

Data Flows Mapping in Fog Computing Infrastructures Using Evolutionary Inspired Heuristic

Claudia Canali and Riccardo Lancellotti

Department of Engineering "Enzo Ferrari",
University of Modena and Reggio Emilia, Modena, Italy
{claudia.canali, riccardo.lancellotti}@unimore.it

Abstract. The need for scalable and low-latency architectures that can process large amount of data from geographically distributed sensors and smart devices is a main driver for the popularity of the fog computing paradigm. A typical scenario to explain the fog success is a smart city where monitoring applications collect and process a huge amount of data from a plethora of sensing devices located in streets and buildings. The classical cloud paradigm may provide poor scalability as the amount of data transferred risks the congestion on the data center links, while the high latency, due to the distance of the data center from the sensors, may create problems to latency critical applications (such as the support for autonomous driving). A fog node can act as an intermediary in the sensor-to-cloud communications where pre-processing may be used to reduce the amount of data transferred to the cloud data center and to perform latency-sensitive operations. In this book chapter we address the problem of mapping sensors over the fog nodes with a twofold contribution. First, we introduce a formal model for the mapping model that aims to minimize response time considering both network latency and processing time. Second, we present an evolutionary-inspired heuristic (using Genetic Algorithms) for a fast and accurate resolution of this problem. A thorough experimental evaluation, based on a realistic scenario, provides an insight on the nature of the problem, confirms the viability of the GAs to solve the problem, and evaluates the sensitivity of such heuristic with respect to its main parameters.

Keywords: Fog computing, Optimization model, Evolutionary programming, Genetic algorithms, Smart Cities

1 Introduction

Cyber-physical systems are producing an ever-growing amount of data through the presence of a large number of geographically-distributed sensors. This ever-increasing amount of data needs to be filtered and processed to support advanced applications that can monitor complex systems. However, as the data size grows, the traditional cloud paradigm becomes inadequate and can result in poor performance, due to the risk of high latency in the responses or due to the risk of creating congestion on the network links of the cloud data centers.

An approach that can increase the scalability and can reduce the latency is the *fog computing* paradigm. Fog computing moves some services from the cloud data center

(located on the network core) towards a set of *fog nodes* located on the network edge, that is the location where the data sources connect to the network. The services that are typically deployed on the fog layer involve data filtering, validation, aggregation, alarm triggering and other pre-processing tasks that can reduce latency and the amount of data transferred towards the cloud data centers. This paradigm has been proposed in [11, 13] as an enabling architecture in different areas. Examples include:

- Applications such as gaming and videoconferencing that need very low and predictable latency
- Geo-distributed monitoring applications like pipeline monitoring or environmental sensing
- Fast mobile applications such as smart connected vehicles and connected rails
- Large-scale distributed control systems including smart grid, smart traffic monitoring, and support for autonomous driving

In this book chapter, we focus mainly on an environmental monitoring application for a smart city, where multiple sensors are deployed over a geographic area to monitor air pollution and traffic. We assume that the infrastructure is structured as in Figure 1 and is composed of three layers: a first *sensor layer* that produces data (the sensors are represented at the bottom of the figure as a set of wireless devices), an intermediate *fog layer* that carries out the preliminary processing of data from the sensors, and a *cloud layer* that is composed by one or more data centers (at the top of the figure) and that is the final destination of the data. Sensors collect information about the city status, such as traffic intensity or air quality [16]. Such data should be collected at the level of a cloud infrastructure to provide value-added services such as traffic or pollution forecast. The proposed fog layer intermediates the communication between the sensors and the cloud to provide scalability and reliability in the smart city services.

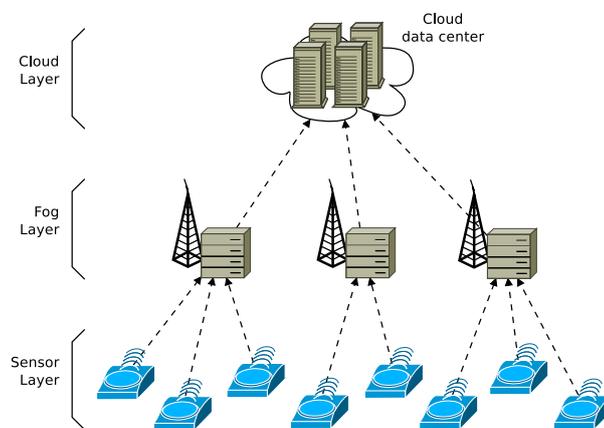


Fig. 1: Fog infrastructure

Our problem is centered on the management of data flows in a fog infrastructure, that is, distributing the data from the sensors among the fog nodes taking into account

both the load balancing issues at the level of each fog node, and the latency in the sensors-to-fog links. This problem is rather new, because most literature on fog infrastructures takes the sensor-to-fog mapping as a fixed parameter depending on the system topology [8], while we assume that, at least on a city-scale scenario, there are degrees of freedom that can be leveraged. As this problem is complex and may present scalability issues, we also introduce an heuristic, based on Genetic Algorithms, to solve the problem. A preliminary version of this research appeared in [5]; however, in this chapter we improve the theoretical model, supporting a more detailed analysis of the problem, and we introduce a more detailed experimental evaluation of our proposal to provide a better insight on the problem and on its solution.

