Università degli Studi di Milano
FACOLTÀ DI SCIENZE E TECNOLOGIE

DIPARTIMENTO DI INFORMATICA
GIOVANNI DEGLI ANTONI

CORSO DI LAUREA MAGISTRALE IN
INFORMATICA

MODELS AND ALGORITHMS FOR THE OPTIMAL ALLOCATION OF SERVICES IN VEHICLE-TO-CLOUD ARCHITECTURES

Supervisor: Prof. Alberto Ceselli
Co-supervisor: Prof. Christian Quadri

Master's Thesis by:
Alessandro Minoli
Matr. 20202A

Academic Year 2023-2024

Ringraziamenti

Ringrazio mamma, papà e Alex perchè, oltre ad avermi permesso di compiere i miei studi, mi supportano (e sopportano) nella vita quotidiana.

Ringrazio Alberto Ceselli per avermi accompagnato durante tutto il mio lavoro di tesi ed essersi dimostrato, oltre che un ottimo professore e ricercatore, una persona gentile, sempre disponibile ad aiutarmi e pronta ad incoraggiarmi.

Ringrazio Christian Quadri e tutte le persone che hanno frequentato *OptLab* negli ultimi mesi per avermi dato saggi consigli ed essere stati amichevoli.

Ringrazio i professori che hanno tenuto i corsi più interessanti del mio percorso magistrale per avermi ispirato con la loro passione, cultura scientifica e umanità:
Giovanni Pighizzini, Luca Prigioniero, Carlo Mereghetti, Beatrice Palano, Massimiliano Goldwurm, Nicolò Cesa-Bianchi, Roberto Cordone e Giovanni Righini.

Ringrazio i miei cugini, i miei zii e i miei nonni.

Ringrazio gli amici di università: pochi ma brillanti e sempre presenti per un caffè.

Ringrazio tutti gli amici del novarese per condividere con me ogni momento.
Una menzione particolare a quelli di *prima o Poi* , *Nerdish* e *Cuore di Cane*.

Contents

Chapter 1

Introduction

This chapter provides a general introduction to the work presented in this Master's Thesis, developed at the Operations Research Lab within the Department of Computer Science at University of Milan.
Section 1.1 introduces the concepts of Micro-services Architecture, Cloud Computing and Edge Computing. In Section 1.2 we define in simple terms the problem addressed in this work, whose foundations are the aforementioned concepts. Finally, Section 1.3 outlines the structure of the thesis.

1.1 Background

1.1.1 Micro-services Architecture

The micro-services architecture [1] is an approach to developing applications as sets of small independent services. Each of the services runs in its own independent process, can be written in any programming language and independently deployed, upgraded or replaced (provided there are no changes to its interfaces).
Services can communicate using lightweight mechanisms. Deciding how to partition a system into micro-services is a fundamental and challenging design decision.
The opposite approach is the monolithic architecture, which is easier to deploy but typically slower to develop, maintain and update. Also, it is less scalable, less caching-friendly and tied to the technology selected at the beginning.
The micro-services architecture also has its drawbacks. It introduces the additional complexity of creating a distributed system, which is difficult to test, and probably to implementing an inter-service communication mechanism. Moreover, it increases memory consumption, as each service requires its own address space.
The micro-services architecture is the major choice for IoT development.

1.1.2 Cloud Computing

Cloud Computing (CC) can be defined as "*a set of network-enabled services, providing scalable, QoS guaranteed, normally personalized, inexpensive computing infrastructures on demand, which could be accessed in a simple and pervasive way*" [2].
Managing complex IT infrastructures often requires users to handle rapidly evolving hardware resources and various software installations. Offloading their data and applications to the remote cloud can be an efficient solution.
CC provides users with services to access hardware, software and data resources transparently via a Wide Area Network (WAN) connection.
The key features of CC services are scalability and flexibility: CC platforms are able to adapt to diverse requirements from a potentially large number of users. These features are enabled by virtualization techniques, where the hardware is partitioned into portions dedicated to each service, and by software solutions for virtualization, such as Virtual Machines (VMs), containers, and their associated management or migration tools. By ensuring abundant resources and high availability, CC has been a successful computing model for the past 20 years. However, CC is becoming unsuitable for many emerging applications. Applications that utilize the 5G technology often rely on stringent latency and reliability constraints. IoT and mobile devices, due to their hardware limitations, have high demands for offloading tasks that lead to bottlenecks in an already congested network.

1.1.3 Edge Computing

Edge Computing (EC) is an umbrella term gathering all architectures that aim to bring computation and storage resources closer to the end users and devices, i.e. not within the core of the network but at the edge. EC goal is to optimize the network and its applications by minimizing the need for long-distance communications between devices and the remote cloud, thereby reducing latency and bandwidth usage.
In a 3-tier network provisioning architecture (Figure 1), Edge servers are the primary resource for augmenting the capabilities of devices. The remote Cloud is used as the last available resource or for delay-tolerant resource-intensive applications.
EC paradigm does not intend to replace CC but to extend it: edge nodes can always access the network core if needed.
EC solutions permit to achieve higher QoS and QoE, as well as improved security, data privacy and reliability compared to CC ones. EC has features such as context-awareness and geolocalization, which are difficult to offer in a traditional cloud environment. While EC architectures are highly scalable in terms of the number of supported devices, there is a trade-off between the number of devices and the complexity of their management processes. They have some limitations in storage capacity, making CC still best suited for long-term data storage. CC is based on centralized

data centers, characterized by high availability and abundant computing resources. In contrast, EC works as a distributed computing standard: edge solutions have an availability that ranges from average to high and offer a moderate amount of computing resources. EC virtualization has some restrictions because of lower processing capabilities, whereas CC allows heavyweight virtualization. Four main edge-level architectures are described in the literature, an overview and comparison of these architectures can be found in [3].

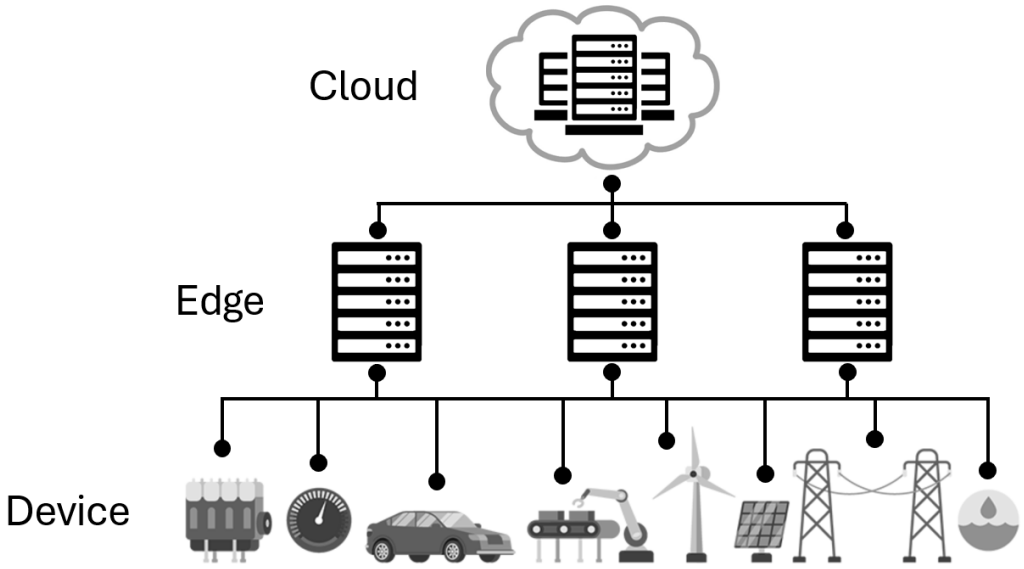


Figure 1: 3-tier network provisioning architecture

1.2 Problem definition

In industrial scenarios, there is often the necessity to run many heterogeneous applications. These applications can be deployed:

- in the remote cloud (CC), if they have no particular latency requirements
- on an edge server (EC), if they have latency requirements and are heavyweight
- in-house, if they have latency requirements and are lightweight

An application is composed of multiple micro-services, each of which runs as a virtual machine (VM) on a network node that has enough resources to host it.

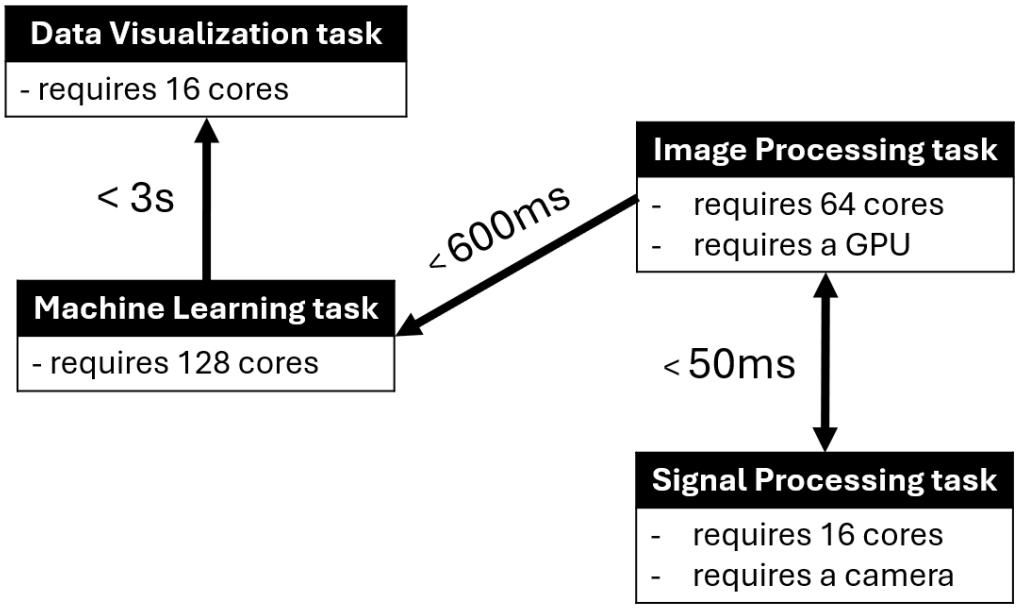


Figure 2: Example of application composed of 4 micro-services

Micro-services that need to communicate with each other must do so via channels that meet specific resource requirements. Figure 2 illustrates an example of application composed of 4 micro-services along with their requirements.
We consider a scenario in which there are some autonomous vehicles that move within a production floor. Multiple applications need to run on a network infrastructure.
The vehicles have limited on-board processing capabilities, and each vehicle can offload some of the computations it needs to perform to a nearby fixed edge server.
We are in a 3-tier Vehicle-to-Cloud provisioning architecture, which is equivalent to the one shown in Figure 1, treating the autonomous vehicles as devices.
This work faces the problem of finding an optimal way to deploy all the applications. More specifically, we want to determine the network node where to host each micro-service, ensuring that all resource constraints, as well as a set of end-to-end requirements, are satisfied and an objective function based on the allocation costs on nodes is minimized. This specific problem, to the best of our knowledge, has not yet been addressed in the literature.

1.3 Structure of the work

This is an overview of the content of the subsequent chapters:

- Chapter 2 summarises some literature related to the subject
- Chapter 3 formalizes the problem and provides for it two integer mathematical programming formulations, a compact and an extended one
- Chapter 4 presents some column generation based algorithms for the problem
- Chapter 5 explains how the key entities of the problem are randomly generated
- Chapter 6 evaluates the performance of the proposed algorithms on a randomly generated dataset
- Chapter 7 reports final thoughts and future perspectives of the work

Chapter 2

Literature review

The current standards provide all the required instruments, functionalities and technologies to actually implement EC solutions. Nevertheless, orchestrating services at the edge, due to the limited amount of resources available and the high dynamics of the environment, remains a challenging task and still an open research issue.
This chapter provides a brief overview of four scientific papers related to the topic.

Mobile Edge Cloud Network Design Optimization (2017) [4]

This article tackles the edge cloud network design problem for mobile access networks. Figure 3 shows an example of a simple mobile edge cloud network.
The scenario is such that there are VMs associated with mobile users to host low-latency applications. Each VM is allocated to a cloudlet, which is a trusted cluster of computers connected to the Internet and accessible to nearby mobile devices.
The design problem consists of determining where to install cloudlet facilities and then how to assign sets of access points to these cloudlets. All decisions must satisfy the service-level agreements (SLA). Additionally, partial user mobility, which causes variations in the network load and service level, and VM orchestration can also be considered. The work demonstrates that including user and VM mobility in the network planning significantly increases the number of users with respected SLA. It also compares two VM mobility modes from one cloudlet to another: bulk migration and live migration. Mathematical programming techniques that combine column generation, iterative rounding, local search, very large scale neighborhood and problem reduction are exploited to achieve high quality solutions in reasonable time.

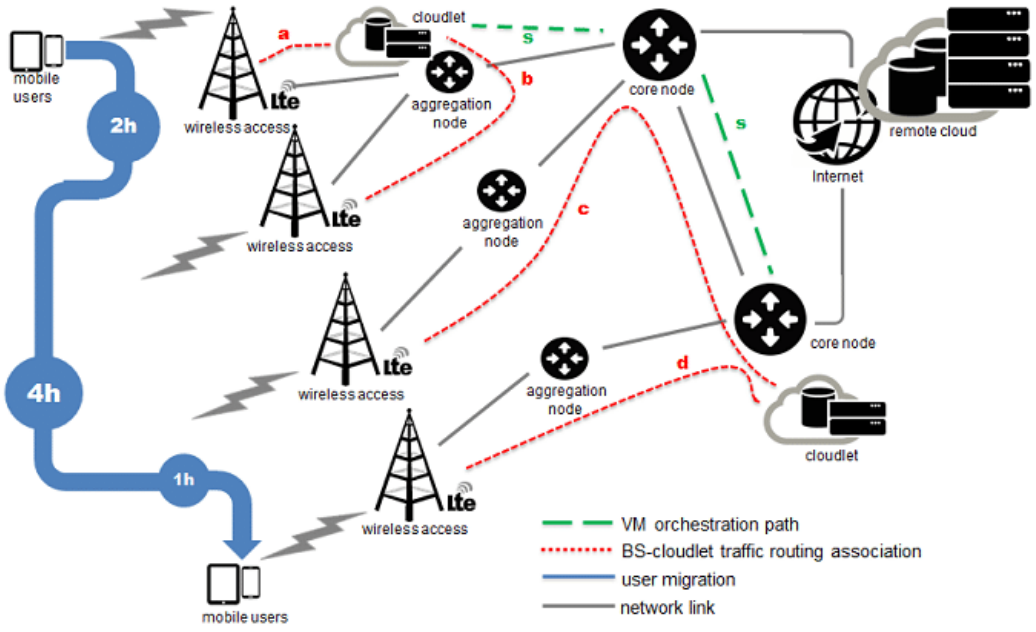


Figure 3: Example of mobile edge cloud network

Prescriptive Analytics for MEC Orchestration (2018) [5]

This article proposes a data-driven approach to solve the problem of assigning base stations (access points) to MEC facilities (cloudlets) over time, while respecting capacity constraints and minimizing network- and user-related costs. BS-to-MEC facility switching operations cannot reasonably be expected to run continuously because each operation implies a switching cost for the network and a latency cost for the user. Therefore, these operations must occur only at specific points in time.
Data analytics for mobile networks have revealed that the network traffic exhibits regular and predictable macroscopic spatiotemporal patterns. By applying clustering techniques, the traffic in a real mobile cellular dataset can be aggregated into a limited number of patterns, within which it remains stable across base stations. These patterns are then integrated in a BS-to-MEC orchestration mathematical programming algorithm, which is based on column generation and heuristic rounding of the variables. This framework outperforms the orchestration decision process that does not use time-period aggregations, achieving an improvement of an order of magnitude in MEC access latency.

Optimal Assignment Plan in Sliced Backhaul Networks (2020) [6]

This article also addresses the problem of planning BS-to-MEC facility associations, but specifically in multi-slice scenarios. Network slicing is a key feature of 5G networks that enables the creation of dynamic logical, independent and vertical partitions of the network to satisfy the requirements of heterogeneous applications. Each component of a slice, namely each node or link, has specific requirements (e.g. computational/storage capacity for nodes and delay/capacity for links), which must be satisfied for effective service provisioning. The slicing process affects the entire mobile network infrastructure, from the core network down to the base stations.
The paper presents a combinatorial optimization model that supports multiple network slices and demonstrates that the multi-sliced approach improves network performance, even when the available resources are limited. Similarly to the previously analyzed paper, the spatiotemporal patterns of network traffic are exploited to plan an optimal utilization of the resources and the algorithm technique employed is column generation.

