
            append $neighbor$ to $TabuList$
            if Len($TabuList$) $> TabuSize$ **then**
                Remove first item from $TabuList$
            end if
            $CurrentSolution \leftarrow neighbor$
            if $Profit(CurrentSolution) > Profit(BestSolution)$ **then**
                $BestSolution \leftarrow CurrentSolution$
            end if
            Break
        end if
    end while
end for
return $BestSolution$

4 | Implementation

The C++ implementation of our heuristic approach was developed and written entirely from scratch. The code is structured around 6 main classes, written with the Standard Library:

- class `User`, presented in section 4.1;
- class `Application`, presented in section 4.2;
- class `ResourceDistribution`, presented in section 4.3;
- class `PartialSolution`, presented in section 4.4;
- class `Solution`, presented in section 4.5.
- class `Platform`, presented in section 4.6;

Table 4.1 provides an overview of the class structure, highlighting the main features and inheritance dependencies.

4.1. User Class

Every instance of this class represents a user participating in the system. Therefore, members of this class are the many different users' parameters presented so far. Moreover, each user is identified by an ID (type `unsigned`), and has a member of type `shared_ptr<Application>` pointing to the application selected by the user, to identify his/her application.
The fundamental method of this class is:
`std::array<bool,2> algorithm1(double r);` .
It takes as inputs the extra cost charged by the platform and solves the user's optimal decision problem presented by 3.1; the return value is a two dimensional array of type `bool`, whose elements are true if and only if the user is choosing the corresponding deployment.

4.2. Application Class

This class is very simple, besides the different parameters introduced in our model $(D_e, D_c, \lambda, \delta, m^{(k)}, \gamma, r^{(1)}, \bar{R})$, it contains a member of type `string` that represents the application name.

4.3. ResourceDistribution Class

As the name suggests, this class is used to represent all the possible edge and cloud resource allocations made by the platform. It has 4 container members, mapping each application by their name to the corresponding value:

- `map<string,double> edgeLoads` : maps each application to the load that the platform assigned to the edge.
- `map<string,double> cloudLoads` : maps each application to the load that the platform assigned to the cloud.
- `map<string,T> edgeServers` : maps each application to the number of edge servers that the platform allocated for it.
- `map<string,T> cloudServers` : maps each application to the number of cloud VMs that the platform allocated for it.

The value type of `edgeServers` and `cloudServers` is a template parameter. This is because this class, and its derivatives, are used both in Algorithm 3.3, where the integrality assumption of the number edge and cloud resources is relaxed (type `double`), and Algorithm 3.4, where the original constraints are satisfied (type `int`).

4.4. PartialSolution Class

This class inherits from `ResourceDistribution`, and for the same reason it is still a class template. In addition to the containers of its parent class, it has the following members:

- `map<unsigned,bool> userFirstDeployment` : maps each user by his/her ID to a boolean value which is true if the user chose the 1° deployment.
- `map<unsigned,bool> userSecondDeployment` : maps each user by his/her ID to a boolean value which is true if the user chose the 2° deployment.
- `vector<string> order` : defines the order in which applications are first allocated to edge nodes.
- `double r` : extra cost for the 2° deployment charged by the platform.
- `double profit` : total net profit for the platform.

This class basically represents the solution to the relaxed problem, presented in section 3.1.

4.5. Solution Class

This class is used to create instances of the solution of the complete Stackelberg game. It inherits from `PartialSolution<int>`, and contains the following additional members:

- `map<unsigned,bool> userEdgeAllocation` : maps each user by his/her ID to a

boolean value which is true if he/she has been assigned by the platform to the edge.

- `map<unsigned,bool> userCloudAllocation`: maps each user by his/her ID to a boolean value which is true if he/she has been assigned by the platform to the cloud.
- `map<string,double> L_e` : maps each application to its edge load per time of execution.
- `map<string,double> L_c` : maps each application to its cloud load per time of execution.

Moreover, for this specif class we defined two comparison operators, one that defines an order relation based only on profit, and one that takes into account more attributes, and it's used in the Tabu Search to distinguish different solutions.

4.6. Platform Class

This class is definitely the most complex one. Besides containing trivial information like costs and availability of edge and cloud resources, it has the fundamental containers of users participating in the systems and applications that the platform is providing. Considering that our heuristic approach performs many operations that require specific user and app assignment and access, I ultimately decided to use the following containers for storing users and applications information:

- `map<unsigned,User> users` : each user is mapped by his/her ID.
- `map<string,shared_ptr<Application» apps` : each application is mapped by its name to a smart pointer pointing to the same instance pointed by the member of User objects.
- `map<string,vector<unsigned» userMap`: each application is mapped to the vector of its users' ID. In this way the platform has a fast and easy way of accessing the users of a specific application.

