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. Indeed, mobile devices are often limited by scarce computational power that prevents AI applications from being processed locally; moreover, cloud serves are located far from the end users, meaning that the processes of sending and receiving data are affected by a transmission delay. A newly popular approach to these difficulties is provided by the edge paradigm, by which the client data is processed by a provider (or platform) in the periphery of the network, as close to the originating source as possible.
In this thesis, we model the provider-users dynamics as a Stackelber Game, and we formulate a Maximum Integer Non-Linear Programming problem from the edge provider perspective, where the objective function is the net difference of the revenue from supporting the users in the execution of AI applications and the costs of edge and cloud resources that has to be used by the provider to guarantee that all performance constraints are satisfied. This problem and our proposed approach to find an approximate solution are a generalization of the work presented in [16], that we extended by considering an arbitrary number of AI applications offered by the edge provider to the users. Moreover, we also refined the final solution provided by our 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 ([14]). According to Fortune [7], the world wide Internet of Things market size is expected to grow from 478.36 billion USD in 2022, to 2465.26 in 2029, as new *smart* policies are being adopted in private homes, public infrastructures and manufactures.
The proliferation of IoT devices has made AI applications increasingly directed towards mobile computing since AI paradigms can be used to process the massive amounts of data generated by IoT devices and produce useful analysis or improve the efficiency of the related services.
For these reasons 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 Natural language processing (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 this thesis we focus on the field of Augmented Reality (AR), where AI applications enables advanced and accurate digital overlays on the real-world environment. This type of applications relies extensively on Deep Neural Networks (DNN) models, which are computationally expensive, making them challenging to deploy on mobile devices that are restricted by limited storage capacities and computational power ([5]). 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.
The resource shortage of local devices and transmission latency of cloud, could be more sensitive when the applications have quality of service (QoS) response time constraints. In order to overcome shortcomings arising from the above-mentioned problems, a new computing paradigm has recently been introduced: the so called edge computing. As explained in [15], it 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 consisting of three main compo-

nents: 1. an edge provider (or platform) with limited number of edge servers, 2. a large number of mobile users who are connected to the edge provider through a network with non-negligible latency, 3. a pool of unlimited cloud VMs connected to the edge platform through a fast network, 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. For the sake of simplicity, we consider single DNN-based AI applications with two different deployments, which determine either the DNN can be run fully on local device or partially processed by remote resources. In order to participate in the system, the users must pay an application 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, we 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 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. Additionally, we implement an iterative search algorithm with past-state memory to optimize further our final solution.

3. We finally evaluate the performance of the proposed approach by analyzing the time required to reach the optimal solution under different conditions and comparing our results with the solution found by commercial software tool.

The remaining portion of this thesis is structured in the subsequent manner. Chapter 1 provides an overview of the literature related to the topics presented in this thesis. Chapter 2 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 3. In Chapter 4, 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 5. The experimental analysis of our approach is discussed in Chapter 6, where we measure the quality of our solutions by comparing them with the results obtained by running a traditional solver. Finally, in chapter 7 we discuss future work.

1 | Related Work

In this Chapter we provide a quick overview of how the topics presented in this thesis have been tackled in the literature.

MEC System models composed by edge servers and cloud virtual machines and resource allocation problems in such environments have recently received a lot of attention from the research community, such as in [4] and in [18]. In the former cited work a wireless energy transfer to IoT devices minimization problem is formulated along with a solution algorithm based on bipartite graph matching; whereas the latter presents a DNN partition placement and resource allocation scheme to maximize the utility function of a system characterized by heterogeneous IoT devices. The interest on this topic has grown because the processing demand for AI applications has sky-rocketed due to the increasing proliferation of IoT devices, meaning that efficient management of edge and cloud resources has become necessary to achieve highly scalable IoT systems; for example, in [22] a dynamic and energy efficient resource allocation mechanism is proposed to manage users' requests based on their priority for a system with a limited amount of edge servers.

It is common practice to model the collaboration between users and edge provider by partitioning the DNN underlying the AI applications, in order to reduce the computational power required by IoT devices. Contrary to our work, where we proposed a unique and fixed DNN partition in two possible deployments, [19] designs a composite collaborative DNN that compresses the model size, where an inference mechanism is implemented between the mobile lightway binary neural network and the edge server to improve training accuracy. [20] formulates the offloading problem as a graph state migration problem which is solved thanks to a Graph Neural Network approach. [21] implements a dynamic, fast and efficient algorithm for generating near-optimal offloading decisions in terms of response time constraint and energy consumption.

Another common feature of the MEC models literature are incentives mechanism to motivate edge providers to put their idle resources at disposal. Indeed, sharing edge nodes is costly in terms of electricity expenses and usage; hence, edge providers should receive monetary compensation for their services. [11] proposes a complex incentive mechanism based on an open market with a double stage auction system: during the assignment stage application users (buyers) secretly bid for an offloading service and are assigned to a potential fog nodes (sellers) that set a price depending on their usage cost; then, in the pricing stage, the set of winning buyers and sellers as well as the clearing prices are determined. In this thesis we follow the Stackelberg Game approach introduced in [16]. Here, a MEC platform objective function is defined, that represents the platform profit and grows with the number of users participating in the systems. On the other hand, users are motivated to run the sensing task (hence, facing the energy costs) by receiving an utility value from the platform as a reward for participating in the system.

An important aspect of edge computing problems is resource management. Obviously, edge and cloud resources are costly and it's in the edge provider interest to minimize their expenses while guaranteeing all quality of service constraints. For this reason, in the literature there has been a multitude of proposed formulations and solutions to the edge computing resource allocation problem. Machine Learning approaches are particularly popular. For example, [9] explores a multi-device hybrid decision-based deep reinforcement learning (DRL) solution to address the problem involving both continuous and discrete variables, while [13] propose a deep-deterministic policy gradient (DDPG) based temporal feature learning attentional network (TFLAN) model to address the multi-resource allocation problem. Once again, in this thesis we follow the approach presented in [16], where an heuristic algorithm based on KKT conditions is proposed in order to find an approximate solution to the MINLP MEC platform optimization problem in a fast and efficient way.

2 | System Model and Problem Formulation

The purpose of this Chapter is to model the Mobile Edge Cloud (MEC) system and introduce all the relevant definitions composing the formulation of the problem.
More specifically, in Section 2.1 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 2.2 contains the mathematical formulation of the users' decision problem and platform optimization problem.

2.1. Mobile Edge Cloud model

In this thesis, we consider a MEC system as shown in Figure 2.1. It consists of an edge provider, located in telecommunication towers, with a limited number of edge servers, a controller agent deployed in the end-user's device that collects some user's information, 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 provider through a fast network with negligible delay, whereas users' connection is affected by a non-negligible delay (5G network).

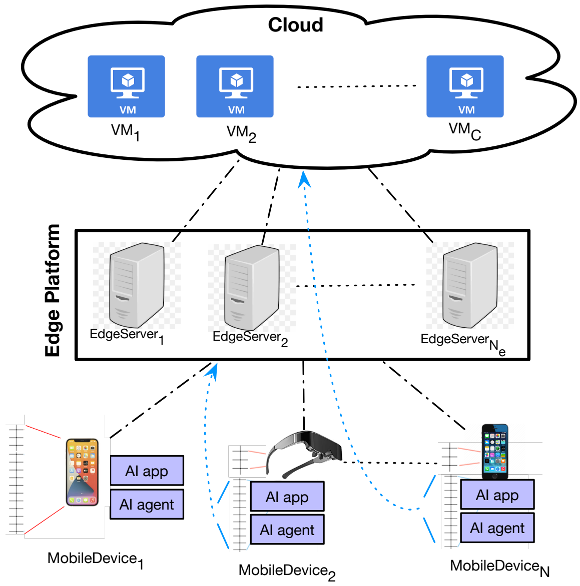


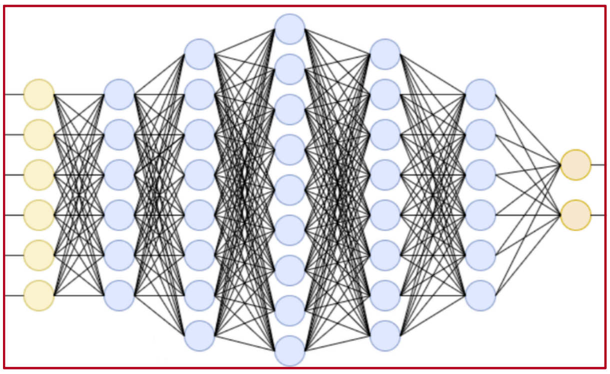
Figure 2.1: Application's resource assignment model.

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 at most one application at the time. Users communicate to the provider to inform which application they are willing to run, meaning that the provider 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:

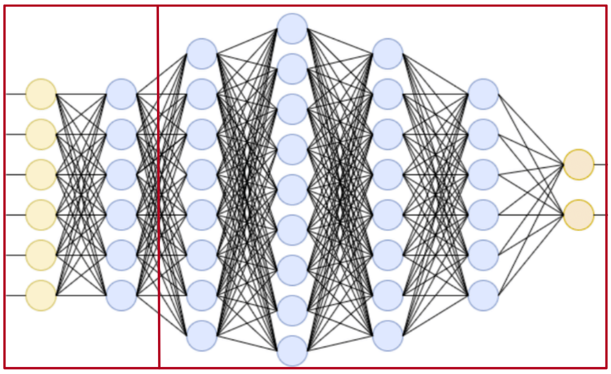
$$\mathcal{U} = \bigcup_{a=1}^{|\mathcal{A}|} \mathcal{U}^{(a)}.$$

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 two alternative deployments indexed by k: an example of AI application is shown in Figure 2.2. The first deployment ($k = 1$, see Figure 2.2a) is a full DNN which can be deployed only on the user device, while the other ($k = 2$, see Figure 2.2b) 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 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

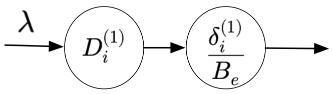
Figure 2.2: Example of AI application component with two deployments

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 $[r_{min}, r_{max}]$.