Multi-user edge service orchestration based on DRL (2023) [7]

This article focuses on the orchestration of session-based services (e.g. online games, video conferences), which are characterized by stateful and long-lived instances involving multiple users. The goal is to optimize a multi-objective function that considers competing goals such as QoS, resource usage, service request waiting time and network operational costs. The orchestrator must operate in an online manner, assuming no prior knowledge of future service requests. Moreover, due to changing network conditions, it has to continuously monitor previously deployed instances and potentially migrate them to enhance QoS or balance the load across edge servers.
Most of the existing literature of this field relies on heuristics and MILP optimization, which are often unable to adapt to new contexts without experiencing performance degradation. The recent introduction of Deep Reinforcement Learning (DRL) has opened up the opportunity for designing new approaches to online decision processes. DRL provides a powerful framework for complex decision tasks and is able to adapt to various system load conditions and service demand patterns.
The article presents an orchestration methodology based on a combination of model-free DRL and a parametric combinatorial model. The obtained solutions are close to offline optimal ones. An high QoS level is guaranteed while utilizing only a small amount of extra resources. Fair load balancing across facilities is maintained and computational times are suitable for near-real-time decisions.

Chapter 3

Models : compact and extended formulation

In this chapter we formally define the problem that was briefly introduced in Section 1.2 and we propose for it two mathematical programming formulations: a compact and an extended one.

3.1 Problem data

We model the network as a directed strongly connected graph $G_{network} = (I, A)$ where:

- the vertices set I represents the network nodes
- the edges set A represents the links (i, j) between pairs of nodes $i \in I$, $j \in I$. The edges are symmetric : $(i, j) \in A$ implies $(j, i) \in A$

We model an application as a directed weakly connected graph $G_{app} = (T, D)$ where:

- the vertices set T represents the micro-services to deploy
- the edges set D represents the communication dependencies (u, v) between pairs of micro-services $u \in T$, $v \in T$

We define :

- a set R of consumable resources (e.g. RAM, storage, bandwidth)
- a set S of properties (e.g. latency, presence of a sensor)

We assume that :

- all applications, meaning all the micro-services along with their communication dependencies, are known at optimization time (i.e. subsequent deployment requests may be handled by subsequent incremental optimization rounds, but each optimization round will be greedy in its choices)

- the routing in the network is independent from the optimization process. Specifically, we assume that each routing path $\mathcal{P}_{ij}$, defined as the sequence of links in A which are traversed by packets sent by node $i \in I$ and received by node $j \in I$, is known and fixed before the optimization process.
  Equivalently, we assume binary coefficients a_{ij}^{lm} are given, taking value 1 if $(l, m) \in \mathcal{P}_{ij}$, 0 otherwise.

The following data are given :

- $Q_i^k \quad \forall i \in I, \forall k \in R \cup S$, which is the availability of resource k on node i
- $Q_{ij}^k \quad \forall (i, j) \in A, \forall k \in R \cup S$, which is the availability of resource k on link (i, j)
- $q_u^k \quad \forall u \in T, \forall k \in R \cup S$, which is the consumption $(k \in R)$ or the requirement $(k \in S)$ of resource k yield by micro-service u
- $q_{uv}^k \quad \forall (u, v) \in D, \forall k \in R \cup S$, which is the consumption $(k \in R)$ or the requirement $(k \in S)$ of resource k yield by connection (u, v)

From this data we compute, for each requirement of resource $k \in S$ and for each pair of network nodes $i \in I, j \in I$:

$$v_{ij}^k = f^k(\{Q_{lm}^k \mid (l, m) \in \mathcal{P}_{ij}\})$$

where f^k is a suitable cumulative consumption function for requirement $k \in S$ along the routing path from i to j.
For example, if k represents latency, it is :

$$v_{ij}^{\text{latency}} = \sum_{(l,m) \in \mathcal{P}_{ij}} Q_{lm}^k$$

The same computation can also be applied to each consumption of resource $k \in R$.
For example, if k represents bandwidth, it is :

$$v_{ij}^{\text{bandwidth}} = \text{MIN}_{(l,m) \in \mathcal{P}_{ij}} \; Q_{lm}^k$$

If mapping a single pair of micro-services alone exceeds the available resource along the path between two network nodes, then it is infeasible to perform that mapping

(although the converse is not necessarily true).

Thus, we compute a set of binary coefficients b_{uv}^{ij}, which are 0 if the resource requirement/consumption q_{uv}^{k} for connection (u, v) exceeds v_{ij}^{k} for some $k \in R \cup S$ and 1 otherwise. Therefore, $b_{uv}^{ij} = 0$ when it is infeasible to map a pair of micro-services u and v to network nodes i and j because the requirement/consumption of some resource $k \in R \cup S$ would be violated on the path from i to j.
Similarly, we compute a set of binary coefficients b_{u}^{i}, which are 0 if the resource requirement q_{u}^{k} alone exceeds Q_{i}^{k} for some $k \in R \cup S$ and 1 otherwise.

Finally, we define a cost c_i for each node $i \in I$.
The specific formula for computing costs is arbitrary, but intuitively :

- c_i should be low for cloud nodes, medium for edge nodes, and high for vehicle nodes
- c_i should be low for border nodes and high for nodes that are very central in the network

3.2 Compact formulation

For the compact formulation of the problem, we use a set of binary decision variables $x_{ui} \in \{0, 1\}$ which are 1 if micro-service $u \in T$ is mapped to node $i \in I$, 0 otherwise.

The objective is to minimize the use of high cost nodes:

$$minimize \quad \sum_{u \in T} \sum_{i \in I} c_i \cdot \ x_{ui} \tag{C_{obj}}$$

The following constraints must be satisfied:

- each micro-service must be mapped to exactly one node:

$$\sum_{i \in I} x_{ui} = 1 \qquad \forall u \in T \tag{C_1}$$

- no micro-service can be placed on a node that does not satisfy all requirements:

$$x_{ui} = 0 \qquad \forall i \in I, \forall u \in T \text{ s.t. } b_{u}^{i} = 0 \tag{C_2}$$

- no pairs of micro-services can be placed on pairs of nodes whose connecting path does not satisfy all requirements:

$$x_{ui} + x_{vj} \leq 1 \qquad \forall i \in I, \forall j \in I, \forall (u,v) \in D \text{ s.t. } b_{uv}^{ij} = 0 \qquad (C_3)$$

- the overall consumption of resources on each node must not exceed the resource availability:

$$\sum_{u \in T} q_u^k \cdot x_{ui} \leq Q_i^k \qquad \forall k \in R, \forall i \in I \qquad (C_4)$$

- the overall consumption of resources on each link must not exceed the resource availability:

$$\sum_{i \in I} \sum_{j \in I} \sum_{(u,v) \in D} q_{uv}^k \cdot a_{ij}^{lm} \cdot x_{ui} \cdot x_{vj} \leq Q_{lm}^k \qquad \forall k \in R, \forall (l,m) \in A \qquad (C_5)$$

The model is compact: a polynomial number of variables and constraints are employed. Since the last family of constraints contains the bilinear terms $x_{ui} \cdot x_{vj}$, it is a Binary (convex) Quadratic Programming (BQP) problem.
While these bilinear terms can be linearized, given the size of the instances we plan to solve, an ILP solver like *GuRoBi* should be capable of managing them automatically.

Improvements on model building time

A simple way to speed up the model building time is to declare constraints C_3 only $\forall i \in I, \forall j \in I, \forall u \in T, \forall v \in T$ s.t. $b_u^i = 1 \wedge b_v^j = 1 \wedge b_{uv}^{ij} = 0$.
When $b_u^i = 0 \vee b_v^j = 0$, the satisfaction of constraints C_3 is implied by constraints C_2.

Constraints C_5 are the most computationally expensive to build. Instead of computing 3 nested summations with many 0 terms for each constraint, we incrementally build the left-hand side LHS_{lm}^k of each constraint following Algorithm 1.

3.3 Extended formulation

We have a set N of applications to be mapped onto the network.
For each application $n \in N$, we define the set:

$$\Omega^n = \{\text{all feasible mappings of application } n \text{ onto the network}\}$$

A mapping for application n is feasible if it satisfies constraints C_1, C_2, C_3, C_4, C_5.
Given a mapping p, we define the following coefficients:

Algorithm 1 Algorithm to compute left-hand sides of constraints C_5

Initialize $LHS^k_{lm} = 0 \quad \forall (l,m) \in A, \forall k \in R$
for all $(u,v) \in D$ **do**
  for all (i,j) *s.t.* $b^i_u = 1 \wedge b^j_v = 1 \wedge b^{ij}_{uv} = 1$ **do**
    for all $(l,m) \in \mathcal{P}_{ij}$ **do**
      for all $k \in R$ **do**
        $LHS^k_{lm} = LHS^k_{lm} + q^k_{uv} \cdot x_{ui} \cdot x_{vj}$
      end for
    end for
  end for
end for

- c_p : the cost of mapping p
- $q^{i,k}_p \ \forall k \in R, \forall i \in I$: the total consumption of resource k on node i yield by mapping p
- $q^{lm,k}_p \ \forall k \in R, \forall (l,m) \in A$: the total consumption of resource k on link (l,m) yield by mapping p

For the extended formulation of the problem, we introduce a set of binary decision variables $\theta^n_p \in \{0,1\} \ \forall n \in N, \forall p \in \Omega^n$ which are 1 if mapping p is selected for application n and 0 otherwise.

The objective is to minimize the total cost of mapping all applications:

$$minimize \quad \sum_{n \in N} \sum_{p \in \Omega^n} c_p \cdot \ \theta^n_p \qquad (E_{obj})$$

The following constraints must be satisfied:

- the overall consumption of resources on each node must not exceed the resource availability:

$$\sum_{n \in N} \sum_{p \in \Omega^n} q^{i,k}_p \cdot \theta^n_p \leq Q^k_i \qquad \forall k \in R, \forall i \in I \qquad (E_1)$$

- the overall consumption of resources on each link must not exceed the resource availability:

$$\sum_{n \in N} \sum_{p \in \Omega^n} q^{lm,k}_p \cdot \theta^n_p \leq Q^k_{lm} \qquad \forall k \in R, \forall (l,m) \in A \qquad (E_2)$$

- exactly one mapping must be chosen for each application:

$$\sum_{p \in \Omega^n} \theta_p^n = 1 \qquad \forall n \in N \tag{E_3}$$

This model employs a polynomial number of constraints and an exponential number of variables. It is formulated as a Binary Linear Programming problem.
The LP relaxation of this model provides stronger lower bounds with respect to the LP relaxation of the compact model. The problem has been partially convexified.

Chapter 4

Algorithms : column generation and heuristics

This chapter begins with a theoretical introduction to column generation and discusses how this technique can be applied to our problem in Section 4.1. A heuristic algorithm to map a single application onto the network is introduced in Section 4.2. Three matheuristics designed to find feasible and good quality solutions to our problem are presented in Section 4.3.

4.1 Column generation

4.1.1 Theoretical background

A high-quality reference for the topic is: *A Primer in Column Generation* [8].

Given an ILP problem, the ideal formulation of its constraint set is the convex hull $conv(X)$ of its feasible integer solutions X (Figure 4). When we solve an ILP to optimality we "convexify" it, as the optimal solution found corresponds to the optimal solution of the LP problem whose polyhedron is $conv(X)$.
Let $x^{(1)}, x^{(2)}, \ldots, x^{(p)}$ be the extreme points of $conv(X)$.
The Minkowsky and Weil theorem states that:

$$\forall x \in X,\ \exists \theta \geq 0 \mid x = \sum_{u=1}^{p} \theta_u \, x^{(u)} \ ,\ \sum_{u=1}^{p} \theta_u = 1$$

Hence, the optimal solution of the ILP can be expressed as:

$$z^* = \min_{x \in \mathbb{R}^n_+} \{cx \mid x \in conv(X)\} = \min_{\theta \in \mathbb{R}^p_+} \{cx \mid x = \sum_{u=1}^{p} \theta_u \, x^{(u)} \ ,\ \sum_{u=1}^{p} \theta_u = 1\}$$

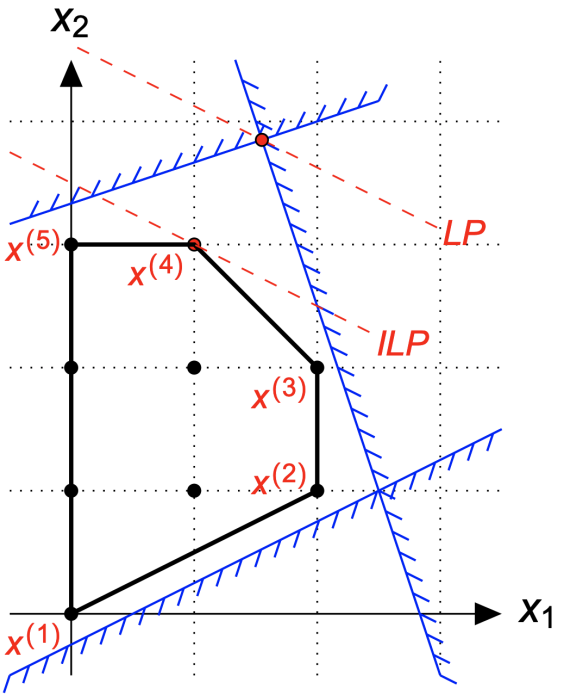


Figure 4: Ideal formulation of an ILP problem

In practice, we often do not know how to convexify the entire ILP but only a certain sub-problem. Given an ILP:

$$z^*_{\text{compact}} = \min_{x \in \mathbb{Z}^n_+} \{cx \mid Dx \geq e \ , \ Fx \geq g\} \tag{1}$$

it can happen that $Q = \{x \in \mathbb{Z}^n_+ \mid Fx \geq g\}$:

- can be convexified as is, in which case $Dx \geq e$ are complicating constraints
- can be decomposed into independent subproblems we are able to convexify (Dantzig-Wolfe decomposition), in which case $Dx \geq e$ are linking constraints
- both of the cases

Therefore, instead of solving (1), we can solve:

$$z^* = \min_{x \in \mathbb{Z}^n_+} \{cx \mid Dx \geq e \ , \ x \in conv(Q)\} \tag{2}$$

To obtain this, we use the extended formulation, based on the substitution $x = \sum_{u=1}^{p} \theta_u \, x^{(u)}$, where $x^{(u)}$ is a generic extreme point of $conv(Q)$:

$$z^*_{\text{extended}} = \min_{\theta \in \mathbb{B}^p} \{ \sum_{u=1}^{p} cx^{(u)}\,\theta_u \mid \sum_{u=1}^{p} Dx^{(u)}\,\theta_u \geq e \ , \ \sum_{u=1}^{p} \theta_u = 1 \ \} \tag{3}$$

The extended formulation (3) remains an integer problem and has a combinatorial number of variables. However, its linear relaxation is tighter than that of (2).

Column generation (CG) is a technique used to solve LP problems with a large set N of variables (columns), such as the linear relaxation of (3).
The approach involves solving a Linear Restricted Master Problem (LRMP), where only a small subset $\overline{N} \subseteq N$ of columns is included in the simplex algorithm tableau. When an optimal primal solution $x*$ for LRMP is found, the optimal dual solution λ^* is also available. A Pricing Problem (PP), which searches for a column $j \notin \overline{N}$ with negative reduced cost $\overline{c}_j$ in the current basic solution, is defined using the dual values from the LRMP and optimized. The coefficients a_i of the column are the decision variables in the PP. The PP is the problem we are able to convexify. If Dantzig-Wolfe decomposition is exploited, multiple pricing problems must be optimized.
For an LP problem with m constraints, the reduced cost of a column is:

$$\overline{c_j} = c_j - \sum_{i=1}^{m} a_{ij}\lambda_i$$

If the PP cannot find a negative reduced cost column, then z^*_{LRMP} equals the optimum z^*_{LMP} of the original Master Problem. Otherwise, column j is added to $\overline{N}$ and the LRMP is re-optimized. LRMP feasibility must always be guaranteed: at the beginning of CG it is initialized with dummy columns of very high cost.

CG typically have a tailing-off effect towards the end, where the bound it provides has very small improvements. Unfortunately, z^*_{LRMP} is a valid lower bound only when the PP can no longer find new negative reduced cost columns. To overcome this issue, the LRMP dual values can be used as Lagrangean multipliers to compute a valid lower bound z^*_{LR} at any point during CG. The value of z^*_{LRMP} is monotone and $\geq z^*_{\text{LMP}}$. The value of z^*_{LR} is not monotone in general and $\leq z^*_{\text{LMP}}$. CG is typically stopped when the gap between z^*_{LRMP} and z^*_{LR} becomes small enough. An example of the explained situation is reported in Figure 5.