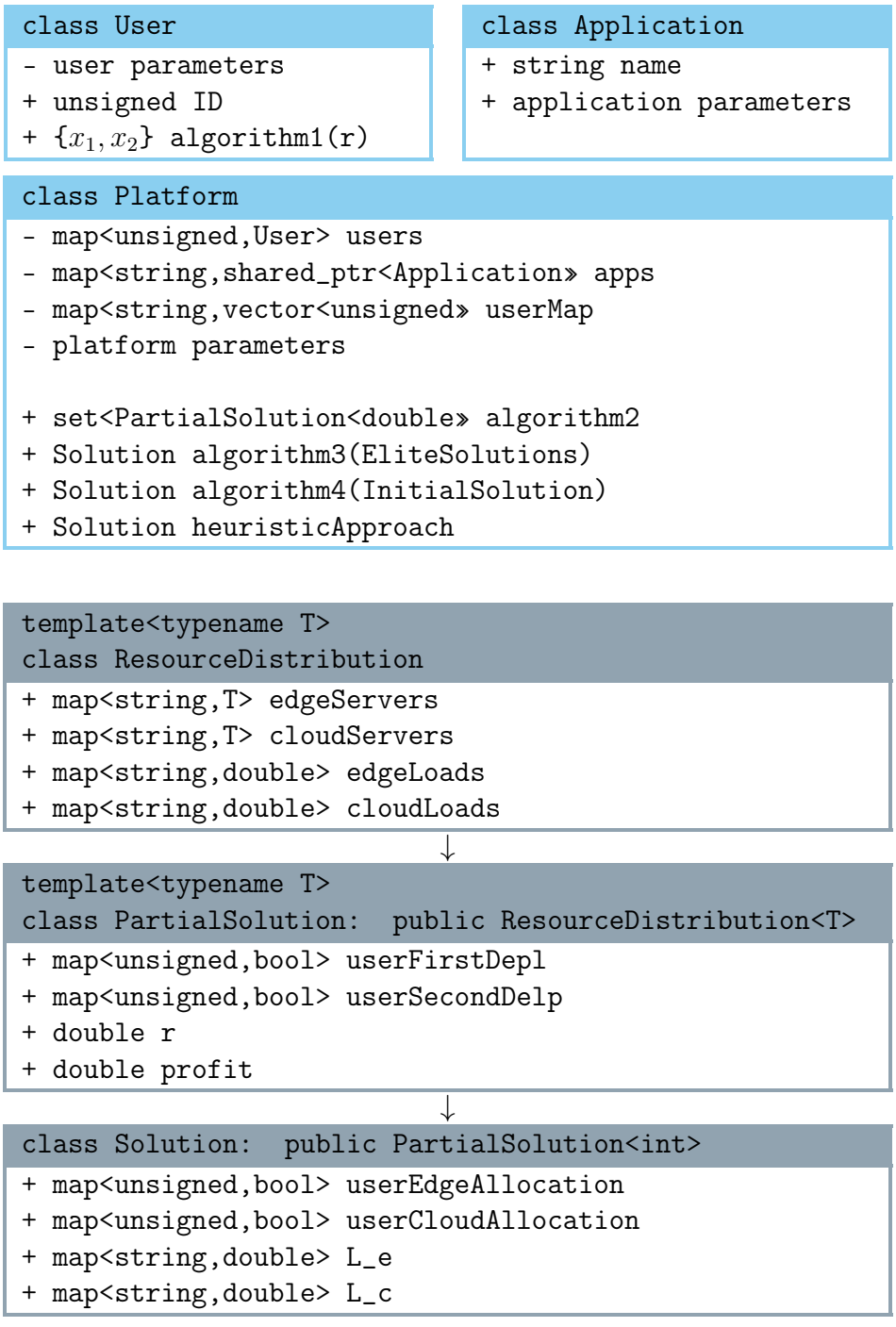
Moreover, all the relevant algorithms and procedures of our heuristic approach are implemented as methods of the class. Below we list some of the main ones[1]:

- `algorithm2` : implementation of Optimal Price Algorithm 3.3.
- `ComputeOptimalVMs`: implementation of Procedure 3.2.
- `algorithm3` : implementation of User's Resource Assignment Algorithm 3.4.
- `CountEdgeVMs`: implementation of procedure 15.
- `algorithm4`: implementation of Tabu Seach Algorithm 3.7
- `MoveCloudToEdge` : implementation of Algorithm 3.5.
- `MoveEdgeToCloud`: implementation of Algorithm 3.6

[1] full documentation can be found in file `Platform.cpp`

- `heuristicApproach`: implementation of our full heuristic approach that calls in succession all the previous methods. It takes as input the the vector of application permutations to be considered in `algorithm2`, the number of partial-solutions to be extended in `algorithm3` and the max number of iterations for `algorithm4`.

Table 4.1: Class Diagram



5 | Experimental Analysis

In this Chapter we show the results of our experimental analysis where we tested our algorithms and their implementation.
Our results revolve around 3 main experiments:

- We measured the scalability of our heuristic approach by computing the time required by the algorithm to reach its optimal solution, for progressively larger systems. These experiment and their results are discussed in Section 5.2.
- We also compared the quality of the results given by the heuristic approach with the optimal solution found by BARON, which is a commercial software tool capable of solving MINLP problems. This experiment is discussed in Section 5.3.
- Finally, in Section 5.4, we analyse in detail the effectiveness of tabu search in improving the final platform profit found by the heuristic approach.

All the experiments presented above are performed on concrete instances of the Stackelber Game, which we created by generating multiple samples of the Mobile Edge Cloud System model, whose parameters are randomly selected using uniform random distributions over some pre-defined intervals. This process is presented with more detail in the Section 5.1.

In order to avoid any potential effect that a random generation of the parameters may have on our experiments, all our results are obtained by averaging the outcomes of 10 different instances of the same problem.

5.1. System setup

The parameters used to generate the system are listed in table 5.1. We start by creating the objects of type `Application`; then, for each application we create the corresponding users. Finally, we initialize an object of type `Platform` and we update its members with the applications and users previously created. The number of users and applications is variable, and will be used as a stress parameter to increase the complexity of the problem.

5.2. Heuristic approach scalability

Our first analysis consists in measuring the time required by the algorithm to complete the execution of the method `heuristicApproach`, presented in section 4.6, under different conditions. These conditions consists of a different choice of the number of users and number of applications, which are the main factors of the computational complexity of

