An optimization view to the design of edge computing infrastructures for IoT applications

Thiago Alves de Queiroz, Claudia Canali, Manuel Iori, Riccardo Lancellotti

Abstract Internet of Things (IoT) based applications have recently experienced a remarkable diffusion in many different contexts, such as automotive, e-health, public security, industrial applications, energy, and waste management. These kinds of applications are characterized by geographically distributed sensors that collect data to be processed through algorithms of Artificial Intelligence (AI). Due to the vast amount of data to be processed by AI algorithms and the severe latency requirements of some applications, the emerging Edge Computing paradigm may represent the preferable choice for the supporting infrastructure. However, the design of edge-computing infrastructures opens several new issues concerning the allocation of data flows coming from sensors over the edge nodes, and the choice of the number and the location of the edge nodes to be activated. The service placement issue can be modeled through a multi-objective optimization aiming at minimizing two aspects: the response time for data transmission and processing in the sensors-edge-cloud path; the (energy or monetary) cost related to the number of turned on edge nodes. Two heuristics, based on Variable Neighborhood Search and on Genetic Algorithms, are proposed and evaluated over a wide range of scenarios, considering a realistic smart city application with 100 sensors and up to 10 edge nodes. Both heuristics can return practical solutions for the given application. The results indicate

Thiago Alves de Queiroz
Institute of Mathematics and Technology, Federal University of Catalão, Catalão-Goiás, Brazil e-mail: taq@ufg.br

Claudia Canali
Department of Engineering "Enzo Ferrari", University of Modena and Reggio Emilia, Modena, Italy, e-mail: claudia.canali@unimore.it

Manuel Iori
Department of Science and Methods for Engineering, University of Modena and Reggio Emilia, Reggio Emilia, Italy, e-mail: manuel.iori@unimore.it

Riccardo Lancellotti
Department of Engineering "Enzo Ferrari", University of Modena and Reggio Emilia, Modena, Italy, e-mail: riccardo.lancellotti@unimore.it

a suitable topology for a network-bound scenario requires less enabled edge nodes comparatively with a CPU-bound scenario. In terms of performance gain, the VNS outperformed in almost every condition the GA approach, reaching a performance gain up to almost 40% when the network delay plays a significant role and when the load is higher. Hence, the experimental tests demonstrate that the proposed heuristics are useful to support the design of edge-computing infrastructures for modern AI-based applications relying on data collected by geographically distributed IoT sensors.

Key words: IoT applications, Artificial Intelligence, Edge Computing, Service Placement, Variable Neighborhood Search, Genetic Algorithms.

1 Introduction

A great variety of Internet of Things (IoT) applications have been developed in the last few years and are getting more and more popularity thanks to their capability of providing society and citizens with improved services and lifestyle. Many applications, ranging from autonomous driving, healthcare, smart cities up to industrial environments, increasingly exploit IoT-based services to support decision-making systems and take advantage of data-driven services [2, 45]. IoT applications are based on the presence of geographically distributed sensors to collect heterogeneous kinds of data about the surrounding environment. Such data are typically sent to a cloud computing data center to be processed using machine learning and Artificial Intelligence (AI) algorithms [31]. However, IoT applications are characterized by very different requirements, which will be pointed out in Section 2.

For some applications, the traditional cloud-based approach may not represent the most convenient choice. For example, for bandwidth-bound applications, the increasing volume of produced data is likely to make transferring and processing data at a remote cloud data center too expensive or not convenient for network constraints. Indeed, a high network utilization may be unwanted due to costs related to the cloud pricing options, even when the network load does not lead to a performance degradation. Other IoT applications have essential requirements related to latency and response time (e.g., real-time contexts) and cannot cope with a remote cloud data center's high network delays. In these cases, an *Edge Computing* paradigm is likely to represent a preferable solution [25].

The main feature of an edge computing infrastructure is a layer of edge nodes located on the network edge, close to the sensors, to host tasks aimed at pre-processing, filtering, and aggregating the data coming from the distributed sensors. The layer of edge nodes, placed in an intermediate position between sensors and the remote cloud data center, presents a twofold advantage. First, it may reduce the data volume transferred to the cloud through pre-processing and filtering performed on the network edge. Second, the intermediate layer of edge nodes reduces latency and response times for latency-bound applications. However, the increased complexity due to the

introduction of the edge nodes layer opens novel issues concerning the infrastructure design.

The allocation of the data flows coming from sensors over the edge nodes and the choice related to the number and location of edge nodes to be activated represent critical aspects for the design and the management process of an edge computing infrastructure. While the problem of an optimized service placement has been largely investigated in terms of resource management within cloud computing data centers [3, 26], it received scarce attention in the edge computing area. This chapter focuses on this critical issue, proposing modeling the service placement based on a multi-objective optimization problem aiming at minimizing two aspects. First, the response time for the data transmission and processing along the path from sensors to edge nodes and then to the cloud data center; second, the (energy or monetary) costs related the number of turned on edge nodes. Due to the complexity related to the optimized design of an edge computing infrastructure that should cope with a non-linear objective function, we propose two approaches based on meta-heuristics, namely Variable Neighborhood Search (VNS) and Generic Algorithms (GA). We evaluate the performance of the two approaches over a range of different scenarios, considering the realistic setting of a smart city application developed in the medium-sized Italian city of Modena. Our experiments show that both heuristics represent promising approaches for designing an edge computing infrastructure, representing effective solutions for supporting IoT applications.

The rest of this chapter is organized as follows. Section 2 describes IoT applications and their classifications based on key requirements. Section 3 focuses on the literature review. Section 4 describes the main performance metrics used to evaluate the behavior of an edge computing infrastructure supporting IoT applications. In contrast, Section 5 formalizes the problem of designing and operating an edge computing infrastructure. Section 6 presents the two considered heuristics based on VNS and GA. Section 7 evaluates how the proposed algorithms can cope with the problem of designing and managing an edge computing infrastructure. Finally, Section 8 presents some concluding remarks.

2 IoT applications: classification and challenges

IoT applications have recently experienced a remarkable diffusion in very heterogeneous fields, ranging from automotive to industry, health, and smart city applications. The reasons motivating this increasing popularity of applications based on a distributed set of sensors collecting data from the environment are multiple. First of all, thanks to the enhancements of the underlying technologies, the required technical equipment (principally the IoT sensors) is becoming increasingly powerful and efficient in terms of energy consumption, reducing its size at the same time, making possible its use in many contexts before not even conceivable. Simultaneously, the capability of sensors of storing and transmitting data has increased along with the availability of wireless connectivity in many physical environments.

In this section, we initially classify IoT applications based on their field of application. Then, we characterize their essential requirements to understand the challenges from the supporting infrastructure's point of view.

2.1 A Classification by Field of Application

2.1.1 Automotive

The global automotive market around artificial intelligence is expected to grow up to $ 8,887.6 million by 2025, with a compound annual growth rate of 45% from 2018 to 2025[1]. *Autonomous driving* holds the promise of reducing traffic fatalities, reducing congestion as well as curbing our carbon footprint. According to ABI, a marketing firm, roughly 8 million (10% of global output) vehicles with self-driving capabilities of level 3 or higher will be shipped in 2025[2]. Connected cars and related city infrastructures also ensure fleet and vehicle health solutions based on collection of data coming from the vehicle, data management at the cloud level, and application of advanced analytics. Furthermore, the concept of *predictive maintenance* is progressively transforming into a practical and straightforward solution integrating vehicles sensors, hardware and software modules, and data transmitters, allowing the tracking of performance and eventual failure factors. Finally, value-added user services and applications are growing in the field of *infotainment*. A wide range of advanced services are offered by companies such as car manufacturers, insurance, service companies and infotainment providers.

2.1.2 Industry 4.0

The interconnection of machines and IoT sensors able to gather useful data to provide real-time valuable information increases productivity, quality, and worker safety. It is expected that industrial markets will capture more than 70% of IoT value - estimated to exceed $4.6 trillion by 2025[3]. For example, leading institutions and firms in Europe have proposed the Reference Architecture Model Industry 4.0 (RAMI 4.0), describing key concepts and standards for this emerging paradigm [43]. *Industrial IoT* (IIoT) and Artificial Intelligence constitute the fundamental basis for the development and success of the so-called Industry 4.0: IIoT allows to continuously collect data from several and heterogeneous sensors and devices, and to forward the collected data to cloud computing data centers ini a secure way. For these reasons, IIoT is able of supporting many Industry 4.0 applications, such as safety and security of industrial processes, quality control, inventory management, optimization of

[1] https://www.alliedmarketresearch.com/automotive-artificial-intelligence-market

[2] https://www.goldmansachs.com/insights/technology-driving-innovation/cars-2025/

[3] https://www.forbes.com/sites/louiscolumbus/2017/12/10/2017-roundup-of-internet-of-things-forecasts/

packing, logistics and supply chain, maintenance processes monitoring and effective detection of failures through the use of machine learning predictive techniques [45].

