An improved multi-objective genetic algorithm with heuristic initialization for service placement and load distribution in edge computing

Adyson Maia, Yacine Ghamri-Doudane, Dario Vieira, Miguel Franklin de Castro

Version of Record: https://www.sciencedirect.com/science/article/pii/S1389128621002127
Manuscript_6a3e732d8a967313a97e155bf7b2d221

An Improved Multi-Objective Genetic Algorithm with Heuristic Initialization for Service Placement and Load Distribution in Edge Computing

Adyson M. Maia[a,b], Yacine Ghamri-Doudane[b], Dario Vieira[c], Miguel F. de Castro[a]

[a] *Federal University of Ceará (UFC), Fortaleza, Brazil*
[b] *University of La Rochelle (ULR), La Rochelle, France*
[c] *Engineering School of Information and Digital Technologies (EFREI), Villejuif, France*

Abstract

Edge Computing (EC) is a promising concept to overcome some obstacles of traditional cloud data centers to support Internet of Things (IoT) applications, especially time-sensitive applications. However, EC faces some challenges, including the resource allocation for heterogeneous applications at a network edge composed of distributed and resource-restricted nodes. A relevant issue that needs to be addressed by a resource manager is the service placement problem, which is the decision-making process of determining where to place different services (or applications). A related issue of service placement is how to distribute workloads of an application placed on multiple locations. Hence, we jointly investigate the load distribution and placement of IoT applications to minimize Service Level Agreement (SLA) violations due to the limitations of EC resources and other conflicting objectives. In order to handle the computational complexity of the formulated problem, we propose a multi-objective genetic algorithm with the initial population based on random and heuristic solutions to obtain near-optimal solutions. Evaluation results show that our proposal outperforms other benchmark algorithms in terms of response deadline violation, as well as terms of other conflicting objectives, such as operational cost and service

Email addresses: adysonmaia@great.ufc.br (Adyson M. Maia), yacine.ghamri@univ-lr.fr (Yacine Ghamri-Doudane), dario.vieira@efrei.fr (Dario Vieira), miguel@great.ufc.br (Miguel F. de Castro)

availability.

Keywords: internet of things, edge computing, 5G network, service placement, load distribution, genetic algorithm, multi-objective optimization

1. Introduction

Internet of Things (IoT) is a paradigm driving a digital transformation in our daily lives with the progressing development of computing and wireless technologies [1]. In the IoT paradigm, smart objects, or things, are essential building blocks, which include mobile phones, vehicles, home appliances, sensors, actuators, and any other embedded devices. These devices will be interconnected in order to exchange data related to real and virtual worlds among themselves through modern communication network infrastructures. The continued growth of IoT brings more connected devices to collect and consume data across the network. Due to the resource constraints of those devices, a common approach is to use Cloud Computing (CC) to execute data analysis remotely. Some of the benefits of using CC are (i) the flexible pricing model (pay-as-you-go), (ii) the on-demand and elastic delivery of virtualized resources (e.g., computing, storage, and network resources), and (iii) the scalable computing model [2].

The backbone of CC is based on building a few large data centers in various parts of the world to serve a huge number of users. However, this means that all the data and requests need to be transmitted to a remote centralized data center, which results in long (network) latencies. For some time-sensitive IoT applications (e.g., factory automation, intelligent transport systems, emergency response, interactive mobile gaming, augmented reality, and mission-critical applications) requiring real-time responses at 10ms or even 1ms [3], the delay caused by transferring data to the cloud is unacceptable. Moreover, some data processing can be made locally without having to be transmitted to the cloud. Even when some decisions are made in the cloud, it may be unnecessary and inefficient to send the large volume of data to the cloud for processing and storing because not all data is useful for decision making and analysis [4]. Another

drawback of CC is the lack of fast and direct access to local contextual information (e.g., precise user location, local network conditions, and users mobility behavior) while provisioning resources to an application [5]. Therefore, these challenges caused by the explosive growth of IoT cannot be addressed only by the Cloud Computing model.

Edge Computing (EC) aims to overcome some limitations of CC by bringing cloud services and resources (e.g., computing, storage, and networking resources) closer to end-users on geo-distributed nodes at the network edges [6, 7]. Some examples of such edge nodes are cellular base stations, routers, switches, and wireless access points. Thus, applications (or services[1]) running on edge nodes in the vicinity of their users (customers or IoT devices) can filter, aggregate, or analyze data close to its source. Consequently, Edge Computing can (i) minimize latency and response time; (ii) reduce core and cloud networks traffic; (iii) reduce load on end-user devices; and minimize power consumption of those devices; (iv) make better location- and context-aware decisions; as well as (v) improve security and privacy [5]. The characteristics of EC ensure a wide range of applications and use cases that can benefit from being deployed at the edge, such as healthcare, augmented and virtual reality, multi-player gaming, interactive multimedia, video analytics, smart environments, industrial control systems, vehicular communications, road traffic monitoring [7].

Compared to a traditional cloud infrastructure, Edge Computing has distinguishing characteristics [4, 8]. First, Cloud Computing usually locates its resources in a few centralized data centers, but there will be a dense geo-distribution of edge nodes in EC. Furthermore, the availability of the cloud services depends on the distance of multi-hop between the end-user and the cloud data centers, while edge nodes are one or few hops away from end-users. Edge nodes are more heterogeneous than cloud servers, i.e., edge nodes come with different form factors and resources capabilities. However, edge nodes are generally more resource-limited than cloud data centers.

[1] we use the notion of application and service interchangeably in this paper

Despite the benefits of Edge Computing, it is still a recent research topic and faces several challenges. We are particularly interested in the service placement problem in the context of EC, which is a resource management issue related to the selection of computer-based nodes (or servers) to host multiple applications according to some demands and constraints to optimize a set of performance-related objectives [9, 10]. Therefore, a service placement scheme maps each application onto some hosting node.

Another challenge is provisioning adequate resources to handle the workloads generated by applications users, which relates to the amount of resources that should be allocated to each application that processes an incoming workload. A simple solution to this problem is using resource over-provisioning to handle high peak loads. However, over-provisioning is unsuitable for EC due to high costs and limited resource capacity at edge nodes [11, 7]. Meanwhile, a service placement solution mappings can be many-to-many, i.e., an application can be placed onto one or more nodes, and a node can host more than one application. In this way, service placement may consider the load balancing to distribute workloads of an application across multiple nodes. Load distribution can be applied to improve resource usage, increase availability, reduce response time, avoid bottlenecks and over-provisioning [12].

Although the service placement and load distribution can be separately optimized in independent procedures, the placement decision may affect the load distribution, and vice versa [13]. Hence, an optimal decision-making process may require a complex joint optimization of those two procedures.

We can find several research works in the literature concerning service placement in Cloud Computing (e.g., [14, 15, 10]). However, these works cannot be directly applied to EC because they do not consider the distinct characteristics of EC. On the other hand, there are some research works on service placement in the context of Edge Computing, such as Tärneberg et al. [9], Gu et al. [16], Skarlat et al. [11], Zhao and Liu [17]. However, these existing works have shortcomings, such as not considering the limited resource capabilities of edge nodes, requirements of time-sensitive applications, or multiple conflicted opti-

mization objectives.

Motivated by the above facts, we jointly address the service placement and load distribution problem in Edge Computing. The formulated problem considers infrastructure constraints and different application characteristics (response deadline, resource demand, scalability, and availability) to minimize not only Service Level Agreement (SLA) violations but also other possibly conflicting objectives (e.g., operational cost and unavailability). Furthermore, we extend our previous Genetic Algorithm (GA) [18, 19] to improve the solutions obtained for the formulated problem. More specifically, the main contributions of this paper are as follows:

- We present a system model where multiple replicas of an application can be placed in different parts of the network to distribute requests (load) among these replicas. Moreover, we assume a static EC scenario where all information required in the service placement and load distribution decision-making process is provided in advance and does not change over time.

- We jointly formulate the service placement and load distribution as an Mixed-Integer Nonlinear Programming (MINLP) optimization problem to minimize the potential occurrence of SLA violations. Thus, a solution for this problem specifies the service placement mapping and the distribution of requests among different nodes. We also transform this MINLP into a Mixed-Integer Linear Programming (MILP) problem to be solved using a linear solver.

- The single-objective problem is extended to include multiple conflicted objectives but still prioritizing time-sensitive applications.

- We propose a meta-heuristic based on GA to solve the multi-objective problem. The proposal is an improvement of our previous works [18, 19] by including heuristic solutions in the initialization of the GA and the MGBM stopping criteria [20]. This improvement aims to speed up the

algorithm and obtain better solutions.

The remainder of the paper is organized as follows. Section 2 reviews the related work. Next, Section 3 presents our system model. We formally formulate the service placement and load distribution problem in Section 4. In Section 5, we describe our proposed GA to solve the formulated optimization problem. Then, Section 6 analyses the performance of our proposal. Finally, Section 7 concludes this paper.

2. Related Work

Virtual Machine (VM) placement is a well-studied topic in Cloud Computing (CC). Pires and Barán [14], Pietri and Sakellariou [15], de Carvalho et al. [21], and Filho et al. [10] published surveys on this topic over the past few years. However, service placement approaches to conventional CC cannot be directly applied to Edge Computing (EC) because they do not consider the distinct characteristics of EC (e.g., a large, distributed, and heterogeneous environment) nor the time-sensitive application requirements.

Computation offloading refers to the transfer of tasks from a device to an external platform, such as the edge and cloud computing. Hence, it enables running intensive computational applications at a device with constrained resources while reducing its energy consumption. Moreover, a crucial part of offloading is deciding whether to offload or not. In [22], the authors present a survey concerning computation offloading in the context of Multi-access Edge Computing (MEC). In this paper, we are not interested in this offloading decision process, but service placement and computation offloading can be seen as complementary problems. Furthermore, we only focus on the static aspect of the service placement problem, where the mapping of applications onto hosting nodes does not change for a long time, and the decision parameters are known in advance.

We can find some approaches in the literature to solve the service placement problem in the context of Edge Computing. Skarlat et al. [11] examine the

service placement in a hierarchical and distributed edge architecture to maximize the number of applications placed on edge nodes rather than the cloud. The work also prioritizes time-sensitive applications to satisfy their response time deadline requirements. Moreover, a control node uses a Genetic Algorithm (GA) to obtain the application placement decision among its child nodes in the hierarchical network infrastructure. However, it does not address the load distribution, and the proposed GA can generate infeasible solutions, which may degrade the algorithm performance.

In [9], the authors discuss the placement of applications in edge nodes to minimize the overall running cost. That is, the work aims to reduce the edge resource consumption due to the assumption that the running cost is proportional to resource usage. The authors present an iterative local search heuristic among neighboring solutions to find a near-optimal solution. A limitation of this work is the assumption of having sufficient resources for all applications at the network edge. Moreover, the work does not consider the requirements of time-sensitive applications.

Zhao and Liu [17] address service placement and load distribution problems in MEC while targeting to minimize the average response time. The authors propose a heuristic strategy that tries to select, for each application, hosting nodes with low average response time among all requests to host the application and receive its requests. Nevertheless, the proposed solution does not take into consideration the response time deadline requirement that is particularly important for latency-sensitive applications.

In [16], the authors also investigate both service placement and load distribution problems in MEC. Gu et al. [16] formulate the optimization problem to minimize an overall cost while satisfying the maximum tolerable delay of time-sensitive medical applications. However, the authors only examine the application deployment in base stations of a mobile network, ignoring other possible locations such as the core network and cloud data centers. In addition, the work assumes that there are sufficient resources in the base stations to deploy all applications while satisfying the delay requirement.

Yang et al. [23] present a study for joint optimization of service placement and load distribution problems in a static scenario. The authors design a two-step heuristic to minimize the average latency of all application requests while satisfying the nodes capacity constraints. First, the heuristic relaxes the problem by disregarding the node capacity constraint, and it solves the problem using a linear programming solver. Then, a greedy strategy tries to obtain a feasible solution for the original problem from the optimal solution of the relaxed problem. A drawback of this work is the assumption that applications are homogeneous, as all requests consume the same amount of resources.

In [24], the authors investigate VM scheduling and placement decision in MEC to (i) maximize infrastructure provider revenue, (ii) minimize SLA violations, and (iii) ensure fairness in resource allocation among service providers. Even though the work investigates SLA violation in terms of response time, it considers the processing time responsible only for the response delay, neglecting the network delay. Furthermore, the work applies a weighted sum scalarization to transform the multi-objective problem into a single-objective one. Unfortunately, setting weights for each objective is not an easy task in a practical scenario for a decision-maker.