Table 5.1: System Parameters

Application Parameters	
$\bar{R}^{(a)}$	Randomly generated with Uniform Distribution in $[3, 10]$ sec.
$\lambda^{(a)}$	Randomly generated with Uniform Distribution in $[2, 5]$ req/sec.
$r^{(a1)}$	Randomly generated with Uniform Distribution in $[2, 5]$ \$/h.
$D_c^{(a)}$	$\frac{\bar{R}^{(a)}}{15}(1 + 0.1X)$ sec, where X is a Uniform random variable in $[-1, 1]$.
$D_e^{(a)}$	$\frac{6}{5}D_c^{(a)}$ sec.
$\delta^{(a)}$	$0.66(1 + 0.1X)$ MB, where X is a Uniform random variable in $[-1, 1]$.
$m^{(ak)}$	$(2 - 0.5k)(1 + 0.1X)$ MB, where X is a Uniform random variable in $[-1, 1]$.
γ	$1.5(1 + 0.1X)$, where X is a Uniform random variable in $[-1, 1]$.
User Parameters	
$D_i^(1)$	$\frac{3}{4}\bar{R}^{(a)}(1 + var \cdot X)$ sec, where X is a Uniform random variable in $[-1, 1]$, default value of var is 0.1
$D_i^{(2)}$	$\frac{1}{6}D_i^{(1)}$ sec,
$p_i^{(1)}$	$5D_i^{(1)}$ W,
$p_1^{(2)}$	$10D_i^{(2)}$ W,
β_i	Randomly generated with Uniform distribution in $[0.491, 0.522]$ $\frac{\$}{kWh}$,
$\bar{M}_i$	5 MB,
B_i	Randomly generated with Uniform distribution in $[9.9, 25.1]$ Mbs,
U_i	Randomly generated with Uniform distribution in $[12, 15]$ $\frac{\$}{h}$,
α_i	0.65,
T_i	50% of users Uniform distribution in $[540, 660]$ sec, 50% of users Uniform distribution in $[1080, 1320]$ sec,
$\bar{E}_i$	Randomly generated with Uniform distribution in $\left[0.9\lambda^{(a)}T_i^2 p_i^{(2)}, 1.2\lambda^{(a)}T_i^2 p_i^{(1)}\right]$ J,
ζ_i	Randomly generated with Uniform distribution in $\left[5.3 \cdot 10^{-5}, 6 \cdot 10^{-5}\right]$ $\frac{\$}{MB}$,
Platform Parameters	
N_e	0.3 $\lvert\mathcal{U}\rvert$,
c	1.5 $\frac{\$}{h}$,
c_e	$\frac{c}{5}$ $\frac{\$}{h}$,
r_{min}	2 $\frac{\$}{h}$,
r_{max}	10.8 $\frac{\$}{h}$,
T	3600 sec.

our algorithms. More specifically, for each value of $|\mathcal{A}|$ (n. of app) in the set $\{3, \ldots, 7\}$ we will run our algorithms taking as the number of users all the values in the range $[50, 1000]$, with an increment of 50 at each step. In order to boost the performance, the code running the algorithms has been parallelized using MPI. We have divided the vector of all permutations that need to be checked among ranks, so that the computational load of `algorithm2` (4.6) is reduced, allowing each rank to reach its optimal solution faster. Moreover, we optimized the algorithms even more by further reducing the number of permutations that are considered by each rank. This optimization achieves huge time savings at the cost of slight decrease in the quality of the final solution (see Appendix A.5).
The results are shown in Figure 5.1, where we observe that even for the most complex system (7 applications - 1000 users) the algorithm finds the optimal solution in just few minutes. Furthermore, the plots suggest a linear relation between the execution time and the number of users, meaning that our approach scales well with the complexity of the problem.
These tests were performed on my PC with 6 cores Intel 3.6GHz and 16GB memory.

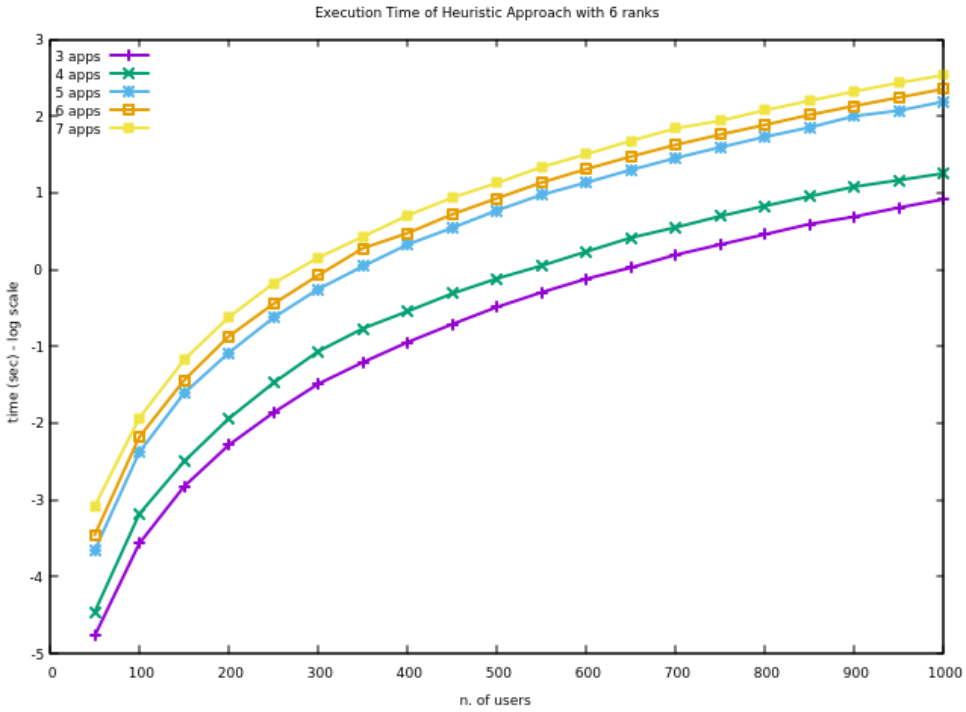


Figure 5.1: Time required for heuristic approach to reach its optimal solution.

5.3. Comparison with BARON

In this Section we compare the solution found by our method with the one found by BARON. We considered a system with 3 applications while the number of users was varied in the range $[10, 100]$ with an increment of 10 users at each step. For each experiment we measured the following attributes of the final solution (for both methods):

- Platform profit,
- time required to reach it,
- total number of edge/cloud resources used,
- total number of users assigned to edge/cloud.