4.1.2 Application to the mapping problem

We have a set N of applications to be mapped onto the network.
Each $n \in N$, as explained in Section 3.1, is modeled a graph $G^n_{app} = (T^n, D^n)$.
We exploit CG technique and Dantzig-Wolfe decomposition by choosing to convexify

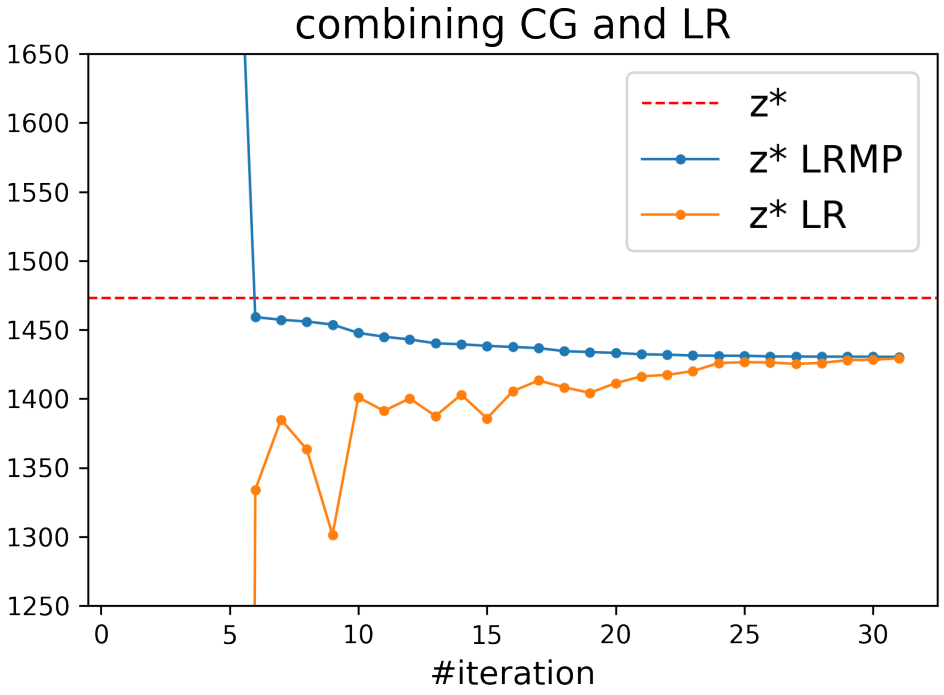


Figure 5: Use of the LR to cut the CG tailing-off effect

the problem of mapping single applications onto the network. In the master problem, we have linking constraints imposing that the mappings, combined together, do not exhaust the available network resources.

Linear Restricted Master Problem

Given $\overline{\Omega^n} \subseteq \Omega^n = \{\text{all feasible mappings of application } n \text{ onto the network}\}\ \forall n \in N$, we introduce a set of continuous decision variables $0 \leq \theta_p^n \leq 1 \quad \forall n \in N, \forall p \in \overline{\Omega^n}$.

The objective function is:

$$minimize \quad z_{\text{LRMP}} = \sum_{n \in N} \sum_{p \in \overline{\Omega^n}} c_p \cdot \theta_p^n \qquad (\text{LRMP}_{obj})$$

The constraints are:

$$\sum_{n \in N} \sum_{p \in \overline{\Omega^n}} q_p^{i,k} \cdot \theta_p^n \leq Q_i^k \qquad \forall k \in R, \forall i \in I$$

$$\sum_{n \in N} \sum_{p \in \overline{\Omega^n}} q_p^{lm,k} \cdot \theta_p^n \leq Q_{lm}^k \qquad \forall k \in R, \forall (l,m) \in A$$

$$\sum_{p \in \overline{\Omega^n}} \theta_p^n = 1 \qquad \forall n \in N$$

To obtain non-negative dual variables, we reverse the signs of all inequalities:

$$-\sum_{n \in N} \sum_{p \in \overline{\Omega^n}} q_p^{i,k} \cdot \theta_p^n \geq -Q_i^k \qquad \forall k \in R, \forall i \in I \qquad \rightarrow \lambda_i^k \geq 0 \qquad (\text{LRMP}_1)$$

$$-\sum_{n \in N} \sum_{p \in \overline{\Omega^n}} q_p^{lm,k} \cdot \theta_p^n \geq -Q_{lm}^k \qquad \forall k \in R, \forall (l,m) \in A \qquad \rightarrow \mu_{lm}^k \geq 0 \qquad (\text{LRMP}_2)$$

To avoid having unrestricted dual variables, we relax the equalities as follows. The optimal solution does not change because the new constraints will always be active.

$$\sum_{p \in \overline{\Omega^n}} \theta_p^n \geq 1 \qquad \forall n \in N \qquad \rightarrow \eta_n \geq 0 \qquad (\text{LRMP}_3)$$

Pricing Problem (PP^n)

The pricing problem PP^n is solved to find the column (i.e. the mapping of application n onto the network) with the minimum reduced cost.
We have a set of binary decision variables $x_{ui} \in \{0,1\}$ which are 1 if micro-service $u \in T^n$ is mapped to node $i \in I$, 0 otherwise.
The following quantities, already described in Section 3.3 and acting as matrix coefficients in LRMP, are associated with each mapping p of application n:

$$c_p = \sum_{u \in T^n} \sum_{i \in I} c_i \cdot x_{ui}$$

$$q_p^{i,k} = \sum_{u \in T^n} q_u^k \cdot x_{ui} \qquad \forall k \in R, \forall i \in I$$

$$q_p^{lm,k} = \sum_{i \in I} \sum_{j \in I} \sum_{(u,v) \in D^n} q_{uv}^k \cdot a_{ij}^{lm} \cdot x_{ui} \cdot x_{vj} \qquad \forall k \in R, \forall (l,m) \in A$$

The objective function is:

$$
\begin{aligned}
\textit{minimize} \quad c_{\mathrm{PP}^n} = & \sum_{u \in T^n} \sum_{i \in I} c_i \cdot \ x_{ui} \\
& - \sum_{k \in R} \sum_{i \in I} \sum_{u \in T^n} - \ q_u^k \cdot x_{ui} \cdot \lambda_i^k \\
& - \sum_{k \in R} \sum_{(l,m) \in A} \sum_{i \in I} \sum_{j \in I} \sum_{(u,v) \in D^n} - \ q_{uv}^k \cdot a_{ij}^{lm} \cdot x_{ui} \cdot x_{vj} \cdot \mu_{lm}^k \\
& - \eta_n
\end{aligned}
$$

The constraints are C_1, C_2, C_3, C_4, C_5.

We can rearrange the objective function as:

$$
\begin{aligned}
\textit{minimize} \quad c_{\mathrm{PP}^n} = & \sum_{u \in T^n} \sum_{i \in I} c_i \cdot \ x_{ui} \\
& + \sum_{k \in R} \sum_{i \in I} \sum_{u \in T^n} [\, q_u^k \cdot \lambda_i^k \,] \cdot x_{ui} \\
& + \sum_{k \in R} \sum_{(l,m) \in A} \sum_{i \in I} \sum_{j \in I} \sum_{(u,v) \in D^n} [\, q_{uv}^k \cdot a_{ij}^{lm} \cdot \mu_{lm}^k \,] \cdot x_{ui} \cdot x_{vj}
\end{aligned}
\qquad (\mathrm{PP}_{obj})
$$

The term $[\, q_u^k \cdot \lambda_i^k \,]$ penalizes the allocation of u on i.
The term $[\, q_{uv}^k \cdot a_{ij}^{lm} \cdot \mu_{lm}^k \,]$ penalizes the simultaneous allocation of u on i and v on j.

Additional details

Figure 6 illustrates how the CG algorithm works.
The LRMP must always be feasible, so it is initialized with a dummy column for each application, which has a very high cost and no consumption of resources. We also add to the LRMP the column corresponding to the optimal mapping of each application onto the network, which can be found by solving the pricing problem with all dual values set to 0. If such a mapping does not exist, the problem is infeasible.
At each CG iteration, given the optimal solution z^*_{LRMP} and the optimal solutions of all pricing problems $c^*_{\mathrm{PP}^n}$ $\forall n \in N$, we compute the dual bound z^*_{LR} given by the Lagrangean relaxation as :

$$
z^*_{\mathrm{LR}} = z^*_{\mathrm{LRMP}} \ + \sum_{n \in N} c^*_{\mathrm{PP}^n}
$$

We stop the CG when the percentage gap between z^*_{LRMP} and z^*_{LR} is < 0.1.
Both the LRMP and the pricing problems are solved using a solver. For efficiency, the LRMP is updated at each iteration instead of being rebuilt. Regarding the pricing

problems, only the objective function must be updated at each iteration. In PP_{obj} the first block of summations remains constant. To speed up the updates of the second and third block of summations, only the terms for which the dual variables are non-zero are built. For the third block, this optimization can be extended further by defining only the terms when also the coefficients a_{ij}^{lm} are non-zero.

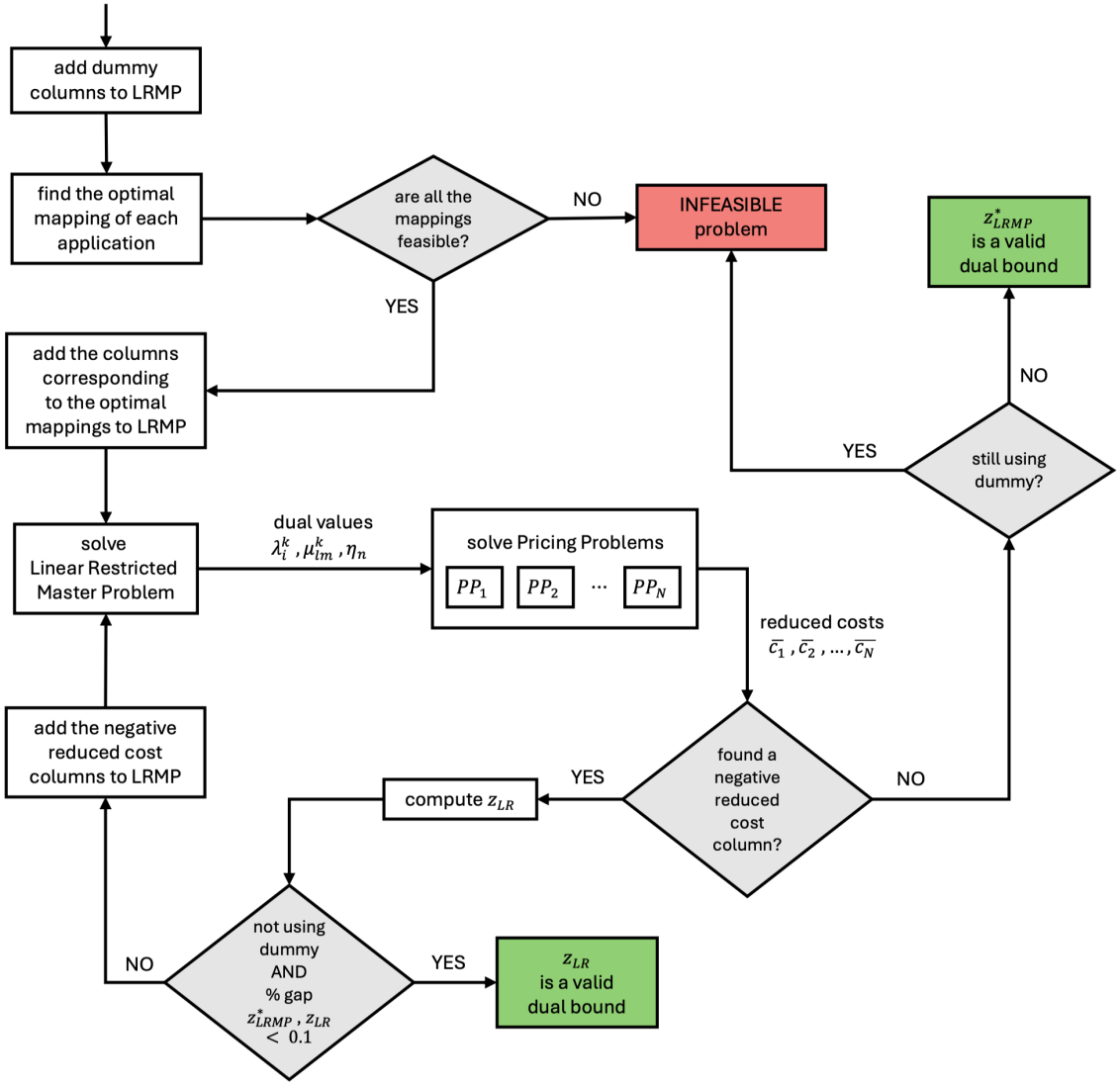


Figure 6: Flowchart of the CG algorithm

4.2 Heuristic mapping of a single application

We devised a constructive metaheuristic for mapping a single application onto the network. It can be used to try to heuristically solve the pricing problems or to generate feasible columns to initialize the LRMP. It is inspired by the semi-greedy algorithm [9] and its idea that elements leading to an optimal solution are a good choice if we use the objective function as a selection criterion, even if they are not strictly the best choice. The metaheuristic exploits randomization, which allows to perform multiple runs of the algorithm to find multiple feasible solutions.
Each step of the algorithm consists in assigning a micro-service $u \in T$ to a node $i \in I$. For each $u \in T$, we define a list of candidate nodes for u as $C_u = [\, i \in I \mid b_u^i = 1 \,]$. The nodes in each candidate list are sorted by assignment costs in the PP, i.e. by the values $\overline{c_{ui}} = c_i + \sum_{k \in R} q_u^k \cdot \lambda_i^k$. We define c_u^{min} and c_u^{max} as the minimum and maximum cost nodes in C_u. To reduce the search space, we restrict each candidate list C_u to include only the nodes with costs between c_u^{min} and $(1-\alpha) \cdot c_u^{min} + \alpha \cdot c_u^{max}$. We set the parameter $\alpha = 0.2$.
The algorithm takes in input :

- the restricted candidate lists C_u , $\forall u \in T$
- the assignment costs $\overline{c_{ui}}$, $\forall u \in T$, $\forall i \in C_u$
- the dual variables μ_{lm}^k and the dual variable η_n

Each run of the algorithm constructs a solution s and stores :

- s_{cost} : the partial cost of solution s
- $s_x[u] \in \{\perp\} \cup \{i \in C_u\}$, $\forall u \in T$: the current assignments of micro-services to nodes in solution s. If u is not yet assigned, then $s_x[u] = \perp$
- R_i^k , $\forall i \in I, \forall k \in R$: the residual availability of resource k on node i
- R_{lm}^k , $\forall (l,m) \in A, \forall k \in R$: the residual availability of resource k on link (l,m)

We define a function **FIX** (u, i) that fixes micro-service u on node i and, accordingly to this operation, updates all the data structures. The function fails if the fixing makes the solution infeasible.
We define a function **PREPROCESS** () that iteratively fixes all micro-services with only one possible candidate node until no more such micro-services are left. This function calls the function **FIX** and, if **FIX** fails, then **PREPROCESS** fails too.

The pseudo-code of the algorithm is provided in Algorithm 2.

To try to generate a feasible column for the original problem, the algorithm can be executed setting all dual values to 0, including those used to compute the costs $\overline{c_{ui}}$, and by disregarding the conditions that stop execution when $s_{cost} \geq 0$.
Choosing a random node in C_u ($|C_u| \geq 2$) with linearly decreasing probabilities means that the probability p_k of choosing node $C_u[k]$, $\forall k \mid 0 \leq k < |C_u|$ is:

$$p_k = \frac{|C_u| - k}{\sum_{j=0}^{|C_u|-1}(|C_u| - j)}$$

For example, if $|C_u| = 10$, then the PMF of the random variable representing index k is reported in Figure 7. Since nodes in C_u are sorted by assignment cost $\overline{c_{ui}}$, this approach increases the likelihood of assigning u to nodes with lower costs.