According to Spinnewyn et al. [25], an EC environment is more susceptible to unpredictable failures than the cloud. As a result, this characteristic significantly affects the reliability of applications deployed in an EC environment. Therefore, the authors investigate the placement of multi-component applications to optimize multi-objectives and satisfy the application minimum availability requirement. This work also applies a scalarization to transform a multi-objective optimization into a single-objective problem. More specifically, it sequentially optimizes the problem in multiple steps. In each step, an objective function is optimized, and the results of previous steps are added as equality constraints. A shortcoming of this work is that it does not consider the requirements of time-sensitive applications. Furthermore, the applied scalarization transformation also requires the decision-maker to have a global preference order among all objectives.

Table 1 presents a comparison of the discussed related work. We can observe that some works address both the service placement and load distribution problems, but not all works adequately address the Quality of Service (QoS) requirements, especially the response deadline requirement for time-sensitive applications. Regarding the optimization object, most research papers have a single objective. Moreover, many multi-objective works transform the problem into a single-objective problem by some scalarization method that requires a global preference order among all optimization objectives. However, this preference order can be hard to adjust or, in many cases, does not even exist in a practical scenario.

Table 1: Related work comparison

Work	**Problem**	**Objective**	**Requirement**	**Solution**
Skarlat et al. [11]	SP	SO	Resource-Based, QoS-Based	MH
Tärneberg et al. [9]	SP	SO	Resource-Based	H
Zhao and Liu [17]	SP, LD	SO	Resource-Based	H
Gu et al. [16]	SP, LD	SO	Resource-Based, QoS-Based	H
Yang et al. [23]	SP, LD	SO	Resource-Based	H
Katsalis et al. [24]	SP, LD	MO2SO	Resource-Based, QoS-Based	D
Spinnewyn et al. [25]	SP	MO2SO	Resource-Based, QoS-Based	MH, H
Our proposal	SP, LD	SO, MO	Resource-Based, QoS-Based	MH

SP: Service Placement, LD: Load Distribution,
SO: Single Objective, MO: Multi-Objective, MO2SO: MO transformed into SO,
MH: Meta-Heuristic, H: Heuristic, D: Deterministic

In order to overcome some limitations of the above-discussed works, we jointly investigate the static service placement and load distribution problems in a distributed, heterogeneous, and resources limited EC environment. In this

paper, we present a single and multi-objective optimization formulations that consider applications, especially for IoT, with different QoS requirements. Moreover, we improve the performance of our previous solutions [18, 19] by including heuristics in the initialization of the proposed meta-heuristic based on GA.

3. System Model

In our system model, an Infrastructure Provider (InP) owns and maintains the network infrastructure comprised of hosting nodes and links connecting these nodes. A hosting node is a general-purpose machine having diverse resource capabilities (e.g., processing, memory, storage, and networking resources) and supporting execution of applications through virtualization technologies, such as VM or container. Moreover, hosting nodes can be located anywhere on the network between end-users and cloud data centers. Some nodes also act as access points for end-user devices.

Application Service Providers (ASPs) create applications used by end-users and rent on-demand resources from an InP to deploy their applications. End-user devices are connected directly with some node over a wired or wireless link. Over time, devices send requests, or tasks, to be processed by an application. A device request is routed among the nodes up to a node hosting the required application. Lastly, the request is processed, and its response is sent back to the device.

ASPs do not directly determine where to place their applications. Usually, an ASP signs an SLA defining the QoS requirements to be fulfilled by the InP. In this way, a service placement scheme is an autonomous decision-making process maintained by the InP that decides where to deploy different applications over the network infrastructure according to some constraints, requirements, and performance goals defined by ASPs and InP.

In order to illustrate the system model, we describe the use case of a cellular network with Edge Computing capabilities (e.g., 5G network), as shown in Figure 1. In this case, applications can be hosted on nodes located on the

Radio Access Network (RAN), Core Network, and Cloud Computing regions. If an application is running on a Base Station (BS) in the RAN region, then a request can only be routed among neighboring Base Stations (BSs) to reduce traffic at the core and to decrease transmission delays. However, not all applications can be deployed to BSs because of the limited computing resources in this region. Hence, some applications are hosted in the core or the cloud while carrying about not violating some placement criteria defined by the providers.

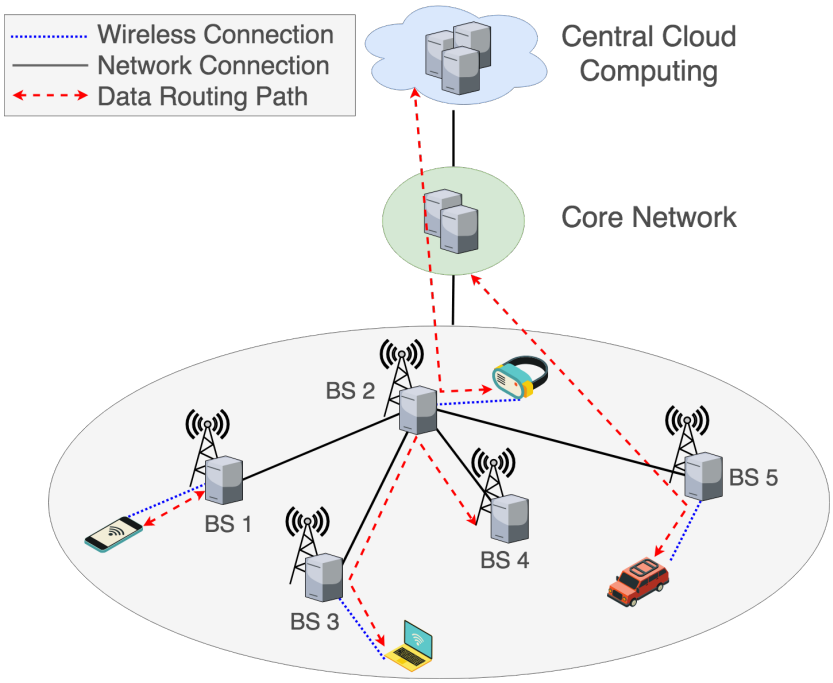


Figure 1: Proposed Edge Computing system for 5G networks.

3.1. Network Model

We model the network as an undirected and connected graph $G = (\mathcal{V}, \mathcal{E})$, where the vertices $\mathcal{V}$ are hosting nodes and the edges $\mathcal{E}$ are network links between the nodes. We assume all vertices are accessible by any other vertex in the graph through multiple hops. In addition, end-user devices and their connections are not represented in G.

Each network link $e = (u, v) \in \mathcal{E}$ has the following property:

- **Transmission Delay** $D_{net}^{a,e}$ is the average amount of time it takes for a request for an application a to be transmitted in the network link e.

3.2. Resource Type Model

The proposed system model allows specifying different types of resources, where $\mathcal{R}$ is the set of these considered resources. For instance, the set $\mathcal{R} = \{CPU, RAM, DISK\}$ is made up of processing (CPU), Random-Access Memory (RAM) and disk storage resources (DISK). RAM and disk storage are measured in bytes, while CPU can be measured in Instructions Per Second (IPS).

3.3. Node Model

A node, or vertex, in the graph G, represents a server with specific resource capabilities to run applications. Although multiple servers may be located on a single network node, we view these multiple servers as a single unit. Therefore, there is precisely one server for each network node, and we use the terms node and server interchangeably throughout this paper. In this way, we can include cloud data centers as a single cloud node in the graph.

Each node $v \in \mathcal{V}$ has the following features:

- **Resource Capacity** C_v^r is a number describing the total capacity of resource $r \in \mathcal{R}$ on node v.
- **Usage Cost** $g_v(\lambda)$ is a function specifying the (monetary) cost of allocating resources for an application with a workload $\lambda \geq 0$ on the node. We define the workload λ_a^v as the (average) arrival rate of requests for application a in node v.
- **Availability** P_v is the probability that the node v will not fail.

We assume that the cloud node has an unlimited capacity for all resources (i.e., $C_{cloud}^r = \infty, \forall r \in \mathcal{R}$) due to the capacity difference between cloud and edge nodes.

3.4. Application Model

Let $\mathcal{A}$ be the set of all different applications to be placed over the network. We consider that one or more instances, or replicas, of these applications can be deployed within the system, but these instances are independent of each other. Furthermore, a node can only host one instance of each application.

An application $a \in \mathcal{A}$ has the following parameters:

- **Response Deadline** D_a is a number specifying the maximum time (i.e., deadline) allowed for responding a request for application a. The response time comprises the network delay plus the processing delay.
- **Maximum Number of Instances** N_a describes how application a scales horizontally. For instance, it can be set to $N_a = 1$, if the service does not allow replicas or $N_a = \infty$ if the maximum number of instances is undefined.
- **Resource Demand** $h_a^r(\lambda)$ denotes how an application scales vertically. In other words, it is a non-decreasing function specifying the (average) amount of resources $r \in \mathcal{R}$ required by a replica of a with a workload λ. For instance, we can define a constant function if the vertical scaling is not supported or an increasing linear function if the amount of resources required is proportional to the workload.
- **CPU Work Size** W_a is a value indicating the average amount of processing required to get a response to a request for a. It is measured by the number of instructions or clock cycles required to process a request.
- **Availability** P_a denotes the probability that an application replica is working without internal failure.

3.5. User Model

End-user devices, or users, are not aware of where applications are deployed and which application instance will handle their requests. Therefore, we can distribute these requests among multiple replicas placed on the system.

Let $\mathcal{U}$ bet the set of all users in the system, then a user $u \in \mathcal{U}$ has the following properties:

- **Requested Application** u_a. It specifies the application $a \in \mathcal{A}$ requested by the user. For the sake of simplicity, we assume each user sends requests for only one application.

- **Request Rate** λ_a is the average request generation rate for an end-user device of application $a \in \mathcal{A}$. We assume that all users of a send requests to this application at the same average rate, which is determined by a Poisson distribution.

- **Attached Node** u_v. A user is always connected to some node $v \in \mathcal{V}$, which acts as an access point.

Then, we define $\mathcal{U}_a^v$ as the set of users connected to node v requesting application a, and $\mathcal{U}_a$ is the set of all users requesting application a in the system.

Table 2 summarizes the notations used in the system model.

4. Problem Statement and Formulation

In a practical scenario, it is not possible to place all applications on the edge of the network given the resource limitations of hosting nodes in this region. Consequently, some applications are deployed further (i.e., in the core network or the cloud) from their end-users. This considerable distance between node and user may result in the response time of a request to exceed the deadline specified by some applications. Moreover, an overloaded server also increases response time, thus distributing the load among application replicas may mitigate this issue. Hence, both service placement and load distribution decisions may result in violations of the response deadline requirement, which is an important metric to be minimized for time-sensitive applications in an Edge Computings environment.

Infrastructure Provider (InP) and Application Service Providers (ASPs) often have many other performance metrics to optimize instead of just a single one.

Table 2: Notations used in the proposed system model

Symbol	**Description**
System Model	
$\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{A}, \mathcal{U}$	Set of network nodes, network links, resource types, applications, and users, respectively
$D_{net}^{a,e}$	Network delay for application (app) a in a link e
C_v^r	Total capacity of resource r on node v
$g_v(\lambda)$	Resource allocation cost on node v for an app with workload λ
P_v	Availability probability of node v
D_a	Maximum tolerable response time of app a
N_a	Maximum number of replicas for app a
$h_a^r(\lambda)$	Demand of resource r for a replica of app a with workload λ
W_a	CPU work size of a request for app a
λ_a	Request generation rate for users of app a
P_a	Availability probability of app a
$\mathcal{U}_a^v$	Set of users connected to node v requesting application a
$\mathcal{U}_a$	Set of all users requesting application a in the system

However, those multiple metrics are, in general, contradicting each other. For instance, an ASP wants to decrease response time while reducing the monetary cost of allocating resources to its applications. On the other hand, decreasing costs implies using cheaper cloud resources and, thus, increasing response time. An InP may also seek to reduce the infrastructure running costs by decreasing the number of active nodes hosting applications. However, minimizing the number of servers in use may result in nodes being overloaded and, consequently, SLA violations.

Given the above context, we formulate the joint problem of service placement and load distribution to minimize deadline violation as a single objective, and multiple objectives in this section. In this paper, we are only interested in the static, or offline, approach of the problem where applications and users do not move for a long time. Moreover, all information required in the decision-making

process is provided in advance.

Table 3 summarizes the notations used in this section, and the remaining of this section is organized as follows. First, Subsection 4.1 presents an estimation of the response time of a request. Then, Subsection 4.2 specifies the variables and constraints of the problem. Subsection 4.3 presents a nonlinear and linear formulation of the single-objective case. Finally, Subsection 4.4 formulates the multi-objective case.