BARON was given a time limit of 2h for each single instance of the problem, and the initial solution had a null value for all the decision variables. BARON runs were performed on a Linux server machine with 40-cores Intel(R) Xeon(R) CPU 2.20 GHz and 64 GB memory, while the heuristic approach was run on my PC with 6 cores Intel 3.6GHz and 16 GB memory.
Figure 5.2 shows the results. As usual, the results are the average of the outcomes of 10 different instances of the same problem.
From the first plot we observe that for small systems $(10, 20, 30$ users) BARON finds better solutions than our approach, consistently improving our profit by $33.8\%, 17.7\%, 16\%$ respectively.
However this behavior is much more inconsistent for larger systems, where the heuristic approach often achieves better results. Indeed, by averaging the net percentage increasing of BARON over the heuristic approach across all experiments, we find that the profit achieved by BARON is only 4.96% higher. Moreover, the heuristic approach consistently improves the profit of the solution with increasingly number of users, as it should be expected; whereas, the performance of BARON is much more volatile.
As for the time required to reach the final solution, we observe that the heuristic approach is multiple orders of magnitude faster than BARON, which reaches the time limitation even for small systems.
Finally, we notice the consistency in which the heuristic approach increases the edge and cloud resource usage with respect to the number of users in the system, always allocating the maximum number of edge nodes available to the platform (coherently with our assumption of cheaper edge resources). Once again, the behavior of BARON is more inconsistent.

5.4. Tabu search

Figures 5.3, 5.4 shows the best and current profit profiles for multiple runs of tabu search. Each run was performed on a system with 500 users and 5 applications; the parameter *var* (see Table 5.1) was set to 30% and the max number of iterations is 500. The attributes of the solution objects that were saved in the tabu list are:

- Profit,
- number of edge and cloud resources allocated for each application,
- total number of users assigned to edge and cloud.

We observe that in the case of *short memory* of the algorithm (tabu list siz = 10), RANDOM neighbor selection seems to work better, as it is capable of improving at least once the profit of the solution; whereas BEST PROFIT is stuck looping across the same solu-

tions.
If we increase the tabu list size to 50 (*long memory* scenario), we see that the results of BEST PROFIT are improved; however also observe that the algorithm falls back in a longer loop.

Finally, we measured the profit increase (in percentage) achieved by tabu search for different values of the parameter *var*, which is indicative of the variance among users. Our hope was that, for increasingly variance among users, the performance of tabu search would increase, since the solution of the heuristic approach would have been less optimal. Hence, we tested different versions of tabu search on a system with 500 users, 5 applications, while the value of *var* was varied in the range $[0.1, 0.9]$ with an increase of 0.1 at each step. Figure 5.5 shows the results, which, as usual, were averaged across 10 different instances of the same problem.
We conclude that the tabu search algorithm we presented in this Thesis is not effective for this framework, as the profit increase never exceeds 0.5%.

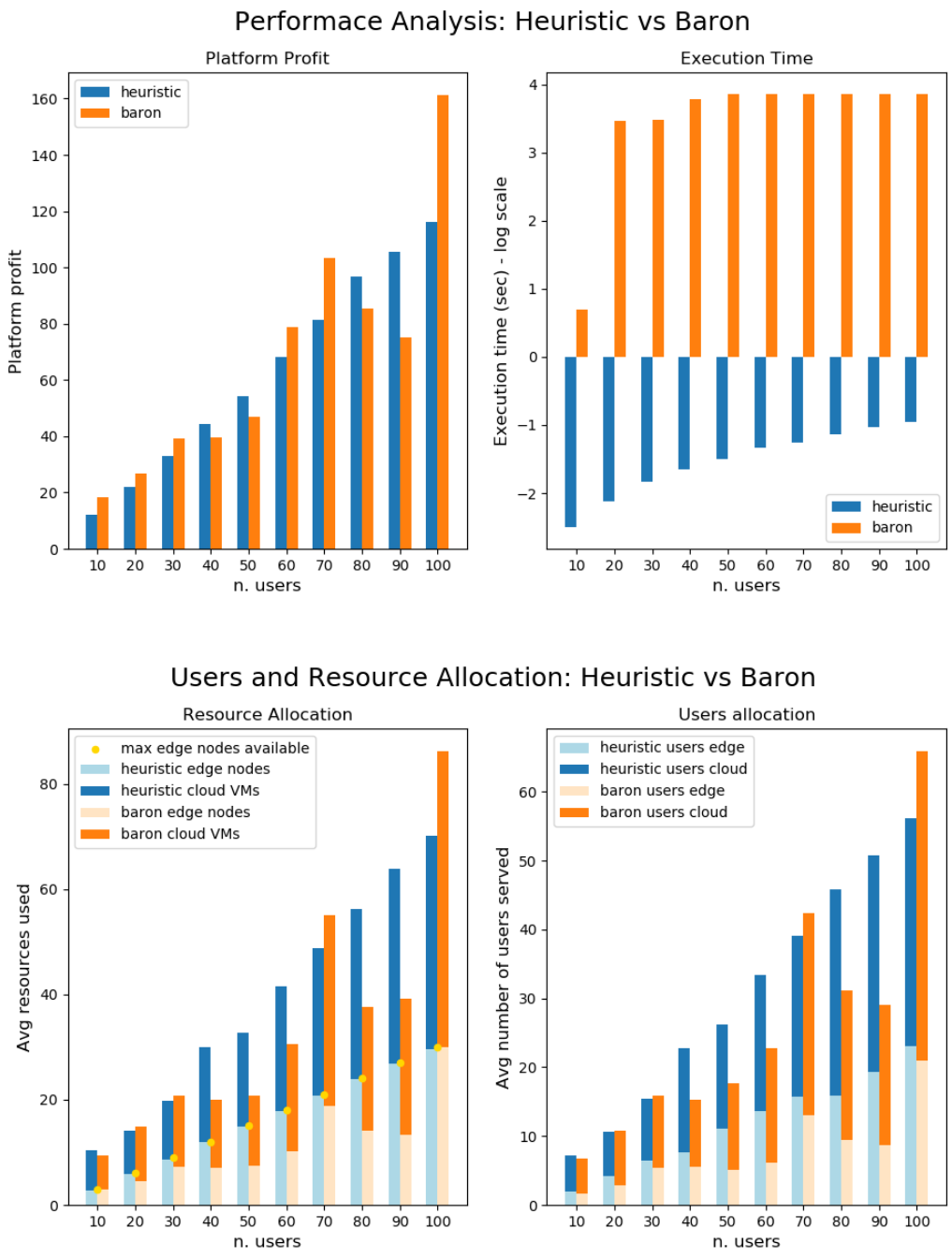


Figure 5.2: BARON and heuristic approach comparison.