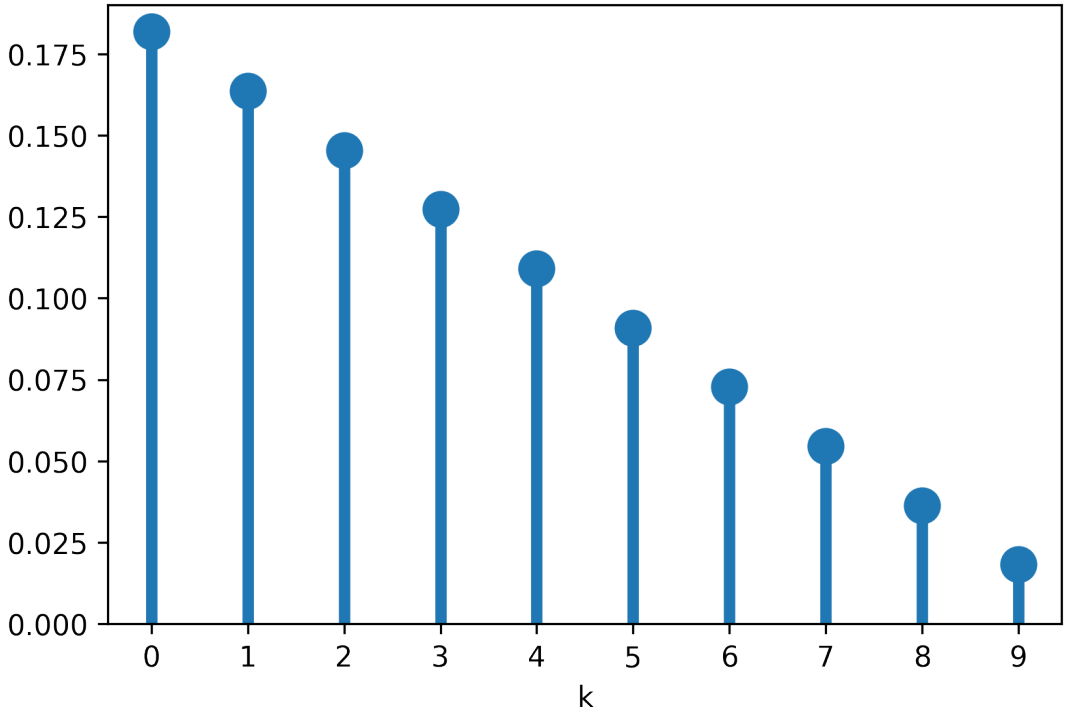


Figure 7: Probabilities of selecting the node at index k of C_u where $|C_u| = 10$

Algorithm 2 Heuristic mapping of a single application

input : C_u , $\overline{c_{ui}}$, μ_{lm}^k , η_n

initialize $s_{cost} = -\eta_n$
initialize $s_x[u] = \perp$, $\forall u \in T$
initialize $R_i^k = Q_i^k$, $\forall i \in I, \forall k \in R$
initialize $R_{lm}^k = Q_{lm}^k$, $\forall (l,m) \in A, \forall k \in R$

procedure - FIX (u, i)

$s_{cost} = s_{cost} + \overline{c_{ui}}$
if $s_{cost} \geq 0$ **then return** FAIL **end if**
$s_x[u] = i$

$R_i^k = R_i^k - q_u^k \ , \forall k \in R$
if $\exists\, v \in T \mid s_x[v] = \perp$, $i \in C_v$ and $k \in R \mid q_v^k > R_i^k$ **then**
remove i from C_v
if $|C_v| = 0$ **then return** FAIL **end if**
end if

for all $(u,v) \in D \mid s_x[v] = \perp$ **do**
remove from C_v all nodes $j \in C_v \mid b_{uv}^{ij} = 0$
if $|C_v| = 0$ **then return** FAIL **end if**
end for

for all $(v,u) \in D \mid s_x[v] = \perp$ **do**
remove from C_v all nodes $j \in C_v \mid b_{vu}^{ji} = 0$
if $|C_v| = 0$ **then return** FAIL **end if**
end for

for all $(u,v) \in D \mid s_x[v] = j$ **do**
$R_{lm}^k = R_{lm}^k - q_{uv}^k \ , \forall k \in R\,, \forall (l,m) \in \mathcal{P}_{ij}$
if $\exists\, k \in R\,, (l,m) \in A \mid R_{lm}^k < 0$ **then return** FAIL **end if**
end for

for all $(v,u) \in D \mid s_x[v] = j$ **do**
$R_{lm}^k = R_{lm}^k - q_{vu}^k \ , \forall k \in R\,, \forall (l,m) \in \mathcal{P}_{ji}$
if $\exists\, k \in R\,, (l,m) \in A \mid R_{lm}^k < 0$ **then return** FAIL **end if**
end for

return SUCCESS

end procedure

---

procedure - PREPROCESS ()

  while TRUE **do**
    found = FALSE
    for $u \in T \mid s_x[u] = \perp$ **do**
      if $|C_u| = 1$ **then**
        found = TRUE
        $i = C_u[0]$
        failed = **FIX** (u, i)
        if failed **then return** FAIL **end if**
      end if
    end for
    if $\neg$ found **then break end if**
  end while

  return SUCCESS

end procedure

failed = **PREPROCESS** ()
if failed **then return** FAIL **end if**
for $u \in T$ scanned ordered by increasing values of $|C_u|$ **do**
  if $s_x[u] = \perp$ **then**
    i = random node in C_u chosen with linearly decreasing probabilities
    failed = **FIX** (u, i)
    if failed **then return** FAIL **end if**
    failed = **PREPROCESS** ()
    if failed **then return** FAIL **end if**
  end if
end for

for $\mu^k_{lm} \mid \mu^k_{lm} > 0 \ \ , \forall (l, m) \in A \, , \forall k \in R$ **do**
  for $(i, j) \mid (l, m) \in \mathcal{P}_{ij}$ **do**
    for $(u, v) \in D$ **do**
      if $s_x[u] = i \ \wedge \ s_x[v] = j$ **then**
        $s_{cost} = s_{cost} + q^k_{uv} \cdot \mu^k_{lm}$
      end if
    end for
  end for
end for

return SUCCESS , s_x **if** $s_{cost} < 0$ **else** FAIL

4.3 Column generation based matheuristics

We want to take advantage of the columns generated during the CG algorithm to quickly find feasible integer solutions for the problem, possibly of good quality.
This section presents three matheuristics that exploit this idea.
A key reference for exploring these techniques is [10].

4.3.1 Discrete LRMP

This technique consists in optimizing the LRMP as an integer problem, considering as variables a restricted set of columns. Typically, these columns are the ones generated during the CG iterations. This approach is straightforward to implement but it has the primary drawback that the discrete LRMP is often infeasible: the available columns cannot be combined into an integer solution. Moreover, the quality of the solutions is poor in many cases. Reasoning on specific problems, it is possible to design procedures to repair DLRMP infeasibility or to enrich the set of available columns. However, significantly improving the tightness of the obtained primal bounds is challenging. Adding too many columns can also slow down the optimization process.

4.3.2 Pure Diving

This technique consists in iteratively rounding up the column that in the optimal solution of the LRMP has the value closest to 1 and re-optimizing the residual LRMP. Before re-optimization, all the columns that cannot contribute to an integer solution for the residual problem, because they would lead to infeasibility, must be removed from the LRMP. The pricing problems must also be updated in order to generate only proper columns. After a maximum of N (number of applications) column roundings the algorithm terminates. It may terminate earlier if the solution to the residual LRMP is integer without explicitly enforcing it or in case of failure (i.e. due to an infeasible residual LRMP or pricing problem).
Re-optimization of the residual LRMP via column generation can produce new columns and thus be time-consuming. It can be done approximately to control the computational effort. The newly generated columns complement the already existing ones to construct a feasible solution. Diving heuristics, exploiting re-optimization as a way to repair infeasibility, are less likely to dead-end with an infeasible residual master compared to the Discrete LRMP heuristic.
Pure Diving (PD) can be viewed as a depth-first search in an LP-based branch-and-bound tree where, at each node, the branch is selected heuristically through rounding. The left side of Figure 8 illustrates this intuition.

The basic PD scheme can be modified to take multiple columns into the solution simultaneously. However, in most cases, rounding one column at a time yields better results. This is probably because a less aggressive fixing allows greater benefits from residual LRMP re-optimization.

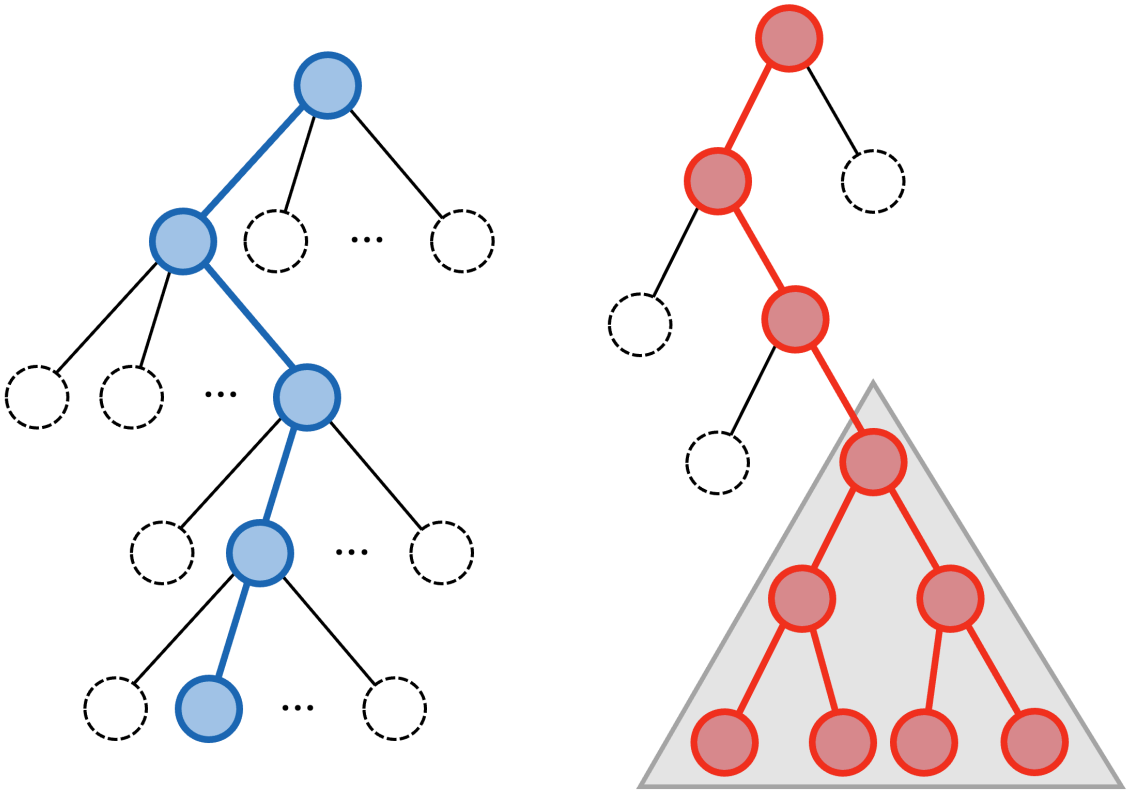


Figure 8: Pure Diving scheme on the left, Rounding + SubMIPing on the right

4.3.3 Rounding + SubMIPing

This technique also involves variable rounding but, instead of fixing entire columns (θ_p^n), it fixes the original variables of the compact model (x_{ui}), which represent single assignments of micro-services to nodes. In the optimal solution of LRMP, the fractional value of a variable x_{ui} is the sum of the fractional values of all columns where u is assigned to node i. Similarly to PD, iteratively the variable in the optimal solution of the LRMP with the value closest to 1 is rounded and the residual LRMP is re-optimized. After fixing $x_{ui} = 1$, before re-optimization, all columns assigning micro-service u to a node $\neq i$ must be removed and the pricing problems must be updated. If no columns are removed (as frequently happens), the re-optimization

phase can be skipped and the next variable is fixed. Also in this case, the optimization of the residual LRMP can be approximate. In Rounding + SubMIPing (R+S), iterations continue until a fraction α of the total number of micro-services has been fixed. The process then enters a SubMIPing phase, where the original problem is optimized with all the previous fixings incorporated in the compact model. Choosing α is a trade-off between avoiding master infeasibility and obtaining a more restricted subproblem to solve exactly. R+S can be viewed as a heuristic depth-first search in the branch-and-bound tree of the original binary integer problem up to a certain depth (that depends on α), followed by an exact optimization step. The right side of Figure 8 illustrates this intuition.
R+S re-optimization is more fine-grained than PD re-optimization, and thus more effective. We expect R+S to find more feasible and good quality solutions with respect to the previous techniques, at the expense of longer computing times caused by the SubMIPing phase.

Chapter 5

Random generation of problem data

This chapter describes the random generation of the key entities in the mapping problem. This process is essential to actually build instances to test the algorithms introduced in the previous chapter.
The random generation of networks is discussed in Section 5.1.
The random generation of applications is discussed in Section 5.2.

5.1 Random generation of networks

The typical network topology we consider includes multiple vehicle nodes, multiple edge nodes and a single cloud node. Given a network with a total number of nodes $n = |I|$, we choose the number of edge nodes n_E to be approximately equal to $\frac{2}{3}\,n$:

$$n_E = \mathcal{U}\{\, \lfloor (\frac{2}{3} - 0.05) \cdot n \rceil \,,\, \lfloor (\frac{2}{3} + 0.05) \cdot n \rceil \,\} \quad \text{where } \mathcal{U} \text{ is the discrete uniform distribution}$$

Once n_E is generated and knowing the number of cloud nodes $n_C = 1$, we can derive the number of vehicle nodes as $n_V = n - n_E - n_C$.

5.1.1 Network topology

The generation process for a random network topology $G_{network}$ is incremental: we first generate only edge nodes, then add vehicle nodes and finally the cloud one.

Edge nodes topology

We model edge nodes topologies using Waxman graphs [11]: a class of random graphs well-suited for modelling the intra-domain part of Internet topology. The nodes of a Waxman graph are uniformly distributed within a rectangular area. The probability of a direct link existing between two nodes i and j is related to their Euclidean distance $d(i, j)$ as follows :

$$P(\{i, j\}) = \beta e^{\frac{-d(i,j)}{L\alpha}}$$

where L is the maximum distance between two nodes and α and β are parameters within the range (0,1]. The probability that two nodes are directly connected decreases exponentially as their Euclidean distance increases. Larger values of β result in graphs with higher link densities, whereas smaller values of α increase the density of short links relative to longer ones. Waxman himself sets $\alpha = \beta = 0.4$, but no agreement exists on appropriate values to be adopted for these two parameters.
Since Waxman graphs are not guaranteed to be connected, we address the issue in the following way. We initially set $\alpha = \beta = 0.01$ and generate a limited number of graphs with n_E nodes. If one of them is connected, then we select it to be our edge topology. If none is connected, then we increment both the parameters by 0.01 and repeat the generation. Figure 9 reports the results of an experiment in which, for topology sizes n_e varying in range $[2, 100]$, we recorded the first parameter settings that yielded a connected graph within 500 generation attempts. Figure 10 shows an example of an edge topology with $n_e = 14$ nodes.

Adding vehicle nodes

Vehicle nodes are the least connected ones in the network. Each vehicle node is assumed to be connected to only one randomly selected edge node, chosen uniformly among all n_e edge nodes. There are no vehicle-vehicle connections. It is possible for two or more vehicle nodes to be connected to the same edge node. Figure 11 represents the topology obtained by adding $n_v = 5$ vehicle nodes to the topology in Figure 10.

Adding the cloud node

The cloud node is the most connected node in the network, it is the one with maximum degree. Let D_E be the degree of the edge node with highest degree in the topology. The cloud node degree D_C is a random value:

$$D_C = \mathcal{U}\{\ MAX(D_E + 1, \lfloor 0.5 \cdot n_e \rceil)\ ,\ MAX(D_E + 1, \lfloor 0.75 \cdot n_e \rceil)\ \}$$

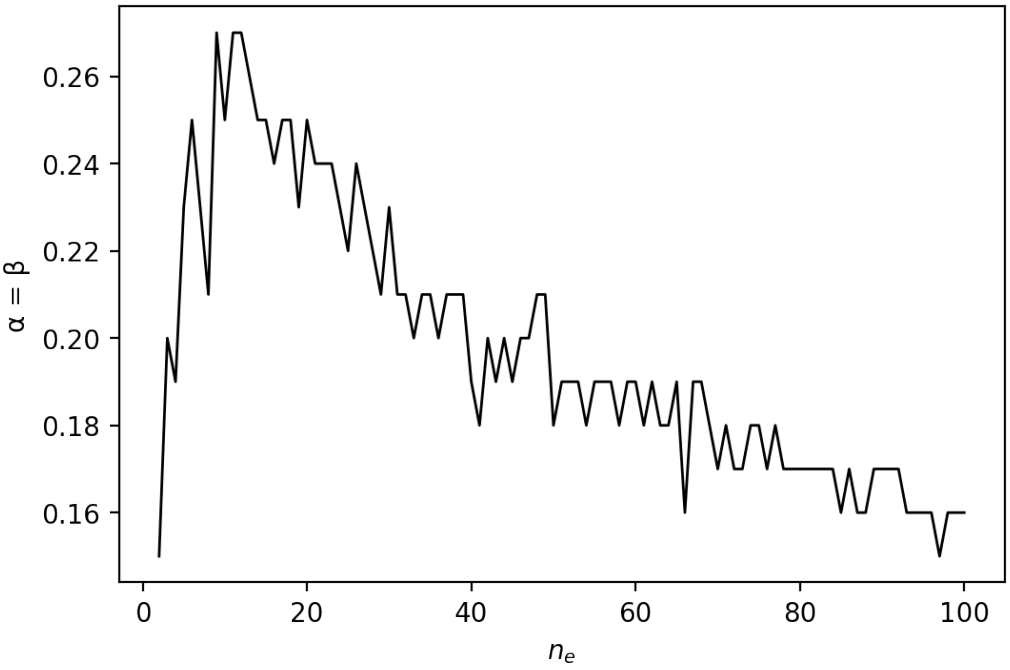


Figure 9: Waxman parameters setting

We assume it to be connected to D_C distinct nodes, chosen uniformly from all n_e edge nodes. There are no cloud-vehicle connections. Figure 12 represents the topology obtained by adding the cloud node to the topology in Figure 11.

Routing paths

The routing path $\mathcal{P}_{ij}$ from node $i \in I$ to node $j \in I$ is selected uniformly from all the shortest paths $i \to j$. Therefore, in general, $\mathcal{P}_{ij} \neq \mathcal{P}_{ji}$.