Table 3: Notations used in the problem formulation

Symbol	**Description**
Problem Formulation	
$\mathcal{F}_a^{u,v}$	Set of requests from users attached to node u to an instance of app a hosted on node v
Q_a^u	Number of requests for app a generated from users attached to node u
Q_a	Total number of requests for app a in the system
$d_a^{u,v}$	Average response time of a request flow $\mathcal{F}_a^{u,v}$
$d_{net}^{a,u,v}$	Average network delay of a request flow $\mathcal{F}_a^{u,v}$
$d_{proc}^{a,u,v}$	Average processing delay of a request flow $\mathcal{F}_a^{u,v}$
λ_a^v	Request arrival rate of app a on node v
μ_a^v	Service rate of app a on node v
Problem Variables	
$x = (\rho, \gamma, \delta)$	Decision variables
ρ_a^v	Whether node v deploys an instance of app a or not
$\gamma_a^{u,v}$	Whether request flow $\mathcal{F}_a^{u,v}$ exists or not
$\delta_a^{u,v}$	Number of requests in the flow $\mathcal{F}_a^{u,v}$
Optimization Objectives	
F	List of objective functions, $F = (f_1, \ldots, f_m)$
$f_{dv}(x)$	Deadline violation of the system given variable x
$f_{cost}(x)$	Overall operational cost given variable x
$f_{fail}(x)$	Average unavailability of applications given variable x

4.1. Response Time Estimation

Since requests can be distributed among multiple replicas of an application, we define a request flow $\mathcal{F}_a^{u,v}$ as the set of requests for application $a \in \mathcal{A}$ generated by users attached to node $u \in \mathcal{V}$ (source node) and handled by a replica of a placed on node $v \in \mathcal{V}$ (target node). In addition, let $\delta_a^{u,v} = |\mathcal{F}_a^{u,v}|$ be the number of requests in a flow. Thus, Equation 1 specifies the average response time $d_a^{u,v}$ of a request flow $\mathcal{F}_a^{u,v}$, where $d_{net}^{a,u,v}$ is the average time to send requests to a from users in u to node v and $d_{proc}^{a,u,v}$ is the average processing time of requests on v. We estimate both network and processing delays in the remainder of this subsection.

$$d_a^{u,v} = d_{net}^{a,u,v} + d_{proc}^{a,u,v} \tag{1}$$

4.1.1. Network Delay

The network delay of a request includes: (i) the communication delay between the requesting end-user device and the node to which it is attached, and (ii) the transmission delay from this latter node to a server hosting the application following a multi-hop routing path. It is important to note that a user attachment node can host the application and process its requests, and therefore the transmission delay of the second part is zero. Since we are examining the static problem case, the communication delay between a device and its attachment node does not affect the placement decision [17]. Therefore, we do not consider this communication delay in the network delay estimation.

We estimate the average network delay of a request flow $\mathcal{F}_a^{u,v}$ as:

$$d_{net}^{a,u,v} = \begin{cases} 0 & \text{if } u = v \\ \sum_{e \in \mathcal{P}_{u \to v}} D_{net}^{a,e} & \text{otherwise} \end{cases} \tag{2}$$

where $\mathcal{P}_{u \to v}$ is the set of links in a routing path from u to v. This set can be determined by some shortest routing path algorithm, such as the Floyd–Warshall algorithm [26, 27]. Therefore, we assume that $d_{net}^{a,u,v}$ is a constant in the static problem.

4.1.2. Processing Delay

For an application replica placed on a node, we model its processing time as an M/M/1 queueing model. This queueing model helps us to estimate the average processing time in different workload conditions. Moreover, we utilize the M/M/1 queue because the aggregated traffic generated by users of an IoT application can be approximated as a Poisson process [28].

We determine the request arrival rate or workload λ_a^v for the application a running on node v as the sum of all requests arriving at this node. Equation 3 expresses this request arrival rate, where $\delta_a^{u,v} \in [0, Q_a^u]$ is an integer variable indicating the size of request flow $\mathcal{F}_a^{u,v}$ (i.e., number of requests in the flow). Moreover, $Q_a^u = \lceil |\mathcal{U}_a^u| \lambda_a \rceil$ is the number of requests for application a generated by users attached to node u, λ_a is the request rate of a user of a, and $|\mathcal{U}_a^u|$ is the cardinality of set $\mathcal{U}_a^u$.

$$\lambda_a^v = \sum_{u \in \mathcal{V}} \delta_a^{u,v} \tag{3}$$

Service times have an exponential distribution with rate parameter μ, where $1/\mu$ is the mean service time in an M/M/1 queue. Thus, we express $1/\mu_a^v$ as the average time to perform the request CPU work W_a with the resources allocated for a replica of application a in node v as:

$$\frac{1}{\mu_a^v} = \frac{W_a}{h_a^{CPU}(\lambda_a^v)} \tag{4}$$

Finally, Equation 5 gives the average processing time of requests for application a running on node v according to Little's law.

$$d_{proc}^{a,u,v} = \frac{1}{\mu_a^v - \lambda_a^v} \tag{5}$$

4.2. Problem Variables and Constraints

In order to jointly formulate the service placement and load distribution problems, we define $x = (\rho, \gamma, \delta)$ as a triple of problem variables. We describe these variables as follows:

1. **Application Placement** $\rho = \{\rho_a^v \mid a \in \mathcal{A} \text{ and } v \in \mathcal{V}\}$ is a set of binary variables, where $\rho_a^v \in \{0, 1\}$ indicates whether a replica of an application a is placed on a node v or not.
2. **Request Flow Existence** $\gamma = \{\gamma_a^{u,v} \mid a \in \mathcal{A} \text{ and } u, v \in \mathcal{V}\}$ is the second set of binary variables, where $\gamma_a^{u,v} \in \{0, 1\}$ specifies whether or not a request flow $\mathcal{F}_a^{u,v}$ exists between nodes u and v for an application a.
3. **Load Distribution** $\delta = \{\delta_a^{u,v} \mid a \in \mathcal{A} \text{ and } u, v \in \mathcal{V}\}$ is a set of variables related to how requests are distributed, where $\delta_a^{u,v} \in \mathbb{Z}^+$ is the size of request flow $\mathcal{F}_a^{u,v}$.

A solution to the studied problem sets values for the above variables. Furthermore, a solution is feasible only if all the following constraints are met:

1. **Number of Instances.** A node can only host a single replica of a given application. Moreover, the number of instances deployed in the system must respect the limits defined by the applications (i.e., N_a), and all of them need to be placed.

$$1 \leq \sum_{v \in \mathcal{V}} \rho_a^v \leq N_a \quad \forall a \in \mathcal{A} \tag{6}$$

2. **Request Flow Existence.** A request flow $\mathcal{F}_a^{u,v}$ only exists if a replica of application a is placed on node v and there are users connected in u requesting a.

$$\gamma_a^{u,v} \leq \rho_a^v Q_a^u \quad \forall a \in \mathcal{A}, \forall u, v \in \mathcal{V} \tag{7}$$

3. **Request Flow Size.** If a flow $\mathcal{F}_a^{u,v}$ exists, its size must be at least one and at most equal to the number of requests generated by users connected to node u.

$$\gamma_a^{u,v} \leq \delta_a^{u,v} \leq \gamma_a^{u,v} Q_a^u \quad \forall a \in \mathcal{A}, \forall u, v \in \mathcal{V} \tag{8}$$

4. **Load Conservation.** The aggregate size of all request flows for application a from the same source node u is equal to the total number of requests for a generated by users connected to this node. In other words,

all requests are distributed to some node.

$$\sum_{v \in \mathcal{V}} \delta_a^{u,v} = Q_a^u \quad \forall a \in \mathcal{A}, \forall u \in \mathcal{V} \tag{9}$$

5. **Node Capacity.** The total amount of resources demanded by applications placed on a server should not exceed its capacity.

$$\sum_{a \in \mathcal{A}} \rho_a^v h_a^r(\lambda_a^v) \leq C_v^r \quad \forall r \in \mathcal{R}, \forall v \in \mathcal{V} \tag{10}$$

6. **Queue Stability.** An M/M/1 queue is stable only if the average service rate is larger than its average arrival rate. This stability needs to be guaranteed for each application placed on a node.

$$\lambda_a^v < \mu_a^v \quad \forall a, v \left(\rho_a^v = 1\right), a \in \mathcal{A}, v \in \mathcal{V} \tag{11}$$

4.3. Single Objective Formulation

Our goal is to minimize the deadline violation of the system, which we define as the highest violation among all request flows in the system. Then, we specify the deadline violation of a request flow $\mathcal{F}_a^{u,v}$ as the positive part of the difference between its average response time and the application response deadline, i.e., $\max\left(0, d_a^{u,v} - D_a\right)$. Equation 12 expresses this objective function, where $[z]^+ = \max(0, z)$ is the positive part of a real number z.

$$\begin{aligned} f_{dv}(x) &= \max_x \{[d_a^{u,v} - D_a]^+\} \\ x &= (\rho, \gamma, \delta) \end{aligned} \tag{12}$$

Then, the static single objective optimization problem is formulated as:

$$\begin{aligned} &\min f_{dv}(x) \\ &x = (\rho, \gamma, \delta) \\ &\text{subject to eqs. (6) to (11)} \end{aligned} \tag{13}$$

The optimization problem (13) is a Mixed-Integer Nonlinear Programming (MINLP) problem because constraints (10) and (11) and the objective function (12) are nonlinear. MINLP is usually difficult to solve due to its high

computational complexity [29]. One way to reduce this complexity is to apply linearization and relaxation techniques. In Appendix A, we describe how to transform problem (13) into a Mixed-Integer Linear Programming (MILP) problem.

4.4. Multiple Objective Formulation

Based on the single objective optimization problem (13) and given $F = (f_1, f_2, \ldots, f_m)$ as a list of m performance-related functions, the multi-objective optimization problem for the static approach is formulated as:

$$\begin{aligned} \min F(x) &= (f_1(x), f_2(x), \ldots, f_m(x)) \\ x &= (\rho, \gamma, \delta) \\ \text{subject to eqs. (6) to (11)} & \end{aligned} \tag{14}$$

According to the InP or ASPs wishes, different objectives can be optimized. Some non-exhaustive performance-related functions are listed below:

- **Deadline Violation** f_{dv}. The violation level of the system defined in the single objective case by Equation 13 is a relevant metric to minimize for latency-sensitive applications.

- **Operational Cost** f_{cost}. Deploying applications on the system is not a free operation, for there is a cost charged, possibly monetary, to ASPs for the resources used by their applications according to the pay-as-you-go pricing model. For this reason, a provider aims to reduce the cost of running a product. Given $g_v(\cdot)$ as the allocation cost function on a node v described in Section 3, then the total operational cost for an application is simply the sum of costs on each node hosting an application replica, i.e., $\sum_{v \in \mathcal{V}} \rho_a^v g_v(\lambda_a^v)$. Considering all applications, a performance metric to minimize is the overall operational cost, which is defined as:

$$\begin{aligned} f_{cost}(x) &= \sum_{a \in \mathcal{A}} \sum_{v \in \mathcal{V}} \rho_a^v g_v(\lambda_a^v) \\ x &= (\rho, \gamma, \delta) \end{aligned} \tag{15}$$

- **Unavailability** f_{fail}. ASPs also want high availability for their applications. An application becomes unavailable when all of its replicas become unavailable. A replica is unavailable when there is a failure in the node hosting it, or an internal failure occurs in its software. In other words, a replica is available if there is no hardware and software failure. We formulate this unavailability of an application as $\prod_{v \in \mathcal{V}} (1 - \rho_a^v P_v P_a)$, where P_v and P_a are the availability probability of node v and application a respectively. In this way, maximizing the average availability or minimizing the average unavailability across all applications is a metric to be optimized, which we formally specify as follows:

$$\begin{aligned} f_{fail}(x) &= \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \prod_{v \in \mathcal{V}} (1 - \rho_a^v P_v P_a) \\ x &= (\rho, \gamma, \delta) \end{aligned} \tag{16}$$

Unlike single objective optimization problems that may have a unique optimal solution, in multi-objective optimization problems, conflicts among objectives usually prevent from having a single optimal solution that can optimize all objectives simultaneously. In this way, improvement of one objective may lead to deterioration of another. For instance, reducing unavailability or increasing availability of applications means raising costs by placing more replicas. On the other hand, decreasing costs implies using more cloud resources as they are generally cheaper than edge resources, and thus increasing the deadline violation. Therefore, it is necessary to search for a set of best optimal compromise solutions by considering trade-offs among the conflicting objectives.