2.1.3 E-Health

The global market size related to IoT-based equipment for healthcare is projected to reach $ 534.3 billion by 2025, with at an annual growth rate of 19.9% over the period 2019-2025[4]. IoT sensors technologies have penetrated various healthcare applications, ranging from monitoring and management of chronic disease, support for home health, education for patients and professionals, up to applications for disaster management. In the field of *medicare interaction*, smart wearable devices are increasingly used to collect data about patients health status. Significant examples of data that are likely to be collected in these applications are heartbeat, glucose level and blood pressure. Such data are collected through sensors located on wearable technologies, and are then sent to smartphones to be collected and analyzed. In this way, critical information about the medical condition of the patients are collected and afterward transmitted to a remote provider to be analyzed in order to provide the required care support. This approach allows the health care provider to remotely monitor a patient in his/her own home or in a care facility, thus reducing the readmission rates. Furthermore, the interactions between individuals and a provider can be aided with live video streams for consultative, treatment and diagnostic services. Another sub-field of application is related to *health tracking*, in terms of health practice and education, that has been increasingly supported by mobile devices. E-Health IoT-based applications can range from continuous monitoring of health parameters or conditions, to targeted text messages for medicare, up to wide-scale alerts about disease outbreaks.

2.1.4 Smart Cities Applications

IoT applications are enabling Smart City initiatives worldwide. The possibility of collecting data by distributed sensors and analyzing them through artificial intelligence algorithms for predictive purpose or supporting decision making processes opened a wide range of opportunities for creating innovative services to improve the lives of their residents [2]. Among the many smart cities applications, we select the following ones as significant examples.

- *Mobility management*. Data collected from distributed sensors reveal patterns of public mobility and transportation. Mobility operators use these data to enhance the traveling experience and eventually improve safety and punctuality. Simultaneously, solutions for personal vehicles can determine the number, location, and speed of moving vehicles. Furthermore, smart traffic lights can automatically manage the lights according to real traffic conditions. Smart services for traffic

[4] https://www.reportlinker.com/p05763769/?utm_source=PRN

management may also forecast, based on historical data, traffic evolution over time in order to prevent potential congestion. Finally, smart parking solutions can predict the occupation or the availability of parking spots for creating dynamic parking maps.

- *Energy and waste management*. The sustainable management of energy and waste in our cities is one of the most important field of IoT-based applications. The data collected by smart meters connected to the network can be sent directly to a public utility for further processing and analysis. In this way, utility companies are able to bill for the exact amount of energy, water and gas consumed by each citizen. Consumption patterns of an entire city can be easily monitored too. Furthermore, waste management applications may facilitate the optimization of waste collection schedules through the accurate tracking of waste levels and analytics allowing route optimization.
- *Public safety*. Applications for public safety for smart cities involve collection, processing and transmission of an heterogeneous and ever increasing amount of video sources, ranging from home surveillance systems, traffic monitoring, up to individual videos. In general, smart technologies deployment can provide surveillance with the capability to identify different entities, ranging from human beings, objects, events and detect patterns of behavior and anomalies, either real-time or post-events, from large bodies of video and audio streams. Another application is related to Public Warning Systems to alert citizen in a certain area and communicate them to take precautions or actions. In this case, IoT devices are connected to IoT service platforms to convey information to the citizens.

2.1.5 Retail

In the retail industry, IoT and artificial intelligence is applied with the final goal to support, guide and improve the production of products for stores and businesses. Remote and accurate monitoring, and sales forecast represent the main benefits that IoT-based innovative solutions may provide to the retail business. Tracking and finalizing consumers' purchases and their product history and specific demographics provides retail consumers with heightened convenience. This trend also benefits retailers by achieving an improved understanding of their audience, thanks to the use of artificial intelligence algorithms on the collected data. A typical application in this field is related to *location-aware advertisement and navigation*. Bluetooth beacons provide shoppers with indoor and outdoor localization and information about shops nearby and help vendors focus their marketing campaigns on actual customer behavior. On the same line, IoT-led data tracking allows retail firms to get advanced metrics on the flow of people within the stores and the best selling points for individual customers. An innovative application consists of *smart mirrors*, which can be adopted within fitting rooms for suggesting other items based on what others have bought using an RFID label-scanning system. Smart mirrors are also used in conjunction with augmented reality, allowing customers to dress virtually without physically doing it.

2.1.6 Smart Agriculture

Smart agriculture, also called precision agriculture may significantly help farmers in order to maximize their productivity while minimizing costs and use of resources (e.g., water, seeds and fertilizer). The deployment of sensors-based fields maps allows the farmers to better understand their farms at a micro-scale level of detail and to reduce the related environmental impacts. The core building block for the development of such smart agriculture are represented by Internet of Things (IoT) and artificial intelligence [8]. Specifically, the role of IoT is fundamental for the automatic collection of data that are subsequently transmitted to cloud data centers for processing. On the other hand, artificial intelligence solutions typically involving artificial neural networks and clustering techniques are then applied to process data to support decision-making. A typical example is related to the identification of the more appropriate amount of water necessary to irrigate fields, that is determined through the analysis of the collected agricultural data. Specifically, an action is taken in order to provide the correct quantity of water as soon as the irrigation need is detected, that is when the field lacks water.

2.2 Challenges of IoT Applications

The previously described applications show significantly different characteristics in terms of collected data, type of required processing, and purpose of the data analysis results. The application characteristics imply very different requirements from computational and networking points of view, leading to consequent challenges for the underlying supporting infrastructure. As an example, autonomous driving capabilities require real-time data transmission with strictly controlled latency margins to be safe: a typical requirement of the so-called level 4-5, that is high-level autonomous driving, is that each status messages is delivered within 10 ms at maximum to enable appropriate and safe vehicle reactions [38]. On the other hand, medicare applications offering video streaming are more likely to be constrained by bandwidth availability. The near coming transition between closed-world, independent applications and general-purpose infrastructures capable of supporting diverse applications without intervention requires hardware and software tools capable of finding adaptive trade-offs among different requirements. In the following, we introduce a taxonomy based on four categories, including the majority of IoT applications. Table 1 summarizes the results.

- **Latency-bound**: applications with strong requirements in terms of response times, as in real-time contexts. Examples of applications: mobility management, public surveillance, autonomous driving, and location-aware advertisement and navigation belong to this category.
- **CPU-bound**: applications characterized by computationally heavy tasks with high processing times where the CPU of the processing nodes is likely to become

a bottleneck. Examples of applications: public safety, autonomous driving, infotainment systems, smart mirrors, and medicare applications are in this category.
- **Bandwidth-bound**: applications characterized by a significant amount of data to be transferred, where the network bandwidth is an essential requirement for delivering services. Examples of applications: public safety, autonomous driving, infotainment, smart mirrors, and remote medicare interaction.
- **Reliability-bound**: applications where the reliability - intended as integrity, security, privacy, and availability - of data is a crucial requirement. Examples of applications: autonomous driving and e-health applications.
- **Stateful**: applications need to access past information for processing incoming data, hence data flows cannot be distributed over multiple nodes. Typically, applications in which aggregated views of the data are needed. Examples are mobility management, predictive maintenance, and smart mirrors.

Table 1 IoT applications essential requirements.

Application	Latency	CPU	Bandwidth	Reliability	Statefulness
Autonomous driving	✓	✓	✓	✓	-
Predictive maintenance	-	-	-	-	✓
Infotainment	-	✓	✓	-	-
Industrial IoT	-	-	-	-	✓
Medicare interaction	-	✓	✓	✓	-
Health tracking	-	✓	-	✓	-
Mobility management	✓	-	-	-	✓
Utility/Waste	-	-	-	-	-
Public safety	✓	✓	✓	-	-
Location-aware Ads/Nav	✓	-	-	-	-
Smart Mirrors	-	✓	✓	-	✓
Smart agriculture	-	-	-	-	✓

3 Literature Review

As discussed in the previous section, an edge computing paradigm may efficiently handle data volume to be processed in IoT applications. In general, these applications require low latency and highly scalable services, motivating edge-based solutions in substituting the standard cloud computing paradigm. There is a vast literature showing the benefits of the edge computing infrastructure, especially when there is a

large volume of data coming from geographically distributed devices [33, 35, 42, 44]. One considerable advantage of this architecture is decentralization, which allows pre-process data (for example, filtering and aggregation processing) before they reach the cloud data centers [19, 35, 39].

A survey on edge computing infrastructures was given in [39], who commented on applications and their challenges. Regarding IoT services, Wen *et al.* [35] discussed concepts and issues that may appear in complex scenarios, as, e.g., smart cities and marine monitoring. These authors handled a generic scenario by using genetic algorithms. Recently, a survey on the integration of edge computing with IoT was given by [28]. The authors discussed many challenges related to the heterogeneity, complexity, and dynamics of applications that need to be taken into account to deliver precise and reliable services. The authors also gave some insights into resiliency and the modeling by game theory. These studies foster the smart city application we are investigating in terms of cost reduction, load balancing, and latency reduction.