Figure 5.3: Short memory scenario.

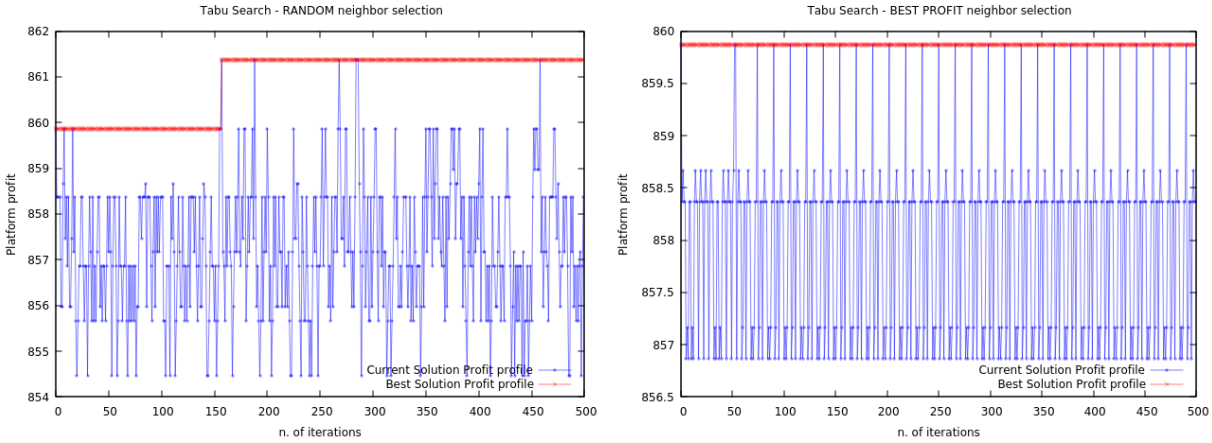
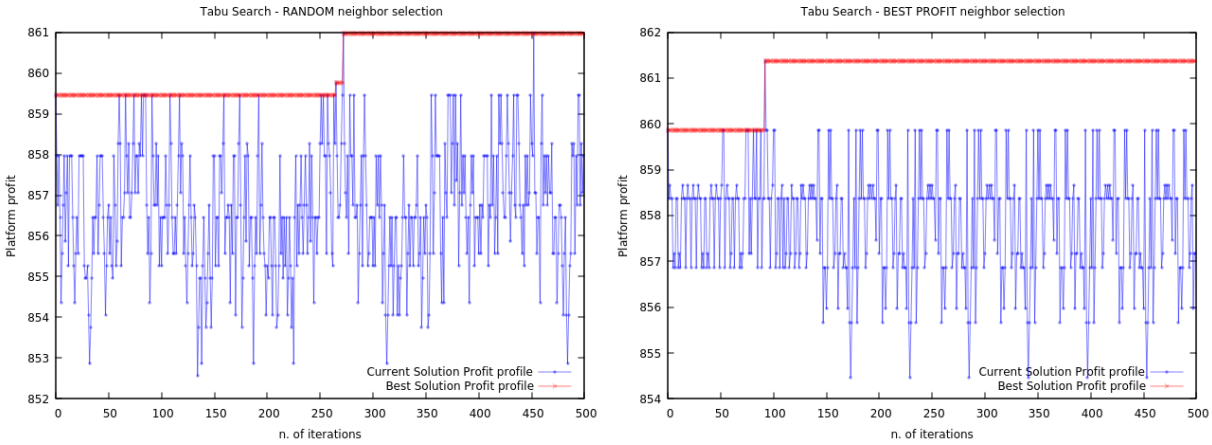


Figure 5.4: Long memory scenario.



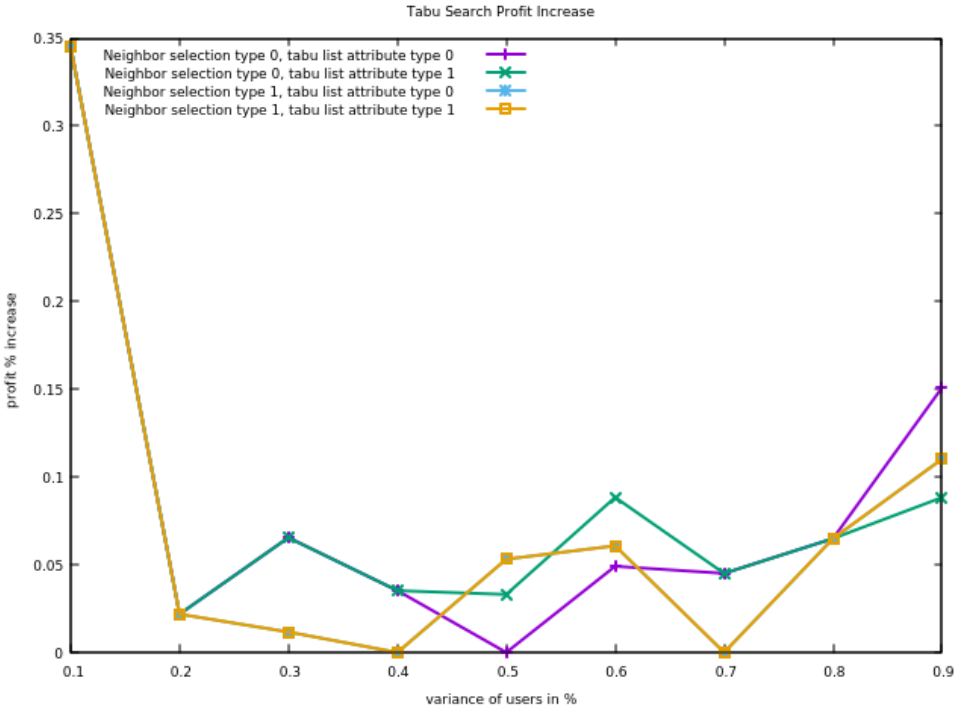


Figure 5.5: Profit inctease by tabu search for different users' variance.