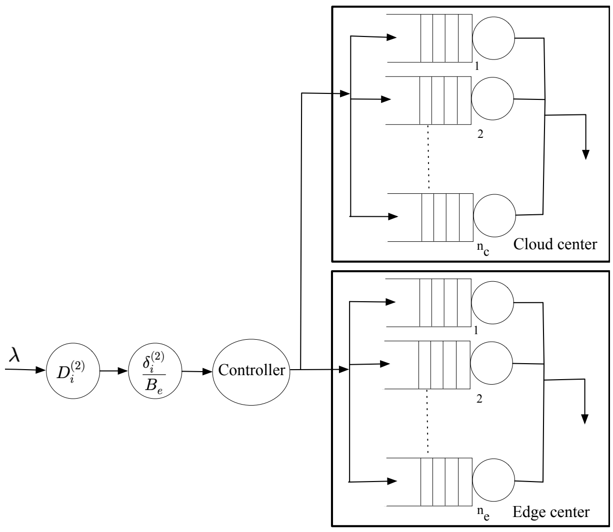
We assume each user is interested in running just one specific AI application; and we denote the utility gain for running their desired application with U_i, $\forall i \in \mathcal{U}$. Therefore, users are willing to participate in the system as long as they have enough energy and memory requirements 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 [10]) $D_i^{(k)}$ 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 [17] while the local IoT devices are modelled as a delay center (see Figure 2.3).
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 2.3: Queuing model of resources.

of the users who choose the second deployment at the edge servers and the cost of remote cloud resources, while the performance response time constraint for each application is fulfilled. 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:

$$x_i^{(k)} = \begin{cases} 1 & \text{if user } i \text{ selected deployment } k \text{ ,} \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

$$y_i^e = \begin{cases} 1 & \text{if user } i\text{'s deployment is served by edge VMs,} \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

$$y_i^c = \begin{cases} 1 & \text{if user } i\text{'s deployment is served by cloud VMs,} \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

Where $x_i^{(k)}$ is a user decision variable, while y_i^e and y_i^c are edge platform decision variables. We have the following constraint for $x_i^{(k)}$ (at most, only one deployment can be selected):

$$\sum_{k=1}^{2} x_i^{(k)} \leq 1 \qquad \forall i \in \mathcal{U}, \tag{2.4}$$

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:

$$y_i^e + y_i^c = x_i^{(2)} \qquad \forall i \in \mathcal{U}. \tag{2.5}$$

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:

$$\sum_{k=1}^{2} m^{(ak)} x_i^{(k)} \leq \bar{M}_i \qquad \forall i \in \mathcal{U}^{(a)},\ \forall a \in \mathcal{A}, \tag{2.6}$$

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

$$E_i \leq \bar{E}_i \qquad \forall i \in \mathcal{U}, \tag{2.7}$$

where E_i is the energy consumption of user i' device and it is defined as follows:

$$E_i = \lambda^{(a)} T_i^2 \sum_{k=1}^{2} p_i^{(k)} x_i^{(k)} \quad \forall i \in \mathcal{U}^{(a)},\ \forall a \in \mathcal{A}. \tag{2.8}$$

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:

$$\Lambda^{(a)} = \sum_{i \in \mathcal{U}^{(a)}} \lambda^{(a)} x_i^{(2)} \qquad \forall\ a \in \mathcal{A}. \tag{2.9}$$

We assume that the edge provider has access to all user parameters such as U_i, $\bar{M}_i$, $\bar{E}_i$, $D_i^{(k)}$, T_i and $p_i^{(k)}$ through the agent running on user devices (See Figure 2.1). 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:

$$L_e^{(a)} = D_e^{(a)} \lambda^{(a)} \sum_{i \in \mathcal{U}^{(a)}} x_i^{(2)} y_i^e \qquad \forall\, a \in \mathcal{A}, \tag{2.10}$$

$$L_c^{(a)} = D_c^{(a)} \lambda^{(a)} \sum_{i \in \mathcal{U}^{(a)}} x_i^{(2)} y_i^c \qquad \forall a \in \mathcal{A}. \tag{2.11}$$

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:

$$\frac{L_e^{(a)}}{n_e^{(a)}} < 1 \iff L_e^{(a)} < n_e^{(a)} \qquad \forall a \in \mathcal{A}, \tag{2.12}$$

$$\frac{L_c^{(a)}}{n_c^{(a)}} < 1 \iff L_c^{(a)} < n_c^{(a)} \qquad \forall a \in \mathcal{A}. \tag{2.13}$$

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:

$$R_i = \sum_{k=1}^{2} D_i^{(k)} x_i^{(k)} + \frac{\delta^{(a)} x_i^{(2)}}{B_i} + \frac{D_e^{(a)} x_i^{(2)} y_i^e}{1 - \frac{L_e^{(a)}}{n_e^{(a)}}} + \frac{D_c^{(a)} x_i^{(2)} y_i^c}{1 - \frac{L_c^{(a)}}{n_c^{(a)}}} \tag{2.14}$$

Where the first expression denotes the running time of the selected deployment on users' device. The second expression indicates the transmission latency from users to the edge provider. Note that by running the first deployment, the users do not send any data to the provider. $\delta^{(a)}$ 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 provider. In the above equation, only the transmission delay between the end users and the edge provider 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 [8]; 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:

$$R_i \leq \bar{R}^{(a)} \quad \forall i \in \mathcal{U}^{(a)},\ \forall a \in \mathcal{A}. \tag{2.15}$$

2.1.1. Edge provider 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:

$$P_e = \sum_{a\in\mathcal{A}} \sum_{i\in\mathcal{U}^{(a)}} \sum_{k=1}^{2} T_i r^{(ak)} x_i^{(k)} - T\left(c_e n_e + c_c n_c\right) \tag{2.16}$$

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., $c_e D_e^{(a)} < c_c D_c^{(a)} \quad \forall a \in \mathcal{A}$.

2.1.2. User cost model

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

$$C_i = \sum_{k=1}^{2} T_i x_i^{(k)} \left(\alpha_i r^{(ak)} + (1-\alpha_i)\beta_i p_i^{(k)} \lambda^{(a)} T_i + x_i^{(2)} \zeta_i \delta^{(a)} \lambda^{(a)}\right), \tag{2.17}$$

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. Each user wants to maximize the difference between the value of running the application U_i and the cost C_i.

2.2. Problem Formulation

In this Section we finally set the mathematical formulation of the constrained optimization problem featuring our model, both from a user and provider perspective.

[2] https://aws.amazon.com/ec2/pricing/

[3] https://azure.microsoft.com/en-us/pricing/details/virtual-machines/windows/

The provider optimization problem is formulated in Subsection 2.2.1 and the users' decision problem in Subsection 2.2.2.

2.2.1. Platform optimization problem

The edge platform optimization problem is formulated as follows:

$$\max_{r,n_e^{(a)},n_c^{(a)},y_i^e,y_i^c} P_e = \sum_{a\in\mathcal{A}}\sum_{i\in\mathcal{U}^{(a)}}\sum_{k=1}^{2} T_i r^{(ak)} x_i^{(k)} - T\left(c_e n_e + c_c n_c\right) \tag{2.18}$$

$$\text{Subject to:} \tag{2.19}$$

$$\sum_{a\in\mathcal{A}} n_e^{(a)} \leq N_e, \tag{2.20}$$

$$\sum_{k=1}^{2} D_i^{(k)} x_i^{(k)} + \frac{\delta^{(a)} x_i^{(2)}}{B_i} + \frac{D_e^{(a)} x_i^{(2)} y_i^e}{1-\frac{L_e^{(a)}}{n_e^{(a)}}} + \frac{D_c^{(a)} x_i^{(2)} y_i^c}{1-\frac{L_c^{(a)}}{n_c^{(a)}}} \leq \bar{R}^{(a)} \quad \forall i\in\mathcal{U}^{(a)},\ \forall a\in\mathcal{A}, \tag{2.21}$$

$$L_e^{(a)} < n_e^{(a)} \qquad \forall a\in\mathcal{A}, \tag{2.22}$$

$$L_c^{(a)} < n_c^{(a)} \qquad \forall a\in\mathcal{A}, \tag{2.23}$$

$$y_i^e + y_i^c = x_i^{(2)} \qquad \forall i\in\mathcal{U}, \tag{2.24}$$

$$r^{(a2)} = r^{(a1)} + \gamma^{(a)} r \quad \gamma^{(a)} > 1, \qquad \forall a\in\mathcal{A}, \tag{2.25}$$

$$r_{min} \leq r \leq r_{max}, \tag{2.26}$$

$$y_i^e, y_i^c \in \{0,1\} \quad \forall\ i\in\mathcal{U}, \tag{2.27}$$

$$n_e^{(a)}, n_c^{(a)} \in \mathbb{Z}_+ \quad \forall a\in\mathcal{A}. \tag{2.28}$$

Where $r^{(a1)}$ is the constant time unit fee for running application a, while $\gamma^{(a)} r$ is the extra cost to be paid by users of application a if they select the second deployment, in which case the platform will have to allocate edge or cloud resources to serve the computational load of the second partition of the DNN. More specifically, $\gamma^{(a)}$ is a positive real number and a fixed parameter, while r is also a positive real number, but it's a platform decision variable.

Figure 2.4 shows an example of the platform profit as a function of r, which is a piecewise function (see section 4.2), in a system with 200 users and 4 applications. The system parameters were generated with uniform random distributions on predefined intervals. Each point of the plot was obtained by solving the Platform optimization problem for a fixed value of r, that was varied in the range $[2, 20]$ with an increment of 0.2 at each step. We observe that clearly the problem is not convex and the platform profit function exhibits an irregular behaviour due to the change in users' decision.

2.2.2. 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

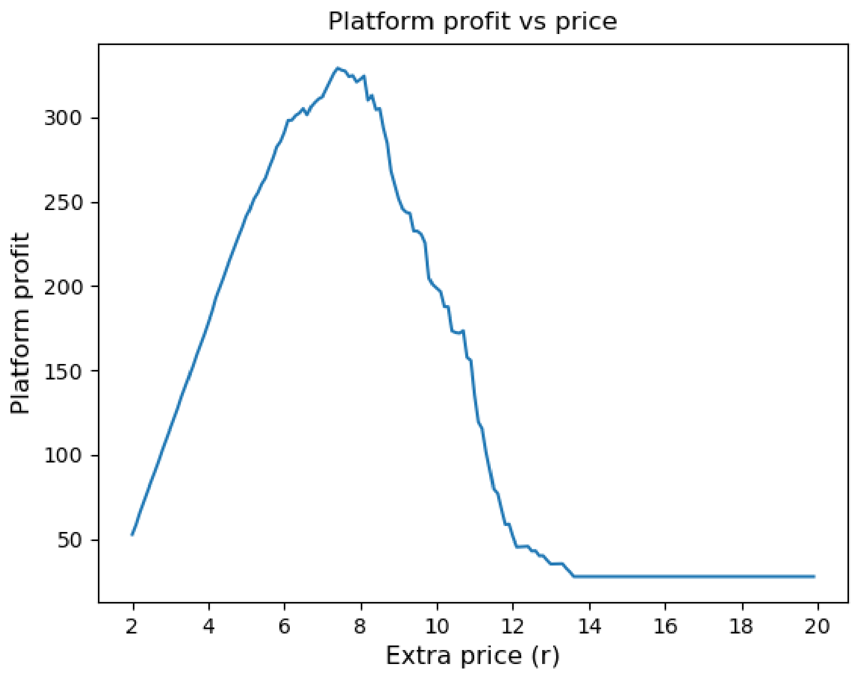


Figure 2.4: Platform profit function for 200 users and 4 applications.