In general, the project of edge computing infrastructures is concerned with allocating services over the infrastructure. This was the case of [15], who handled issues related to the power consumption and transmission delay in the edge nodes to cloud data centers communication. On the other hand, the work of [40] modeled the load sharing issues in an edge to edge communication. In the context of industrial applications, Foukalas [18] presented a distributed intelligent IoT platform. The author used the cognitive IoT concept, with machine learning classifiers having an overall knowledge of the application, for monitoring and control purposes. Experiments were conducted on a scenario of predictive maintenance in smart factories, demonstrating the effectiveness of the proposed approach. Recently, Caiza *et al.* [9] discussed the main issues related to architecture, security, latency, and energy consumption in the context of IIoT. The authors commented on the vital role that edge computing plays to deal with these issues, which certainly include the fast processing and real-time storage of large volumes of data. Although our application may carry some similarities with these studies, our initial assumption considers a long-range communication network where multi-hop links exist between sensors. Then, each edge node can serve any sensor.

Our study considers an application to support smart city services and other studies also support this kind of application. For example, Tang *et al.* [33] proposed a system structured on four layers to handle it. Wang *et al.* [34] examined the coupling resource management problem, which is related to the multiple requisitions a sensor node is required to attend and then may result in failures of services. With the aim of an edge computing paradigm, the authors could efficiently introduce buffer and controller operations to obtain a sustainable system. In the survey of Zahmatkesh and Al-Turjman [42], the benefits of buffer/caching are discussed when edge computing is used in IoT communications. The authors investigated caching techniques using artificial intelligence and machine learning approaches, showing their potential and future challenges. Concerning the smart traffic monitoring, Dhingra *et al.* [16] used edge computing in a framework for congestion monitoring and traffic light management. The final results demonstrated a reduction in response time and an increase in the bandwidth. The proposed approach is also flexible, because it does not impose a

fixed number of layers and let the number of edge nodes be determined according to the application's requirements.

Determining the number of edge nodes turned on and which sensors each of these nodes will serve is a typical facility location problem. This problem has been extensively explored in the operational research literature, proposing different algorithms to handle it. This problem aims at determining the number and where to open facilities (location decisions), besides which customers each open facility will serve (allocation decisions). The objective is to minimize the location and allocation decisions while satisfying the customers' demand [14].

Facility location problems were surveyed in [17, 22, 1, 13]. Farahani *et al.* [17] focused their review on multi-criteria problems, including bi-objective, multi-objective, and multi-attribute problems. They also commented on solutions methods. In the review of [22], the problems were classified according to the use of aggregation rules to handle situations with a large number of customers. The authors described aggregation error measurements, besides commenting on conditional location problems. On the other hand, Ahmadi-Javid *et al.* [1] focused their review on healthcare applications, presenting a framework to classify the problems following the emergency type and location management. The recent survey of Turkoglu and Genevois [13] considers service facility location problems, with a detailed classification of them based on purpose, space, distance, time, parameters, capacity, facilities, objectives, competition, application field, solution method, type of experiment, and other features.

Regarding IoT applications, Klinkowski *et al.* [24] handled the problem of data center locations with light-path provisioning in elastic optical networks. Silva and Fonseca [32] considered the optimization of human mobility using an edge computing infrastructure. The objective is to reduce the processing in the cloud data centers and the latency of users. Both [24] and [32] started dealing with mathematical models, but due to their inefficiency, they derived heuristic approaches. Recently, Canali and Lancellotti [10, 11] considered the optimization of a service placement problem with the presence of edge infrastructures. In [10], these authors outlined the problem, while in [11], they presented a GA to examine a smart city application. On the other hand, the GA-based heuristic proposed in this chapter is a clear step ahead concerning these studies, because we introduce the possibility of locating edge nodes to reduce cost and power consumption, besides respecting a service level agreement.

4 Performance modeling in edge computing infrastructures

In this section, we present the main performance metrics that can be used to evaluate an Edge computing infrastructure's behavior supporting IoT applications. We can distinguish two classes of metrics: efficiency metrics, related to the utilization of resources needed to operate, and performance metrics that concern the ability to operate effectively.

4.1 Efficiency metrics

An Edge computing infrastructure needs resources to provide its services. Resources range from hardware components to energy to economic costs related to maintenance and administration. In the following of this analysis, we will refer in general to costs because all the resources needed to support IoT applications can be easily mapped into economic costs. Costs can occur both at deployment time or when running IoT applications on the infrastructure. An example of deployment-time is the purchase of the hardware for the infrastructure and its installation. We usually refer to these costs as *Capital Expenditures* (CAPEX). Examples of costs related to the operation of the infrastructure are the energy cost and maintenance. We refer to them as to *Operating Expenditures* (OPEX).

Another way to classify the costs related to the edge computing infrastructure is to consider if the cost is global or related to its infrastructure. For example, costs related to setting up a control center for the edge computing infrastructure are not strongly related to the size of the infrastructure. On the other hand, the cost of the purchase and installation of edge nodes is directly correlated with the number of nodes to install. Similarly, energy costs are directly related to the number of active edge nodes.

The costs occur both when the infrastructure is deployed (usually, in this case, we consider the CAPEX related to the edge nodes that should be installed) and when the infrastructure operates (in this case, the focus is on OPEX such as energy costs for the nodes). It is worth noting that, when the load is subject to variations over time, the infrastructure can be designed to cope with the peak load; however, during off-peak hours, a fraction of the nodes can be turned off to reduce energy costs.

4.2 Performance metrics

Another class of metrics is related to the performance experienced by the IoT application. These metrics typically can be related to the response time when data flows are processed in the edge infrastructure or can concern the infrastructure's availability.

Among response-time related performance metrics, the most common approach is to consider the average response time in a stationary scenario (when there is no transient effect due to sudden variations of the incoming load). An alternative metric is to consider a different estimation of the response time, typically taking into account 90-percentile or 95-percentile, representing a worst-case scenario excluding pathologically long request processing, for example, due to timeouts of transmission errors.

Availability-related metrics can take into account metrics such as service availability over a while (typical values range from 95% availability up to the 5-nines availability for critical services that must be online for 99.999% of the considered period). Another metric to measure a service's ability to process requests is con-

sidering the number of requests processed and the number of requests dropped, for example, due to the infrastructure's failures or due to overload in the edge nodes. To this aim, we define the *Throughput* as the number of requests processed in a considered period and the *Goodput* as the number of requests correctly processed in the same time frame. The difference between Throughput and Goodput measures the requests not processed (that is dropped) due to overload or other infrastructure problems. From this metric, we can infer the request drop rate, which is the percentage of requests that cannot be satisfied.

All the considered performance metrics can be used to define some *Service Level Agreement* (SLA) that defines the contract between the provider of the service and its users. An SLA can define a maximum acceptable response time (defined as a limit on the average or on some percentile of the response time), a minimum service availability that must be guaranteed, or a minimum drop rate. Failing to respect an SLA for the service provider usually results in a penalty that reduces the revenues.

5 Problem Formulation

We now formalize the problem of designing and operating an edge computing infrastructure supporting an IoT application. We assume the problem to be both CPU-bound and network-bound, where the network concerns may be either due to network latency or bandwidth issues. Furthermore, we assume our IoT application is a stateful application, where all data from an IoT device (sensor) must be sent to the same edge node for processing. We use as the basis of our model a facility location problem. We aim at (1) identifying a subset of potential edge nodes that should serve requests from IoT devices (referred to as sensors) and (2) mapping data flows from sensors to edge nodes and from edge nodes to cloud data centers to reduce cost and optimize performance.

If the problem concerns the infrastructure (infrastructure deployment) design phase, we consider a set of potential locations where edge nodes can be installed. We know an expected incoming load. Based on a performance model, we determine how many locations should be selected to host an edge node to satisfy SLA requirements while minimizing the infrastructure CAPEX. Similarly, during the edge infrastructure operations (infrastructure management), we can solve the same problem with different input data. In this case, we have a set of edge nodes, and we want to decide the minimum set of nodes to turn on to satisfy the SLA while minimizing the OPEX.

5.1 Model parameters

To provide a model for the problem, we assume a stationary scenario. Let $\mathcal{S}$ be a set of geographically distributed IoT devices (sensors) producing data at a constant rate. The generic sensor i produces data at a rate λ_i. An intermediate layer of edge nodes

will process the data from the sensors. The edge nodes can perform operations such as filtering, aggregation, or anomaly detection with low latency. In our problem, we consider a set $\mathcal{N}$ of potential edge nodes locations. A decision variable E_j is used to decide if edge node j is to be used. If we consider the problem of infrastructure deployment, E_j is used to decide if a potential edge node is to be actually deployed. In the problem of infrastructure management, the variable E_j is used to decide if the edge node is to be turned on or off, based on present load conditions. Each edge node j can process data at a rate μ_j and is associated with a cost c_j that is either a representation of OPEX or CAPEX, depending on the scenario in which the problem is applied. Furthermore, the transmission of data between the sensors and the edge nodes occurs with a delay δ_{ij}, with i being the sensor and j the edge node.