The concept of Pareto dominance [30] plays a vital role to find a set of best trade-off solutions that cannot be improved in any of the objectives without degrading at least one of the other objectives. Formally, a feasible solution x_1 Pareto dominates another solution x_2, expressed as $x_1 \prec x_2$, only if

$$\begin{aligned} f_i(x_1) \leq f_i(x_2) \qquad & \forall i \in \{1, 2, \ldots, m\} \\ \text{and } f_j(x_1) < f_j(x_2) \qquad & \exists j \in \{1, 2, \ldots, m\} \end{aligned} \tag{17}$$

The set of all solutions that are not dominated by any other solution is called

Pareto optimal set or Pareto front. These non-dominated solutions are considered equally good if there is no additional preference information. However, for the optimization problem (14), it is important to improve the performance of time-sensitive applications, which can be considered a preference information. Moreover, we assume that a decision maker either does not have a preference order for other objectives or has difficulties in obtaining it. Motivated by this fact, we propose the following modification of Pareto dominance.

Preferred Dominance. Let f_1 be the highest priority function to be optimized among the list of objective functions $F = (f_1, f_2, \ldots, f_m)$. A feasible solution x_1 dominates another solution x_2, expressed as $x_1 \prec^1 x_2$, only if

$$\begin{aligned} &|f_1(x_1) - f_1(x_2)| \leq \xi \text{ and } x_1 \prec x_2 \text{ for } (f_2, \ldots f_m) \\ \text{or if } &f_1(x_1) < f_1(x_2) - \xi \end{aligned} \tag{18}$$

where ξ is a constant approximation error. In other words, the dominance operator $\prec^1$ prioritizes a selected function f_1, and then it is sufficient that a solution x_1 has a smaller value than another solution x_2 in f_1 in order for x_1 dominates x_2. Otherwise, if $f_1(x_1)$ and $f_1(x_2)$ have close values, then the Pareto dominance operator, defined in Equation 17, with the remaining objective functions is used instead. Therefore, we can select a performance-related function to time-sensitive applications as a priority goal. For instance, we can set the deadline violation in Equation 12 as the primary objective to be optimized, i.e., $f_1 = f_{dv}$.

5. Genetic Algorithm Proposal

Although well-known linear solvers, such as IBM ILOG CPLEX[2], can solve MILP problems, these problems are generally NP-Hard [17]. Moreover, the linear version of Problem (13) in Appendix A is highly time-consuming due to a large number of variables. Another limitation of these solvers is that they

[2] `https://www.ibm.com/products/ilog-cplex-optimization-studio`

only address single-objective optimization problems. A way to overcome this limitation is transforming a multi-objective optimization into a single-objective problem through scalarizing methods, such as weighted sum. However, these methods assume that there is a global preference order among all objectives to be optimized, which may be hard to define in practical cases. Furthermore, in many multi-objective optimizations, it is difficult to obtain the exact Pareto optimal set. Therefore, we propose a meta-heuristic based on Genetic Algorithm (GA) to obtain near-optimal solutions. We utilize a genetic approach because this meta-heuristic is not limited to linear, unconstrained, or single-objective problems. In other words, a GA can solve a nonlinear, constrained, and multi-objective problem, such as Problem (13). Moreover, a GA can be implemented in a parallel environment [31].

GA is a method for solving optimization problems inspired by the process of natural selection. In a GA, a population of candidate solutions, called individuals, to an optimization problem is evolved toward an optimal solution [32, 33]. Each individual has a corresponding chromosome that encodes the solution. For instance, a traditional chromosome is represented by a string, or vector, of binary values, but other encodings are also possible. Moreover, a chromosome is associated with a fitness level, which is correlated to the objective function value of the solution it encodes. GA is an iterative process where at each step, called generation, it creates a new population by recombining or randomly mutating chromosome elements of selected individuals of the current population to produce offspring that make up the next generation. Individuals are selected stochastically, but those with better fitness are preferred over those that are less fit. Usually, GA terminates when either produces a maximum number of generations or reaches a satisfactory fitness level.

GA is typically applied for unconstrained optimization problems. A common way of incorporating constraints into a GA is through penalty functions that add a certain value to the objective function based on the amount of constraint violation present in a specific solution. Nonetheless, it may be extremely difficult to estimate good penalty factors or even generate a single feasible solution for

some complex optimization problems [34].

An alternative method to handle constrained-optimization problems in GAs is to develop (i) special solution representations to simplify the shape of the search space, and (ii) special operators to preserve the feasibility of solutions at all times. Gonçalves and Resende [35] propose a Biased Random-Key Genetic Algorithm (BRKGA) where chromosomes are represented as a vector of randomly generated real numbers in the interval $[0, 1]$. A deterministic algorithm, named decoder, takes any chromosome as input and associates it with a feasible solution of an optimization problem, for which an objective value or fitness can be computed. In other words, a chromosome gives instructions on how to build a feasible solution for a particular problem.

Although BRKGA can handle constrained problems, it is generally applied for single-objective optimizations. On the other hand, GAs are suited for multi-objective problems due to the simultaneous evaluation of many candidate solutions, but it is necessary to incorporate the idea of Pareto optimality during the better fit selection process. Hence, we extend BRKGA by including the population classification strategy of the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [36]. NSGA-II is a well-known multi-objective genetic algorithm that uses dominance and diversity preservation mechanisms to rank individuals of a population.

Figure 2 shows the flowchart of our genetic algorithm that combines BRKGA and NSGA-II. In the next subsections, we detail the main evolutionary operations of the proposed genetic algorithm.

5.1. Chromosome representation

The chromosome representation proposed and the description of its parts are given below:

$$\begin{aligned} C = \Big[& I_1, I_2, \ldots, I_{|\mathcal{A}|}, \\ & S_1^1, S_1^2, \ldots, S_1^{|\mathcal{V}|}, \ldots, S_{|\mathcal{A}|}^1, S_{|\mathcal{A}|}^2, \ldots, S_{|\mathcal{A}|}^{|\mathcal{V}|}, \\ & E_1, E_2, \ldots, E_Q \Big] \end{aligned}$$

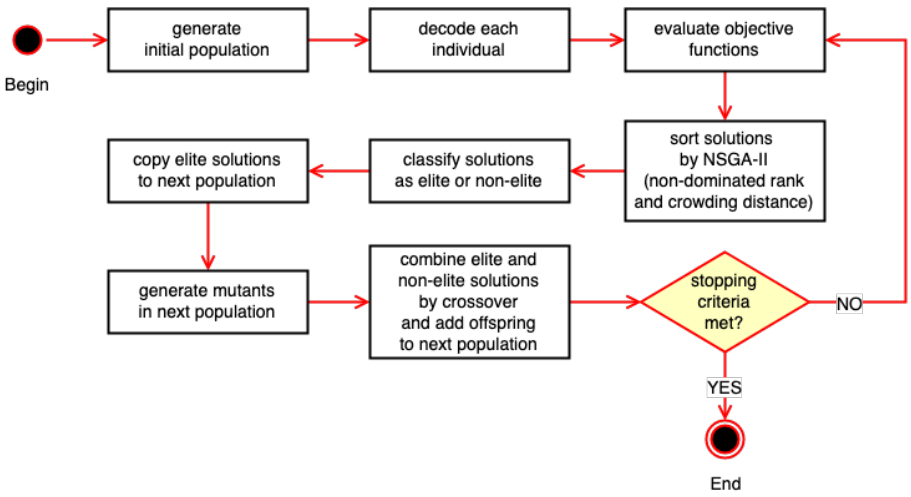


Figure 2: Proposed genetic algorithm flowchart.

1. I_a defines the number of replicas for an application a.
2. S_a^v specifies the priority to place a replica of application a in node v.
3. E_q is related to the order of a request q assigned to a replica, where $q \in \{1, 2, \ldots, Q\}$ and $Q = \sum Q_a^u$ is the total number of requests.

Decoder Algorithm. The proposed Algorithm 1 decodes the above chromosome representation into a feasible solution. Its basic idea is to first select the potential placement locations of each application (lines 4 to 7). For this, it takes the first part of the chromosome (I_a) to delimit the maximum number of possible nodes hosting an application replica on line 5. Then, nodes with high values in the second part of the chromosome (S_a^v) are chosen as potential deployment sites for an application. The cloud is also added as a possible location to ensure that there are resources to deploy at least one replica of each application on the network.

The second step of Algorithm 1 is the load/request distribution (lines 8 to 20). It creates a sequence of all requests conforming to the third part of the chromosome (E_q). Following this sequence, requests are assigned one at a time among the nodes selected in the first step of the algorithm. For a request, the decoder looks for a node with short response time and sufficient resources to

receive it. When the first envisioned target node is found, it sets to place a replica of the requested application on this node and assigns the request to this replica. It also updates the number of available resources and the response time of the target node after increasing the workload. Note that only the processing delay may be increased with a growing workload, thus using the estimated response time to select the target node is a way to distribute load among nodes without ignoring the network delay.

Finally, the decoder algorithm verifies that the maximum number of replicas of an application is respected and replaces surplus replicas with the cloud node (lines 21 to 24).

In order to exemplify how the Algorithm 1 works, let us define a simple system model with three nodes (one base station, the core, and the cloud), and one application, as shown in Figure 3a. This application allows a maximum of four replicas to be placed in the system (i.e., $N_1 = 4$), and it has three requests. Moreover, the base station has resources available to receive a maximum of one request, while the core node can receive two requests, and the cloud has no restriction. Figure 3b shows the chromosome of an individual in this defined system. This chromosome leads the decoder algorithm to select only two nodes as candidates to host the application because $\lceil I_1 N_1 \rceil = \lceil 0.5 * 4 \rceil = 2$. More specifically, the base station and core nodes are selected because they have higher values in the second part of the chromosome (i.e., $S_1^{BS} = 0.7, S_1^{core} = 0.4$). The third part of the chromosome indicates that request E_2 is assigned first and followed by requests E_1 and E_3. According to the shortest response time order in the second step of Algorithm 1, a replica of the application with request E_2 is placed in the base station and then another replica with the remaining requests E_1 and E_3 is placed in the core node. Figure 3c shows the feasible solution decoded from the exemplified chromosome.

Complexity Analysis. Let $A = |\mathcal{A}|, V = |\mathcal{V}|$ and $R = |\mathcal{R}|$. The first outermost loop of Algorithm 1 (lines 4 to 7) has complexity $O(AV \log V)$ due to the sorting procedure on line 6. Line 8 has complexity $O(Q \log Q)$. Assuming

Algorithm 1: Proposed Chromosome Decoder.

Data: *individual*

Result: Decoded solution (ρ, γ, δ)

initialize $\rho_a^v, \gamma_a^{u,v}, \delta_a^{u,v} \leftarrow 0$;

initialize $d_a^{u,v} \leftarrow d_{net}^{a,u,v}$;

$I, S, E \leftarrow individual.chromosome$;

`/* Part I: Node Selection */`

forall $a \in \mathcal{A}$ **do**

  $n \leftarrow \min(|\mathcal{V}|, \lceil I_a N_a \rceil)$;

  $V_a \leftarrow$ select n nodes with higher S_a^v, $v \in \mathcal{V}$;

  $V_a \leftarrow V_a \cup \{\text{cloud}\}$;

`/* Part II: Request Distribution */`

$L \leftarrow$ list of requests sorted by E in descending order;

forall $r \in L$ **do**

  $a \leftarrow app_r$; `// requested application of` r

  $u \leftarrow source_r$; `// source node of` r

  sort nodes v in V_a by $d_a^{u,v}$ in ascending order;

  forall $v \in V_a$ **do**

    if *assigning* r *to* v *respects Constraints* (10) *and* (11) **then**

      $\rho_a^v \leftarrow 1$;

      $\gamma_a^{u,v} \leftarrow 1$;

      $\delta_a^{u,v} \leftarrow \delta_a^{u,v} + 1$;

      update free resources on v given (ρ, γ, δ);

      update $d_a^{u,v}$ by Equation 1 and current (ρ, γ, δ);

      break;

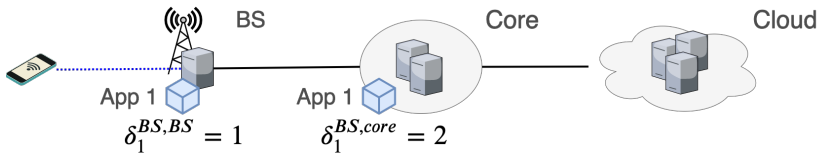
`/* Part III: Feasibility Verification */`

forall $a \in \mathcal{A}$ **do**

  $n \leftarrow N_a - \sum_{i \in \mathcal{V}} \rho_a^i$;

  if $n > 0$ **then**

    replace n replicas of a with the cloud node;



(a) System model.