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

$$\max_{x_i^{(k)}} \sum_{k=1}^{2} x_i^{(k)} \left(U_i - T_i \left(\alpha_i r^{(ak)} + (1-\alpha_i)\beta_i p_i^{(k)} \lambda^{(a)} T_i + x_i^{(2)} \zeta_i \delta^{(a)} \lambda^{(a)} \right) \right) \tag{2.29}$$

$$\text{Subject to:} \tag{2.30}$$

$$\sum_{k=1}^{2} x_i^{(k)} \leq 1, \tag{2.31}$$

$$\lambda^{(a)} T_i^2 \sum_{k=1}^{2} p_i^{(k)} x_i^{(k)} \leq \bar{E}_i, \tag{2.32}$$

$$\sum_{k=1}^{2} m^{(ak)} x_i^{(k)} \leq \bar{M}_i, \tag{2.33}$$

$$x_i^{(k)} \in \{0, 1\} \qquad \forall k = 1, 2. \tag{2.34}$$

We introduced users' utility U_i inspired by an incentive mechanisms like [16], 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:

Table 2.1: 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.
$t_i^{(k)}$	Binary variable to show if deployment k is convenient to run on user i device.
z_i	Binary variable to show if the cost for deployment 1 is less than the cost of deployment 2 for user-i.

- if the memory and energy of user's device are enough to run the first deployment and $U_i \geq C_i^{(1)}$[4], the following condition must hold ($\forall i \in \mathcal{U}^{(a)},\ \forall a \in \mathcal{A}$):

$$D_i^{(1)} < \bar{R}^{(a)}$$

- if the memory and energy of the device is enough to run the second deployment and $U_i \geq C_i^{(2)}$, this condition must hold:

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

For the reader's convenience, the variables and parameters of the model are summarized in Table 2.1 & Table 2.2.

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

Table 2.2: 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.
$\gamma^{(a)}$	Coefficient of r for computing the extra price for deployment 2 of application a.
T	Time horizon of platform resource allocation problem.
ζ_i	Coefficient to convert the data transfer size to monetary cost.

3 | 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.

As we mentioned in previous section, 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, for any given value of r, the platform can anticipate users' decision without contacting them by solving problem 2.29 in a centralized manner. This operation is trivial as it only requires to to check few conditions presented in algorithm 3.1.

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 $x_i^{(1)}, x_i^{(2)}$.

Given that the platform can predict users' behavior, 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:

$$\max_{r, n_e^{(a)}, n_c^{(a)}, y_i^e, y_i^c} P_e = \sum_{a \in \mathcal{A}} \sum_{i \in \mathcal{U}^{(a)}} \sum_{k=1}^{2} T_i r^{(ak)} x_i^{(k)} - T\left(c_e n_e + c_c n_c\right) \tag{3.1}$$

Subject to: (3.2)

$$\mathbf{x}_i^*(\mathbf{r}) = \arg\max_{\mathbf{x}_i} \{ \sum_{k=1}^{2} x_i^{(k)} \left(U_i - T_i \alpha_i r^{(ak)} + (1-\alpha_i)\beta_i p_i^{(k)} \lambda^{(a)} T_i + x_i^{(2)} \zeta_i \delta^{(a)} \lambda^{(a)} \right) : \tag{3.3}$$

$$\sum_{k=1}^{2} x_i^{(k)} \leq 1, \quad \lambda T_i^2 \sum_{k=1}^{2} p_i^{(k)} x_i^{(k)} \leq \bar{E}_i, \tag{3.4}$$

$$\sum_{k=1}^{2} m^{(ak)} x_i^{(k)} \leq \bar{M}_i, \tag{3.5}$$

$$x_i^{(k)} \in \{0,1\} \quad \text{for } k = 1,2\}, \qquad \forall \ i \in \mathcal{U}^{(a)}, \ \ \forall a \in \mathcal{A}. \tag{3.6}$$

$$\sum_{a \in \mathcal{A}} n_e^{(a)} \leq N_e, \tag{3.7}$$

$$\sum_{k=1}^{2} D_i^{(k)} x_i^{(k)} + \frac{\delta^{(a)} x_i^{(2)}}{B_i} + \frac{D_e^{(a)} x_i^{(2)} y_i^e}{1 - \frac{L_e^{(a)}}{n_e^{(a)}}} + \frac{D_c^{(a)} x_i^{(2)} y_i^c}{1 - \frac{L_c^{(a)}}{n_c^{(a)}}} \leq \bar{R}^{(a)} \quad \forall i \in \mathcal{U}^{(a)}, \ \forall a \in \mathcal{A}, \tag{3.8}$$

$$L_e^{(a)} < n_e^{(a)} \qquad \forall a \in \mathcal{A}, \tag{3.9}$$

$$L_c^{(a)} < n_c^{(a)} \qquad \forall a \in \mathcal{A}, \tag{3.10}$$

$$y_i^e + y_i^c = x_i^{(2)} \qquad \forall i \in \mathcal{U} \tag{3.11}$$

$$r^{(a2)} = r^{(1)} + \gamma^{(a)} r, \quad \gamma^{(a)} > 1 \quad \forall a \in \mathcal{A} \tag{3.12}$$

$$r_{min} \leq r \leq r_{max} \tag{3.13}$$

$$y_i^e, y_i^c \in \{0,1\} \qquad \forall \ i \in \mathcal{U}, \tag{3.14}$$

$$n_e^{(a)}, n_c^{(a)} \in \mathbb{Z}_+. \qquad \forall a \in \mathcal{A}. \tag{3.15}$$

We remark that the optimality constraint (3.3)-(3.6) can be replaced by a set of linear constraints by introducing few additional binary variables and parameters, whose definition is reported below:

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

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

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

Notice that $s_i^{(k)}$ and $t_i^{(1)}$ are parameters since they depend only on fixed parameters of user-i and his/her application. On the contrary, $t_i^{(2)}$ and z_i are binary variables, since

they depend on r, which we recall is a platform decision variable.
The linear constraints equivalent to (3.3)-(3.6) are shown in the following lines (where we denote the cost of running deployment k for user i as $C_i^{(k)}$):

$$
\begin{aligned}
&-(1-t_i^{(k)})M \leq U_i - C_i^{(k)} \leq t_i^{(k)}M \qquad \text{for } k=1,2 && (3.16)\\
&-(1-z_i)M \leq C_i^{(2)} - C_i^{(1)} \leq z_i M && (3.17)\\
&x_i^{(k)} \leq s_i^{(k)} t_i^{(k)} \qquad \text{for } k=1,2 && (3.18)\\
&x_i^{(1)} \geq s_i^{(1)} t_i^{(1)} + s_i^{(2)}(z_i^{(12)} - 1) && (3.19)\\
&x_i^{(2)} \geq s_i^{(2)} t_i^{(2)} - s_i^{(1)} z_i^{(12)} && (3.20)\\
&\sum_{k=1}^{2} x_i^{(k)} \leq 1 && (3.21)\\
&x_i^{(k)}, t_i^{(k)}, z_i^{(12)} \in \{0,1\} \quad \text{for } k=1,2 && (3.22)
\end{aligned}
$$

where M is a large enough positive real number.

The Stackelberg game (3.1)–(3.22) 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 (3.8). Therefore, in the next Section we propose an heuristic approach to solve the problem faster.
This method has already been presented in [16] for a similar problem. This document shows its adaptation to the framework of our model, where multiple applications are considered.

4 | 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 [16], which is used to solve a similar problem.

The heuristic approach is based on three main ingredients. First, Section 4.1 shows that, under the assumption of a fixed price, a solution close to the optimum for the edge provider resource allocation problem (i.e., the decisions for $n_e^{(a)}$ and $n_c^{(a)}$) can be identified by solving a convex sub problem. Secondly, in Section 4.2 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 4.3.

Finally, as the original contribution of this thesis, section 4.4 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 4.1 highlights the main steps we have just presented, that together frame our proposed approach.

4.1. Relaxed Problem with fixed price

In this section we show that the edge provider 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 considering new simplified response time constraints. This solution will then allow to solve the overall Stackelberg game quickly using a heuristic approach. To reach this goal, we first formulate the relaxed problem in Section 4.1.1 and then we solve the problem through KKT condition first by assuming a system with single application in Section 4.1.2 and then we extend our solution for a system with multiple application in Section 4.1.3. Finally, in section 4.1.4 we provide the algorithm to find the approximate optimal solution of problem Equation 4.4.

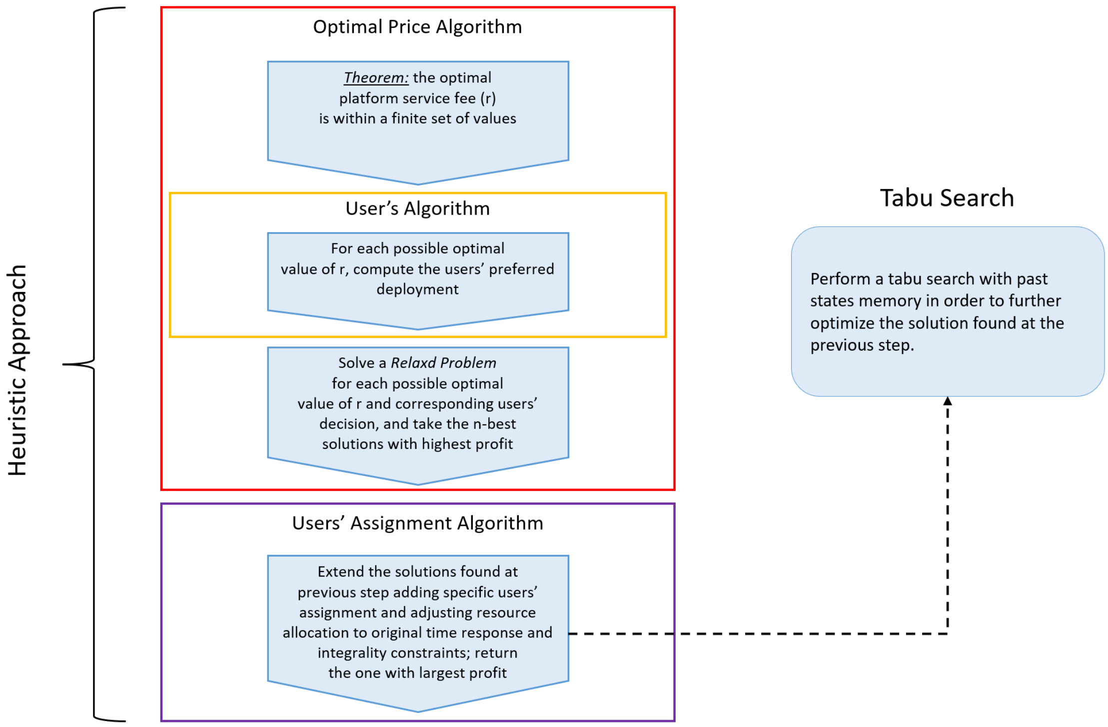


Figure 4.1: Flow chart of our proposed approach.