Finally, we consider the third layer of our architecture, that consists in set of cloud data centers C. When data is sent from an edge node j to a cloud data center k, it incurs in a delay δ_{jk}.

To considered problem uses the following binary decision variables.

- E_j, to define if a (potential) edge node located at position j is available to process data from sensors.
- x_{ij}, to define if sensor i is sending data to edge node j
- y_{jk}, to define if edge node j is sending data to cloud data center k;

The main symbols of the model are summarized in Table 2.

Table 2 Summary of notations used in the proposed model.

Parameters of the model	
$\mathcal{S}$	Sensors set
$\mathcal{N}$	Edge nodes set
C	Cloud data centers set
λ_i	Outgoing data rate from sensor i
λ_j	Incoming data rate at edge node j; $\lambda_j = \sum_{i \in \mathcal{S}} x_{ij}\lambda_i$
$1/\mu_j$	Processing time at edge node j
c_j	Cost for deploying a potential edge node j (or for keeping an edge node powered on)
δ_{ij}	Communication delay from sensor i to edge node j
δ_{jk}	Communication delay from edge j to cloud k
Indices used in notation	
i	Index of a sensor
j	Index of an edge node
k	Index of a cloud data center
Decision variables	
E_j	Enabling of edge node j
x_{ij}	Communication occurs between sensor i and edge node j
y_{jk}	Communication occurs between edge node j and cloud data center k

5.2 Objective functions and SLA

The considered problem is based on the problem in [10] and takes into account two different performance metrics. The first performance metric is related to the infrastructure cost that we aim to minimize. Ideally, the lower is the number of edge nodes that we use, the lower is the global cost. The second performance metric is related to the average response time, and it plays a double role in our problem formulation. On the one hand, we consider an SLA on the average response time to guarantee that the response time remains below a given value. On the other hand, as long as the infrastructure's cost remains the same, we prefer solutions that are characterized by a lower average response time.

Concerning the infrastructure cost, we can define the total cost as:

$$C = \sum_{j \in \mathcal{N}} c_j E_j \tag{1}$$

For the second performance metric, we should remember that response time is the sum of three contributions: T_{netSE} that is the network delay related to the sensor-to-edge latency, T_{netEC} that is the delay related to the edge-to-cloud latency, and T_{proc} that is the time due to data processing on the edge nodes.

The network delay components can be modeled as follows:

$$T_{netSE} = \frac{1}{\sum_{i \in \mathcal{S}} \lambda_i} \sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{N}} \lambda_i x_{ij} \delta_{ij} \tag{2}$$

$$T_{netEC} = \frac{1}{\sum_{j \in \mathcal{N}} \lambda_j} \sum_{j \in \mathcal{N}} \sum_{k \in C} \lambda_j y_{jk} \delta_{jk} \tag{3}$$

For the T_{netSE} definition in Eq. (2), we weight the delays δ_{ij} between each sensor i and each edge node j by the amount of traffic passing through that link, which is $\lambda_{ij} x_{ij}$. The sum is then normalized dividing by the total traffic $\sum_i \lambda_i$. In a similar way, we define T_{netEC} in Eq. (3). The main difference is the use of the term λ_j to refer to the load incoming into each edge node j and forwarded from the edge node to the cloud layer. We can define λ_j as:

$$\lambda_j = \sum_{i \in \mathcal{S}} x_{ij} \lambda_i, \quad \forall j \in \mathcal{N} \tag{4}$$

The component concerning the processing time T_{proc} is modeled using the queuing theory considering an M/G/1 system. According to the PASTA theorem [36] and to the Pollaczek-Kinchine formula used to describe the M/G/1 average response time [21], we have:

$$T_{proc} = \frac{1}{\sum_{j \in \mathcal{N}} \lambda_j} \sum_{j \in \mathcal{N}} \lambda_j \left(\frac{1}{\mu_j} + \frac{\lambda_j \mu_j V_j^2}{2(\mu_j - \lambda_j)} \right) \tag{5}$$

Where V_j^2 is the variance of the service time process. Assuming a variance of the service time close to the one of the inter-arrival time ($2/\mu_j^2$), as in other example sin literature [4, 11], we can simplify Eq. (5) in

$$T_{proc} = \frac{1}{\sum_{j \in \mathcal{N}} \lambda_j} \sum_{j \in \mathcal{N}} \lambda_j \frac{1}{\mu_j - \lambda_j} \tag{6}$$

It is worth mentioning that we do not consider the cloud layer's details (such as the computation time at the cloud data center level) in the model of our problem. Indeed, this aspect is not meaningful for optimizing the edge infrastructure.

This model for the processing time is used also to describe the SLA for the edge computing infrastructure. Specifically, we expect the average response time to stay below T_{SLA}:

$$T_{SLA} = K\frac{1}{\overline{\mu}} + \overline{\delta}_{SF} + \overline{\delta}_{FC} \tag{7}$$

Where K is constant value (in cloud systems it is common to consider K=10 [4]); $1/\overline{\mu}$ is the average service time in an edge node; $\overline{\delta}_{SE}$ and $\overline{\delta}_{EC}$ are the average sensor-to-edge and edge-to-cloud delays, respectively.

5.3 Optimization problem

We can define the model for the edge computing infrastructure problem supporting an IoT application as follows.

Minimize:

$$C = \sum_{j \in \mathcal{N}} c_j E_j \tag{8}$$

$$T_R = T_{netSE} + T_{netEC} + T_{proc} \tag{9}$$

Subject to:

$$T_R \leq T_{SLA} \tag{10}$$

$$\lambda_j < E_j \mu_j, \quad \forall j \in \mathcal{N} \tag{11}$$

$$\sum_{j \in \mathcal{N}} x_{ij} = 1, \qquad \forall i \in \mathcal{S} \tag{12}$$

$$\sum_{k \in C} y_{jk} = E_j, \qquad \forall j \in \mathcal{N} \tag{13}$$

$$E_j \in \{0, 1\}, \quad \forall j \in \mathcal{N} \tag{14}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i \in \mathcal{S}, j \in \mathcal{N} \tag{15}$$

$$y_{jk} \in \{0, 1\}, \quad \forall j \in \mathcal{N}, k \in C \tag{16}$$

The two objective functions, (8) and (9), are related, respectively, to the minimization of: **cost**, which depends on the number of used edge nodes; and **response time**, which is the delay in sensor-edge-cloud data transit expressed through the function introduced in the previous section. The response time objective is subordinated to the cost one, meaning that we aim to minimize (9) as long as the improvement for this objective function does not affect (8).

The model includes the following constraints. Constraint (10) places a limit for the average response time: this value must not violate the *Service Level Agreement* (SLA). Constraints (11) guarantee that no overload occurs on the edge nodes. Hence, the incoming data rate on every node must be lower than the processing rate. For a node that is powered down, no processing occurs. Constraints (12) ensure that exactly one edge node will process the data of each sensor. In a similar way constraints (13) guarantee that exactly one cloud data center receives the data for each edge node. Finally, constraints (14), (15) and (16) describe the binary nature of the decision variables.

6 Heuristics

Several options are available when we aim to solve a problem using a heuristic algorithm. On the one hand, greedy heuristics are usually quite fast. On the other hand, the performance of a greedy algorithms depends on the nature of the problem. Local minima and a non-convex domain, which may hinder their ability to find a solution for the problem. In designing an edge computing platform, the objective function is non-linear. Furthermore, the feasibility domain of the problem is not guaranteed to be convex. For this reason, we explore solutions that differ from the classical greedy approach.

Due to the number of edge nodes and sensors, the problem is characterized by a high dimensionality that may hinder the performance of branch and bound approaches, due to the large solution space to explore.

For these reasons we focus on meta-heuristics that provide a flexible approach to the problem. The ability of these meta-heuristics to solve successfully a broad and heterogeneous set of problems is known in literature [7]. In particular, we consider two approaches, namely, Variable Neighborhood Search (VNS) and Generic Algorithms.

6.1 Variable Neighborhood Search

The *Variable Neighborhood Search* (VNS) is a meta-heuristic used to solve hard nonlinear and combinatorial optimization problems. Some examples include its application to vehicle routing [37], portfolio selection [6], cutting and packing [30], scheduling [29], among others.

VNS is a single solution-based method, where a solution passes through a shaking and local search phases to become globally optimal concerning all neighborhood structures. These structures represent how to explore the current solution's neighborhood towards new solutions further investigated in the local search phase. When an improved solution is found, VNS comes back to the first neighborhood, or else it continues to the next neighborhood [27, 20].

The VNS iterates through the following general steps, given an initial solution: shaking phase, to obtain a new solution by applying a neighborhood structure; local search phase, to improve the new neighbor solution; and, an acceptance phase, to change the current solution by the one from the previous phase. Once the current solution is changed, the search restarts from the first neighborhood structure; otherwise, it considers the next structure. We present Algorithm 1 with the implemented VNS for the edge computing infrastructure problem. During the search for a solution, we do not accept solutions that violate constraints (10) to (13).