0.5	0.7	0.4	0.1	0.6	0.8	0.3
I_1	S_1^{BS}	S_1^{core}	S_1^{cloud}	E_1	E_2	E_3

(b) Chromosome as an input parameter.

ρ_1^{BS}	ρ_1^{core}	ρ_1^{cloud}	$\delta_1^{BS,BS}$	$\delta_1^{BS,core}$	$\delta_1^{BS,cloud}$
1	1	0	1	2	0

(c) Feasible solution decoded.

Figure 3: Decoding example for Algorithm 1.

that checking the satisfaction of constraints on line 14 can be done in $O(R)$ and the update of variables between lines 15 and 19 has complexity $O(1)$. Then the complexity of the second outermost loop (lines 9 to 20) is $O(QV \log V + QVR)$. Given that the number of resources types is much less than the number of nodes (i.e., $R \ll V$), then we have $O(QV \log V + QVR) = O(QV \log V)$. The loop between lines 21 and 24 has complexity $O(AV)$. Therefore, Algorithm 1 has complexity $O((A+Q)V \log V + Q \log Q)$.

5.2. Initial Population

BRKGA initializes the first population with individuals as vectors of random values. An alternative is to populate the starting population with a few solutions obtained with another heuristic for the specific problem being solved while the remaining individuals are randomly generated. This strategy may help speed up the convergence of the algorithm and improve the quality of the final solutions. Hence, we apply the following heuristics that encode some feasible solutions on

the proposed chromosome representation:

- **Cloud**. A simple solution is to place all applications in the cloud. According to Algorithm 1 (lines 4-7), it is sufficient that $I_a = 0$ to only select the cloud node. Thus, the solution is encoded as

$$I_a = 0,\ S_a^v = 0,\ E_q = 0 \quad \forall a \in \mathcal{A}, \forall v \in \mathcal{V}, \forall q \in [1, Q]$$

- **Deadline**. Another heuristic is to prioritize requests for applications with shorter deadline requirements. In addition, the heuristic selects as many nodes as possible, i.e., $I_a = 1$, for an application to reduce response time and deadline violations. Given D_{app_q} as the application deadline requirement of request q, then the heuristic solution is encoded as

$$\begin{aligned} I_a &= 1, && \forall a \in \mathcal{A} \\ S_a^v &= 0, && \forall a \in \mathcal{A}, \forall v \in \mathcal{V} \\ E_q &= 1 - \frac{D_{app_q}}{\max_{a \in \mathcal{A}} D_a} && \forall q \in [1, Q] \end{aligned}$$

- **Net Delay**. Zhao and Liu [17] propose a heuristic that selects nodes with the lowest network latency for all other nodes as candidates to host an application. We encode this heuristic as follows:

$$\begin{aligned} I_a &= 1, && \forall a \in \mathcal{A} \\ S_a^v &= 1 - \frac{\sum_{i \in \mathcal{V}} d_{net}^{a,i,v}}{\max_{j \in \mathcal{V}} \sum_{i \in \mathcal{V}} d_{net}^{a,i,j}}, && \forall a \in \mathcal{A}, \forall v \in \mathcal{V} \\ E_q &= 0 && \forall q \in [1, Q] \end{aligned}$$

- **Cluster**. For each application, a heuristic is to place replicas of it in regions where its end-users are located. We can define a region as a set of close nodes where there are users attached to them (i.e., $|\mathcal{U}_a^v| > 0$). Moreover, we apply the K-medoids clustering technique [37] to detect those regions. The K-medoids algorithm is a variation of the classical

K-means algorithm. Despite the similarity between these two algorithms, K-medoids chooses points in the dataset as centers, or medoids, of the clusters and can be used with an arbitrary distance function, while in K-means, a cluster center is not necessary a point in the dataset. Moreover, Park and Jun [37] propose a simple and fast K-metoid algorithm with time complexity of $O(nk)$, where n is the dataset size, and k is the number of clusters. These characteristics allow us to partition the subset of nodes where $|\mathcal{U}_a^v| > 0$ in any connected graph using the network delay $d_{net}^{a,u,v}$ as the distance function. Let $\mathcal{M}_a$ be the set of medoids obtained after the execution of the clustering algorithm for an application a with a maximum of N_a clusters. Then, the heuristic to place replicas close to end-users prioritizes nodes near to a center of $|\mathcal{M}_a|$ regions, which is formally encoded as

$$
\begin{aligned}
I_a &= 1, && \forall a \in \mathcal{A} \\
S_a^v &= 1 - \frac{\min_{i \in \mathcal{M}_a} d_{net}^{a,i,v}}{\max_{j \in \mathcal{V}} \min_{i \in \mathcal{M}_a} d_{net}^{a,i,j}}, && \forall a \in \mathcal{A}, \forall v \in \mathcal{V} \\
E_q &= 0 && \forall q \in [1, Q]
\end{aligned}
$$

- **Combined Solution**. Given $\mathcal{S}$ as a set of random-key chromosome vectors and s_i the i-th element of a vector $s \in \mathcal{S}$ with length n, then we can combine two or more heuristic solutions by summing their encoded vectors as specified below:

$$
s_i^+ = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} s_i \quad \forall i \in \{1, \ldots, n\}
$$

- **Inverted Solution**. In order to add more diversity to the initial population, we can add complementary, or inverted, solutions to those obtained by the above heuristics. Since $s_i \in [0, 1]$ is valid for each element of a chromosome vector s, we can obtain the inverted solution by doing the following operation:

$$\bar{s}_i = 1 - s_i \quad \forall i \in \{1, \ldots, n\}$$

5.3. Next Population and Stopping Criteria

At the beginning of an iteration or generation t, our genetic algorithm decodes all individuals in the current population into feasible solutions. Next, it applies the sorting procedure of NSGA-II to rank individuals based on a dominance operator and a mechanism to preserve diversity. In this sorting procedure, we can either use our proposed dominance operator $\prec^1$ or the Pareto dominance operator $\prec$ to check if a solution is better than another in a multi-objective context.

After the population sorting, the proposed GA partitions the population into two groups: (i) the best-ranked individuals as an elite group, and (ii) the remaining individuals as non-elite ones. Then, it adds the elite individuals without modification in the next generation. This elitist strategy keeps track of good solutions found during the algorithm iterations, which results in a monotonically improving heuristic. It is important to note that elite group size is a parameter of the algorithm, but a relatively small value is generally used.

In addition to the elite individuals, the next population is also composed of (i) mutant and (ii) offspring individuals. We utilize the BRKGA procedure to generate mutant individuals as simple random-generated vectors. Our GA completes the remainder of the next population with offspring. Each new offspring individual is produced by combining one solution selected at random from the group of elite individuals and one from the set of non-elite individuals. Similar to BRKGA, we use parameterized uniform crossover mechanism [38] to combine two parent solutions and obtaining a new offspring. In this crossover mechanism, an offspring inherits a gene (i.e., the value of a vector element) of its elite parent with a probability $P_{\text{elite}} \geq 0.5$.

The above-discussed procedure for generating the next population is repeated until a stopping criterion is met. We use the stopping criterion for multi-objective problems proposed by Martí et al. [20], which is called MGBM

after the authors surnames. MGBM tries to detect situations where no further progress will be made in the GA by designing a progress estimator $\hat{I}_{\text{mdr}}(t)$. Thus, our GA terminates either when it reaches a maximum number of generations (i.e., $t \geq t_{\max}$) or when the progress estimation falls below a defined threshold (i.e., $\hat{I}_{\text{mdr}}(t) < \hat{I}^{\min}_{\text{mdr}}$). Here, $t_{\max}$ and $\hat{I}^{\min}_{\text{mdr}}$ are parameters to be set, and $t \geq 1$ is the generation/iteration index.

6. Performance Analysis

In this section, we present the performance (i.e., the optimality) results of our proposed Genetic Algorithm (GA) by comparing it with benchmarking algorithms over a cellular network (5G) with Edge Computing (EC) capabilities.

This section is structured as follows. First, Subsection 6.3 presents the performance metrics. Next, Subsection 6.2 describes the evaluated algorithms. Then, Subsection 6.3 details the experiment setup. In Subsection 6.4, we define the values of key parameters of the proposed GA. Finally, we analyze the obtained experimental results in Subsection 6.5.

6.1. Performance Metrics

We select the deadline violation f_{dv}, operational cost f_{cost}, and service unavailability f_{fail} as performance-related functions to be optimized. We chose these functions because they are relevant in the context of EC, and there are conflicts between them, as discussed in Section 4.4. It is important to note that we can use other objectives since the proposed algorithm has no restrictions on this aspect.

6.2. Evaluated Algorithms

An overview of the compared algorithms is given below:

- ***MILP*** returns the optimal solution for the linear and relaxed version of (13) (see Appendix A). The optimal solution is found by the branch and cut technique in the CPLEX linear solver. We also use a timeout parameter to stop the solver tool.

- ***Cloud*** simply places everything in the cloud node.
- ***NetDelay+DL*** combines the *Net Delay* and *Deadline* heuristics presented in Section 5.2.
- ***Cluster+DL*** combines the *Cluster* and *Deadline* heuristics presented in Section 5.2.
- ***MOHGA*($\prec^1$)** is the proposed GA for multi-objective using the heuristic initialization and the dominance operator $\prec^1$. The result of this algorithm is the best-ranked individual because we are only interested in obtaining a single solution.
- ***MOHGA*($\prec$)** is similar to $MOHGA(\prec^1)$ but using the Pareto dominance operator $\prec$ instead.
- ***MOGA*($\prec^1$)** is the same as $MOHGA(\prec^1)$ but without using the heuristic initialization. That is, the first population is only randomly generated.
- ***SOHGA*(f)** uses the proposed GA with heuristic initialization to optimize a single objective function $f \in \{f_{dv}, f_{cost}, f_{fail}\}$.

6.3. Analysis Setup

We conduct the experiment in Python with the CPLEX solver [39] to evaluate the performance of the above-mentioned algorithms in a 5G network scenario. In this scenario, Base Stations (BSs) are equally distributed in a grid area, and there is a network link between neighboring BSs. There are also hosting nodes in different network parts (i.e., BS, core, and cloud), and their capacities are reduced as they descend from cloud to BSs.

We use the three types of applications specified for 5G networks, with the following characteristics [40]:

- **massive Machine Type Communications (mMTC)** has low resource usage, high deadline, and a large number of users;

Table 4: Performance Evaluation Parameters

Parameter	**Value**
System	
CPU (MIPS)	Cloud: ∞, Core: 200000, BS: 40000
Storage Disk (MB)	Cloud: ∞, Core: 32000, BS: 16000
RAM (MB)	Cloud: ∞, Core: 8000, BS: 4000
Node Availability P_v (%)	Cloud: 99.9, Core: 99.0, BS: 90.0
Cost G_v, G_v^r	Cloud: 0.025, Core: 0.05, BS: 0.1
User Proportion (%)	70 mMTC, 20 eMBB, 10 URLLC
App. Proportion (%)	34 mMTC, 33 eMBB, 33 URLLC
Applications	
Max. Replicas N_a	$[1, \lvert\mathcal{V}\rvert]$
Deadline D_a (ms)	mMTC: $[100, 1000]$, URLLC: $[1, 10]$, eMBB: $[10, 50]$
λ_a (requests/ms)	mMTC: $[0.0002, 0.001]$, eMBB: $[0.001, 0.01]$, URLLC: $[0.02, 0.2]$
App. Availability P_a (%)	mMTC, eMBB: $[80.0, 90.0]$, URLLC: $[90.0, 99.0]$
CPU Work W_a (MI)	mMTC, URLLC: $[1, 5]$, eMBB: $[1, 10]$
RAM, Disk $H_1^{a,r}, H_2^{a,r}$	mMTC, URLLC: $[1, 10]$, eMBB: $[1, 50]$
CPU $H_1^{a,r}, H_2^{a,r}$	W_a, $W_a/D_a + 1$
Net. Delay $D_{net}^{a,u,v}$ (ms) neighbor BS-BS, BS-Core	mMTC, URLLC: $[1, 2]$, eMBB: $[1, 5]$
Net. Delay $D_{net}^{a,u,v}$ (ms) Core-Cloud	mMTC, URLLC: $[10, 12]$, eMBB: $[10, 15]$

An interval $[a, b]$ means that a value is chosen randomly within this range.

- **Ultra Reliable Low Latency Communications (URLLC)** has low resource usage, a strict deadline, and a small volume of users;
- **enhanced Mobile Broadband (eMBB)** has high resource usage, a medium deadline, and an intermediate number of users.