Bibliography

[1] H. Haile, K. J. Grinnemo, S. Ferlin, P. Hurtig, and A. Brunstrom. End-to-end congestion control approaches for high throughput and low delay in 4g/5g cellular networks. *Computer Networks*, 186:107692, 2021. ISSN 1389-1286. doi: https://doi.org/10.1016/j.comnet.2020.107692. URL `https://www.sciencedirect.com/science/article/pii/S1389128620312974`.

[2] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. *Quantitative system performance: computer system analysis using queueing network models*. Prentice-Hall, 1984.

[3] H. Sedghani, D. Ardagna, M. Passacantando, M. Z. Lighvan, and H. S. Aghdasi. An incentive mechanism based on a Stackelberg game for mobile crowdsensing systems with budget constraint. *Ad Hoc Networks*, 123:102626, 2021. ISSN 1570-8705. doi: https://doi.org/10.1016/j.adhoc.2021.102626. URL `https://www.sciencedirect.com/science/article/pii/S1570870521001505`.

[4] U. Tadakamalla and D. A. Menasce. Autonomic resource management for fog computing. *IEEE TCC*, pages 1–1, 2021. doi: 10.1109/TCC.2021.3064629.

A | Appendix A

A.1. Proof of **Theorem 3.1**

Proof. We denote the response time of edge resources for deployment a as follows:

$$f(n_e^{(a)}, \Lambda_e^{(a)}) = \frac{\Lambda_e^{(a)}}{\Lambda^{(a)}} \cdot \frac{D_e^{(a)} n_e^{(a)}}{n_e^{(a)} - D_e^{(a)} \Lambda_e^{(a)}}$$

The first and second derivatives of f with respect to $n_e^{(a)}$ are:

$$\frac{\partial f(n_e^{(a)}, \Lambda_e^{(a)})}{\partial n_e^{(a)}} = -\frac{1}{\Lambda^{(a)}} \frac{(\Lambda_e^{(a)})^2 (D_e^{(a)})^2}{\left[n_e^{(a)} - \Lambda_e^{(a)} D_e^{(a)}\right]^2} \tag{A.1}$$

$$\frac{\partial^2 f(n_e^{(a)}, \Lambda_e^{(a)})}{\partial {n_e^{(a)}}^2} = \frac{1}{\Lambda^{(a)}} \frac{2(\Lambda_e^{(a)})^2 (D_e^{(a)})^2}{\left[n_e^{(a)} - \Lambda_e^{(a)} D_e^{(a)}\right]^3} \tag{A.2}$$

The first and second derivatives of f respect to $\Lambda_e^{(a)}$ are:

$$\frac{\partial f(n_e^{(a)}, \Lambda_e^{(a)})}{\partial \Lambda_e^{(a)}} = \frac{1}{\Lambda^{(a)}} \cdot \frac{D_e^{(a)} (n_e^{(a)})^2}{\left[n_e^{(a)} - \Lambda_e^{(a)} D_e^{(a)}\right]^2} \tag{A.3}$$

$$\frac{\partial^2 f(n_e^{(a)}, \Lambda_e^{(a)})}{\partial {\Lambda_e^{(a)}}^2} = \frac{1}{\Lambda^{(a)}} \cdot \frac{2(D_e^{(a)})^2 (n_e^{(a)})^2}{\left[n_e^{(a)} - \Lambda_e^{(a)} D_e^{(a)}\right]^3} \tag{A.4}$$

The Partial derivative is:

$$\frac{\partial^2 f(n_e^{(a)}, \Lambda_e^{(a)})}{\partial \Lambda_e^{(a)} \partial n_e^{(a)}} = \frac{\partial^2 f(n_e^{(a)}, \Lambda_e^{(a)})}{\partial n_e^{(a)} \partial \Lambda_e^{(a)}} = -\frac{1}{\Lambda^{(a)}} \left(\frac{2\Lambda_e^{(a)} (D_e^{(a)})^2 n_e^{(a)}}{\left[n_e^{(a)} - \Lambda_e^{(a)} D_e^{(a)}\right]^3} \right) \tag{A.5}$$

So we can write the Hessian of $f(n_e^{(a)}, \Lambda_e^{(a)})$ as follows:

$$\nabla^2 f(n_e^{(a)}, \Lambda_e^{(a)}) = \frac{2(D_e^{(a)})^2}{\Lambda^{(a)} \left(n_e^{(a)} - D_e^{(a)} \Lambda_e^{(a)}\right)^3} \begin{bmatrix} (\Lambda_e^{(a)})^2 & (\Lambda_e^{(a)}) n_e^{(a)} \\ \Lambda_e^{(a)} n_e^{(a)} & (n_e^{(a)})^2 \end{bmatrix} \tag{A.6}$$

Notice that $tr(\nabla^2 f) > 0$ and $det(\nabla^2 f) = 0$, meaning that the Hessian is positive semi-definite and $f(n_e^{(a)}, \Lambda_e^{(a)})$ is convex.
Since the response time constraints (3.3) are the sum of multiple convex function, they are convex. □

****

A.2. Proof of Theorem 3.2

Proof. Problem 3.14 is equivalent to:

$$\begin{aligned} &\min_{n_e^{(a)}} \ c_e n_e \\ &\sum_{a \in \mathcal{A}} n_e^{(a)} \leq N_e \\ &n_e^{(a)} \geq \frac{\bar{\bar{R}}^{(a)} D_e^{(a)} \Lambda^{(a)}}{\bar{\bar{R}}^{(a)} - D_e^{(a)}} \quad \forall a \in \mathcal{A}. \end{aligned}$$