Node costs

We enforce the following properties:

- allocating a micro-service on the cloud node is always much more convenient than allocating it on an edge node
- allocating a micro-service on an edge node is always much more convenient than allocating it on a vehicle node

To satisfy these requirements, the cost of allocating a micro-service on the cloud node is fixed to 10, the allocation cost on edge nodes is $\in [50, 100]$ and the allocation cost

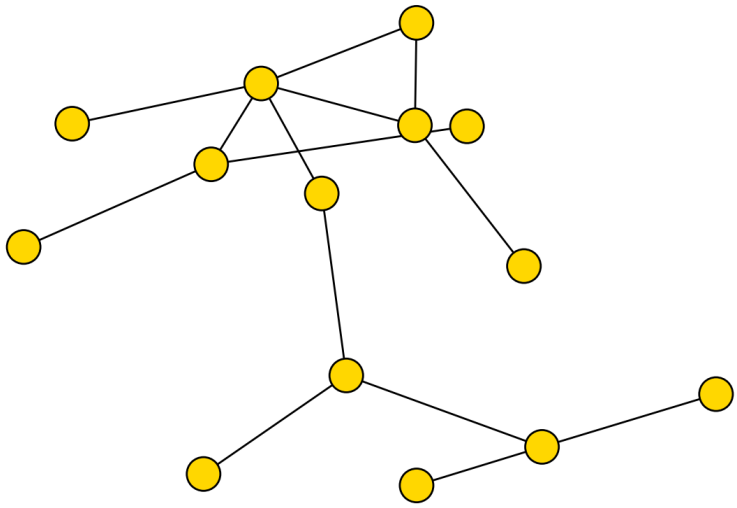


Figure 10: Example of random edge topology

on vehicle nodes is $\in [150, 200]$.
Additionally, within nodes of the same class, we enforce this property:

- allocating micro-services on nodes that are central in the network is less convenient than allocating them on border nodes

To achieve this, the harmonic centrality of each node is calculated, and the the centrality scores are linearly scaled to integers values within the range $[50, 100]$ for edge nodes and $[150, 200]$ for vehicle nodes. Harmonic centrality [12] is a popular centrality measure, defined for each node $i \in I$ as the sum of the reciprocals of the shortest path distances from all other nodes to i :

$$C_i = \sum_{j \in I \ | \ j \neq i} \frac{1}{d(j,i)} \quad \text{where } d(j,i) \text{ is the shortest-path distance between } j \text{ and } i$$

Higher values of C indicate higher centrality.
To scale values x from range $[p, q]$ to values x' within $[p', q']$, the following transformation is applied:

$$x' = (q' - p') \cdot \frac{x - p}{q - p} + p'$$

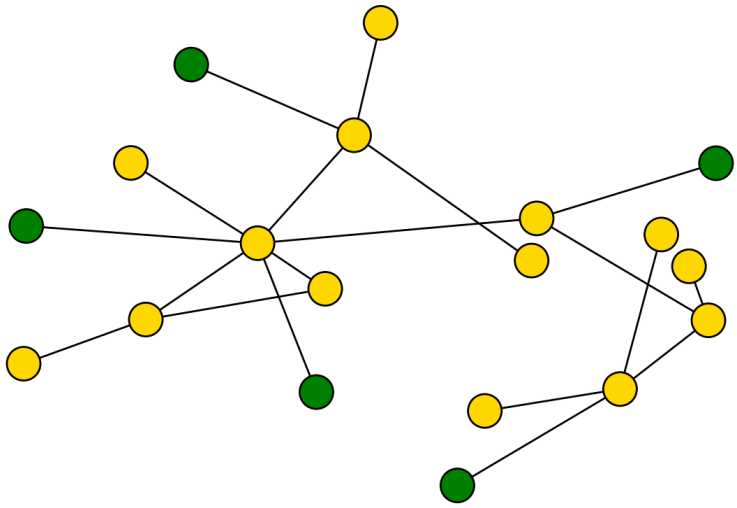


Figure 11: Example of random edge + vehicle topology

5.1.2 Network properties and resources

Although the problem we aim to solve is modeled in a general setting, to actually generate some instances we limit the model to use a specific set of resources and properties. It is common for certain resources (or properties) to be defined only for network nodes or links. If a resource or property k is only defined on nodes/links, it can be assumed to have infinite availability on links/nodes. Consequently, constraints C_4 or C_5 are trivially satisfied for k, and thus they can safely be removed from the model.

We consider the following resources as belonging to the set R :

- #core $\in \mathbb{N}$, a resource defined on network nodes
- bandwidth $\in \mathbb{N}$, a resource defined on network links

We consider the following properties as belonging to the set S :

- has_camera $\in \{0, 1\}$, a property defined on network nodes
- has_gpu $\in \{0, 1\}$, a property defined on network nodes
- latency $\in \mathbb{N}$, a property defined on network links

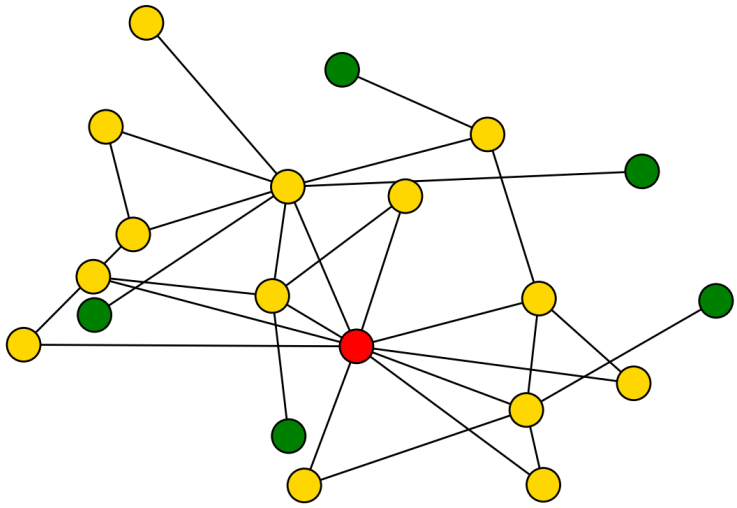


Figure 12: Example of random edge + vehicle + cloud topology

Resource #core

We assume the number of available cores to be infinite for the cloud node and, in most cases, greater for edge nodes than for vehicle nodes. We define :

$$Q_i^{\#\text{core}} = \begin{cases} \mathcal{U}\{\ \underline{c}_v^n\ ,\ \bar{c}_v^n\ \} & \text{if } i \text{ is a vehicle node} \\ \mathcal{U}\{\ \underline{c}_e^n\ ,\ \bar{c}_e^n\ \} & \text{if } i \text{ is an edge node} \\ +\infty & \text{if } i \text{ is the cloud node} \end{cases} \qquad \forall i \in I$$

where $\underline{c}_v^n, \bar{c}_v^n, \underline{c}_e^n$ and $\bar{c}_e^n$ are parameters of the random generation such that $\underline{c}_v^n < \underline{c}_e^n$ and $\bar{c}_v^n < \bar{c}_e^n$.

Resource bandwidth

We assume the bandwidth on links to be distributed in the same way across all the network, irrespective of the type of nodes that specific links connect. The bandwidth is symmetric: it is identical for packets from i towards j and vice versa $\forall (i,j) \in A$. We define :

$$Q_{ij}^{\text{bandwidth}} = Q_{ji}^{\text{bandwidth}} = \mathcal{U}\{\ \underline{b}^n\ ,\ \bar{b}^n\ \} \qquad \forall (i,j) \in A$$

where $\underline{b}^n$ and $\bar{b}^n$ are parameters of the random generation.

Property has_camera

We assume that the cloud node does not have a camera and that vehicle nodes are more likely to have a camera than edge nodes. We define :

$$Q_i^{\text{has\_camera}} = \begin{cases} \mathcal{B}\{\ p_v^n\ \} & \text{if } i \text{ is a vehicle node} \\ \mathcal{B}\{\ p_e^n\ \} & \text{if } i \text{ is an edge node} \\ 0 & \text{if } i \text{ is the cloud node} \end{cases} \qquad \forall i \in I$$

where p_v^n and p_e^n are parameters of the random generation such that $p_v^n > p_e^n$, and $\mathcal{B}$ is the Bernoulli distribution.

Property has_gpu

We assume that the cloud node does not have a GPU and that edge nodes are more likely to have a GPU than vehicle nodes. We define :

$$Q_i^{\text{has\_gpu}} = \begin{cases} \mathcal{B}\{\ q_v^n\ \} & \text{if } i \text{ is a vehicle node} \\ \mathcal{B}\{\ q_e^n\ \} & \text{if } i \text{ is an edge node} \\ 0 & \text{if } i \text{ is the cloud node} \end{cases} \qquad \forall i \in I$$

where q_v^n and q_e^n are parameters of the random generation such that $q_v^n < q_e^n$.

Property latency

We assume unit latency across all the network:

$$Q_{ij}^{\text{latency}} = 1 \qquad \forall (i,j) \in A$$

This implies that we can interpret latency as a #hop property : requiring a routing path to have latency $\leq k$ is equivalent to requiring the path to traverse no more than k links.

5.1.3 Format of network files

To describe a network, we have one file to represent its topology and another file to specify its properties and resources.
Topology files for networks follow this format :

- line 1 : "< number of nodes $n = |I|$ >" , (nodes are named from 0 to $n - 1$)
- line 2 : "< space-separated list of links $(i, j) \in A \mid i < j$ >"
- line 3 : "< space-separated list of node costs c_i , $\forall i \in I$ >"
- lines ≥ 4 : "$< i > < j >$: < space-separated list of nodes in $\mathcal{P}_{ij}$ >" , $\forall i, j \in I$

Properties and resources files for networks follow this format :

- line 1 : "core"
- line 2 : "< space-separated list of $Q_i^{\#\text{core}}$, $\forall i \in I$ >"
- line 3 : "has_camera"
- line 4 : "< space-separated list of $Q_i^{\text{has\_camera}}$, $\forall i \in I$ >"
- line 5 : "has_gpu"
- line 6 : "< space-separated list of $Q_i^{\text{has\_gpu}}$, $\forall i \in I$ >"
- line 7 : "bandwidth"
- lines : "$< i >, < j > < Q_{ij}^{\text{bandwidth}} >$" , $\forall (i, j) \in A \mid i < j$
- line 7+1+$|A/2|$: "latency"
- lines : "$< i >, < j > < Q_{ij}^{\text{latency}} >$" , $\forall (i, j) \in A \mid i < j$

An example of network files is provided in Appendix A.1.

5.2 Random generation of applications

As we did with networks, also for applications we handle separately the generation of their topology and their requirements of properties and resources.
Each application consists of a number of micro-services $m = |T|$. We decided to keep $3 \leq m \leq 15$, as real-world scenarios often focus on mapping many small- to medium-sized applications rather than a few large ones.

5.2.1 Application topology

We assume that each randomly generated application topology G_{app} is either a path, a tree or a graph. It is reasonable to believe that mapping a simple application (e.g. a path of micro-services) is generally easier than mapping a complex one (e.g. a graph of micro-services with many communication dependencies).

Path topology

The simplest application topology we can generate is a path of micro-services. Obviously, there is only one possible path topology for each setting of m. Figure 13 is the path topology with $m = 7$ micro-services.

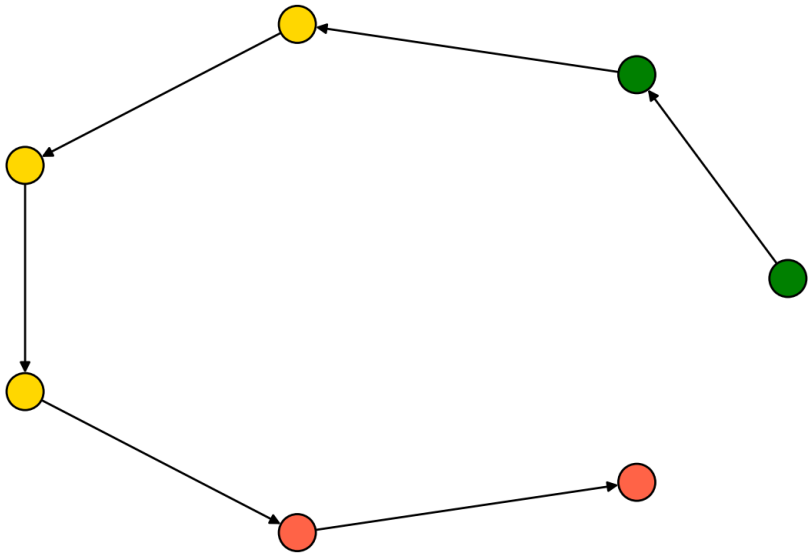


Figure 13: Example of an application with path topology

Tree topology

A more complex application topology we can generate is a rooted tree with all arcs reversed. To randomly generate such a topology for a given m:

- initialize m nodes, named from 0 to $m-1$, with no edges
- for i in $1 \dots m$: add the edge (i, j) where $j = \mathcal{U}\{0, i-1\}$

Node 0 is the root of the inverse rooted tree, it has no outgoing arcs. There are multiple possible tree topologies for each value of m. Figure 14 shows an example of a tree topology with $m = 12$ micro-services. The red node in the bottom-left corner is the root.

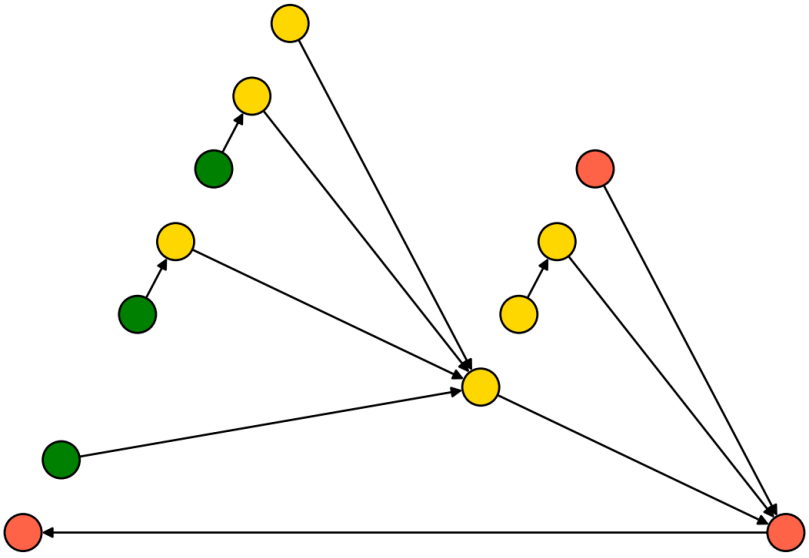


Figure 14: Example of an application with tree topology

Graph topology

The graph application topology is constructed by adding arcs to a tree topology. The added arcs are chosen uniformly at random from all arcs that are not the $m-1$ ones already in the tree. The number of added arcs is modeled as a discrete random variable $X_m = \lfloor Y_m \rfloor$, where Y_m is a random variable having a truncated normal distribution with parameters $\mu = \lfloor \frac{m}{3} \rceil$, $\sigma = \frac{m}{3}$, lower bound $a = 1$ and upper bound $b = m$. We are allowing the number of arcs to double at most, ensuring the resulting

graphs to remain quite sparse in general. Figure 15 shows the empirical probability mass function for X_m given $m = 12$. Figure 16 shows an example of a graph topology with $m = 12$ micro-services.

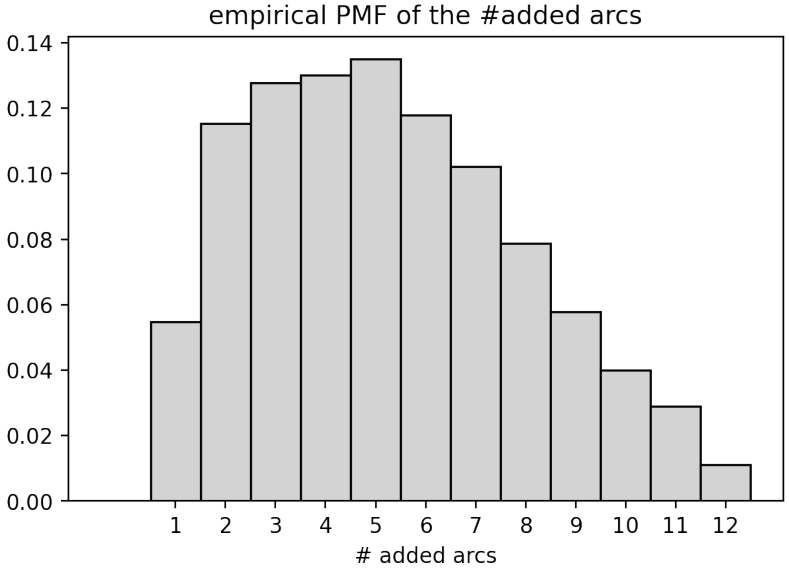


Figure 15: Empirical PMF of the number of arcs to add X_m for $m = 12$

5.2.2 Application properties and resources

In Section 5.1.2 we defined the resources availability for each network node and link. This section introduces the resources consumption of each micro-service and each communication dependency between micro-services.

Types of micro-services

In the network context, the distinction between edge, vehicle and cloud nodes is precise. In the application context we do not establish a priori that a micro-service will be placed on a specific type of node. Instead, we set the resource requirements of micro-services in a way that makes them more likely to be mapped to a specific type of node (e.g. a micro-service requiring many cores is likely to be mapped to the cloud node, while a micro-service requiring a GPU cannot be mapped to the cloud node). A micro-service that is likely to be mapped to a vehicle, edge or cloud node is referred to as a vehicle, edge or cloud micro-service.