Algorithm 1: VNS FOR THE LOCATION-ALLOCATION PROBLEM.

```
1  x ← an initial solution generated by a random constructive heuristic;
2  while the stopping criteria are not reached do
3  |   k ← 1;
4  |   while k ≤ K_max do
5  |   |   x′ ← random solution in the neighborhood structure N_k(x);
6  |   |   x″ ← apply the local search on x′;
7  |   |   if f(x″) < f(x) then
8  |   |   |   x ← x″;
9  |   |   |   k ← 1;
10 |   |   end
11 |   |   else k ← k + 1;
12 |   end
13 end
14 return x;
```

In Algorithm 1, we code a solution as a matrix of integers. Each matrix's line represents an edge node and contains the sensors it serves. The last cell of each matrix's line keeps the cloud data center that serves the edge node. When there is no sensor in a matrix's line, then that edge node is off. Notice that each sensor will appear in exactly one matrix's line. In the initial solution, we select the closest cloud data center for each edge node to serve it. Similarly, each sensor is served by the closest edge node. In the case an edge node has reached the T_{SLA}, then no other sensor can be served by it. In the latter, the sensor is served by its second-closest edge node, and so on until the edge nodes serve all sensors.

The inner loop of lines 4-12 in Algorithm 1 ends when all neighborhood structures are visited for the current solution. It means this solution is globally optimal with regards to all these structures. In line 5, a new solution is generated randomly on the current solution's neighborhood (i.e., the shaking phase). Next, we apply the local search on this new solution, attempting to improve it. The local search is based on

trying all possible allocations of sensors in edge nodes and swaps of sensors in edge nodes. Algorithm 2 describes the local search phase.

A solution has two objectives: (i) the cost associated with the number of edge nodes on; and (ii) the delay in sensor-edge-cloud transit of data. We assume the first objective is used to guide the VNS. It means a given solution is better than another if its first objective is smaller, or if their first objectives are equal but its second is smaller. Notice that if the current solution is improved, the search restarts from the first neighborhood structure. Besides that, our VNS considers $K_{max} = 5$ neighborhood structures based on swap and move operations. In particular:

- N_1: select (randomly) an edge node n_1, the farthest sensor s_1 which is served by n_1, the edge node n_2 that is the closest to s_1, and the sensor s_2 which is served by n_2 that is the closest to n_1. Now, let s_1 be served by n_2 and s_2 by n_1.
- N_2: let $\mathcal{N}_{on}$ be the set of edge nodes on. Let $r_j = \lambda_j/\mu_j$ be the load of each edge node j in this set. Calculate the average load of the edge nodes on as $\bar{r} = \left(\sum_{j\in\mathcal{N}_{on}} r_j\right)/|\mathcal{N}|$. Then, select (randomly) $n_1 \in \mathcal{N}_{on}$ whose load $r_1 > \bar{r}$. If one exists, select the farthest sensor s_1 which is served by n_1. Next, select the edge node $n_2 \in \mathcal{N}_{on}$ with the lowest load r_2 and closest to s_1. Now, let s_1 be served by n_2.
- N_3: let $\mathcal{N}_{on}$ be the set of edge nodes on. Select (randomly) an edge node n_1 from this set. Then, compute the average load with all sensors and edge nodes on, except n_1, as $\tilde{r} = \left(\sum_{i\in\mathcal{S}} \lambda_i\right)/\left(\sum_{j\in\mathcal{N}_{on}\setminus\{n_1\}} \mu_j\right)$. If $\tilde{r} < 1$, then for each sensor s_1 which is served by n_1, let s_1 be served by the closest edge node in $\mathcal{N}_{on}\setminus\{n_1\}$.
- N_4: let $\mathcal{N}_{on}$ and $\mathcal{N}_{off}$ be the sets of edge nodes on and off, respectively. If $\mathcal{N}_{off}$ is not empty, select (randomly) an edge node n_1 from this set. On the other hand, select the edge node $n_2 \in \mathcal{N}_{on}$ whose average response time is the highest one. Now, let all sensors of n_2 to be served by n_1.
- N_5: select (randomly) an edge node on and let it be served by the closest cloud data center.

Algorithm 2: LOCAL SEARCH PHASE.

```
x ← an input solution;
k ← 1;
while k ≤ L_max do
    x' ← the best solution in the neighborhood structure M_k(x);
    if f(x') < f(x) then
        x ← x';
        k ← 1;
    end
    else k ← k + 1;
end
return x;
```

The local search phase is presented in Algorithm 2. Given an input solution, it iterates through $L_{max} = 2$ neighborhood structures. At each structure (line 4), the best possible solution in the current solution's neighborhood is chosen. This means that all possible movements defined by the neighborhood structure are tried, so the one whose neighbor solution has the best improvement is considered. The search restarts from the first structure if the current solution is improved. The two neighborhood structures M_k are defined as:

- M_1: for each pair of edges nodes n_1 and n_2, and sensor s_1 which is served by n_1, now let s_1 be served by n_2. Among all these movements/possibilities, choose the one that most reduce the solution cost.
- M_2: for each pair of edges nodes n_1 and n_2, and sensors s_1 which is served by n_1 and s_2 which is served by n_2, now let s_1 be served by n_2 and s_2 by n_1. Among all these movements/possibilities, choose the one that most reduce the solution cost.

6.2 Genetic Algorithm

We now discuss an alternative heuristic for handling the optimization problem based on genetic algorithms (GAs). Evolutionary programming has been used in literature to solve similar problems, like allocating VMs in a cloud data center [41].

In GAs, we model a possible solution as a *population* composed by *individuals*. For each individual, we encode the solution of the problem in the form of a *chromosome*. A generic individual i is represented through its chromosome C^i, which is a sequence of *genes* with a fixed length. We can write that $C^i = \{c^i_j\}$ with each gene representing a parameter characterizing that individual's solution.

The algorithm starts with an initial, randomly-generated, population of individuals. The objective function for the optimization problem plays the role of a *fitness function* for the GA. Such function is applied to every individual, to assign a *fitness score* to each chromosome. After this initialization, the population evolves through some *generations*. At each generation, the fitness score of each chromosome is updated. The evolution is carried out applying the following operators to the population:

- **Selection** is an operator that decides if an individual should be preserved in the passage from the K^{th} generation to the next. The fitness score of each individual is used to discard individuals with undesirable characteristics.
- **Mutation** randomly alters a gene in an individual (more complex mutations may involve more than one gene, for example swapping their values). In GAs, the role of mutation is to add new genetic material, allowing exploring new areas within the solution space.
- **Crossover** combines two separate individuals creating two new offspring individuals. The offspring is created by exchanging a fraction of the chromosomes of the parents. In GAs, crossover aims to spread positive combinations of genes through the population.

These operations are combined to define an evolutionary strategy. In particular, we adopt the strategy described in [5] as *Simple Strategy*. In every generation, the selection operator is applied to select (and possibly replicate) the fittest individuals. Unfit individuals are likely to be pruned from the genetic pool. After the selection, the new population undergoes the application of the crossover and mutation operators. For each individual, mutation and crossover occurs with a probability defined as P_{mut} and P_{cx}, respectively. For crossover, the offspring (two individuals for each crossover) replaces the parents. In a similar way, mutated individuals replace the originals.

In the following, we discuss the details of the GA applied to our problem. We discuss the chromosome structure and the specific mutation and crossover operators designed for the considered problem.

6.2.1 Problem definition

If we consider the problem statement, we observe that the decision variables y_{jk} for mapping edge nodes to cloud data centers define a subproblem that can be easily solved by mapping each edge node to the nearest cloud data center. For this reason, we can discard from the genetic algorithm implementation this part of the problem, and we can consider y_{ij} as another problem parameter rather than a decision variable. This reduces the solution space to explore and accelerate the algorithm convergence.

Another design choice aims to simplify the management of the double objective function that characterize our problem. Rather than embedding the ability to change the number of active edge nodes, we relay on constraint (10) to infer this value. We observe that the definition of T_{SLA} in (7) presents three components related to T_{proc}, T_{netSE} and T_{netEC}. We focus in the first part and we assume to have an homogeneous population of nodes where $\mu_j = \mu \; \forall j \in \mathcal{N}$. Furthermore, we assume that the sensors' data rate are the same for every sensor, that is $\lambda_i = \lambda \; \forall j \in \mathcal{S}$. Finally, since we are considering the ideal case that is a lower bound on the processing time, we can impose a perfect load balancing among the edge nodes. We can thus write that:

$$T_{proc} = \frac{1}{NE \cdot \mu - |\mathcal{S}| \cdot \lambda} \leq \frac{K}{\mu} \tag{17}$$

Where $NE = \sum_i E_i$ is the number of active edge nodes. From this we can define the minimum number of edge nodes as:

$$NE = \left\lceil |\mathcal{S}| \cdot \frac{\lambda}{\mu} \cdot \frac{K+1}{K} \right\rceil \tag{18}$$

To minimize the first objective function, we start the algorithm with some nodes equal to NE and, if no feasible solution is found, we increase NE, and re-iterate the genetic algorithm. This approach ensures that the objective function (8) is minimized so that the genetic algorithm can focus on the second objective function (9). Indeed

we can use the definition of T_R in equation (9) as the fitness function in our genetic algorithm, with the note that lower response time means better fitness.