Therefore, if condition 3.18 is satisfied, the optimal number of edge servers is:

$$n_e^{(a)} = \frac{\bar{\bar{R}}^{(a)} D_e^{(a)} \Lambda^{(a)}}{\bar{\bar{R}}^{(a)} - D_e^{(a)}} \quad \forall a \in \mathcal{A}$$

□

****

A.3. Proof of Theorem 3.3

Proof. The edge platform problem where no cloud resources are used can be formulated as follows:

$$\begin{aligned} \min \ & \beta_e p_e n_e \\ \text{s.t.} \ & n_e \leq N_e \\ & \frac{D_e n_e}{n_e - D_e \bar{\Lambda}} \leq \bar{\bar{R}} \\ & n_e - D_e \bar{\Lambda} > 0 \end{aligned}$$

that is equivalent to

$$\begin{aligned} \min \ & \beta_e p_e n_e \\ \text{s.t.} \ & n_e \leq N_e \\ & n_e \geq \frac{\bar{\bar{R}} D_e \bar{\Lambda}}{\bar{\bar{R}} - D_e} \end{aligned}$$

whose optimal solution is

$$n_e^* = \frac{\bar{\bar{R}} D_e \bar{\Lambda}}{\bar{\bar{R}} - D_e},$$

provided that the feasible region of the latter problem is nonempty, i.e.,

$$\bar{\Lambda} \leq \frac{N_e(\bar{\bar{R}} - D_e)}{\bar{\bar{R}} D_e},$$

that is the total load is small enough.
If the total load does not satisfies the above condition, then edge resources are saturated and cloud resources need to be used. Thus, the edge platform problem is:

$$\begin{aligned} \min_{(\Lambda_e, n_c)} \quad & c_c n_c \\ \text{s.t.} \quad & \frac{D_e N_e \Lambda_e}{N_e - D_e \Lambda_e} + \frac{n_c D_c(\Lambda - \Lambda_e)}{n_c - D_c(\Lambda - \Lambda_e)} \leq \bar{\bar{R}} \Lambda \\ & 0 < \Lambda_e < \Lambda \\ & N_e - D_e \Lambda_e > 0 \\ & n_c - D_c(\Lambda - \Lambda_e) > 0 \end{aligned}$$

The latter problem is convex and standard constraints qualifications hold (e.g., Slater condition is satisfied), hence it is equivalent to the corresponding KKT system:

$$\begin{aligned} & \mu_1 \left[\frac{D_e N_e^2}{(N_e - D_e \Lambda_e)^2} - \frac{D_c n_c^2}{(n_c - D_c(\Lambda - \Lambda_e))^2} \right] - \mu_2 + \mu_3 + D_e \mu_4 - D_c \mu_5 = 0 \\ & c_c - \mu_1 \frac{D_c^2 (\Lambda - \Lambda_e)^2}{(n_c - D_c(\Lambda - \Lambda_e))^2} - \mu_5 = 0 \\ & \frac{D_e N_e \Lambda_e}{N_e - D_e \Lambda_e} + \frac{n_c D_c(\Lambda - \Lambda_e)}{n_c - D_c(\Lambda - \Lambda_e)} \leq \bar{\bar{R}} \Lambda \\ & \mu_1 \geq 0, \quad \mu_1 \left[\frac{D_e N_e \Lambda_e}{N_e - D_e \Lambda_e} + \frac{n_c D_c(\Lambda - \Lambda_e)}{n_c - D_c(\Lambda} - \Lambda_e) - \bar{\bar{R}} \Lambda \right] = 0 \\ & \Lambda_e > 0, \quad \mu_2 \geq 0, \quad \mu_2 \Lambda_e = 0 \\ & \Lambda_e < \Lambda, \quad \mu_3 \geq 0, \quad \mu_3(\Lambda - \Lambda_e) = 0 \\ & N_e - D_e \Lambda_e > 0, \quad \mu_4 \geq 0, \quad \mu_4(N_e - D_e \Lambda_e) = 0 \\ & n_c - D_c(\Lambda - \Lambda_e) > 0, \quad \mu_5 \geq 0, \quad \mu_5[n_c - D_c(\Lambda - \Lambda_e)] = 0 \end{aligned}$$

Notice that constraints imply $\mu_2 = \mu_3 = \mu_4 = \mu_5 = 0$. Moreover, it follows from second equation that $\mu_1 > 0$, thus first constraint holds as an equality, i.e.,

$$\frac{D_e N_e \Lambda_e}{N_e - D_e \Lambda_e} + \frac{n_c D_c(\Lambda - \Lambda_e)}{n_c - D_c(\Lambda - \Lambda_e)} = \bar{\bar{R}} \Lambda,$$

that is equivalent to

$$n_c = \frac{\left[\bar{\bar{R}} \Lambda - \frac{D_e N_e \Lambda_e}{N_e - D_e \Lambda_e} \right] D_c(\Lambda - \Lambda_e)}{\bar{\bar{R}} \Lambda - \frac{D_e N_e \Lambda_e}{N_e - D_e \Lambda_e} - D_c(\Lambda - \Lambda_e)}. \tag{A.7}$$

Since $\mu_1 > 0$, first equation implies that

$$\frac{D_e N_e^2}{(N_e - D_e \Lambda_e)^2} = \frac{D_c n_c^2}{(n_c - D_c(\Lambda - \Lambda_e))^2},$$

hence

$$\frac{\sqrt{D_e}N_e}{N_e - D_e\Lambda_e} = \frac{\sqrt{D_c}n_c}{n_c - D_c(\Lambda - \Lambda_e)},$$

that is equivalent to

$$n_c = \frac{\sqrt{D_e}D_cN_e(\Lambda - \Lambda_e)}{N_e(\sqrt{D_e} - \sqrt{D_c}) + D_e\sqrt{D_c}\Lambda_e} \tag{A.8}$$