4.1.1. Relaxed Problem Formulation

As we already mentioned, thanks to the controller agent, the provider can obtain the users' parameters and find their desired deployment for a fixed price by solving the users optimization problem following algorithm 3.1.
Note that, the incoming load of each application $\Lambda^{(a)}$ (defined in 2.9) depends 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 provider 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:

$$\Lambda_e^{(a)} + \Lambda_c^{(a)} = \Lambda^{(a)} \tag{4.1}$$

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

$$\bar{\bar{R}}^{(a)} = \bar{R}^{(a)} - \mathbb{E}_a\left[D_i^{(2)} x_i^{(2)}\right] - \mathbb{E}_a\left[\frac{\delta^{(a)} x_i^{(2)}}{B_i}\right] \qquad \forall a \in \mathcal{A}, \tag{4.2}$$

where $\mathbb{E}_a\left[D_i^{(2)}x_i^{(2)}\right]$ denotes the expected demand time of running the first part of DNN 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 of the first part of DNN to the provider. The rationale behind (4.2) 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 (2.21) is redefined on the edge provider side as follows:

$$\frac{\Lambda_e^{(a)}}{\Lambda^{(a)}} \cdot \frac{D_e^{(a)}n_e^{(a)}}{n_e^{(a)} - D_e^{(a)}\Lambda_e^{(a)}} + \frac{\Lambda_c^{(a)}}{\Lambda^{(a)}} \cdot \frac{D_c^{(a)}n_c^{(a)}}{n_c^{(a)} - D_c^{(a)}\Lambda_c^{(a)}} \leq \bar{\bar{R}}^{(a)} \quad \forall a \in \mathcal{A}. \tag{4.3}$$

The following result can be demonstrated.

Theorem 4.1. *The response time constraints in* (4.3) *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 provider 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:

$$\min_{n_e^{(a)},n_c^{(a)},\Lambda_e^{(a)},\Lambda_c^{(a)}} \quad c_e n_e + c_c n_c \tag{4.4}$$

$$\text{Subject to:} \tag{4.5}$$

$$\sum_{a\in\mathcal{A}} n_e^{(a)} \leq N_e \tag{4.6}$$

$$\forall a \in \mathcal{A}: \tag{4.7}$$

$$\frac{\Lambda_e^{(a)}}{\Lambda^{(a)}} \frac{D_e^{(a)}n_e^{(a)}}{n_e^{(a)} - D_e^{(a)}\Lambda_e^{(a)}} + \frac{\Lambda_c^{(a)}}{\Lambda^{(a)}} \frac{D_c^{(a)}n_c^{(a)}}{n_c^{(a)} - D_c^{(a)}\Lambda_c^{(a)}} \leq \bar{\bar{R}}^{(a)} \tag{4.8}$$

$$D_e^{(a)}\Lambda_e^{(a)} < n_e^{(a)} \tag{4.9}$$

$$D_c^{(a)}\Lambda_c^{(a)} < n_c^{(a)} \tag{4.10}$$

$$\Lambda_e^{(a)} + \Lambda_c^{(a)} = \Lambda^{(a)}, \tag{4.11}$$

$$n_e^{(a)}, n_c^{(a)} \geq 0 \tag{4.12}$$

$$\Lambda_e^{(a)}, \Lambda_c^{(a)} \geq 0. \tag{4.13}$$

Where as we already mentioned, we defined: $n_c = \sum_{a\in\mathcal{A}} n_c^{(a)}$ and $n_e = \sum_{a\in\mathcal{A}} n_e^{(a)}$. 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 start by proving, in section 4.1.2 that an analytical solution for the problem Equation 4.4 can be found in the case of a system with one application only. Next, in section 4.1.3 we are going to extend this result to find an approximate solution for the general case of a system with multiple applications.

4.1.2. Single Application System

Now we provide an analytical solution to problem rely on KKT condition. Equation 4.4 in the case of a system with one application, i.e. $\mathcal{A} = \{1\}$. This result is presented in the following Theorem, where, for simplicity, we omit superscript a.

Theorem 4.2. *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 provider 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. □

4.1.3. Multiple Applications System

In this section we present our approach for solving the relaxed problem Equation 4.4. Our approach is based on the assumption of cheaper edge servers, which means that it's in the provider interest to allocate users' load in the edge resources as far as they are available, while resorting to cloud VMs only when it is strictly necessary. Therefore, our proposed approach consists in assigning the load of every application one by one to the edge, until the edge nodes are saturated. Then, the remaining applications will 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.
In order to implement our approach, we need to be able to compute the optimal number of edge nodes and cloud VMs to assign to each application. These results are shown in *Only Edge* and *Only Cloud* scenarios as following:

Only Edge Scenario

In this scenario we compute the optimal number of servers for each application under the assumption of *only edge*, 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 Equation 4.4 becomes:

$$\min_{n_e^{(a)}} c_e n_e \tag{4.14}$$

$$\text{s.t.} \quad \sum_{a\in\mathcal{A}} n_e^{(a)} \leq N_e \tag{4.15}$$

$$\frac{D_e^{(a)} n_e^{(a)}}{n_e^{(a)} - D_e^{(a)}\Lambda^{(a)}} \leq \bar{\bar{R}}^{(a)} \qquad \forall a \in \mathcal{A}, \tag{4.16}$$

$$n_e^{(a)} - D_e^{(a)}\Lambda^{(a)} > 0 \qquad \forall a \in \mathcal{A} \tag{4.17}$$

Theorem 4.3. *If*

$$\sum_{a\in\mathcal{A}} \frac{\bar{\bar{R}}^{(a)} D_e^{(a)} \Lambda^{(a)}}{\bar{\bar{R}}^{(a)} - D_e^{(a)}} \leq N_e, \tag{4.18}$$

the solution of problem 4.14 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. □

Only Cloud Scenario

In this scenario, we compute the optimal number of servers for each application under the assumption of *only cloud*, 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 Equation 4.4 becomes:

$$\min_{n_c^{(a)}} c_c n_c \tag{4.19}$$

$$\text{s.t.} \quad \frac{D_c^{(a)} n_c^{(a)}}{n_c^{(a)} - D_c^{(a)}\Lambda^{(a)}} \leq \bar{\bar{R}}^{(a)} \tag{4.20}$$

$$n_c^{(a)} - D_c^{(a)}\Lambda^{(a)} > 0 \qquad \forall a \in \mathcal{A} \tag{4.21}$$

The solution is given by:

Theorem 4.4. *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. □

4.1.4. Relaxed Problem Solution

In this section we provide a detailed explanation of our proposed approach for solving problem Equation 4.4. This method is outlined in algorithm 4.1 that receives as input the load and the order of priority for each application. The idea is to assign the applications given the order to the edge as far as the edge servers are available, because of cheaper edge nodes compare with cloud, and then the rest of application will be assigned to the cloud.

We first compute the optimal number of edge resources ($n_e^{(a)}$) for all applications under

Algorithm 4.1 Heuristic solution to relaxed problem

1: **procedure** ComputeOptimalVMs(order, $\{\Lambda^{(a)}\}_{a\in\mathcal{A}}$)
2: $\quad n_e^{(a)} \leftarrow$ Solution through Theorem 4.3
3: $\quad$ **if** $\sum_{a\in\mathcal{A}} n_e^{(a)} \le N_e$ **then**
4: $\quad\quad \Lambda_e^{(a)} \leftarrow \Lambda^{(a)}, \quad \Lambda_c^{(a)} \leftarrow 0, \quad n_c^{(a)} \leftarrow 0 \quad \forall a \in \mathcal{A}$
5: $\quad$ **else**
6: $\quad\quad n_c^{(a)} \leftarrow$ Solution through Theorem 4.4
7: $\quad\quad UsedEdgeVMs \leftarrow \sum_{a\in\mathcal{A}} n_e^{(a)},\ idx \leftarrow |\mathcal{A}|$
8: $\quad\quad$ **while** $UsedEdgeVMs > N_e$ **do**
9: $\quad\quad\quad idx \leftarrow idx - 1,\ \tilde{a} \leftarrow order[idx]$
10: $\quad\quad\quad UsedEdgeVMs \leftarrow UsedEdgeVMs - n_e^{(\tilde{a})}$
11: $\quad\quad\quad \Lambda_c^{(\tilde{a})} \leftarrow \Lambda^{(\tilde{a})},\ n_e^{(\tilde{a})} \leftarrow 0$
12: $\quad\quad$ **end while**
13: $\quad\quad \bar{N}_e \leftarrow N_e - \sum_{a\in\mathcal{A}} n_e^{(a)}$
14: $\quad\quad n_e^{(\tilde{a})}, \Lambda_e^{(\tilde{a})}, n_c^{(app)}, \Lambda_c^{(\tilde{a})} \leftarrow$ Solution through Theorem 4.2, given $r, x, \bar{N}_e$
15: $\quad\quad$ **for** $j \leftarrow 1$ **to** $idx - 1$ **do**
16: $\quad\quad\quad a \leftarrow order[j]$
17: $\quad\quad\quad \Lambda_e^{(a)} \leftarrow \Lambda^{(a)},\ n_c^{(a)} \leftarrow 0$
18: $\quad\quad$ **end for**
19: $\quad$ **end if**
20: **end procedure**

OnlyEdge scenario given by Theorem 4.3 (line 2). If the maximum number of available nodes in the edge is enough to cover all applications, we simply assign all applications to the edge and the optimal solution is found (lines 3-5). Otherwise, we compute $n_c^{(a)}$ for all applications through Theorem 4.4 (line 6) and start to saturate the edge servers first, and then assign the rest of deployments to the cloud. To reach this goal, given the order of applications ($order \in Sym(\mathcal{A})^1$), we assign $n_e^{(1)}, n_e^{(2)}, \ldots, n_e^{(m)}$ number of edge nodes

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

to the applications 1 to m until the remaining edge resources are not enough to cover the load of the m^{th} application (lines 7-12). At this point, we divide the load of the m^{th} application into two parts, one will be served in the edge using the remaining edge nodes, and the other part will be served 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 4.2 (lines 13-14). Now that the edge resources are saturated, we assign all the remaining applications to the cloud, by setting $n_e^{(a)} = 0$ for all those of applications (line 15-19).

4.2. 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 the permutations of applications. In reality, the extra cost for the second deployment belongs to the set of the platform decision variables, meaning that the total price of deployments, as well as the computational load of applications are not fixed parameters and depend on the extra cost of the second deployment. 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 proposed in previous section 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, a close up frame of the profit function related to 20 users and 2 applications system is shown in figure 4.2. This plot shows that, the profit function of the provider is a piecewise function, and 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 selected deployment. 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, 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:

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

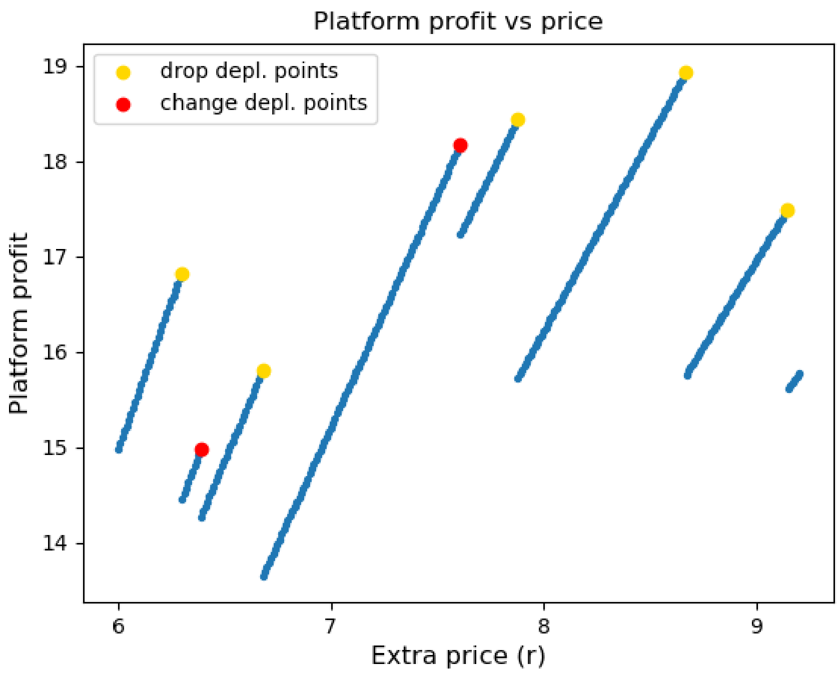


Figure 4.2: Platform profit profile with 20 users and 2 applications.