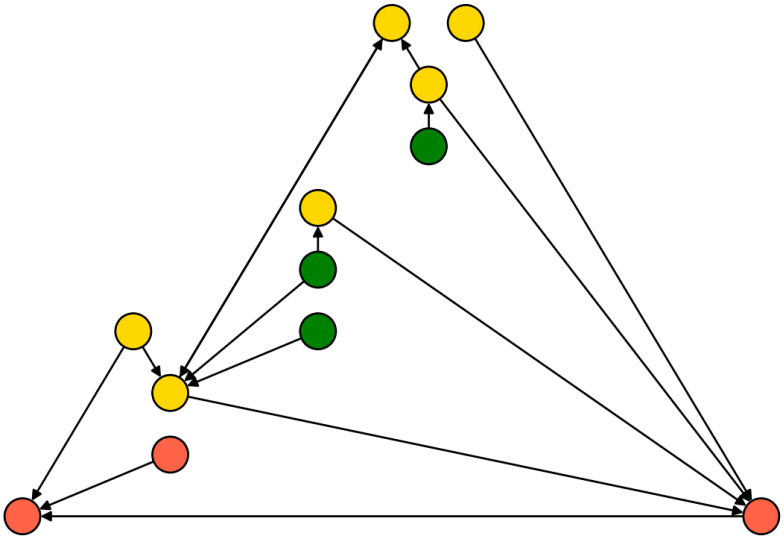


Figure 16: Example of an application with graph topology

Given an application with m micro-services, we set:

$$m_e = \lceil (m-2) \cdot 0.6 \rceil \quad \text{, where } m_e \text{ is the number of edge micro-services}$$

$$m_v = \lceil (m-m_e)/2 \rceil \quad \text{, where } m_v \text{ is the number of vehicle micro-services}$$

$$m_c = m - m_e - m_v \quad \text{, where } m_c \text{ is the number of cloud micro-services}$$

For applications with path and tree topology, we define the m_v nodes with the highest shortest-path distance from the only node with no outgoing arcs as vehicle micro-services. Similarly, we define the m_c nodes with the lowest shortest-path distance from the only node with no outgoing arcs as cloud micro-services. The remaining nodes are defined as edge micro-services. Since applications with graph topology are obtained by adding arcs to a tree, their micro-service types are determined before introducing the additional arcs by treating the structure as a tree. In Figure 13, 14 and 16 vehicle micro-services are shown in green, edge micro-services in yellow and cloud micro-services in red.

Resource #core

We define :

$$q_u^{\#\text{core}} = \begin{cases} \mathcal{U}\{\ \underline{c}_v^a\ ,\ \bar{c}_v^a\ \} & \text{if } u \text{ is a vehicle micro-service} \\ \mathcal{U}\{\ \underline{c}_e^a\ ,\ \bar{c}_e^a\ \} & \text{if } u \text{ is an edge micro-service} \\ \mathcal{U}\{\ \underline{c}_c^a\ ,\ \bar{c}_c^a\ \} & \text{if } u \text{ is a cloud micro-service} \end{cases} \qquad \forall u \in T$$

where $\underline{c}_v^a, \bar{c}_v^a, \underline{c}_e^a, \bar{c}_e^a, \underline{c}_c^a$ and $\bar{c}_c^a$ are parameters of the random generation such that $\underline{c}_v^a < \underline{c}_e^a < \underline{c}_c^a$ and $\bar{c}_v^a < \bar{c}_e^a < \bar{c}_c^a$.

Resource bandwidth

We define :

$$q_{uv}^{\text{bandwidth}} = \mathcal{U}\{\ \underline{b}^a\ ,\ \bar{b}^a\ \} \qquad \forall (u,v) \in D$$

where $\underline{b}^a$ and $\bar{b}^a$ are parameters of the random generation.

Property has_camera

We define :

$$q_u^{\text{has\_camera}} = \begin{cases} \mathcal{B}\{\ p_v^a\ \} & \text{if } u \text{ is a vehicle micro-service} \\ \mathcal{B}\{\ p_e^a\ \} & \text{if } u \text{ is an edge micro-service} \\ 0 & \text{if } u \text{ is a cloud micro-service} \end{cases} \qquad \forall u \in T$$

where p_v^a and p_e^a are parameters of the random generation such that $p_v^a > p_e^a$.

Property has_gpu

We define :

$$q_u^{\text{has\_gpu}} = \begin{cases} \mathcal{B}\{\ q_v^a\ \} & \text{if } u \text{ is a vehicle micro-service} \\ \mathcal{B}\{\ q_e^a\ \} & \text{if } u \text{ is an edge micro-service} \\ 0 & \text{if } u \text{ is a cloud micro-service} \end{cases} \qquad \forall u \in T$$

where q_v^a and q_e^a are parameters of the random generation such that $q_v^a < q_e^a$.

Property latency

We define :

$$q_{uv}^{\text{latency}} = \begin{cases} 2 & \text{with probability } l_2 \\ 3 & \text{with probability } l_3 \\ 4 & \text{with probability } l_4 \end{cases} \qquad \forall (u,v) \in D$$

where l_2, l_3 and l_4 are parameters of the random generation such that $l_2 + l_3 + l_4 = 1$.

5.2.3 Format of application files

To describe an application, we have one file to represent its topology and another file to specify its properties and resources.
Topology files for applications follow this format :

- line 1 : "< number of micro-services $m = |T|$ >" , (micro-services are named from 0 to $m - 1$)
- line 2 : "< space-separated list of communication dependencies $(u, v) \in D$ >"

Properties and resources files for applications follow this format :

- line 1 : "core"
- line 2 : "< space-separated list of $q_u^{\#\text{core}}$, $\forall u \in T$ >"
- line 3 : "has_camera"
- line 4 : "< space-separated list of $q_u^{\text{has\_camera}}$, $\forall u \in T$ >"
- line 5 : "has_gpu"
- line 6 : "< space-separated list of $q_u^{\text{has\_gpu}}$, $\forall u \in T$ >"
- line 7 : "bandwidth"
- lines : "$< u >, < v > < q_{uv}^{\text{bandwidth}} >$" , $\forall (u, v) \in D$
- line 7+1+$|D|$: "latency"
- lines : "$< u >, < v > < q_{uv}^{\text{latency}} >$" , $\forall (u, v) \in D$

An example of application files is reported in Appendix A.2.

Chapter 6

Experiments

This chapter analyzes the performances of the algorithms presented in Chapter 4 on a dataset of instances. The creation of the dataset, which exploits the random generation techniques introduced in Chapter 5, is discussed in Section 6.1.
The computational results of the algorithms are reported in Section 6.2.

6.1 Dataset

We considered network topologies with a number of nodes $n \in \{30, 40, 50, 60, 70\}$.
For each value of n we generated 4 different random network topologies. For each topology, we then generated 5 random configurations of properties and resources.

We considered applications with a number of micro-services $m \in \mathbb{N} \mid 3 \leq m \leq 15$.
For each m we generated the following random application topologies: the path topology, 4 different tree topologies and 4 different graph topologies. For each topology, we then generated 5 random configurations of properties and resources.

Table 1 reports the settings of all the parameters required to generate networks and applications. These are the parameters that has been presented in Chapter 6.

The dataset is composed of instances where we fix: a network of size n, a configuration of properties and resources for the network, k applications to map onto the network and a configuration of properties and resources for each application.
We generated 3 instances for each combination of $n \in \{30, 40, 50, 60, 70\}$ and $k \in \{2, 3, ..., 30\}$. The dataset has 435 instances in total.
Once n is fixed, the network topology and the resource availabilities of each instance are chosen uniformly from the previously generated ones.

	availability Q	consumption q
#core	$\underline{c}_v^n = 10$, $\bar{c}_v^n = 150$ $\underline{c}_e^n = 100$, $\bar{c}_e^n = 400$ $(+\infty)$	$\underline{c}_v^a = 5$, $\bar{c}_v^a = 75$ $\underline{c}_e^a = 80$, $\bar{c}_e^a = 250$ $\underline{c}_c^a = 350$, $\bar{c}_c^a = 900$
has_camera	$p_v^n = 0.8$, $p_e^n = 0.65$	$p_v^a = 0.5$, $p_e^a = 0.35$
has_gpu	$q_v^n = 0.65$, $q_e^n = 0.8$	$q_v^a = 0.35$, $q_e^a = 0.5$
bandwidth	$\underline{b}^n = 100$, $\bar{b}^n = 600$	$\underline{b}^a = 50$, $\bar{b}^a = 200$
latency	(always 1)	$l_2 = 0.15$, $l_3 = 0.45$, $l_4 = 0.4$

Table 1: Parameters setting for generating networks and applications

Each of the k applications has a topology with probabilities of 0.2 to be a path, 0.4 to be a tree and 0.4 to be a graph, and is chosen from all the previously generated application topologies (of any size).

For an instance composed of a network of size n and a total number of micro-services across all the applications M, we define its density as:

$$density\ (\text{ instance }) = \begin{cases} \lfloor \frac{M}{n} \rfloor + 0.5 \ , & \text{if } \frac{M}{n}\ mod\ 1 \leq 0.5 \\ \lceil \frac{M}{n} \rceil \ , & \text{otherwise} \end{cases}$$

6.2 Computational results

The code has been written in *Python* and makes extensive use of the library *gurobipy*, which is an interface to the *GuRoBi* Optimizer (version 11.0.2).
It is available at : `https://github.com/nemolino/master-thesis`
The machine used to run the experiments has an Apple M1 CPU with 8 physical cores and 16GB of RAM.

6.2.1 Root node relaxation

Computing the LP relaxation of the compact model of our problem (Section 3.2) is the most straightforward method to obtain a dual bound for the optimal solution. However, this bound is not easily computable due to the presence of the quadratic constraints C_5 in the formulation.

To deal with this issue, we consider the root node relaxation (RNR) instead of the standard LP relaxation. RNR is the LP relaxation of the problem at the root node of the branch-and-bound tree, i.e. the first LP relaxation that is solved after presolving. Nowadays, commercial solvers have advanced capabilities of preprocessing formulations and strengthen them by generating a lot of useful cuts. As a result, RNR dual bounds are typically tighter than those obtained from the basic LP relaxation and, in our case, quite fast to compute. We calculate the RNR of all the instances in the dataset to identify the trivially infeasible ones. We find that 234/435 instances have infeasible RNR and can thus be excluded from further analysis. The forthcoming experiments will only consider the 201 instances left.
Figure 17 reports the percentage of instances with feasible RNR and their execution times, grouping instances by network size, number of applications and density. It is evident that RNR feasibility is related to instance density: all instances with density ≥ 4 are infeasible. The execution time to compute RNR appears to increase for instances with density close to 4. When the density is much larger or smaller, the instances are either trivially infeasible or trivially feasible. In Section 6.2.3 we will compare the RNR bounds with those obtained from column generation.

6.2.2 Compact model

To asses the actual hardness of the problem and establish a baseline for future performance comparisons, we solve the compact formulation of the instances using the *GuRoBi* solver. We impose a 30 minutes time limit on the solve time.
The solver terminates each run with one of four possible statuses:

- OPTIMAL: an optimal integer solution has been found for the instance
- $>$ 30 min with PB (primal bound): a feasible integer solution has been found for the instance, but it has not been proven to be optimal
- $>$ 30 min with no PB: so far, no feasible integer solution has been found for the instance, but the instance has not been proven to be infeasible
- INFEASIBLE: the instance has been proven to be infeasible

In Figure 18 we report the solver status at the end of each run, grouping instances by network size, number of applications and density. Once again, higher-density instances are more prone to be infeasible and, in this case, also more difficult to solve. Table 2 presents some statistics, aggregated by number of applications, on the solve time of instances where the time limit has not been reached.

6.2.3 Column generation

We apply Column generation (CG) to the instances and, restricting ourselves to those which the solver finds infeasible or for which it cannot find a feasible solution within 30 minutes (they are 32/210), it happens that :

- in 3 cases, the compact model is infeasible but CG provides a lower bound
- in 20 cases, the compact model does not provide a primal bound within the time limit, while CG successfully provides a lower bound
- in 6 cases, the compact model is infeasible and CG is also infeasible. In these situations, there is no computing time gain in identifying infeasibility with CG if it iterates until the end, as it cannot remove dummy columns from the base. Instead, if CG is infeasible due to an infeasible pricing problem, it can stop immediately without even iterating.
- in 3 cases, the compact model does not provide a primal bound within the time limit and CG is infeasible. These situations result in a significant computing time gain in by identifying with CG, even when it iterates until the end trying to remove dummy columns from the base.

CG finds an integer solution 6 times. These integer solutions are also optimal but they only refer to easy instances consisting of 2 applications.

In Figure 19 we report statistics on CG execution time. The results are restricted to instances for which CG finds a lower bound. We treat the cases in which CG fails as outliers because their execution time can be either very short (infeasible pricing) or very long (infeasible master). It is worth noting that optimizing pricing problems takes, on average 90%, of the total CG computing time.
A comparison between the tightness of RNR and CG bounds is presented in Figure 20. The gaps are calculated based on the optimum obtained by the solver or, if it is not available, on the cost of a feasible solution. CG almost always provides better bounds with respect to RNR. RNR is tighter than CG in only 5/210 instances, with a maximum improvement of 0.07 percentage points. Figure 21 shows the percentage improvements of CG bounds over RNR bounds. All statistics are grouped by network size, number of applications and density.

6.2.4 Column generation with heuristic pricing

We apply the metaheuristic described in Section 4.2 to try to solve the pricing problems. The algorithm for searching for negative reduced cost columns is executed

multiple times until either 10 valid columns are found or a time limit of 50ms is reached. Heuristic pricing on PP^n is performed until it fails to find a column within the time limit for the first time. From that point onward, the pricing problem is solved exactly.
In Figure 22 we report the percentage computing time improvement achieved by CG with heuristic pricing compared to basic CG. Except in cases with few applications, the approach on average reduces computing times. In some cases, the improvement exceeds 20%. However, for almost any application size, there are instances for which the computing time increases. We hypothesize that in these situations the meta-heuristic finds columns that, while having negative reduced cost, are poor quality with respect to the optimal one. As a consequence, CG needs more iterations to converge and thus can have a longer overall runtime.

6.2.5 Column generation based matheuristics

The matheuristics have been applied to the instances for which CG provided a lower bound, they are 187/201. The number of instances in which each heuristic succeeds is shown in Figure 23, with results aggregated by the number of applications. The grey bars represent the maximum number of possible successes for each group. This excludes the cases where heuristics are applied but have no chance to succeed due to the instance being infeasible (only 3 such cases). A comparison of the average execution times of the three techniques is presented in Figure 24. Statistics regarding the percentage gaps between the obtained primal bounds and the optimal solutions are reported in Table 3.

Discrete LRMP

DLRMP heuristic succeeds in 31/184 cases and finds the optimum 12 times.
There are no cases where DLRMP finds a solution while PD or R+S fail.
DLRMP provides a better bound than PD in 15 cases and than R+S in 5 cases.
It never succeeds when the solver reaches the time limit.
It never succeeds for instances with more than 8 applications.

Pure Diving

PD heuristic succeeds in 107/184 cases and finds the optimum 9 times.
There are only 2 cases where PD finds a solution while R+S fails.
Additionally, it provides a better bound than R+S in 4 cases.
It succeeds when the solver reaches the time limit with primal bound in 8/42 cases.
It never succeeds for instances with more than 14 applications.

Rounding + SubMIPing

We set the parameter $\alpha = 0.6$, as it leads to master infeasibility in only 2 cases, produces very tight gaps and keeps the SubMIPing times fairly low.
R+S heuristic succeeds in 145/184 cases and finds the optimum 23 times.
It succeeds when the solver reaches the time limit with primal bound in 31/42 cases.
It succeeds when the solver reaches the time limit with no primal bound in 1/20 case.
It never succeeds for instances with more than 21 applications.
The SubMIPing part reaches the 10-minute time limit in 9 cases.