Then, we randomly assign the value of the application parameters based on the above characteristics and some predictions for 5G discussed by Schulz

et al. [3] (response deadline and request rate). For evaluation purposes, we assume that it is sufficient to use relative values for the parameters among different application types instead of applying more realistically accurate values. In addition, we also assume that the application resource demand $h_a^r(\cdot)$ and node usage cost $g_v(\cdot)$ functions are linear according to Equations (19) and (20), respectively, where H_1^a, H_2^a, G_v, and G_v^r are constants.

$$h_a^r(\lambda) = H_1^{a,r}\lambda + H_2^{a,r} \tag{19}$$

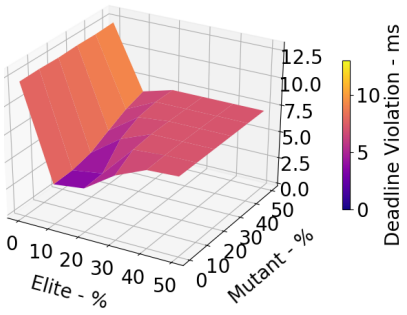
$$g_v(\lambda_a^v) = G_v\rho_a^v + \sum_{r\in\mathcal{R}} G_v^r h_a^r(\lambda_a^v) = G_v\rho_a^v + \sum_{r\in\mathcal{R}} G_v^r \left(H_1^{a,r}\lambda_a^v + H_2^{a,r}\right) \tag{20}$$

In order to have different user densities, users are distributed either uniformly or through isotropic Gaussian blobs [41] in the grid area. Then, a user is attached to the nearest BS. Finally, each test case is executed 30 times to obtain results with a 95% confidence interval [42]. Table 4 summarizes the major experiment parameters.

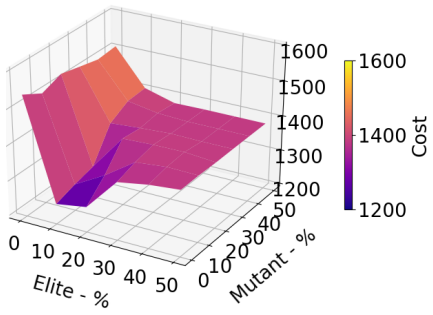
6.4. Different Parameters Settings

Regarding the parameters of the proposed $MOHGA(\prec^1)$, we analyze its performance in terms of the optimization objectives against different values of these parameters.

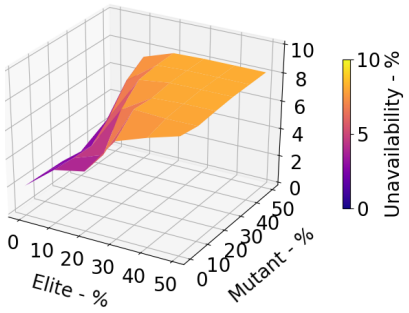
Elite and Mutant Group Size. In a GA with elitist strategy and random mutants, some parameters to be defined are the number of individuals in the elite and mutant sets. Figure 4 presents the influence of these parameters in the objective functions for $MOHGA(\prec^1)$ with a population size of 100. We observe that values in the range between 10% and 20% for these parameters result in better results for all three objectives. These values allow the algorithm to have a diversity of solutions within the search space and still benefit from the elitist strategy. Therefore, we select the number of elite and mutant individuals to be both 10% of all population.



(a) Max. Deadline Violation



(b) Overall Operational Cost

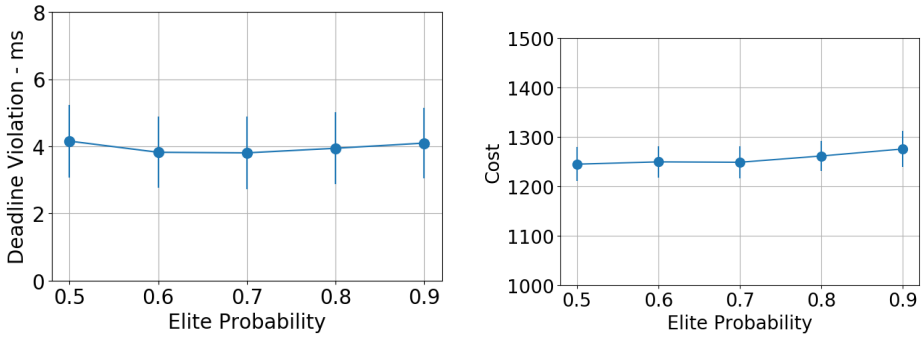


(c) Avg. Service Unavailability

Figure 4: Performance of different elite and mutant group sizes.

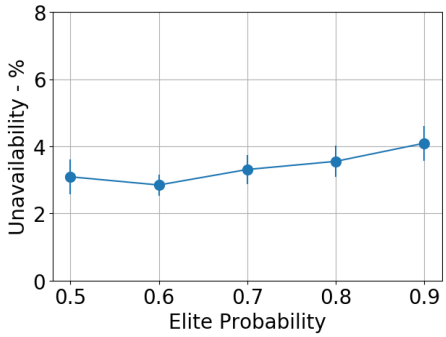
Elite Probability. In the crossover operation described in Section 5.3, a parameter to be defined is the probability P_{elite} to an offspring inheriting a gene of its elite parent. Figure 5 presents the impact of different values of P_{elite} on the objectives functions. We can see that this parameter does not have much influence on the objectives, but $P_{elite} = 0.6$ has a slightly better results, specially for service availability in Figure 5c.

Stopping Criteria Threshold. We examine the performance of different stopping threshold $\hat{I}^{min}_{mdr}$ values for $MOHGA(\prec^1)$ with a maximum of 100 generations and a population size of 100 for each generation. Regarding the optimization objectives, the algorithm performs better when the threshold is below 0.2. However, a small threshold implies that the algorithm iterates over more generations, consequently, resulting in longer execution time, as shown in Figure 6d. Hence,



(a) Max. Deadline Violation

(b) Overall Operational Cost



(c) Avg. Service Unavailability

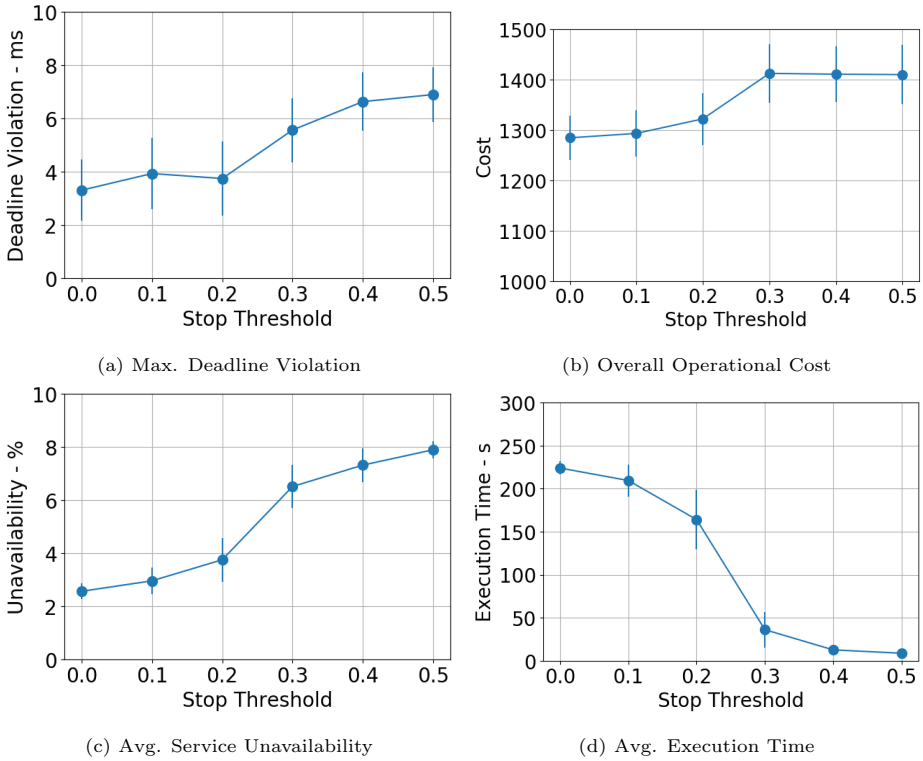
Figure 5: Performance of different elite probability P_{elite} values.

we select $\hat{I}^{min}_{mdr} = 0.1$ as a trade-off between optimality and execution time.

6.5. Results and Discussion

We evaluated performance of the examined algorithms for each optimization objective in scenarios with different amounts of applications, users, and hosting nodes.

Max. Deadline Violation. Figure 7a presents the impact on the system deadline violation level by increasing the number of deployed applications in a scenario with 5x5 BSs and $10k$ users. In all algorithms, the violation level grows as more applications compete for the fixed amount of node resources. More specifically, *Cloud* heuristic has the worst results due to the distance between the users and the cloud node. Meanwhile, the optimum solutions of *MILP*



(a) Max. Deadline Violation

(b) Overall Operational Cost

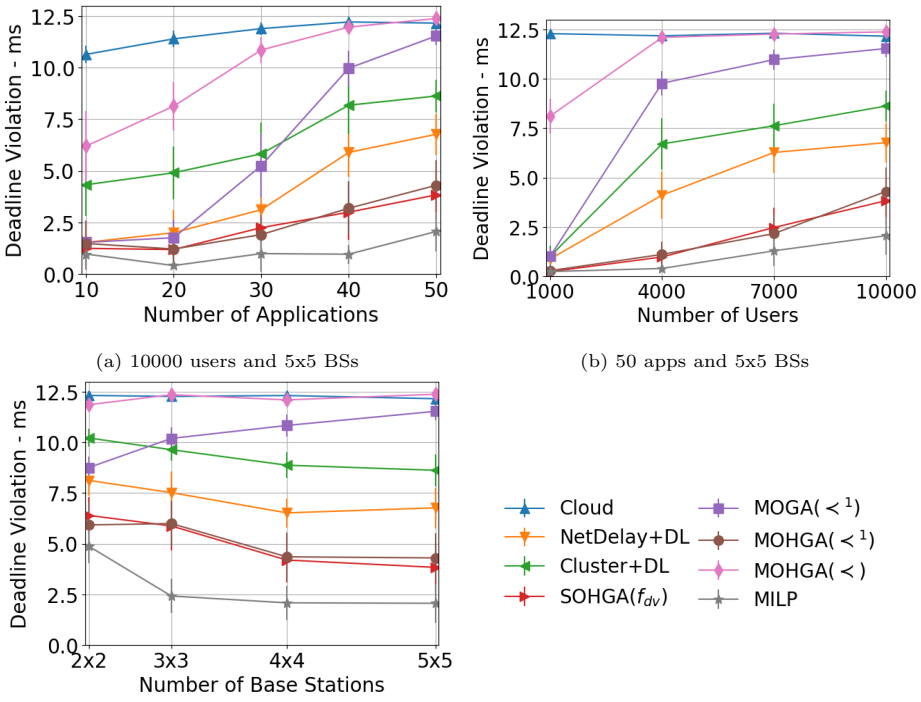
(c) Avg. Service Unavailability

(d) Avg. Execution Time

Figure 6: Performance of different stopping criteria threshold $\hat{I}_{mdr}^{min}$ values.

present, as expected, the best results. When we compare $MOHGA(\prec^1)$ with $MOGA(\prec^1)$ and $MOHGA(\prec)$, a drastic performance improvement of this GA is observed due to the inclusion of the heuristics initialization and preferred dominance operator $\prec^1$. $MOHGA(\prec^1)$ also outperforms both *NetDelay+DL* and *Cluster+DL* heuristics, which are used at its initialization. Furthermore, the multi-objective $MOHGA(\prec^1)$ has similar results obtained by only optimizing the deadline violation in $SOHGA(f_{dv})$ because the dominance operator $\prec^1$ prioritizes time-sensitive applications with strict deadline requirements. Both $MOHGA(\prec^1)$ and $SOHGA(f_{dv})$ perform near the optimum results of *MILP*.