Note that since the cost of running the applications with the first deployment is independent of r, there are no dropping points for deployment one.

- **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 first and second deployment 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 4.5. *The optimal price solution lays at one of the points of discontinuity or in a right neighborhood of them.*

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.
Notice that whenever the value r is equal to a changing deployment point there is no unique optimal solution for the corresponding user (assuming that all energy and memory constraints are satisfied), since, by definition, the changing deployment point is the value for r that makes the costs for both deployments equal. Similarly, when r is at a dropping deployment point, running the second deployment or dropping from the system have the same utility for the user. In algorithm 3.1 we assumed that whenever one of these two cases happen, the user will always choose to run the second deployment. This assumption essentially means that the platform profit is left continuous with respect to the extra price r; equivalently, the intervals in which the users' decision are fixed are given by:

$$(\mathrm{disc}_i, \mathrm{disc}_j],$$

where disc_i denotes a changing or a dropping deployment point for some user $i \in \mathcal{U}$. Therefore, 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, since they lay at the end of the interval in which users' decisions are fixed. However, notice that we have constraint 3.13 on the variable r, meaning that some discontinuity points might not be admissible for the problem. For this reason, we also need to check the the right neighborhood of the discontinuity points. Hence, the set of r-values candidates is determined by:

$$\mathcal{R} = \{\mathrm{change}_i,\ \mathrm{drop}_i,\ \mathrm{change}_i + \epsilon,\ \mathrm{drop}_i + \epsilon\ \forall i \in \mathcal{U}\} \tag{4.22}$$

□

4.2.1. Optimal Price Algorithm

At this point we know that the optimal price r is within a finite set of values given by Theorem 4.5. In order to identify the optimal price r we follow the approach outlined in algorithm 4.2. Namely, after receiving as input all users' parameters (line1), the platform computes the dropping and changing deployment points for each user (lines 3-7) and adds them to $\mathcal{R}$. For each admissible value of $r \in \mathcal{R}$, we solve each user's problem through algorithm 3.1 (lines 8-10); once the users' decision are known, we compute the load for each application according to 2.9 (lines 11), 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 (lines 12-14), through algorithm 4.1 and we append the corresponding solution to a previously initialized list (lines 15-16). Finally, we sort the solution-list by profit and we return the n best elements with the highest profit (lines 20-21), where n is an hyperparameter.
The reason why we do not consider just the elite 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 3.1. Therefore, by selecting more elite solutions, we have an higher chance that one of them will approximate better the optimal solution of the complete problem.

Algorithm 4.2 Optimal prices algorithm

Input: N, users' parameters, $r_{\min}$, $r_{\max}$, $\gamma^{(a)}$, $\delta^{(a)}$, $\lambda^{(a)}$, $r^{(a1)}$ $\forall a \in \mathcal{A}$, orders
Initialization: *Solutions* $\leftarrow []$
for $i \leftarrow 1$ **to** N **do**
  $\text{change}_i = \frac{(1-\alpha_i)\beta_i\lambda^{(a)}T_i(p_i^{(1)}-p_i^{(2)})-\zeta_i\lambda^{(a)}\delta^{(a)}}{\alpha_i\gamma^{(a)}}$.
  $\text{drop}_i = \frac{U_i-T_i[\alpha_i r^{(a1)}-(1-\alpha_i)\beta_i p_i^{(2)}\lambda^{(a)}T_i-\zeta_i\delta^{(a)}\lambda^{(a)}]}{\alpha_i\gamma^{(a)}}$
  Add $\text{drop}_i,\ \text{change}_i,\ \text{drop}_i+\epsilon,\ \text{change}_i+\epsilon$ to $\mathcal{R}$
end for
for r **in** $\mathcal{R}$ **do**
  if $r_{min} \leq r \leq r_{max}$ **then**
    $x_i \leftarrow$ Solve user-i problem given r, $\forall i \in \mathcal{U}$ through algorithm 3.1
    $\Lambda^{(a)} \leftarrow \sum_{i \in \mathcal{U}^{(a)}} \lambda^{(a)} x_i^{(2)} \quad \forall a \in \mathcal{A}$
    orders $\leftarrow$ Permutations of applications
    for order **in** orders **do**
      ComputeOptimalVMs(order,$\{\Lambda^{(a)}\}_{a\in\mathcal{A}}$)
      $P \leftarrow$ 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*

4.3. Users' Resource Assignment

At this point we have a set of *elite solutions* that were found by solving the Relaxed problem of section 4.1 applied on the points of discontinuity. These solutions however, are not admissible solutions in the sense of the complete Stackelberg problem Equation 3.1. Indeed, they lack the specific users' assignment variables y_i^e, y_i^c and the amount of resources allocated was computed considering simplified response time constraints and neglecting the integrality of $n_e^{(a)}$ and $n_c^{(a)}$.
Our purpose is now to extend these approximate 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 y_i^c. Moreover, we have to slightly modify the number of edge and cloud resources assigned to each application, so that all the original response time and integrality constraints are satisfied. Once we have a set of admissible solutions the platform will choose the one with the greatest profit. This solution will be the final output of the heuristic approach.
The process of extending the partial solutions is presented in algorithm 4.3, where we use the following notations:

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

$$LD_i = \bar{R}^{(a)} - D_i^{(2)} x_i^{(2)} - \frac{\delta^{(a)} x_i^{(a)}}{B_i} \qquad \forall i \in \mathcal{U}^{(a)},\ \forall a \in \mathcal{A}. \tag{4.23}$$

- $\text{List}_e^{(a)}$, $\text{List}_c^{(a)}$ 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.
- $\mathcal{U}_2^{(a)} = \{ i \in \mathcal{U}^{(a)},\ x_i^{(2)} = 1 \}$ is the set of users of application a, opting for the second deployment.
- We define the violation of user i as follows:

$$V[i] = D_i^{(2)} + \frac{\delta^{(a)}}{B_i} + \frac{D_e^{(2)}}{1 - L_e^{(a)}/n_e^{(a)}} - \bar{R}^{(a)}, \quad \forall i \in \mathcal{U}_2^{(a)}. \tag{4.24}$$

  If $V[i]$ is positive, it means that the response time constraint is not satisfied for user-i.

The algorithm receives as input the set of *elite solutions* and the response time constraints $R^{(a)}$; the best profit and best solution are initialized to null values (lines 1-2). Next, we start looping on the set of *elite solutions*: we extract from the current element the values for $r, x, n_e^{(a)}, n_c^{(a)}, \Lambda_e^{(a)}, \Lambda_c^a$, and for each application we run procedure ComputeEdgeServers(k) (lines 3-7). This function basically adjusts the number of edge nodes assigned to the current application so that the original time response constraints for all users are satisfied; a complete explanation is provided in section 4.3.1.
Once we have the new number of edge nodes for each application, we check if the sum across all applications exceeds the maximum available number of edge resources (line 8), we compute the maximum number of applications that can be fit in the edge according to the given order of the current elite solution (lines 9-13). Next, we change users' assignment for the first application whose load doesn't fit in the edge, by running procedure MoveUser(i,a) where i is the user with the highest local delay, until the remaining edge nodes are enough to cover the load of the application (lines 14-18).
Now that all edge resources are saturated, we move all the remaining applications to the cloud, by running procedure MoveApplication(a) (lines 19-23). Then, for each application we compute the required number of cloud VMs to satisfy all response time constraints through procedure ComputeCloudVMs(a) (lines 24-26). These corresponding procedures are detailed in section 4.3.1. Finally, the solution is now admissible with respect to the problem Equation 2.18, we compute its profit (line 27) and we update the two variables best profit and best solutions (lines 28-32). In the end, the best solution is returned (line 33).

4.3.1. Procedures

This section provides a detail explanation of procedures used in Algorithm 4.3.

CountEdgeVMs(k)

This procedure first checks if the application is fully assigned to the edge (line 2), if this is the case, all users opting for the second deployment are assigned to the edge (line 3).

Then, it computes the violation of all users (line 4), if they're all negative the returned output is just the smallest integer greater than the initial number of edge nodes (lines 5-6). On the contrary, if there is some user with positive violation, the new number of edge nodes is the minimum integer that make the time response constraint for the user with maximum violation satisfied (lines 7-10).
If instead the the current application has a positive number of cloud VMs assigned, we sort the users by local delay in increasing order (lines 12-14), and split them among edge and cloud. More specifically, we serve all users with position below $\frac{\Lambda_c^{(a)}}{\lambda^{(a)}} + 1$ to the cloud, and the remaining users to the edge (lines 15-18). This is because we want to keep the equilibrium between edge and cloud load given by $\Lambda_c^{(a)}$ and $\Lambda_e^{(a)}$. Finally, the new number of edge nodes is computed as the minimum integer that satisfies the response time constraint of the user with maximum local delay served in the edge.

MoveUser(i,a)

This function receives as input the user that will be moved from edge to cloud and his/her application. The variables $L_e^{(a)}$ and $L_c^{(a)}$ are updated (lines 2-3) and the new number of edge nodes assigned to application a is computed, as the minimum integer that satisfies the response time constraint for the user with the highest local delay in the edge.

MoveApplication(a)

This function receives as input the application that will be served entirely in the cloud. All users are assigned to the cloud (line 2) and the variables $L_e^{(a)}, L_c^{(a)}, \Lambda_e^{(a)}, \Lambda_c^{(a)}$ are updated accordingly. The number of edge resources is set to 0 since there are no users served in the edge.

ComputeCloudVMs(a)

This function counts the required cloud VMs for the application in input to satisfy the response time constraints for all users. If the cloud load of the application is positive, the number of cloud VMs is computed as the minimum integer that satisfies the response time constraint for the user with the highest local delay.