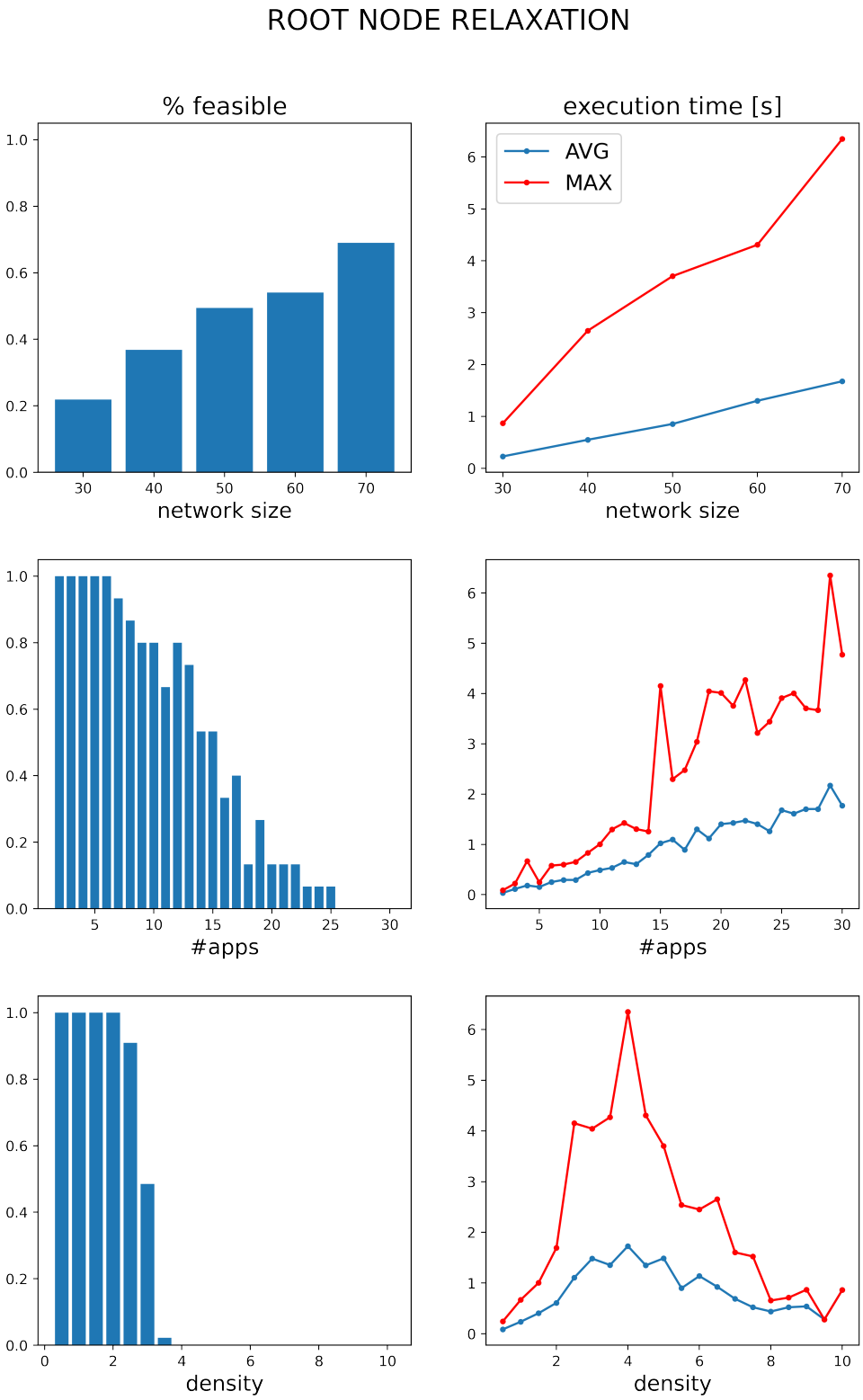


Figure 17: Root node relaxation statistics

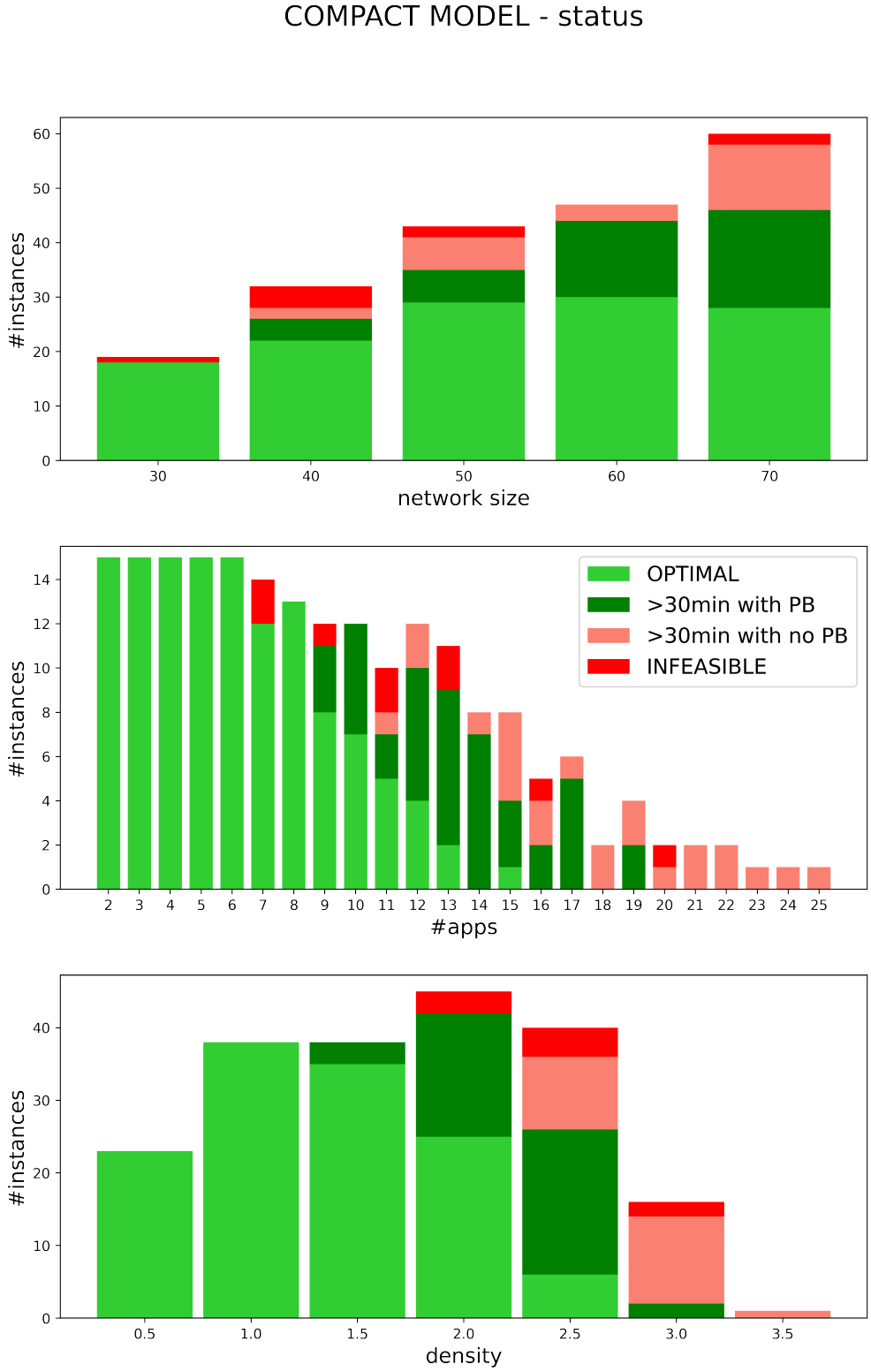


Figure 18: Compact model solve statuses

			compact model time [s]			
#apps	compact model status	count	min	mean	std	max
2	OPTIMAL	15	0.02	0.18	0.14	0.56
3	OPTIMAL	15	0.04	0.67	0.58	2.15
4	OPTIMAL	15	0.12	14.85	53.19	207.09
5	OPTIMAL	15	0.40	38.68	125.69	491.68
6	OPTIMAL	15	0.44	13.65	16.07	57.52
7	INFEASIBLE	2	0.38	1.28	1.27	2.18
	OPTIMAL	12	1.43	13.27	15.33	48.17
8	OPTIMAL	13	1.55	32.38	47.52	157.71
9	INFEASIBLE	1	13.02	13.02		13.02
	OPTIMAL	8	3.46	159.38	208.91	489.46
	>30 min with PB	3				
10	OPTIMAL	7	9.85	259.81	369.23	1004.15
	>30 min with PB	5				
11	INFEASIBLE	2	0.74	0.96	0.32	1.19
	OPTIMAL	5	5.03	487.50	616.21	1499.80
	>30 min with no PB	1				
	>30 min with PB	2				
12	OPTIMAL	4	109.38	335.82	212.75	609.32
	>30 min with no PB	2				
	>30 min with PB	6				
13	INFEASIBLE	2	2.68	6.43	5.29	10.17
	OPTIMAL	2	362.39	412.34	70.65	462.30
	>30 min with PB	7				
14	>30 min with no PB	1				
	>30 min with PB	7				
15	OPTIMAL	1	57.12	57.12		57.12
	>30 min with no PB	4				
	>30 min with PB	3				
16	INFEASIBLE	1	13.05	13.05		13.05
	>30 min with no PB	2				
	>30 min with PB	2				
17	>30 min with no PB	1				
	>30 min with PB	5				
18	>30 min with no PB	2				
19	>30 min with no PB	2				
	>30 min with PB	2				
20	INFEASIBLE	1	65.02	65.02		65.02
	>30 min with no PB	1				
21	>30 min with no PB	2				
22	>30 min with no PB	2				
23	>30 min with no PB	1				
24	>30 min with no PB	1				
25	>30 min with no PB	1				