The growth in users number also affects the demand of node resources and, consequently, the level rise of deadline violation, as shown in Figure 7b. *Cloud* heuristic is an exception in this figure with constant results due to the unlimited



(a) 10000 users and 5x5 BSs

(b) 50 apps and 5x5 BSs

(c) 10000 users and 50 apps

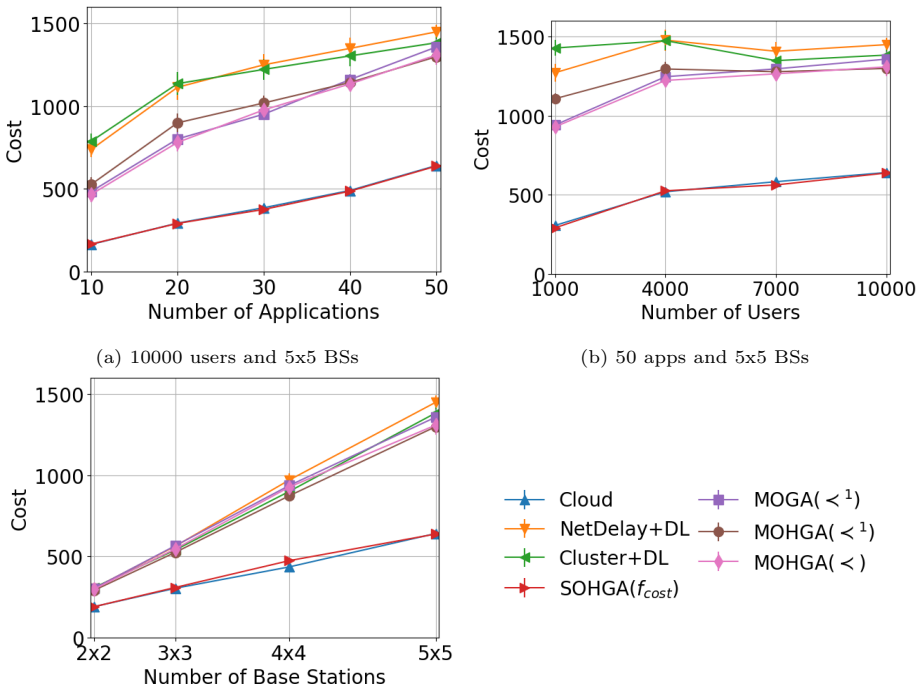
Legend

Figure 7: Maximum Deadline Violation

capacity of the cloud node. Besides that, the growth slope is less steep for *MOHGA*($\prec$), *MOGA*($\prec^1$), *Cluster+DL*, and *NetDelay+DL* algorithms when the number of users is greater than $4k$ due to surplus requests being processed in the cloud node. The other algorithms, *MILP*, *SOHGA*(f_{dv}), and *MOHGA*($\prec^1$), tend to have a linear growth behavior presumably because resource demand also increases linearly with the number of users in the tested scenario.

On the other hand, more nodes mean more resources available to meet application requirements. Figure 7c shows the deadline violation performance of the algorithms by varying the number of base stations with 50 applications and $10k$ users. *Cluster+DL*, *NetDelay+DL*, *MILP*, *SOHGA*(f_{dv}), and *MOHGA*($\prec^1$) exhibit violation decrease by adding more nodes, while *Cloud* and *MOHGA*($\prec$) have similar results in all tested variations. However, *MOGA*($\prec^1$) produces worse results with more base stations, possibly due to a higher number of chro-

mosome values combinations.



(a) 10000 users and 5x5 BSs

(b) 50 apps and 5x5 BSs

(c) 10000 users and 50 apps

Legend

Figure 8: Overall Operational Cost

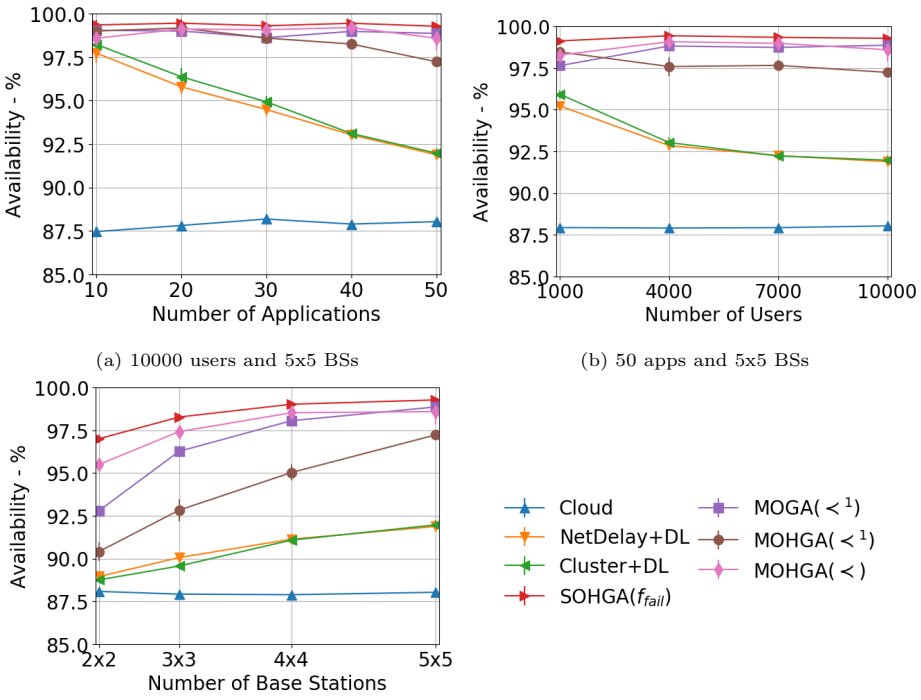
Overall Operational Cost. Figure 8 presents the performance of the compared algorithms according to cost function f_{cost}. In the tested scenarios, the optimum solution to minimize the overall operational cost is to place only one replica of each application in the cloud node, which has the cheapest resources. Thus, *Cloud* heuristic is the optimum solution in this case. As solution *SOHGA*(f_{cost}) includes the *Cloud* heuristic and only optimizes the cost function f_{cost}, it also achieves an optimal solution. In addition, the multi-objective algorithms *MOHGA*($\prec^1$), *MOHGA*($\prec$), and *MOGA*($\prec^1$) have slightly better results than *NetDelay+DL* and *Cluster+DL* by varying the number of applications and users, as shown in Figures 8a and 8b, respectively.

In Figure 8c, all algorithms present a linear cost increase by varying the

number of nodes but with a fixed number of users and applications. This behavior is explained by how we estimate the number of requests generated. More specifically, we assume that there is at least one request coming from a node with users attached when the number of requests is defined as $Q_a^u = \lceil |\mathcal{U}_a^u| \lambda_a \rceil$ in Section 4.1.2. Moreover, users will be more distributed in the test scenario by increasing the number of BSs. Consequently, there are more nodes with attached users and, thus, more requests are generated even if the number of users and applications does not change. Then, increasing the number of requests implies more resource consumption and higher operational costs.

Avg. Service Availability. Figures 9a, 9b, and 9c relate the impact of service availability, or unavailability, with the variation of the number of applications, users, and nodes, respectively. *Cloud* heuristic has the worst results due to the placement of a single replica for each application, and all compared genetic algorithms outperform *NetDelay+DL* and *Cluster+DL*. Moreover, $MOHGA(\prec^1)$ has slightly lower availability than other GAs in Figures 9a and 9b. Meanwhile, the performance difference between $MOHGA(\prec^1)$ and the other GAs is most notable in Figure 9c. This performance degradation may be explained by the trade-off of solution $MOHGA(\prec^1)$ having less deadline violation. That is, $MOHGA(\prec^1)$ may try to place more replicas of time-sensitive applications to reduce deadline violations. Then, fewer resources are available to place many replicas of time-tolerant applications. Consequently, the average availability across all applications is impacted in $MOHGA(\prec^1)$ as there are more time-tolerant than time-sensitive applications in our tests.

In Figures 9a and 9b, we observe a considerable loss of availability for *NetDelay+DL* and *Cluster+DL* heuristic by increasing the number of applications or users on a 5x5 BSs grid. As these heuristics may prioritize the same edge nodes for applications with some similar characteristics (e.g., network latency and user distribution), so there is more competition for those nodes by increasing the number of applications or users. As a result, *NetDelay+DL* and *Cluster+DL* deploy fewer application replicas resulting in decreased service avail-



(a) 10000 users and 5x5 BSs

(b) 50 apps and 5x5 BSs

(c) 10000 users and 50 apps

Legend

Figure 9: Average Service Availability

ability. Meanwhile, the other compared algorithms do not have large variations in their performance, which may be explained by the use of the cloud node to receive the increased resource demand.

Figure 9c shows that the majority of compared algorithms improve their performance by adding more hosting nodes in the system. As the number of nodes grows, it is possible to place more application replicas on the system, and thus, increasing service availability. *Cloud* is the only algorithm without improvement as the number of replicas deployed per application is fixed.

7. Conclusion

In this paper, we jointly formulated the static service placement and load distribution in an IoT and Edge Computing environment as an optimization

problem by considering diverse application characteristics (e.g., response deadline, resource demand, scalability, and availability). The formulated problem aims to minimize SLA infringements, caused by violations of the deadline requirement, and other possibly conflicting objectives (e.g., operational cost and unavailability). Then, we proposed a multi-objective genetic algorithm based on BRKGA and NSGA-II to obtain feasible solutions close to the Pareto optimal front. We also modified the Pareto dominance operator to prioritize applications with strict deadline requirements. Furthermore, we included heuristic solutions during the initialization of the proposed genetic algorithm to improve its solution results.

We analyzed the efficiency of the proposed algorithm through simulations. Our experimental results show that the proposed multi-objective GA achieves values close to the optimum of the MILP formulation in terms of deadline violation, and still generally outperforms the benchmark heuristics for the other analyzed objectives (operational cost and service availability). Moreover, we observed that the results of the proposed GA are related to the deadline prioritization and the heuristic initialization.

As future work, we plan to investigate the dynamic (or online) service placement and load distribution problems, which includes application migration, user mobility, and other dynamic changes in the network.

Acknowledgements

This work was partially financed by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, and by the FUI23 Project, SCORPION, in France.

Appendix A. Linearization and Relaxation

We transform the nonlinear problem (13) into a Mixed-Integer Linear Programming (MILP) problem by linearizing and relaxing its objective function, and constraints (10) and (11). These transformations are described as follows.

Node Capacity Constraint. For an application a, its monotonic resource demand function $h_a^r(\lambda)$ may be nonlinear for a specific resource type $r \in \mathcal{R}$. In this case, the function $h_a^r(\lambda)$ can be replaced by an over linear estimator $\hat{h}_a^r(\lambda)$ in the domain interval $[0, Q_a]$, as shown in Equation A.1, where $H_1^{a,r}$, $H_2^{a,r}$ are constants, and Q_a is equal to $\sum_{v \in \mathcal{V}} Q_a^v$. That is, $h_a^r(\lambda)$ is substituted by a linear function $\hat{h}_a^r(\lambda)$ such that $h_a^r(\lambda) \leq \hat{h}_a^r(\lambda)$ for all $\lambda \in [0, Q_a]$. The idea here is that $\hat{h}_a^r(\lambda)$ results in a resource demand equal to or greater than that originally required by $h_a^r(\lambda)$. For instance, a naive replacement can be the constant function $\hat{h}_a^r(\lambda) = H_2^{a,r}$, where $H_2^{a,r} = \max\{h_a^r(x) \mid 0 \leq x \leq Q_a\}$.

$$\hat{h}_a^r(\lambda) = H_1^{a,r}\lambda + H_2^{a,r} \tag{A.1}$$

Given that requests only arrive at servers running the requested application according to Equations (3), (7) and (8), we have:

$$\rho_a^v \lambda_a^v = \lambda_a^v \tag{A.2}$$

By applying Equations (A.1) and (A.2) to Equation 10, the node capacity constraint can be rewritten as:

$$\sum_{a \in \mathcal{A}} (\lambda_a^v H_1^{a,r} + \rho_a^v H_2^{a,r}) \leq C_v^r \quad \forall r \in \mathcal{R}, \forall v \in \mathcal{V} \tag{A.3}$$

Queue Stability Constraint. In order to have a linear queue stability constraint, we apply Equations (4) and (A.1) to Constraint (11) and obtain:

$$\lambda_a^v \left(H_1^{a,CPU} - W_a\right) + H_2^{a,CPU} > 0 \quad \forall a, v\,(\rho_a^v = 1)\,, a \in \mathcal{A}, v \in \mathcal{V} \tag{A.4}$$

However, it must remove the strictness of the above inequality to obtain a standard form of a MILP problem. For this, it is added a small constant $\Theta \in (0, 1]$. Furthermore, both sides of the inequality are multiplied by ρ_a^v to ensure the queue existence constraint. Then, we further have:

$$\rho_a^v \lambda_a^v \left(H_1^{a,CPU} - W_a\right) + \rho_a^v H_2^{a,CPU} \geq \rho_a^v \Theta \quad \forall a \in \mathcal{A}, \forall v \in \mathcal{V} \tag{A.5}$$