6.2.2 Chromosome encoding

We now consider how to encode a solution in a chromosome. We must embed in the gene sequence two types of information:

- Which edge nodes are active (that is on or selected for deployment) among a set of potential ones. This information represents the E_i decision variables;
- How to map sensors and cloud data centers over the available edge nodes (that would represent the x_{ij} decision variables)

To this aim, we divide the chromosome into two parts.

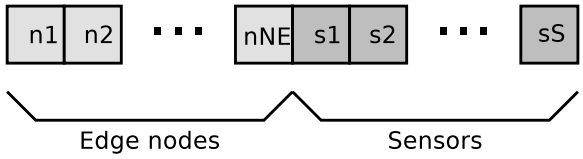

Fig. 1 Solution modeling as a chromosome

The first part contains NE genes $\{n_1, \ldots, n_{NE}\}$ that list the active edge nodes; each generic gene n_i has a value that ranges from 1 to $|\mathcal{N}|$. In this first part of the chromosome, each gene must have a different value from the other, that is $\forall i, j \in [1, NE]$ we have that $n_i = n_j \iff i = j$.

The second part of the chromosome contains $S = |\mathcal{S}|$ genes $\{s_1, \ldots, s_S\}$. Each gene contains a value $\in [1, NE]$ to map the sensors to the list of active edge nodes to data exchange. For the generic sensor $i \in [1, S]$, we have a value e, meaning that sensor i will transmit data to edge node n_e.

The final chromosome will be a concatenation of the two previously described parts: $\{n_1, \ldots, n_{NE}, s_1, \ldots, s_S\}$

6.2.3 Genetic operators

Given our problem's nature, we have to adapt to the typical genetic operators. First of all, the initialization process must ensure that the basic rules of chromosome encoding are respected. To this aim, the first NE genes must contain no duplicates, and each part of the chromosome must have values in the correct range (that is $[1, |\mathcal{N}|]$ for the first NE genes, $[1, NE]$ for the other S genes).

We can use a standard operator for the selection operator because all the problem-dependent information is embedded in the fitness function. Specifically, we rely on

the tournament selection operator. The operator picks K elements randomly within the population and returns the one with the best fitness score.

The mutation operator is implemented as a variation of the uniform mutation operator. Each gene is mutated with a given probability. Again, the mutation, like the initialization, must ensure that the resulting chromosome is feasible for the selected problem encoding.

The crossover operates separately on the two parts of the chromosome. First, we select random parts of the first NE genes, making sure that no duplicates are introduced. Next, we operate on the second part of the chromosome. The particular case to handle is when an edge node n is used in the parent node by a sensor s, but the same node is no longer available in the offspring. In this case, we look among the edge nodes available in the offspring, the node n' that is the closest (from a delay point of view) to n. For each sensor s in the offspring that would refer to n, we use the index of n' to create a feasible chromosome that can inherit some positive feature from the parent.

7 Experimental Evaluation

We now evaluate how the proposed algorithms can cope with designing and managing an edge computing infrastructure. Specifically, we first describe the experimental setup, and then we proceed with a performance comparison between the VNS and GA.

7.1 Experimental scenario

We refer in our analysis to a realistic IoT application project supported by an edge computing infrastructure. In particular, we consider a smart city project developed into the medium-sized Italian city of Modena (closer to 185.000 inhabitants). The IoT application aims to monitor car, bicycle, and pedestrian traffic using a geographically distributed set of sensors that collect information on the traffic proximity sensors. Depending on the setup, the sensors can also capture low-resolution images. For the considered application, sensors must be placed in the city's main streets. In particular, we identify the sensors' location by mapping the street names using the Open Street Map APIs[5], using features developed as part of the PAFFI framework [12]. The sensors send the collected data to the edge nodes, which perform pre-processing tasks using AI techniques. In particular, edge nodes can filter and aggregate the proximity sensor readings and, if available, can analyze images from the camera using neural networks to detect cars, bicycles, and pedestrians. For the edge nodes' potential locations, we have a list of buildings belonging to the municipality that

[5] https://wiki.openstreetmap.org/wiki/API_v0.6

could be used to host these nodes. Finally, all the pre-processed data are then sent to a cloud data center hosted on the municipality data center.

In the sensor prototypes used in this analysis, communication relies on long-range wireless connectivity boards, such as LoRaWAN[6] or IEEE 802.11ah/802.11af [23]. Due to the long-range of these communication technologies, each sensor can potentially communicate with every edge node. On the other hand, the available bandwidth decreases with the sensor-to-edge distance. Consequently, according to other studies in literature [10, 11], the communication delay between sensors and edge nodes is inversely proportional to the physical distance among the communicating entities.

Throughout our experimental evaluation we focus on a scenario where $|\mathcal{S} = 100|$, $|\mathcal{N} = 10|$, and $|C| = 1$. Other experiments with different setup confirm the main findings of our study and are not reported. Instead, we consider a broader analysis of our system's workload scenario to explore the ability of the considered algorithms to adapt to different conditions. In particular, each scenario is defined by three parameters. The first parameter, common to all configurations, is the *sensor data rate* λ, which is the same for all sensors. We consider a preliminary prototype of the smart city applications for traffic monitoring, where each sensor provides a reading every 10 seconds, meaning that $\lambda_i = \lambda = 0.1$, $\forall i \in \mathcal{S}$. This setting is suitable when designing an architecture aiming to support a stationary scenario, for example, in the first design phase of the smart city project. An alternative approach (more focused on a dynamic infrastructure management problem and not considered in our experiments) would consider a variable data rate.

The second parameter is the *average utilization* of the system ρ defined as $\frac{\sum_{i \in \mathcal{S}} \lambda_i}{\sum_{j \in \mathcal{N}} \mu_j}$. Since λ is already defined, variations in the system utilization depend mainly on the edge nodes' processing rate μ. Indeed, several computing apparatuses (typically in the form of an embedded board with processing elements, memory, storage, and communications support commonly used in IoT deployments) are available for this task. The CPU frequency, the presence of multiple cores (typically form 1 to 8), and even the support for dynamic voltage and frequency scaling may affect the μ parameter of each node. For the ρ parameter, we consider a wide range of values: $\rho \in \{0.1, 0.2, 0.5, 0.8, 0.9\}$. The last parameter is $\delta\mu$, which is the ratio between average network delay δ, and the processing time $1/\mu$. The parameter can be used to define the *CPU-bound or network-bound nature* of the scenario. For the $\delta\mu$ parameter, we consider values ranging orders of magnitude with $\delta\mu \in \{0.01, 0.1, 1, 10\}$. Our analysis ranges from CPU-bound scenarios (e.g., when $\delta\mu = 0.01$), where computing time is significantly higher than transmission time, to network-bound cases (e.g., when $\delta\mu = 10$) where data transmission dominates the response time. In our setup, this variability of the CPU to network weight refers mainly to the presence and resolution of images transferred over low-bandwidth links.

The goal of our analysis is to deploy the minimum infrastructure that ensures the respect of SLA as defined in Eq. (7). In our analysis, the constant K is set to 10. To define which nodes are to be deployed, we assume the cost c_j of an edge node at position j is equal to 1, for all $j \in \mathcal{N}$. This assumption is consistent with

[6] https://lora-alliance.org/

the observation that all the potential locations for edge nodes already belongs to the municipality. Our problem is thus reduced to minimizing the number of edge nodes used. For the experimental comparison, we evaluate the two considered alternatives described in Sec. 6:

- *Variable Neighborhood Search (VNS)*: described in Subsection 6.1;
- *Genetic Algorithm (GA)*: described in Subsection 6.2

For the VNS, we run it for 300 seconds or 3000 iterations (the first to reach stops the VNS), while the Genetic algorithm is limited to 300 generations with a population of 200 individuals.

7.2 Experimental results

We now discuss the most significant results of our performance analysis. In particular, we summarize in Tab. 3 the scenario parameters and the performance of the considered algorithms. The first two columns contain the scenario-defining parameters ρ and $\delta\mu$. Next, we present the critical value T_{SLA} used to define the maximum acceptable response time. It is worth noting that T_{SLA} changes as a function of both ρ and $\delta\mu$: the dependency from $\delta\mu$ is clearly evident from Eq. (7). The dependency from ρ is because we consider λ as constant, and $1/\mu$ changes as a function of ρ. Finally, we show the values of the two objective functions for the two algorithms: VNS and GA. The first (obj_1) is the number of edge nodes used, while the second (obj_2) is the average response time.

A discussion of the results is provided in the following. First, we observe that for every scenario, each of the proposed algorithms can identify a solution that satisfies the SLA requirement (this is easily verified comparing the columns obj_2 for the two algorithms with the third column T_{SLA}). A second significant observation is that both algorithms find the same minimum number of edge nodes to use to guarantee acceptable performance (to this aim, we can refer to the obj_1 columns in Tab. 3). Hence, our analysis's first conclusion is that both algorithms are viable to tackle the considered problem.