Algorithm 4.3 Users' resource assignment

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

---

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

---

procedure $\textsc{MoveApplication}$(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**
      $\bar{n}_e^{(a)} \leftarrow \lceil n_e^{(a)} \rceil$, $\bar{n}_c^{(a)} \leftarrow 0$
    else
      $j \leftarrow$ user in $\mathcal{U}_2^{(a)}$ with maximum V
      $LD_j \leftarrow \bar{R} - D_j^{(a)} - \frac{\delta^{(a)}}{B_j}$
      $\bar{n}_e^{(a)} \leftarrow \lceil L_e^{(a)} \frac{LD_j}{LD_j - D_e^{(a)}} \rceil$
    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
    $j \leftarrow \frac{\Lambda_c^{(a)}}{\lambda^a} + 1$
    $y_i^c \leftarrow 1, \qquad \forall i \in \mathcal{U}_2^{(a)} : i < j$
    $y_i^e \leftarrow 1, \qquad \forall i \in \mathcal{U}_2^{(a)} : i \geq j$
    $LD_j \leftarrow \bar{R} - D_j^{(a)} - \frac{\delta^{(a)}}{B_j}$
    $\bar{n}_e^{(a)} \leftarrow \lceil L_e^{(a)} \frac{LD_j}{LD_j - D_e^{(a)}} \rceil$
  end if
end procedure

procedure COMPUTECLOUDVMS(a)
  if $\Lambda_c^{(a)} > 0$ **then**
    $j \leftarrow \mathrm{List}_c^{(a)}[0]$
    $LD_j \leftarrow \bar{R} - D_j^{(a)} - \frac{\delta^{(a)}}{B_j}$
    $\bar{n}_c^{(a)} \leftarrow \lceil L_c^{(a)} \frac{LD_j}{LD_j - D_c^{(a)}} \rceil$
  end if
end procedure

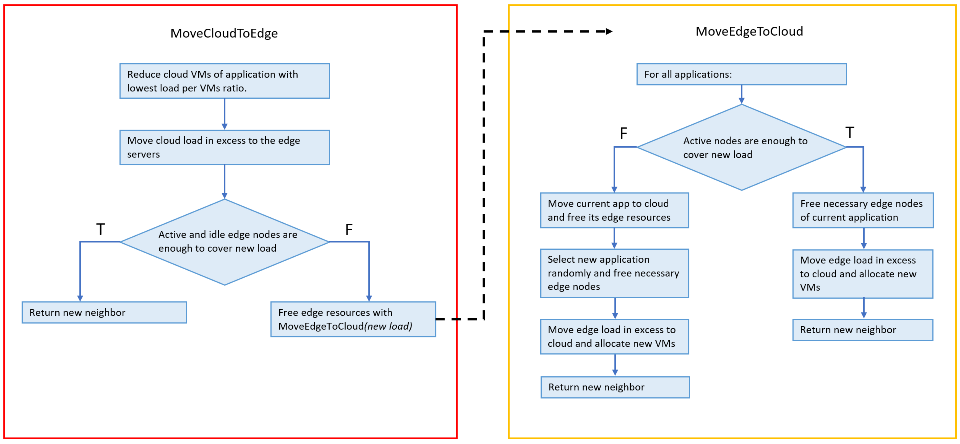


Figure 4.3: Flowchart of neighbor generation process

4.4. Tabu Search

After applying the proposed heuristic approach we have an approximate optimal solution based on the resource allocation of the relaxed problem (4.4). We recall that the relaxed problem considers a convex and simplified response time 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 applications 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.
In the following sections, generating the neighbors of a solution (Section 4.4.1) tabu search algorithm (Section 4.4.2) are described in detail.

4.4.1. Generating the Neighbors

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, by moving users from edge to cloud or vice versa. This process basically consists of two algorithms: 4.4 and 4.5, that we present in the following sections. Figure 4.3 outlines the main steps of these algorithms.

Move Cloud to Edge Algorithm

This algorithm 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 application a (line 2), and according to that lowest load, compute the number of users that are currently

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.
Hence, we take the user with the highest local delay in the cloud (line 5) and compute the maximum number of users that can be served in the cloud after we reduced the number of cloud VMs allocated for application a (line 6). The difference between the current number of users served in the cloud and the maximum number of users that can be served in the cloud after resource reduction is the number of cloud users that need to be moved in the edge (line 7). Once we have that, we take the users with the lowest LD among those currently served in the cloud and we move them in the edge by updating the relevant variables (lines 8-9). The reason why we chose to move the users with the lowest local delay is that we want the load transferred to the edge to be as light as possible, so that we have an higher change that the active or idle nodes are enough to cover it.
Indeed, now that some users have been transferred to the edge, we need to check if the edge nodes currently allocated to application a are enough to satisfy the load and time response constraints. We do that by computing the minimum amount of edge nodes necessary to satisfy the response time constraint for the user in the edge with the highest local delay (lines 10-11): the difference between this number and the currently allocated number of edge nodes will be the number of edge nodes that we need to add to application a so that all constraints are satisfied (line 12). If this difference is less or equal to 0 we don't need to take any action, and we return the new generated neighbor (lines 14-15). If instead, it is strictly greater than 0 and less than the number of idle nodes we use them to cover the difference and return the new generated neighbor (lines 16-19). Finally, if the idle nodes are not enough to cover the difference we call algorithm 4.5 to free the necessary resources from other applications, and return the new generated neighbor (lines 20-23).

Move Edge to Cloud Algorithm

In this section we explain in detail algorithm 4.5 which is used in algorithm 4.4 to move one or more applications (a') from edge to cloud in order to release the required resource ($\text{diff\_}n_e^{(a)}$) in edge to serve application a.
The algorithm receives as input all the variables and parameters regarding the current solution, and the number of edge resources that need to be freed for application a (line 1). An empty list is initialized to stored all the generated neighbors (line 2). Then, we loop on all applications with some active nodes allocated (line 3): we compute the number of users served in the edge (line 4) and we check if the edge nodes allocated are enough to cover the number of edge resources that we want to free for application a (line 5). If this is the case, we take the user with the highest local delay in the edge (line 6) and we compute the maximum number of users that can be served in the edge for the current application a' if we drop the allocated edge resources by $\text{diff\_}n_e$ (lines 6-7). Once we have that, we know how many users we need to move to cloud (line 8). Once again, we move the users with lowest local in the edge to cloud (line 9) and update all the relevant solution variables (lines 10-11). Then, we need to compute the new amount of resources to serve the new cloud users; to do that, we take the user with highest local delay in the cloud and compute the new number of cloud VMs for application a' accordingly (lines

12-13). After that we append the new generated neighbor to the previously initialized neighbors list.
If the condition at line 5 is not satisfied, we move application a' in the cloud by running algorithms 1 and 1 (lines 16-17) and we randomly select another application a'' to free the remaining resources needed for application a (line 18). At this point we repeat the same process presented in lines 6-14 but for application a'' (lines 19-28).
Finally, after looping on all applications, we return the list of neighbors (lines 29-31).

Algorithm 4.4 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**
    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 $\text{diff\_}n_e^{(a)}$ idle edge nodes
    $\text{Idle}_e = \text{Idle}_e - \text{diff\_}n_e^{(a)}$
    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*

4.4.2. Tabu Search Algorithm

Algorithm 4.6 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 must be performed by the algorithm (line 1). At the start, current solution and best solution are set equal to the initial solution (line 2). 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. More specifically, the attributes of the solution that were saved in the tabu list are:

Algorithm 4.5 MoveEdgeToCloud

Input: $n_c^{(a)}, n_e^{(a)}, y^e, y^c, x_i^{(a)}, LD_i^{(a)}, D_e^{(a)}, D_c^{(a)}, L_e^{(a)}, L_c^{(a)}$ $List_e^{(a)}$, $List_c^{(a)}$, 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 $n_e^{(a')} \geq$ 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 lowest 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
    $MoveApplication(a')$
    $ComputeCloudVMs(a')$
    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 $neighbor\_list$

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

The algorithm starts by generating all feasible neighbors of current solution ($N(CurrentSolution)$) by applying the procedures mentioned in subsection 4.4.1. 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, and the algorithm stops when $MaxIter$ has been reached and returns the best solution.

Algorithm 4.6 Tabu Search Algorithm

```
1:  Input: InitialSolution, TabuSize, MaxIter
2:  Initialization: BestSolution, CurrentSolution ← InitialSolution, TabuList ← ∅
3:  for i ← 1toMaxIter do
4:      Neighbors ← Generate all feasible neighbors of CurrentSolution (Section 4.4.1)
5:      while True do
6:          if all Neighbors ∈ TabuList then
7:              return BestSolution
8:          end if
9:          neighbor ← SelectNeighbor(Neighbors, method)
10:         if neighbor ∈ TabuList then
11:             Remove neighbor from Neighbors
12:             neighbor ← SelectNeighbor(Neighbors, method)
13:         else
14:             append neighbor to TabuList
15:             if Len(TabuList) > TabuSize then
16:                 Remove first item from TabuList
17:             end if
18:             CurrentSolution ← neighbor
19:             if Profit(CurrentSolution) > Profit(BestSolution) then
20:                 BestSolution ← CurrentSolution
21:             end if
22:             Break
23:         end if
24:     end while
25: end for
26: return BestSolution
```

5 | 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 5.1;
- class `Application`, presented in section 5.2;
- class `ResourceDistribution`, presented in section 5.3;
- class `PartialSolution`, presented in section 5.4;
- class `Solution`, presented in section 5.5.
- class `Platform`, presented in section 5.6;

Table 5.1 provides an overview of the class structure, highlighting the main features and inheritance dependencies. The source code can be found in GitHub.[1].

5.1. 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.

5.2. 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.

[1] https://github.com/EttoreBusani/master-thesis.git

5.3. 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 4.2, where the integrality assumption of the number edge and cloud resources is relaxed (type `double`), and Algorithm 4.3, where the original constraints are satisfied (type `int`).

5.4. 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 4.1.

5.5. 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.

5.6. 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[2]:

- `algorithm2` : implementation of Optimal Price Algorithm 4.2.
- `ComputeOptimalVMs` : implementation of Procedure 4.1.
- `algorithm3` : implementation of User's Resource Assignment Algorithm 4.3.
- `CountEdgeVMs` : implementation of procedure 1.
- `algorithm4` : implementation of Tabu Seach Algorithm 4.6
- `MoveCloudToEdge` : implementation of Algorithm 4.4.
- `MoveEdgeToCloud` : implementation of Algorithm 4.5

[2]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 5.1: Class Diagram

```
class User
- user parameters
+ unsigned ID
+ {x_1, x_2} algorithm1(r)
```

```
class Application
+ string name
+ application parameters
```

```
class Platform
- map<unsigned,User> users
- map<string,shared_ptr<Application» apps
- map<string,vector<unsigned» userMap
- platform parameters

+ set<PartialSolution<double» algorithm2
+ Solution algorithm3(EliteSolutions)
+ Solution algorithm4(InitialSolution)
+ Solution heuristicApproach
```

```
template<typename T>
class ResourceDistribution
+ map<string,T> edgeServers
+ map<string,T> cloudServers
+ map<string,double> edgeLoads
+ map<string,double> cloudLoads
```

↓

```
template<typename T>
class PartialSolution: public ResourceDistribution<T>
+ map<unsigned,bool> userFirstDepl
+ map<unsigned,bool> userSecondDelp
+ double r
+ double profit
```

↓

```
class Solution: public PartialSolution<int>
+ map<unsigned,bool> userEdgeAllocation
+ map<unsigned,bool> userCloudAllocation
+ map<string,double> L_e
+ map<string,double> L_c
```

6 | Experimental Analysis

In this Chapter we show the results of our experimental analysis where we tested our algorithms and their implementation.
The 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 6.2.
- We also compared the quality of the results given by the heuristic approach with the optimal solution found by BARON [1], which is a commercial software tool capable of solving MINLP problems. This experiment is discussed in Section 6.3.
- Finally, in Section 6.4, 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 6.1.
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.