Finally, applying Equation A.2 to the above result, we obtain the following linear queue stability constraint:

$$\lambda_a^v \left(H_1^{a,CPU} - W_a\right) + \rho_a^v H_2^{a,CPU} \geq \rho_a^v \Theta \quad \forall a \in \mathcal{A}, \forall v \in \mathcal{V} \tag{A.6}$$

Objective Function. Problem (13) has a $\min\max f(x)$ objective, which is nonlinear because max function is nonlinear. This type of problem can be transformed into one without max function by replacing $\min\max f(x)$ for $\min z$, where z is a new variable, and adding a new constraint relating this variable to $f(x)$ (i.e., $f(x) \leq z$). In this way, the objective is linear, but it is necessary to check the linearity of the new constraint. Based on this transformation, we can add the following constraint in the problem:

$$\gamma_a^{u,v} d_a^{u,v} - D_a \leq \varepsilon \quad \forall a \in \mathcal{A}, \forall u, v \in \mathcal{V} \tag{A.7}$$

where $\varepsilon \geq 0$ is a new variable indicating the system deadline violation that we want to minimize. Moreover, given Equations (1), (2), (4), (5) and (A.1), we rewrite Constraint (A.7) as:

$$\begin{aligned}\left(\gamma_a^{u,v}\lambda_a^v d_{net}^{a,u,v} - \varepsilon\lambda_a^v - \lambda_a^v D_a\right)\left(H_1^{a,CPU} - W_a\right) \\ + \gamma_a^{u,v}\left(H_2^{a,CPU} d_{net}^{a,u,v} + W_a\right) - H_2^{a,CPU}\left(D_a + \varepsilon\right) \leq 0 \\ \forall a \in \mathcal{A}, \forall u, v \in \mathcal{V}\end{aligned} \tag{A.8}$$

However, in Constraint (A.8), both $\gamma_a^{u,v}\lambda_a^v$ and $\varepsilon\lambda_a^v$ are bilinear terms (i.e., multiplication of two variables). We can relax these terms to obtain linear ones using McCormick envelopes [43]. That is, we replace these bilinear terms with new variables ($\varphi_a^{u,v} = \gamma_a^{u,v}\lambda_a^v$ and $\psi_a^v = \varepsilon\lambda_a^v$) and add the following new linear constraints in the problem:

$$0 \leq \gamma_a^{u,v} \leq 1 \text{ and } 0 \leq \lambda_a^v \leq Q_a \text{ and } 0 \leq \varepsilon \leq E \qquad \forall a \in \mathcal{A}, \forall u, v \in \mathcal{V} \tag{A.9a}$$

$$0 \leq \varphi_a^{u,v} \leq \lambda_a^v \text{ and } Q_a\left(\gamma_a^{u,v} - 1\right) + \lambda_a^v \leq \varphi_a^{u,v} \leq \gamma_a^{u,v} \qquad \forall a \in \mathcal{A}, \forall u, v \in \mathcal{V} \tag{A.9b}$$

$$0 \leq \psi_a^v \leq \lambda_a^v E \text{ and } \varepsilon Q_a + \lambda_a^v E - E Q_a \leq \psi_a^v \leq \varepsilon Q_a \qquad \forall a \in \mathcal{A}, \forall u, v \in \mathcal{V} \tag{A.9c}$$

where E is a constant specifying the maximum deadline violation allowed. Then, we can rewrite Constraint (A.8) with the two new variables to have a

linear constraint:

$$\left(\varphi_a^{u,v} d_{net}^{a,u,v} - \psi_a^v - \lambda_a^v D_a\right)\left(H_1^{a,CPU} - W_a\right) + \gamma_a^{u,v}\left(H_2^{a,CPU} d_{net}^{a,u,v} + W_a\right) - H_2^{a,CPU}\left(D_a + \varepsilon\right) \leq 0 \quad \forall a \in \mathcal{A}, \forall u, v \in \mathcal{V} \tag{A.10}$$

Linear Formulation. Let $\varphi = \{\varphi^{u,v} \mid a \in \mathcal{A} \text{ and } u, v \in \mathcal{V}\}$, $\psi = \{\psi_a^v \mid a \in \mathcal{A} \text{ and } v \in \mathcal{V}\}$, and $x = (\rho, \gamma, \delta, \varepsilon, \varphi, \psi)$. Then, we use the above linearizations and relaxations to formulate the MILP problem of the static single objective case as follows:

$$\begin{aligned} &\min_x \varepsilon \\ &\quad x = (\rho, \gamma, \delta, \varepsilon, \varphi, \psi) \\ &\text{s.t. eqs. (6) to (9), (A.3), (A.6), (A.9) and (A.10)} \end{aligned} \tag{A.11}$$

It is important to note that a solution to problem (A.11) is also feasible for problem (13), but it may present a higher objective value ε when applied to the original problem due to the bilinear relaxation.

References

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash, Internet of things: A survey on enabling technologies, protocols, and applications, IEEE Communications Surveys Tutorials 17 (2015) 2347–2376.

[2] J. Pan, J. McElhannon, Future edge cloud and edge computing for internet of things applications, IEEE Internet of Things Journal 5 (2018) 439–449.

[3] P. Schulz, M. Matthe, H. Klessig, M. Simsek, G. Fettweis, J. Ansari, S. A. Ashraf, B. Almeroth, J. Voigt, I. Riedel, A. Puschmann, A. Mitschele-Thiel, M. Muller, T. Elste, M. Windisch, Latency critical iot applications in 5g: Perspective on the design of radio interface and network architecture, IEEE Communications Magazine 55 (2017) 70–78.

[4] P. Hu, S. Dhelim, H. Ning, T. Qiu, Survey on fog computing: architecture, key technologies, applications and open issues, Journal of Network and Computer Applications 98 (2017) 27 – 42.

[5] H. Liu, F. Eldarrat, H. Alqahtani, A. Reznik, X. de Foy, Y. Zhang, Mobile edge cloud system: Architectures, challenges, and approaches, IEEE Systems Journal 12 (2018) 2495–2508.

[6] R. Roman, J. Lopez, M. Mambo, Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges, Future Generation Computer Systems 78 (2018) 680 – 698.

[7] K. Bilal, O. Khalid, A. Erbad, S. U. Khan, Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers, Computer Networks 130 (2018) 94 – 120.

[8] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, A. Ahmed, Edge computing: A survey, Future Generation Computer Systems 97 (2019) 219 – 235.

[9] W. Tärneberg, A. Mehta, E. Wadbro, J. Tordsson, J. Eker, M. Kihl, E. Elmroth, Dynamic application placement in the mobile cloud network, Future Generation Computer Systems 70 (2017) 163 – 177.

[10] M. C. S. Filho, C. C. Monteiro, P. R. Inácio, M. M. Freire, Approaches for optimizing virtual machine placement and migration in cloud environments: A survey, Journal of Parallel and Distributed Computing 111 (2018) 222 – 250.

[11] O. Skarlat, M. Nardelli, S. Schulte, S. Dustdar, Towards qos-aware fog service placement, in: 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC), 2017, pp. 89–96. doi:10.1109/ICFEC.2017.12.

[12] M. Xu, W. Tian, R. Buyya, A survey on load balancing algorithms for virtual machines placement in cloud computing, Concurrency and Computation: Practice and Experience 29 (2017) e4123.

[13] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, K. K. Leung, Dynamic service migration and workload scheduling in edge-clouds, Performance Evaluation 91 (2015) 205 – 228. Special Issue: Performance 2015.

[14] F. L. Pires, B. Barán, A virtual machine placement taxonomy, in: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2015, pp. 159–168. doi:10.1109/CCGrid.2015.15.

[15] I. Pietri, R. Sakellariou, Mapping virtual machines onto physical machines in cloud computing: A survey, ACM Comput. Surv. 49 (2016) 49:1–49:30.

[16] L. Gu, D. Zeng, S. Guo, A. Barnawi, Y. Xiang, Cost efficient resource management in fog computing supported medical cyber-physical system, IEEE Transactions on Emerging Topics in Computing 5 (2017) 108–119.

[17] L. Zhao, J. Liu, Optimal placement of virtual machines for supporting multiple applications in mobile edge networks, IEEE Transactions on Vehicular Technology 67 (2018) 6533–6545.

[18] A. M. Maia, Y. Ghamri-Doudane, D. Vieira, M. F. de Castro, Optimized placement of scalable iot services in edge computing, in: 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), 2019, pp. 189–197.

[19] A. M. Maia, Y. Ghamri-Doudane, D. Vieira, M. F. de Castro, A multi-objective service placement and load distribution in edge computing, in: 2019 IEEE Global Communications Conference (GLOBECOM), 2019, pp. 1–7. doi:10.1109/GLOBECOM38437.2019.9014303.

[20] L. Martí, J. García, A. Berlanga, J. M. Molina, A stopping criterion for multi-objective optimization evolutionary algorithms, Information Sciences 367-368 (2016) 700 – 718.

[21] J. O. de Carvalho, F. Trinta, D. Vieira, O. A. C. Cortes, Evolutionary solutions for resources management in multiple clouds: State-of-the-art

and future directions, Future Generation Computer Systems 88 (2018) 284 – 296.

[22] P. Mach, Z. Becvar, Mobile edge computing: A survey on architecture and computation offloading, IEEE Communications Surveys Tutorials 19 (2017) 1628–1656.

[23] L. Yang, J. Cao, G. Liang, X. Han, Cost aware service placement and load dispatching in mobile cloud systems, IEEE Transactions on Computers 65 (2016) 1440–1452.

[24] K. Katsalis, T. G. Papaioannou, N. Nikaein, L. Tassiulas, Sla-driven vm scheduling in mobile edge computing, in: 2016 IEEE 9th International Conference on Cloud Computing (CLOUD), 2016, pp. 750–757. doi:10.1109/CLOUD.2016.0104.

[25] B. Spinnewyn, R. Mennes, J. F. Botero, S. Latré, Resilient application placement for geo-distributed cloud networks, Journal of Network and Computer Applications 85 (2017) 14 – 31. Intelligent Systems for Heterogeneous Networks.

[26] R. W. Floyd, Algorithm 97: Shortest path, Commun. ACM 5 (1962) 345–.

[27] S. Warshall, A theorem on boolean matrices, J. ACM 9 (1962) 11–12.

[28] F. Metzger, T. Hoßfeld, A. Bauer, S. Kounev, P. E. Heegaard, Modeling of aggregated iot traffic and its application to an iot cloud, Proceedings of the IEEE 107 (2019) 679–694.

[29] S. Burer, A. N. Letchford, Non-convex mixed-integer nonlinear programming: A survey, Surveys in Operations Research and Management Science 17 (2012) 97 – 106.

[30] W. Stadler, A survey of multicriteria optimization or the vector maximum problem, part i: 1776–1960, Journal of Optimization Theory and Applications 29 (1979) 1–52.

[31] Y. Cui, Z. Geng, Q. Zhu, Y. Han, Review: Multi-objective optimization methods and application in energy saving, Energy 125 (2017) 681 – 704.

[32] D. E. Goldberg, J. H. Holland, Genetic algorithms and machine learning, Machine Learning 3 (1988) 95–99.

[33] J. H. Holland, et al., Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence, MIT press, 1992.

[34] C. A. C. Coello, Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art, Computer Methods in Applied Mechanics and Engineering 191 (2002) 1245 – 1287.

[35] J. F. Gonçalves, M. G. C. Resende, Biased random-key genetic algorithms for combinatorial optimization, Journal of Heuristics 17 (2011) 487–525.

[36] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: Nsga-ii, IEEE Transactions on Evolutionary Computation 6 (2002) 182–197.

[37] H.-S. Park, C.-H. Jun, A simple and fast algorithm for k-medoids clustering, Expert Systems with Applications 36 (2009) 3336 – 3341.

[38] W. M. Spears, K. D. De Jong, On the virtues of parameterized uniform crossover, Technical Report, NAVAL RESEARCH LAB WASHINGTON DC, 1995.

[39] I. I. CPLEX, V12. 1: User's manual for cplex, International Business Machines Corporation 46 (2009) 157.

[40] N. Alliance, 5g white paper, Next generation mobile networks, white paper (2015) 1–125.

[41] Scikit-learn, Generate isotropic gaussian blobs for clustering, 2019. URL: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html, [Online; accessed 17-July-2019].

[42] R. Jain, The art of computer systems performance analysis, volume 182, John Wiley & Sons Chichester, 1991.

[43] G. P. McCormick, Computability of global solutions to factorable nonconvex programs: Part i — convex underestimating problems, Mathematical Programming 10 (1976) 147–175.