Concerning the number of nodes, which is the first objective of the optimization problem, we show a plot in Fig. 2 that presents a comparison between the actual number of edge nodes used in the infrastructure and the estimation provided by Eq. (18) (we consider the real value, before applying the $\lceil \cdot \rceil$ operator). We compare the curve of the required number of edge nodes as a function of ρ for two values of $\delta\mu$: a CPU bound scenario where $\delta\mu = 0.1$ (represented as squares in Fig. 2) and a network-bound scenario $\delta\mu = 10$ (represented as circles). For both considered scenarios, we provide both the estimate from Eq. (18) (in the line with empty circles and squares) and the objective function (marked with filled circles and squares). We observe that the estimation for the number of edge nodes is a suitable option for a first rough sizing of the infrastructure. The number of edge nodes used is the upper integer compared to the estimation with just one exception. When $\rho = 0.8$,

Table 3 Experimental results.

Parameters			GA		VNS	
ρ	$\delta\mu$	T_{SLA} [s]	obj_1 [#edge]	obj_2 [s]	obj_1 [#edge]	obj_2 [s]
0.1	0.01	1.002	2	0.201	2	0.201
0.1	0.1	1.017	2	0.209	2	0.207
0.1	1	1.172	2	0.293	2	0.268
0.1	10	2.724	2	1.126	2	0.871
0.2	0.01	2.003	3	0.603	3	0.602
0.2	0.1	2.034	3	0.621	3	0.614
0.2	1	2.345	3	0.794	3	0.729
0.2	10	5.449	3	2.525	3	1.867
0.5	0.01	5.009	6	3.072	6	3.070
0.5	0.1	5.086	6	3.121	6	3.096
0.5	1	5.862	6	3.607	6	3.374
0.5	10	13.622	6	8.461	6	6.010
0.8	0.01	8.014	10	4.013	10	4.048
0.8	0.1	8.138	10	4.123	10	4.487
0.8	1	9.380	9	9.343	9	8.724
0.8	10	21.796	9	18.986	9	12.835
0.9	0.01	9.016	10	9.014	10	9.006
0.9	0.1	9.155	10	9.142	10	9.054
0.9	1	10.552	10	10.437	10	9.552
0.9	10	24.520	10	23.175	10	14.496

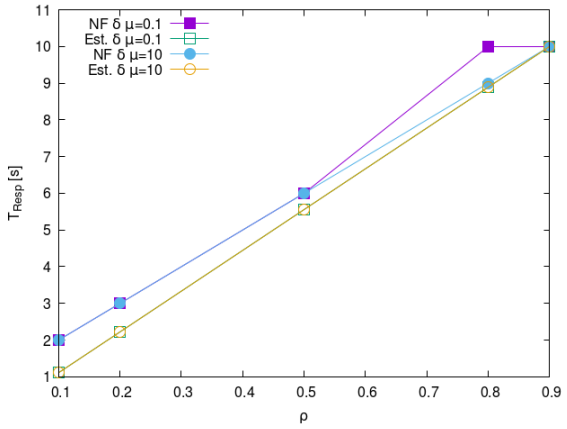

Fig. 2 Number of edge nodes

the expected number of nodes is 8.89. We can identify a suitable topology for a network-bound scenario where an optimized linking between the 100 sensors and 9 enabled edge nodes compensate for slightly unbalanced load distribution among the edge nodes (we cannot evenly distribute 100 sensors over 9 edge nodes). On the other hand, in a CPU-bound scenario, finding a feasible solution and 10 edge nodes are required.

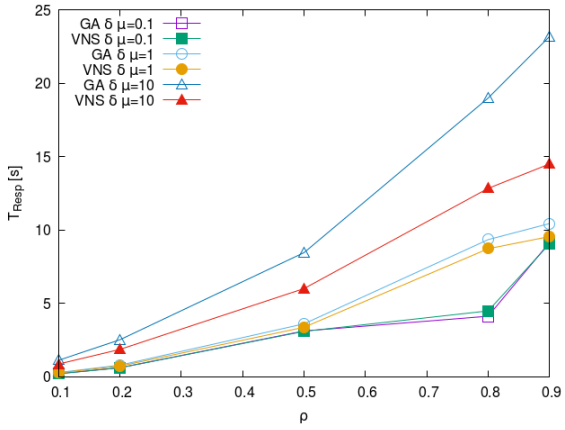

Fig. 3 Response time

Further analysis is the plot of the response time as a function of the load ρ for different $\delta\mu$ scenarios, as in Fig. 3. Specifically, the curves with squares refer to $\delta\mu = 0.1$, the curves with circles refer to $\delta\mu = 1$, and the curves with triangles refer to $\delta\mu = 10$. For every scenario, we compare the performance of the VNS (filled symbols) with the GA (empty symbols). We observe that the two algorithms present similar behavior, with times that (while remaining below the SLA requirement) grow with both the utilization ρ and the $\delta\mu$ parameter.

To fully understand the two algorithms' relative performance, the final comparison shows a heatmap of the relative response time. In particular, we measure the performance gain defined as:

$$R_{\%} = \frac{T_R^{GA} - T_R^{VNS}}{T_R^{VNS}} \cdot 100 \tag{19}$$

When $R_{\%}$ is positive, the VNS is faster than the GA alternative, while when the value is negative, GA provides better performance.

The heatmap in Fig. 4 shows that the VNS outperforms in almost every condition the GA approach. In particular, the performance gain is higher when the network delay plays a significant role and when the load is higher, as testified by the large predominance of red hues in the upper and right part of the graph. Indeed, the VNS iteration is specifically tailored to the considered problem's characteristics, while the GA operators are more generic. Furthermore, the crossover operator is not highly optimized for the considered double meaning of the two parts of the chromosome and may not guarantee the passage of good characteristics from the parents to offspring. This effect is more evident when we need to cope with high network delays.

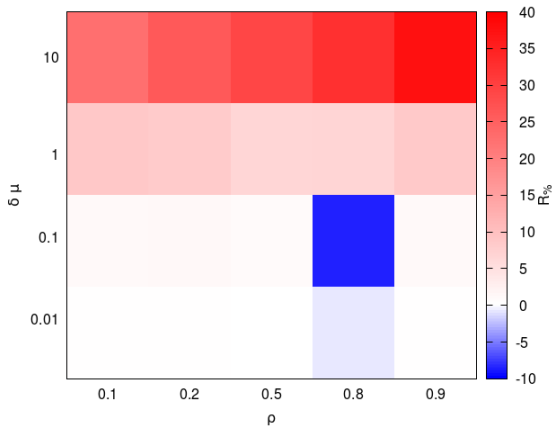

Fig. 4 Comparison of heuristics

8 Conclusion and Future Work

Modern IoT applications typically rely on significant amount of data collected by geographically distributed sensors to be processed by artificial intelligence algorithms with different goals. Depending on the field of application and the characteristics of data collected and provided service, the IoT applications may present strong requirements in terms of latency, CPU, bandwidth, reliability, and statefulness. In this scenario, the Edge Computing paradigm may represent a preferable choice with respect to traditional cloud computing systems to reduce network traffic and improve latency by placing computational resources at the edge of the network. However, edge computing opens new issues for the design of the infrastructure and the deployment of IoT applications. Specifically, critical aspects are related to the allocation of data flows coming from sensors over the nodes of the edge layer and the identification of the number and location of edge nodes to be activated.

In this chapter we formalize the problem of designing and operating an edge computing infrastructure supporting, assuming the IoT application to be both CPU-bound and network-bound, where network concerns may be either due to latency or bandwidth issues. We propose a multi-objective optimization problem to minimize both the response time and the cost associated with the number of activated edge nodes. To face the complexity of the problem that involves a non-linear objective function, the chapter presents and evaluates two heuristics, namely Variable Neighborhood Search (VNS) and Generic Algorithms (GA). The performance of the two heuristics are evaluated over a wide range of scenarios based on the realistic setting of a smart city application developed in a medium-sized Italian city.

The experimental evaluation proves that both heuristics can effectively support the design of an edge computing infrastructure. They returned a feasible solution satisfying the service level agreement for all scenario parameters, from CPU-bound to network-bound ones. Each heuristic's solution requires the same number of edge

nodes to be active. On the other hand, in terms of relative response time, the VNS approach is overall superior, reaching a performance gain up to almost 40% in scenarios characterized by high network delays and load. This chapter represents an important step along a research line aimed at the identification of promising solutions for the development of future infrastructures supporting IoT applications. In future works, we plan to extend the solutions to handle the possible heterogeneity of the edge nodes and dynamic scenarios where the load can change over time.