It follows from (A.7)–(A.8) that

$$\sqrt{D_c}(N_e - D_e\Lambda_e)\Big[(N_eD_e + \bar{\bar{R}}\Lambda D_e - N_e\sqrt{D_cD_e})\Lambda_e$$
$$-N_e\bar{\bar{R}}\Lambda + N_e\Lambda\sqrt{D_cD_e}\Big] = 0.$$

Since $N_e - D_e\Lambda_e > 0$, we get that the optimal edge load is

$$\Lambda_e^* = \frac{N_e\Lambda(\bar{\bar{R}} - \sqrt{D_cD_e})}{N_eD_e + \bar{\bar{R}}\Lambda D_e - N_e\sqrt{D_cD_e}}.$$

Finally, we get from (A.7) the optimal number of cloud resources:

$$n_c^* = \frac{D_c\Lambda[\bar{\bar{R}}D_e\Lambda - N_e(\bar{\bar{R}} - D_e)]}{N_e(\sqrt{D_e} - \sqrt{D_c})^2 + D_e\Lambda(\bar{\bar{R}} - D_c)}.$$

□

****

A.4. Proof of **Theorem 3.4**

Proof. Analogously to the only edge scenario, the optimization problem 3.19 is equivalent to:

$$\min_{n_c^{(a)}} c_cn_c$$
$$n_c^{(a)} \geq \frac{\bar{\bar{R}}^{(a)}D_c^{(a)}\Lambda^{(a)}}{\bar{\bar{R}}^{(a)} - D_c^{(a)}} \quad \forall a \in \mathcal{A}.$$

Therefore, the optimal number of cloud VMs is:

$$n_c^{(a)} = \frac{\bar{\bar{R}}^{(a)}D_c^{(a)}\Lambda^{(a)}}{\bar{\bar{R}}^{(a)} - D_c^{(a)}} \quad \forall a \in \mathcal{A}$$

□

****

A.5. Heuristic approach optimization

In this Section we propose an optimization to our heuristic approach. It works by reducing the number of applications priority-orders that need to be checked by the algorithm. More specifically, we restrict the set of permutations by considering as equivalent all the permutations that differs only by the elements on the last $|\mathcal{A}| - k$ positions; where k is an hyperparameter. For example, if $|\mathcal{A}| = 5$ and $k = 2$, the following permutations are regarded as equal by the algorithm (therefore, only one is considered):

- 1 2 3 4 5
- 1 2 3 5 4
- 1 2 4 3 5
- 1 2 4 5 3
- 1 2 5 3 4
- 1 2 5 4 3

this is because the permutations only differs by the last 3 elements, while the elements on the first k-th positions are equal.
The idea behind it is that for instances of practical interest of our problem, the availability of platform edge servers is sufficient to cover the load of only few applications, meaning that, regardless of the order, the majority of them will be served in the cloud. Therefore, the positioning of applications after a certain index is not going to affect the quality of the final solution, or it will affect it very slightly.
By doing so we achieve huge time-savings, as we improve the computational complexity with respect to the number of applications from $|\mathcal{A}|!$ to $\frac{|\mathcal{A}|!}{(|\mathcal{A}|-k)!}$.
Moreover, to see whether this optimization has a significant impact on the quality of our solution, we run the algorithm for different values of the hyperparameter k. The system used for this test is composed of 5 applications, while the number of users is varied in the range $[50, 1000]$. As always, each results is the average of 10 outcomes generated from different instances of the same problem.
As expected, figure A.1 shows that reducing the permutations checked by the algorithm has no impact on the final profit of the heuristic algorithm, even for $k = 1$.

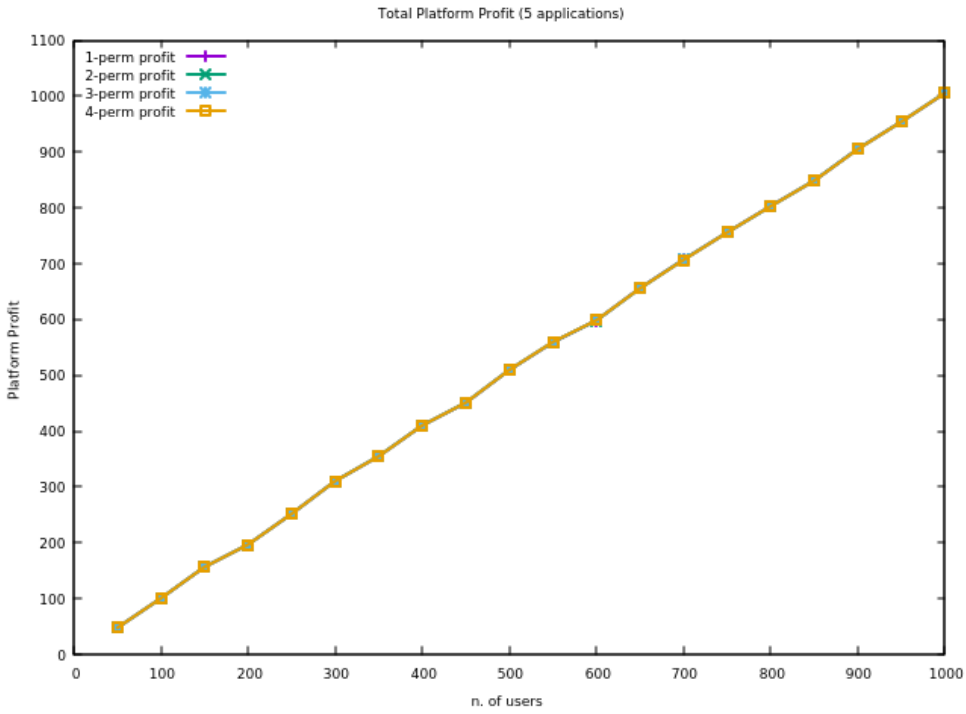


Figure A.1: Heuristic approach final profit for different values of k.

List of Figures

List of Tables