Table 2: Compact model times

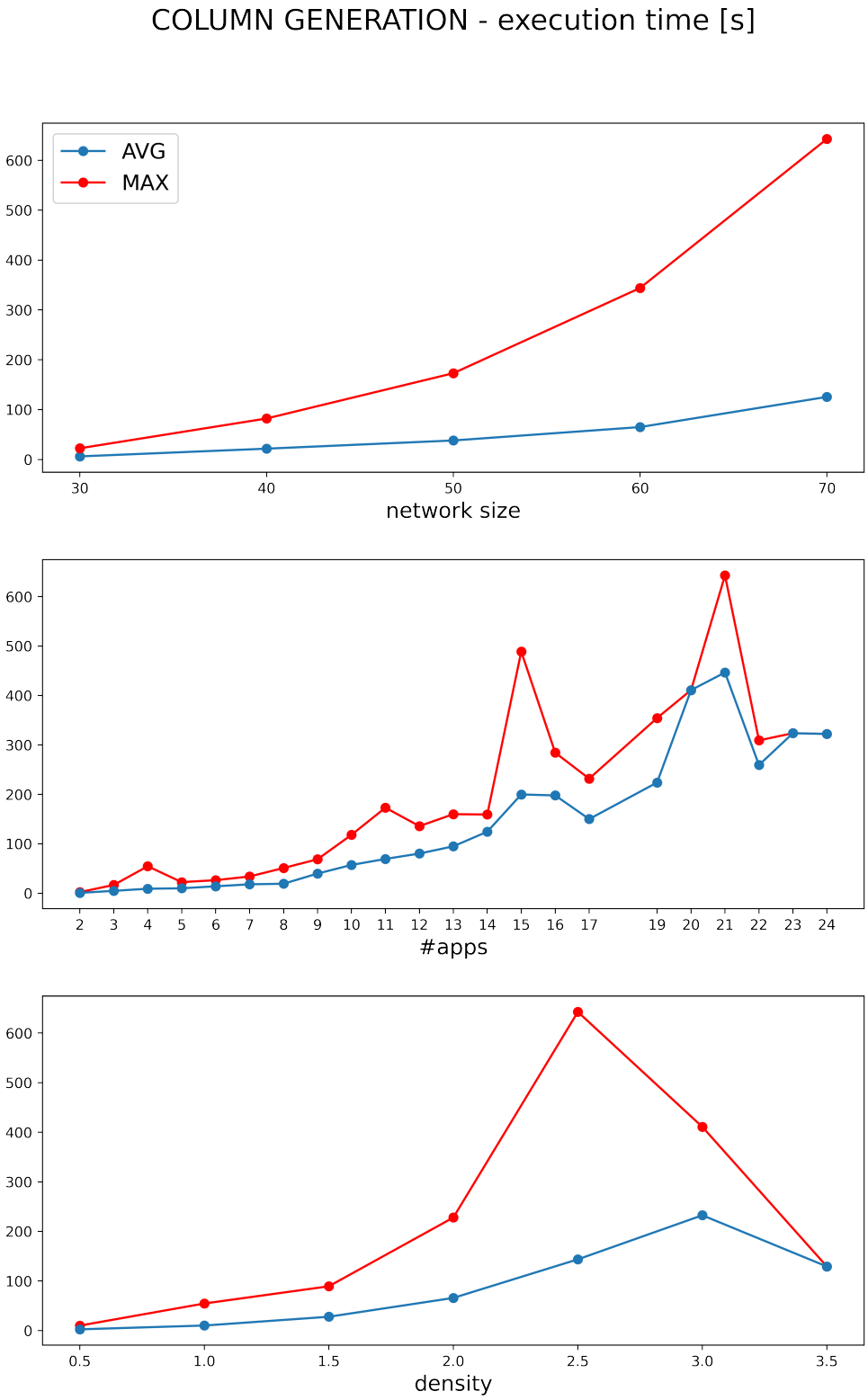


Figure 19: Column generation times

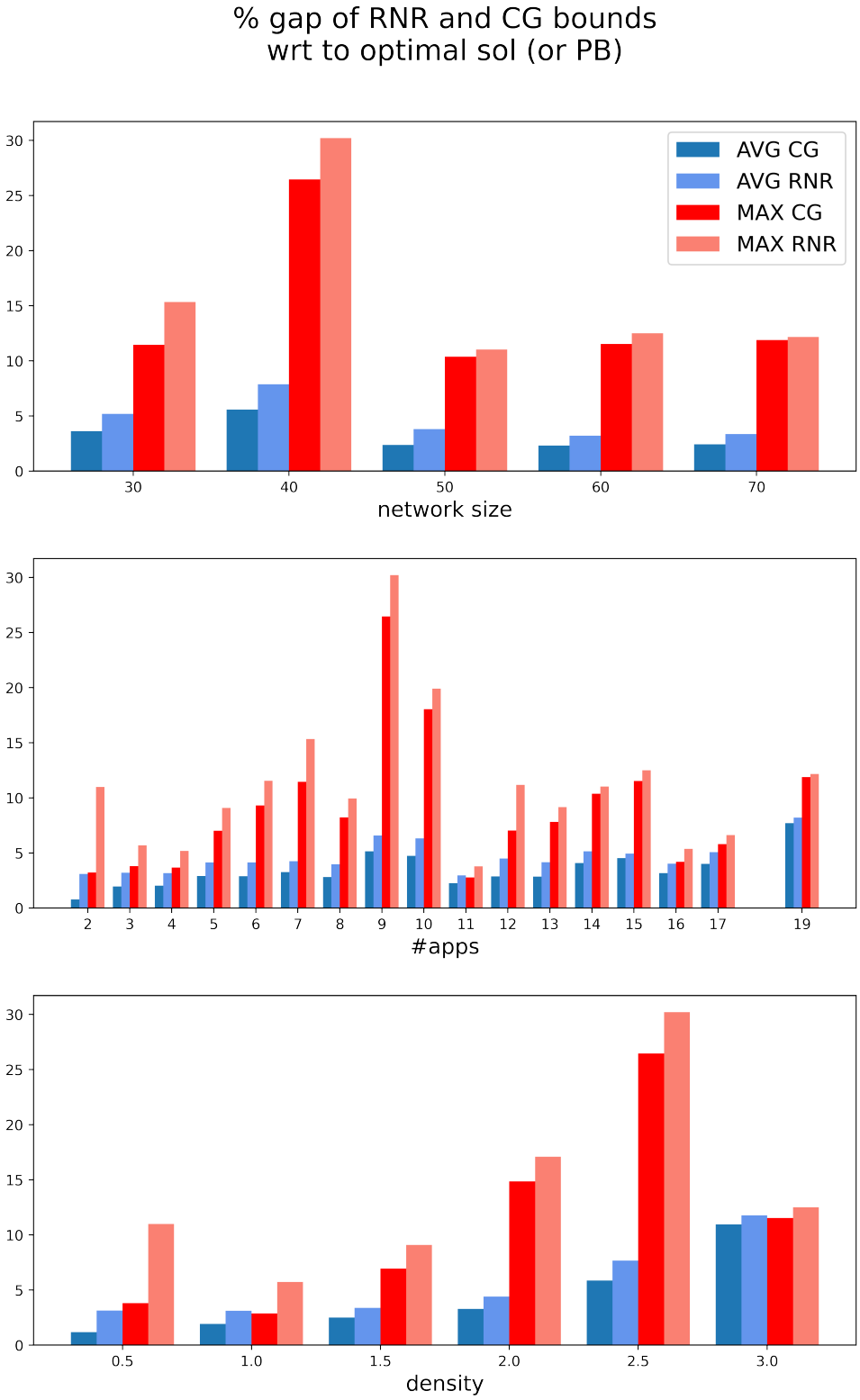


Figure 20: Percentage gaps from the optimum

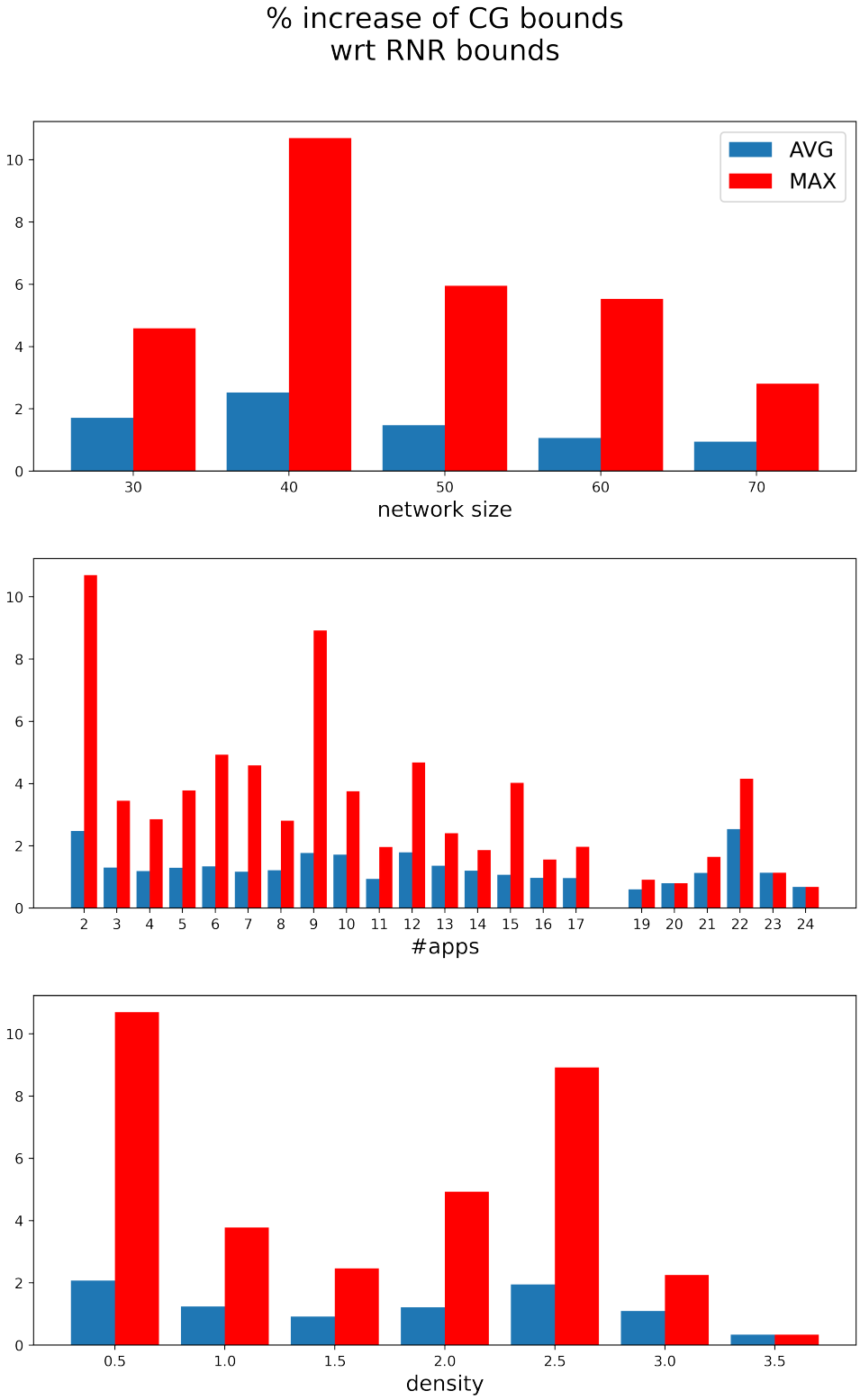


Figure 21: Percentage bound improvements

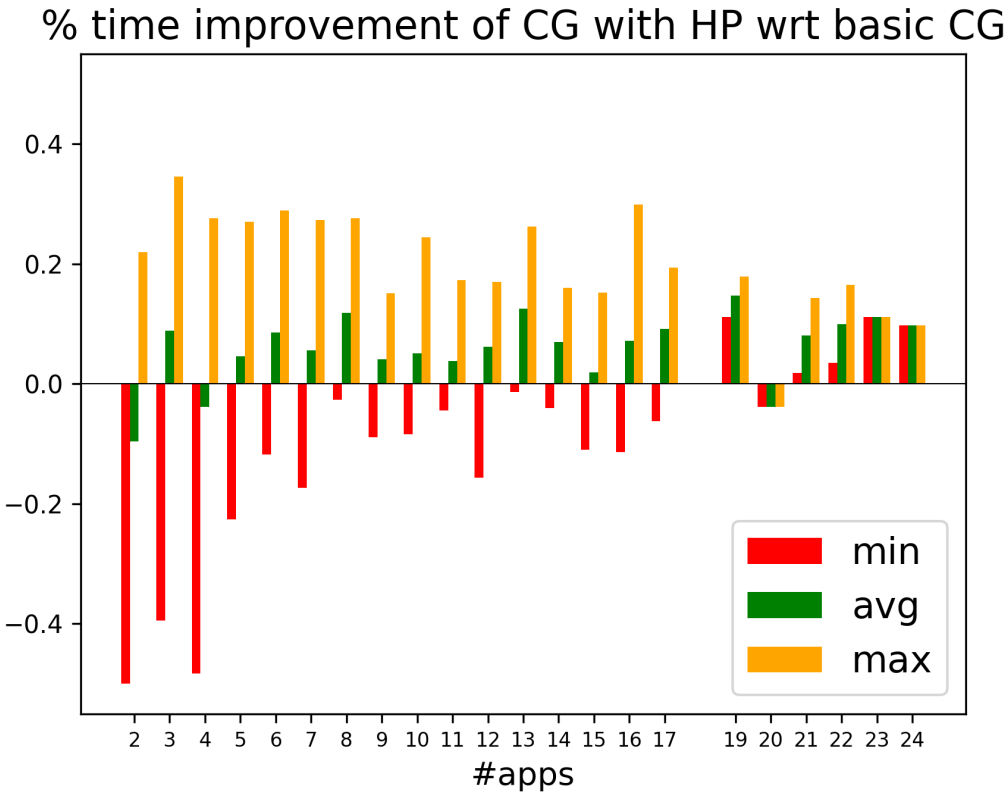


Figure 22: Percentage time improvements of CG with heuristic pricing

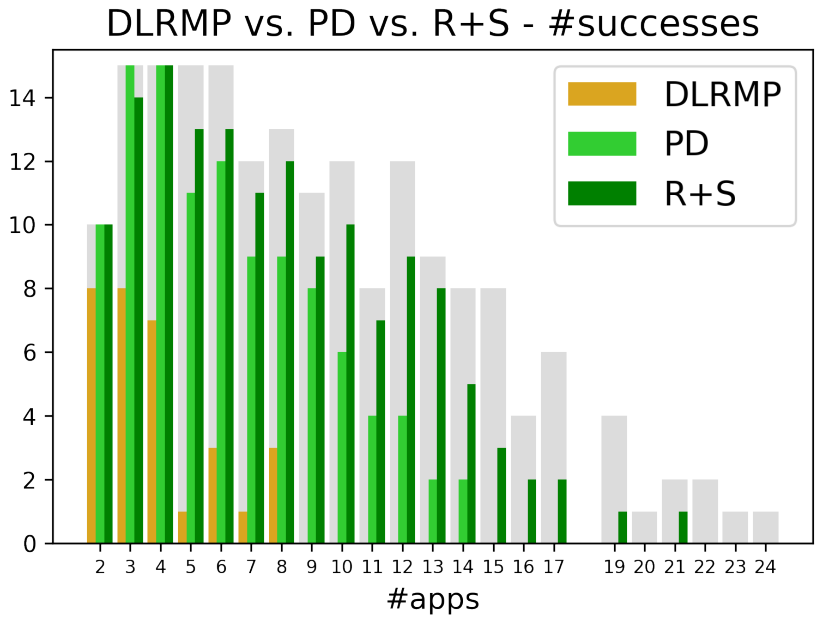


Figure 23: Matheuristics #successes

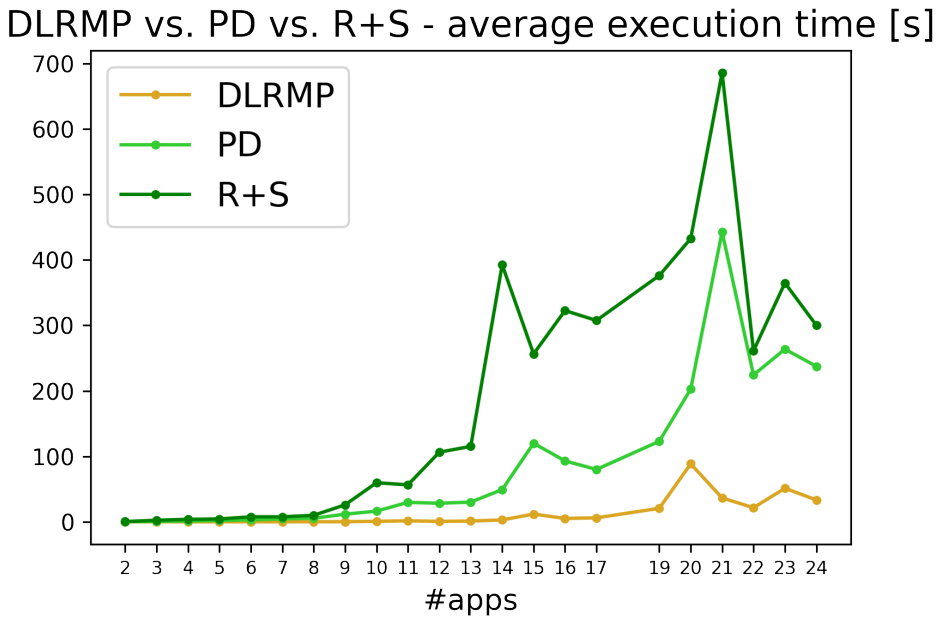


Figure 24: Matheuristics average times

		GAP DLRMP			
#apps	count	min	mean	std	max
2	8	0.00	0.08	0.16	0.42
3	8	0.00	1.67	1.87	5.52
4	7	0.00	1.38	1.83	5.23
5	1	0.00	0.00		0.00
6	3	0.00	2.59	3.06	5.96
7	1	0.17	0.17		0.17
8	3	0.61	1.22	0.53	1.55

		GAP PD			
#apps	count	min	mean	std	max
2	10	0.00	0.79	1.04	3.12
3	15	0.00	1.59	1.28	3.66
4	15	0.00	1.60	1.04	3.83
5	11	0.06	1.53	0.88	3.06
6	12	1.10	2.79	1.29	4.74
7	9	0.21	1.71	1.45	4.64
8	9	1.04	2.47	1.01	4.08
9	8	2.16	2.88	0.76	4.35
10	6	1.59	1.97	0.36	2.59
11	4	0.71	3.48	2.16	5.31
12	4	2.66	3.66	1.26	5.38
13	2	1.95	2.31	0.51	2.67
14	2	2.72	3.99	1.80	5.26

		GAP R+S			
#apps	count	min	mean	std	max
2	10	0.00	0.29	0.49	1.18
3	14	0.00	0.61	0.6	1.58
4	15	0.00	0.32	0.39	1.26
5	13	0.00	0.61	0.51	1.29
6	13	0.12	0.86	0.44	1.48
7	11	0.00	0.46	0.5	1.62
8	12	0.24	0.61	0.6	2.42
9	9	0.05	0.63	0.57	1.75
10	10	-0.10	1.28	2.37	7.81
11	7	0.20	0.70	0.27	1.05
12	9	0.04	1.21	2.64	8.22
13	8	-3.23	0.29	1.56	2.21
14	5	0.09	0.25	0.11	0.40
15	3	0.88	1.29	0.37	1.59
16	2	0.25	0.62	0.52	0.99
17	2	0.30	1.21	1.29	2.12
19	1	-0.94	-0.94		-0.94
21	1	-	-	-	-

Table 3: Matheuristics gaps

Chapter 7

Conclusions

The problem of mapping multiple applications onto the network infrastructure has been formalized. We presented two mathematical formulations for it, a compact and extended one. By exploiting the extended formulation, which naturally yields a decomposition pattern into single applications, we applied the column generation technique to obtain dual bounds. We devised a constructive metaheuristic algorithm to solve the pricing problems, as an alternative to using a general-purpose integer solver. We implemented three CG-based matheuristics to try to derive feasible and good-quality mappings from CG information.
Since the problem is new, no dataset of instances was available for testing the algorithms. We proposed a random generator of topologies and resource requirements of networks and applications. We provided a dataset of problem instances, generated with the tool just mentioned.
As experiments, we examined the performance of GuRoBi solver on the compact formulation of the instances. We compared the quality of CG bounds with respect to the root node relaxation bounds, noting that CG ones are remarkably tighter. We observed that solving the pricing problems heuristically is in general beneficial. In the worst cases, however, the computing times slightly increase instead of decreasing. Developing faster exact algorithms or better heuristics for solving the pricing problem could be a spin-off of this work. The three matheuristics had an increasing level of performance. The most advanced one, based on rounding variables of the compact model and subMIPing, produced solutions of high quality, at the expense of longer computing times. Future work could include online resolution methods for the problem, possibly exploring data-driven paradigms such as Reinforcement Learning.

Appendix A

Instance files examples

A.1 Network files

Network topology

The following file is an example of network topology file:

```
5
0,1 0,4 1,2 1,4 2,3 2,4
50 100 100 175 10
0 0 : 0
0 1 : 0 1
0 2 : 0 1 2
0 3 : 0 1 2 3
0 4 : 0 4
1 0 : 1 0
1 1 : 1
1 2 : 1 2
1 3 : 1 2 3
1 4 : 1 4
2 0 : 2 4 0
# ... omitted lines ...
2 4 : 2 4
3 0 : 3 2 4 0
# ... omitted lines ...
3 4 : 3 2 4
4 0 : 4 0
# ... omitted lines ...
4 4 : 4
```

Network properties and resources

The following file is an example of properties and resources file for the above topology:

```
core
129 249 214 43 1000000000
has_camera
1 0 1 1 0
has_gpu
0 1 0 1 0
bandwidth
0,1 168
0,4 245
1,2 515
1,4 502
2,3 397
2,4 282
latency
0,1 1
0,4 1
1,2 1
1,4 1
2,3 1
2,4 1
```

A.2 Application files

Application topology

The following file is an example of application topology file:

```
5
1,0 2,0 3,1 4,1
```

Application properties and resources

The following file is an example of properties and resources file for the above topology:

```
core
715 172 237 36 47
has_camera
0 0 0 0 1
has_gpu
0 1 1 0 0
bandwidth
1,0 130
2,0 71
3,1 122
4,1 171
latency
1,0 2
2,0 3
3,1 3
4,1 4
```

Bibliography

[1] Namiot Dmitry and Sneps-Sneppe Manfred. On micro-services architecture. *International Journal of Open Information Technologies*, 2(9):24–27, 2014.

[2] Lizhe Wang, Gregor Von Laszewski, Andrew Younge, Xi He, Marcel Kunze, Jie Tao, and Cheng Fu. Cloud computing: a perspective study. *New generation computing*, 28:137–146, 2010.

[3] Gonçalo Carvalho, Bruno Cabral, Vasco Pereira, and Jorge Bernardino. Edge computing: current trends, research challenges and future directions. *Computing*, 103(5):993–1023, 2021.

[4] Alberto Ceselli, Marco Premoli, and Stefano Secci. Mobile edge cloud network design optimization. *IEEE/ACM Transactions on Networking*, 25(3):1818–1831, 2017.

[5] Alberto Ceselli, Marco Fiore, Angelo Furno, Marco Premoli, Stefano Secci, and Razvan Stanica. Prescriptive analytics for mec orchestration. In *2018 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 1–9. IEEE, 2018.

[6] Christian Quadri, Marco Premoli, Alberto Ceselli, Sabrina Gaito, and Gian Paolo Rossi. Optimal assignment plan in sliced backhaul networks. *IEEE Access*, 8:68983–69002, 2020.

[7] Christian Quadri, Alberto Ceselli, and Gian Paolo Rossi. Multi-user edge service orchestration based on deep reinforcement learning. *Computer Communications*, 203:30–47, 2023.

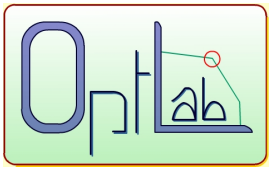
[8] Jacques Desrosiers and Marco E Lübbecke. A primer in column generation. In *Column generation*, pages 1–32. Springer, 2005.

[9] J Pirie Hart and Andrew W Shogan. Semi-greedy heuristics: An empirical study. *Operations Research Letters*, 6(3):107–114, 1987.

[10] Ruslan Sadykov, François Vanderbeck, Artur Pessoa, Issam Tahiri, and Eduardo Uchoa. Primal heuristics for branch and price: The assets of diving methods. *INFORMS Journal on Computing*, 31(2):251–267, 2019.

[11] Bernard M Waxman. Routing of multipoint connections. *IEEE journal on selected areas in communications*, 6(9):1617–1622, 1988.

[12] Paolo Boldi and Sebastiano Vigna. Axioms for centrality. *Internet Mathematics*, 10(3-4):222–262, 2014.



Project developed at the Operations Research Laboratory (OptLab)
http://optlab.di.unimi.it