References

1. A. Ahmadi-Javid, P. Seyedi, and S. S. Syam. A survey of healthcare facility location. *Computers & Operations Research*, 79:223 – 263, 2017.
2. A. H. Alavi, P. Jiao, W. G. Buttlar, and N. Lajnef. Internet of Things-enabled Smart Cities: State-of-the-art and Future Trends. *Measurement*, 129:589 – 606, 2018.
3. D. Ardagna, M. Ciavotta, and R. Lancellotti. A Receding Horizon Approach for the Runtime Management of IaaS Cloud Systems. In *Proc. of 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE, 2014.
4. D. Ardagna, M. Ciavotta, R. Lancellotti, and M. Guerriero. A hierarchical receding horizon algorithm for QoS-driven control of multi-IaaS applications. *IEEE Transactions on Cloud Computing*, pages 1–1, 2018.
5. T. Back, D. Fogel, and Z. Michalewicz. *Evolutionary Computation 1: Basic Algorithms and Operators*. CRC Press, 2002.
6. A. Bačević, N. Vilimonović, I. Dabić, J. Petrović, D. Damnjanović, and D. Džamić. Variable neighborhood search heuristic for nonconvex portfolio optimization. *The Engineering Economist*, 64(3):254–274, 2019.
7. S. Binitha, S. S. Sathya, et al. A survey of bio inspired optimization algorithms. *International Journal of Soft Computing and Engineering*, 2(2):137–151, 2012.
8. F. Bu and X. Wang. A smart agriculture iot system based on deep reinforcement learning. *Future Generation Computer Systems*, 99:500 – 507, 2019.
9. G. Caiza, M. Saeteros, W. Oñate, and M. V. Garcia. Fog computing at industrial level, architecture, latency, energy, and security: A review. *Heliyon*, 6(4):e03706, 2020.
10. C. Canali and R. Lancellotti. A Fog Computing Service Placement for Smart Cities based on Genetic Algorithms. In *Proc. of International Conference on Cloud Computing and Services Science (CLOSER 2019)*, Heraklion, Greece, May 2019.
11. C. Canali and R. Lancellotti. GASP: Genetic Algorithms for Service Placement in fog computing systems. *Algorithms*, 12(10), 2019.
12. C. Canali and R. Lancellotti. Paffi: Performance analysis framework for fog infrastructures in realistic scenarios. In *2019 4th International Conference on Computing, Communications and Security (ICCCS)*, pages 1–8, Oct 2019.
13. D. Celik Turkoglu and M. Erol Genevois. A comparative survey of service facility location problems. *Annals of Operations Research*, 292:399–468, 2020.
14. L. Cooper. Location-allocation problems. *Operations Research*, 11(3):331–343, 1963.
15. R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang. Optimal Workload Allocation in Fog-Cloud Computing Toward Balanced Delay and Power Consumption. *IEEE Internet of Things Journal*, 3(6):1171–1181, Dec 2016.
16. S. Dhingra, R. B. Madda, R. Patan, P. Jiao, K. Barri, and A. H. Alavi. Internet of things-based fog and cloud computing technology for smart traffic monitoring. *Internet of Things*, page 100175, 2020.
17. R. Z. Farahani, M. SteadieSeifi, and N. Asgari. Multiple criteria facility location problems: A survey. *Applied Mathematical Modelling*, 34(7):1689 – 1709, 2010.

18. F. Foukalas. Cognitive iot platform for fog computing industrial applications. *Computers & Electrical Engineering*, 87:106770, 2020.
19. S. S. Gill, S. Tuli, M. Xu, I. Singh, K. V. Singh, D. Lindsay, S. Tuli, D. Smirnova, M. Singh, U. Jain, H. Pervaiz, B. Sehgal, S. S. Kaila, S. Misra, M. S. Aslanpour, H. Mehta, V. Stankovski, and P. Garraghan. Transformative effects of IoT, Blockchain and Artificial Intelligence on cloud computing: Evolution, vision, trends and open challenges. *Internet of Things*, 8:100118, 2019.
20. P. Hansen, N. Mladenović, and J. A. Moreno Pérez. Variable neighbourhood search: methods and applications. *Annals of Operations Research*, 175(1):367–407, 2010.
21. P. G. Harrison and N. M. Patel. *Performance Modeling of Communication Networks and Computer*. Addison-Wesley, 1993.
22. C. Irawan and S. Salhi. Aggregation and non aggregation techniques for large facility location problems - a survey. *Yugoslav Journal of Operations Research*, 25:313–341, 2015.
23. E. Khorov, A. Lyakhov, A. Krotov, and A. Guschin. A survey on IEEE 802.11 ah: An enabling networking technology for smart cities. *Computer Communications*, 58:53–69, 2015.
24. M. Klinkowski, K. Walkowiak, and R. Goścień. Optimization algorithms for data center location problem in elastic optical networks. In *2013 15th International Conference on Transparent Optical Networks (ICTON)*, pages 1–5, June 2013.
25. F. Liu, G. Tang, Y. Li, Z. Cai, X. Zhang, and T. Zhou. A Survey on Edge Computing Systems and Tools. *Proceedings of the IEEE*, 107(8):1537–1562, 2019.
26. A. Marotta and S. Avallone. A Simulated Annealing Based Approach for Power Efficient Virtual Machines Consolidation. In *Proc. of 8th International Conference on Cloud Computing (CLOUD)*. IEEE, 2015.
27. N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.
28. J. Moura and D. Hutchison. Fog computing systems: State of the art, research issues and future trends, with a focus on resilience. *Journal of Network and Computer Applications*, 169:102784, 2020.
29. T. A. d. Queiroz and L. R. Mundim. Multiobjective pseudo-variable neighborhood descent for a bicriteria parallel machine scheduling problem with setup time. *International Transactions in Operational Research*, 27(3):1478–1500, 2020.
30. L. F. M. Santos, R. S. Iwayama, L. B. Cavalcanti, L. M. Turi, F. E. de Souza Morais, G. Mormilho, and C. B. Cunha. A variable neighborhood search algorithm for the bin packing problem with compatible categories. *Expert Systems with Applications*, 124:209 – 225, 2019.
31. U. S. Shanthamallu, A. Spanias, C. Tepedelenlioglu, and M. Stanley. A brief survey of machine learning methods and their sensor and IoT applications. In *2017 8th International Conference on Information, Intelligence, Systems Applications (IISA)*, 2017.
32. R. A. C. Silva and N. L. S. Fonseca. On the location of fog nodes in fog-cloud infrastructures. *Sensors*, 19(11), 2019.
33. B. Tang, Z. Chen, G. Hefferman, T. Wei, H. He, and Q. Yang. A hierarchical distributed fog computing architecture for big data analysis in smart cities. In *Proceedings of the ASE BigData & SocialInformatics 2015*, ASE BD&SI '15, pages 28:1–28:6, New York, NY, USA, 2015. ACM.
34. T. Wang, Y. Liang, W. Jia, M. Arif, A. Liu, and M. Xie. Coupling resource management based on fog computing in smart city systems. *Journal of Network and Computer Applications*, 135:11 – 19, 2019.
35. Z. Wen, R. Yang, P. Garraghan, T. Lin, J. Xu, and M. Rovatsos. Fog orchestration for internet of things services. *IEEE Internet Computing*, 21(2):16–24, Mar 2017.
36. R. W. Wolff. Poisson arrivals see time averages. *Operations Research*, 30(2):223–231, 1982.
37. Z. Xu and Y. Cai. Variable neighborhood search for consistent vehicle routing problem. *Expert Systems with Applications*, 113:66 – 76, 2018.
38. N. Yamanaka, G. Yamamoto, S. Okamoto, T. Muranaka, and A. Fumagalli. Autonomous driving vehicle controlling network using dynamic migrated edge computer function. In *2019 21st International Conference on Transparent Optical Networks (ICTON)*, Angers, France, 2019.

39. S. Yi, C. Li, and Q. Li. A survey of fog computing: Concepts, applications and issues. In *Proceedings of the 2015 Workshop on Mobile Big Data*, Mobidata '15, pages 37–42, New York, NY, USA, 2015. ACM.
40. A. Yousefpour, G. Ishigaki, and J. P. Jue. Fog computing: Towards minimizing delay in the internet of things. In *2017 IEEE International Conference on Edge Computing (EDGE)*, pages 17–24, June 2017.
41. Z. I. M. Yusoh and M. Tang. A penalty-based genetic algorithm for the composite saas placement problem in the cloud. In *IEEE Congress on Evolutionary Computation*, pages 1–8, July 2010.
42. H. Zahmatkesh and F. Al-Turjman. Fog computing for sustainable smart cities in the iot era: Caching techniques and enabling technologies - an overview. *Sustainable Cities and Society*, 59:102139, 2020.
43. F. Zezulka, P. Marcon, I. Vesely, and O. Sajdl. Industry 4.0 – an introduction in the phenomenon. *IFAC-PapersOnLine*, 49(25):8 – 12, 2016. 14th IFAC Conference on Programmable Devices and Embedded Systems PDES 2016.
44. C. Zhang. Design and application of fog computing and internet of things service platform for smart city. *Future Generation Computer Systems*, 112:630 – 640, 2020.
45. P. Zheng, H. Wang, and Z. Sang. Smart manufacturing systems for industry 4.0: Conceptual framework, scenarios, and future perspectives. *Frontiers of Mechanical Engineering*, 13:137 – 150, 2018.