6.1. System setup

All experiments use the same system setup that we present in this section, with the only exception of the number of users and applications, which is specified upon presentation of each experiment.
The parameters used to generate the system are listed in table 6.1. The values for $\bar{R}^{(a)}, \lambda^{(a)}, \delta^{(a)}, m^{(ak)}$ are based on the YOLOv4 object recognition neural network proposed in [3]. Edge and cloud side execution time ($D_e^{(a)}, D_c^{(a)}$) are inspired by [6], where we scaled by a factor of 40 the computational time of edge servers to more realistically reproduce the performance of a GPU-based server used for AI applications.
In [6] the neural network was profiled on real user-side hardware, a Jetson TX2 Edge device (Processor: 6 Core Denver ARM A57, GPU: 256 Core Pascal – 1.3 TFLOPS (FP16)); inspired by this we set the user side execution time for the whole application ($D_i^{(1)}$) proportional to $\bar{R}^{(a)}$ and add a noisy term to create heterogeneity. The execution time for the partitioned application is proportional to the time for the full application considering

a neural network splitting point proposed in [3]. The energy consumption on users' device for first and second deployment $p_i^{(1)}, p_i^{(2)}$ is proportional to the corresponding execution time. The values of parameters B_i and β_i are based on 5G networks and electricity cost in Italy in 2022 ([12], [2]). The memory (M_i) and energy (E_i) maximum capacity of users' device is set in such a way that all users have enough memory to run the full application, while 77% of users, on average, won't have enough energy to run the full application. The cloud VM cost per second for the platform (c_c) is the one for the virtual machines of AWS[1] and Azure[2], while the cost for edge nodes (c_e) is a fraction of it, since they are owned by the platform.
The number of users and applications vary, and will be used as a stress parameter to increase the complexity of the problem.

6.2. Heuristic approach scalability

Our first analysis consists in measuring the time required by our proposed approach to reach its optimal solution. We recall that our approach is composed of 3 main algorithms (see figure 4.1): Optimal Price Algorithm (4.2), User's Algorithm (3.1) and Users' Assignment Algorithm (4.3). The hyperparameters of the algorithms are:

- n: which is the size of the *EliteSoluions* set; namely, the number of solutions of the Relaxed Problem returned by Optimal Price Algorithm, which are then passed to Users' Assignment Algorithm.
- *orders*: which is the vector containing the applications-orders which are tested by the Optimal Price Algorithm when solving each instance of the Relaxed problem.

In this experiment we set $n = 10$, while *orders* is a subset of $Sym(\mathcal{A})$ taken following a method explained in Appendix A.5 (here we use $k = 2$).
We run the heuristic approach for different choices of the number of users and number of applications, which are the main factors of the computational complexity of 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 (*orders*) among ranks, so that the computational load of Optimal Price Algorithm is reduced, allowing each rank to reach its optimal solution faster. The results are shown in Figure 6.1, 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 polynomial 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.

[1] https://aws.amazon.com/ec2/pricing/
[2] 5https://azure.microsoft.com/en-us/pricing/details/virtual-machines/windows/

Table 6.1: 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_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.

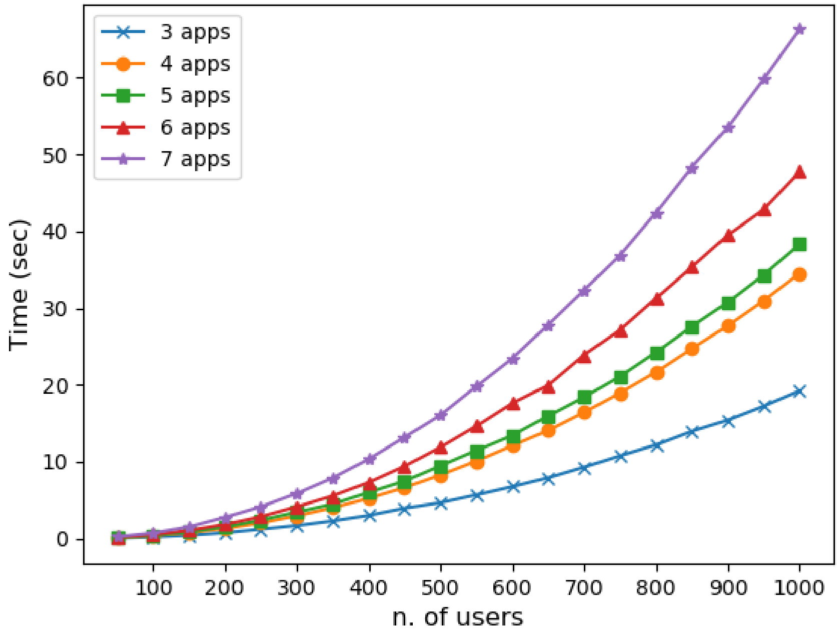


Figure 6.1: Time required for heuristic approach to reach its optimal solution.

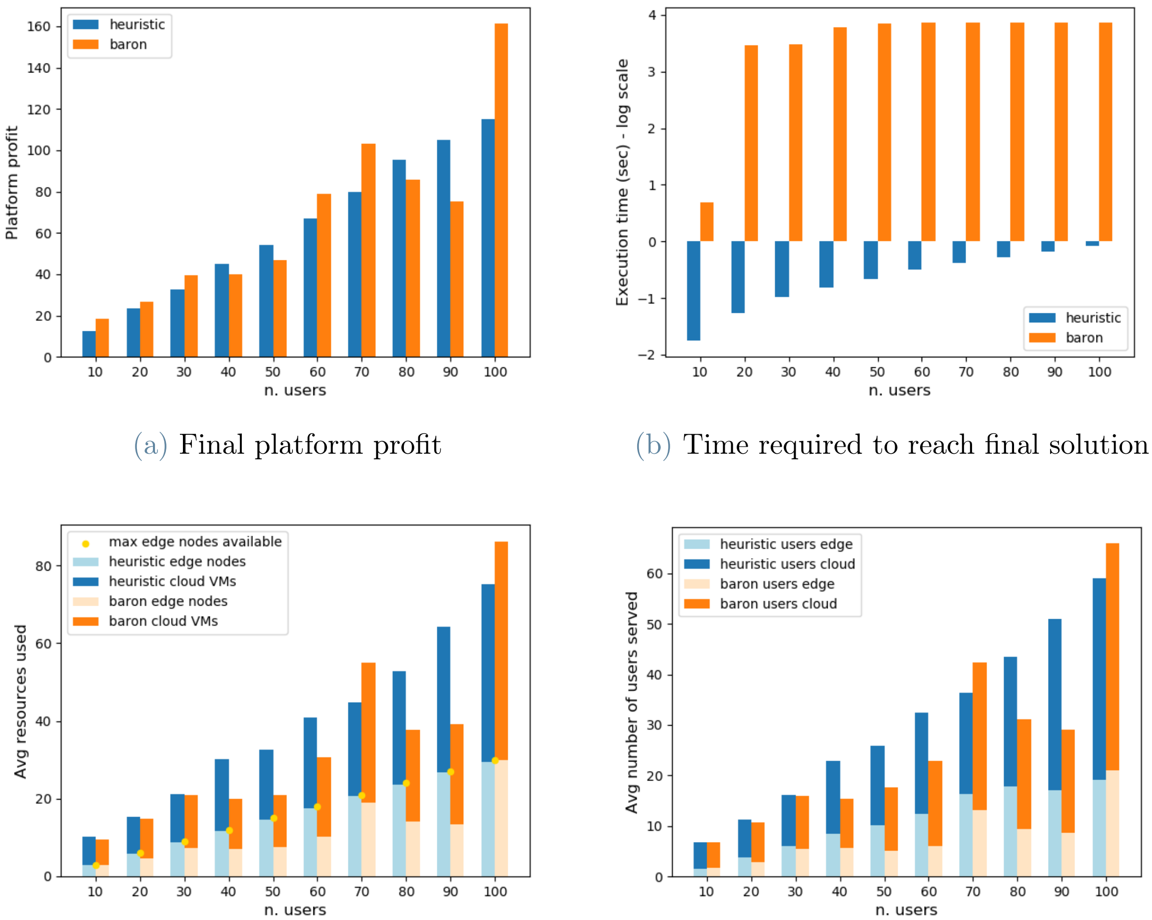
6.3. Comparison with BARON

In this Section we compare the solution found by our method with the one found by BARON 22.3.21. 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 6.2 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 $31.9\%, 11.6\%, 16.5\%$



(a) Final platform profit

(b) Time required to reach final solution

(c) Average of resources used

(d) Average assignment of users

Figure 6.2: Heuristic approach vs BARON comparison

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 4.66% 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.

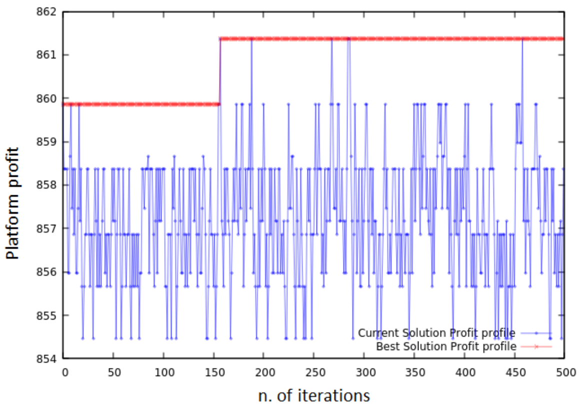
6.4. Tabu search

Figure 6.3 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 6.1) 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:

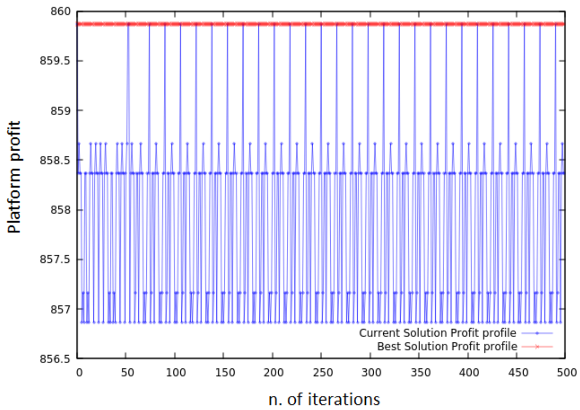
- Profit,
- number of edge and cloud resources allocated for each application,
- total number of users assigned to edge and cloud.

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 solutions.
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.

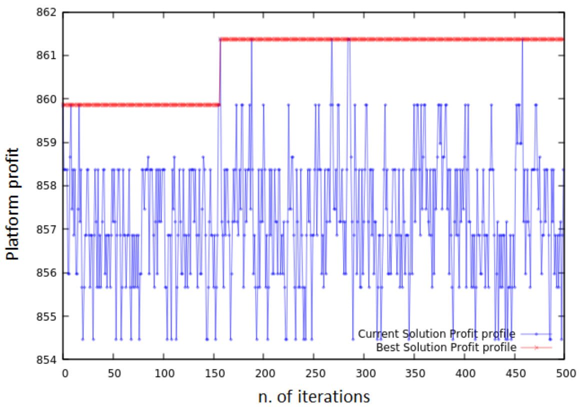
Finally, we measured the profit increase (in percentage) achieved by tabu search for different values of the parameter *var*, which is indicative of the variance among users. Our hope was that, for increasingly variance among users, the performance of tabu search would increase, since the solution of the heuristic approach would have been less optimal. Hence, we tested different versions of tabu search (different neighbor selection method and tabu list attributes) 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 6.4 shows the results, which, as usual, were averaged across 10 different instances of the same problem. Neighbor selection type 0 is RANDOM selection, while type 1 is BEST PROFIT. Tabu list attribute type 0 is the same as we used previously, namely it considers profit,



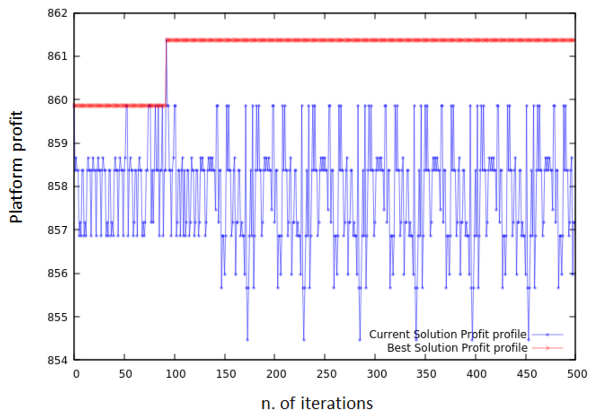
(a) Random neighbor selection, short memory scenario



(b) Best profit neighbor selection, short memory scenario



(c) Random neighbor selection, long memory scenario



(d) Best profit neighbor selection, long memory scenario

Figure 6.3: Current and best profit profiles of tabu search algorithm.

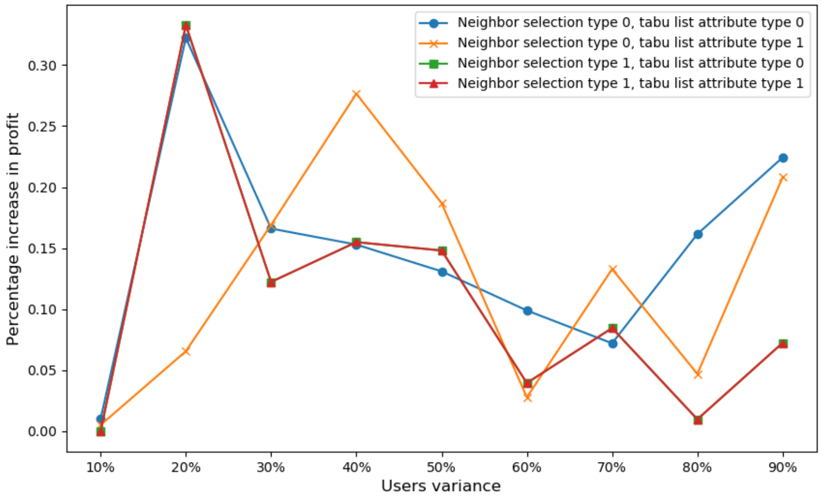


Figure 6.4: Profit increase by tabu search for different users' variance.

number of edge and cloud resources allocated for each application and total number of users assigned to edge and cloud; attribute type 1 is the same but without considering the number of users.

We conclude that the tabu search algorithm we presented in this Thesis is not much effective for this framework, as the profit increase never exceeds 0.5%.

7 | Conclusions and Future Work

In this Chapter we make some considerations on our results and discuss possible future developments to our model and solution.

In this thesis, we have formulated a profit optimization problem with an incentive mechanism for the edge platform based on a Stackelberg Game and we have developed an algorithm to solve our problem in a fast and efficient way. Moreover, we used a commercial software tool named BARON [1] as benchmark to evaluate the performance of our C++ implementation of the algorithm. We found that, in most cases, our approach was able to match closely and, at times, improve the solution achieved by BARON, loosing only less than 5% of the profit averaged across different tests. The main advantage of our approach however, is its execution time, which is multiple orders of magnitude faster than BARON, even when tested on a slower system. For example, for problem instances where BARON was reaching its time-limit constraint of 2 hours, our algorithm completes its execution in less than 1 second.
A natural extension to our model and solution would be to consider the more general case of multiple deployments scenario, where each application and its corresponding DNN have an arbitrary number of splitting points. The generalization of the users' decision problem and its solution is a straight-forward adaptation of the current formulation to a domain with increased dimension. The solution and formulation of the relaxed optimal allocation problem 4.4 on the other hand, is a bit more complicated, as the problem rises in complexity, since the platform not only has to find the best resource allocation and load partition among applications, but also among deployments of the same application.

The other main contribution of this thesis is the tabu search algorithm, whose purpose was to further optimize the profit achieved by our approach by altering the resource and users' assignment of the final solution. Unfortunately, we conclude that the current implementation of the algorithm is not effective in this particular framework, as the increase in profit achieved by tabu search did not exceed 0.5% in all our tests. However, changes can be made to the algorithm hoping to achieve better results. For example, we could try to modify the neighbors generation process (4.4.1) at each iteration, hoping to escape the local maximum and eventually improve the final solution.

Bibliography

[1] Minlp: Baron. https://minlp.com/baron-solver.

[2] Average monthly electricity wholesale price in italy from january 2019 to january 2023. https://www.statista.com/statistics/1267548/italy-monthly-wholesale-electricity-price/.

[3] C.-Y. W. Alexey Bochkovskiy and H.-Y. M. Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv*, 2020.

[4] M. Bolourian and H. Shah-Mansouri. Energy-efficient task offloading for three-tier wireless powered mobile edge computing. *IEEE Internet of Things Journal*, 2023.

[5] G. P. H. Z. S. X. S. Z. X. Chen. Joint optimization of video-based ai inference tasks in mec-assisted augmented reality systems. *IEEE Open Journal of the Communications Society*, 2023.

[6] E. I. Chirica. An approach and a tool for the performance profiling and prediction of partitioned deep neural networks in computing continua environments. *Thesis of Politecnico di Milano*, 2022.

[7] Fortune. Internet of things (iot) forecast. https://www.fortunebusinessinsights.com/industry-reports/internet-of-things-iot-market-100307, 2021.

[8] 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.

[9] F. Z. X. Z. R. Q. H. Han Hu, Dingguo Wu and H. Zhu. Intelligent resource allocation for edge-cloud collaborative networks: A hybrid ddpg-d3qn approach. *IEEE Transactions on Vehicular Technology*, 2023.

[10] 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.

[11] G. X. Lijun Sun and R. Yu. Tafs: A truthful auction for iot application offloading in fog computing networks. *IEEE Internet of Things Journal*, 10, 2022.

[12] I. Open Signal. 5g experience report. https://www.opensignal.com/reports/2022/05/italy/mobile-network-experience-5g., 2022.

[13] A. QADEER and M. J. LEE. Deep-deterministic policy gradient based multi-resource allocation in edge-cloud system: A distributed approach. *IEEE*, 2023.

[14] M. S. A. M. M. F. H. A. E. M. M. Salim. Toward secured iot-based smart systems using machine learning. *IEEE Open Journal of the Communications Society*, 11 (22725710):20827 – 20841, 2023.

[15] M. Satyanarayanan. The emergence of edge computing. *IEEE Open Journal of the Communications Society*, pages 30–39, 2017.

[16] 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`.

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

[18] Y. J. S.-s. L. TaeYoung Kim, HyungBin Park and S. Lee. Partition placement and resource allocation for multiple dnn-based applications in heterogeneous iot environments. *IEEE Internet of Things Journal*, 2023.

[19] P. R. L. L. F. I. C. P. F. I. S. D. F. I. Yakun Huang, Xiuquan Qiao and J. Chen. A lightweight collaborative deep neural network for the mobile web in edge cloud. *IEEE TRANSACTIONS ON MOBILE COMPUTING*, 21, 2022.

[20] Y. M. Zhenchuan Sun and C. Yu. Graph-reinforcement-learning-based task offloading for multiaccess edge computing. *IEEE Internet of Things Journal*, 10, 2023.

[21] H. W. M. I. C. T. M. I. Ziru Zhang, Nianfu Wang and I. Ruidong Li, Senior Member. Mr-dro: A fast and efficient task offloading algorithm in heterogeneous edge/cloud computing environments. *IEEE Internet of Things Journal*, 10, 2023.

[22] L. T. J. S. M. I. I. R. S. M. Zubair Sharif, Member and S. M. Mamoun Alazab. Adaptive and priority-based resource allocation for efficient resources utilization in mobile-edge computing. *IEEE INTERNET OF THINGS JOURNAL,*, 10, 2023.

A | Appendix A

A.1. Proof of Theorem 4.1

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 (4.3) are the sum of multiple convex function, they are convex. □

****

A.2. Proof of Theorem 4.3

Proof. Problem 4.14 is equivalent to:

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

Therefore, if condition 4.18 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 4.2

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 4.4

Proof. Analogously to the only edge scenario, the optimization problem 4.19 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. Selection of Application Order

In this Section we explain our proposed method of selection of the application-orders considered by the algorithm, among all the possible permutations. The method is quite simple: 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 with respect to the scenario where all permutations are considered, 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; clearly, for progressively lower values of k the final profit should be lower, as less and less possible applications-orders are being considered by the algorithm. The system used for this test is composed of 5 applications, while the number of users is varied in the range $[100, 1000]$. 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 $k = 1$.

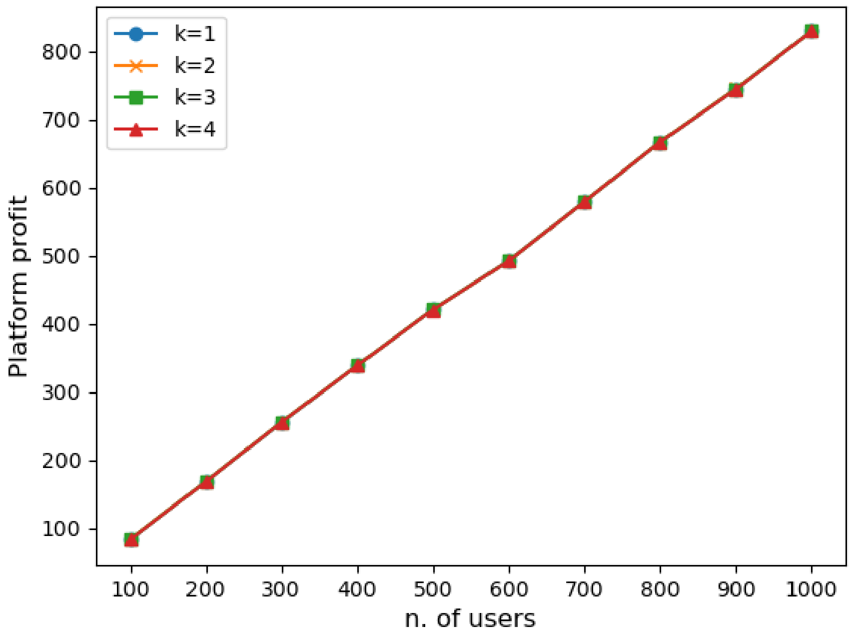


Figure A.1: Heuristic approach final profit for different values of k.

List of Figures

List of Tables