Specifically, our contribution can be summarized as follows: (1) we present an innovative approach to model fog infrastructures, defining an optimization problem for the sensor-to-fog mapping problem; (2) we introduce a GAs-based heuristic to solve this new problem (other approaches based on this heuristics were applied only to traditional cloud data centers [20] and/or to Web Service composition [10]); (3) we provide a thorough experimental analysis of the problem, with the aim to give insight on the nature of the problem and on the performance of the proposed heuristic solution.

The book chapter is organized as follows. Section 2 discusses the related work, while Section 3 formalizes as an optimization problem the model of the considered application. Section 4 presents the heuristic algorithms proposed to solve the problem. Section 5 introduces the experimental testbed and discusses the results that confirms the viability of our approach. Finally, Section 6 concludes the paper with some final remarks.

2 Related work

The significant increase in the amount of data generated by modern infrastructures, joined to the need of processing the same data to provide value-added services, motivates the research community to explore the edge-based solutions (such as the so-called *fog computing*) as an evolution of the traditional cloud-based model. Indeed, when we have a large amount of data originated at the border of a network, a centralized cloud model becomes inefficient, while an approach that pushes some operation (such as filtering or aggregation of the data) towards the network edge has been demonstrated to be viable alternative in several papers [8, 15, 17, 18].

In [18], Yi *et al.* propose a survey with significant application scenarios (and the corresponding design and implementation challenges) of fog computing. A more widespread discussion of open issues, open research directions and challenges of fog computing for IoT services and smart cities is proposed in [17]. This vision is consistent with the reference scenario proposed in this chapter. Indeed, we refer to a similar smart-city sensing application and we consider, as the main challenge, the combined goal of reducing communication latency, while preserving load balancing.

Some studies focus on the allocation of services that process data from the fog nodes over a distributed cloud infrastructure. For example, Deng *et al.* [8] investigate how power consumption and transmission delay can be balanced in fog-to-cloud communication, proposing an optimization model. Another interesting study [19] focuses

on minimizing the service delay in a IoT-fog-cloud scenario, where fog nodes implement a policy based on fog-to-fog load sharing. It is noteworthy to consider that, in both these studies, the mapping of data sources over the fog nodes is not taken into account. Indeed, [8] assumes single-hop wireless links between sensors and fog nodes, while communication in [19] is based on application-specific domains. Our research is complementary with respect to these issues, as we consider a network layer capable of multi-hop links so that each sensor can communicate with each fog node, motivating our sensors-to-fog mapping.

Another area that is close to our research is that of fog computing infrastructures supporting smart cities. For example, Tang *et al.* [15] propose a hierarchical 4-layer fog computing system for smart cities. This layering leverages the nature of a geographic distribution in a large set of sensors that carry out latency-sensitive tasks, where a fast control loop is required to guarantee the safety of critical infrastructure components. Moreover, unlike our research, also this paper assumes that the sensors-to-fog nodes mapping is fixed so that a fog node communicates only with a local set of sensors that are deployed in the neighborhood.

A different research [6] is more focused on the vision of Data Stream Processing (DSP) applications. In particular, the paper addresses the operator placement problem: DSP operators must be mapped on the fog nodes with the goal of maximizing the QoS. The problem is described as an Integer Linear Programming (ILP) problem. However, the authors of [6] assume that the incoming data flow can be split to support parallel processing. Our research considers a more generic application scenario where this assumption may not be acceptable. Research on genetic algorithms (GAs) have been proposed in the area of cloud computing. In particular, Yusoh *et al.* [20] rely on GAs to propose a solution of Software as a Service (SaaS) Placement Problem that is scalable. Another significant example is [10] where a QoS-aware service composition problem in cloud systems is solved using GAs.

Finally, this chapter extends a previous research [5], where a similar problem is considered. However, in the present study improves both the model, with a more effective and simplified presentation of the problem model, and the experimental section, introducing a thorough discussion of the parameters characterizing the problem and their impact on the ability of commercial solvers and of the proposed solution to reach a high quality solution for the problem.

3 Problem definition

3.1 Problem overview

The reference architecture in Figure 1 is used as the basis of our problem modeling. In particular, we consider a set of sensors \mathcal{S} distributed over an area and we assume that these sensors are producing data with a known intensity, such that the generic sensor $i \in \mathcal{S}$ produces a packet of data with a frequency λ_i (the reader may refer to Table 1 for a summary of the symbols used in the model). In our model, we introduce some simplifying assumptions that is, we consider a stationary scenario where the data rate of sensors remains stable over time and where sensors do not move. Furthermore, we

anticipate that, in the experiments we will consider sensors to be homogeneous, that is λ_i is the same for every sensor, even if the model is capable to capture an heterogeneous scenario as well.

The fog layer is composed by a set of nodes \mathcal{F} that collect the data from the sensors and carry out tasks such as validation, filtering and aggregation on them, to guarantee a fast and scalable processing of data even in the case where data processing is computationally demanding. In our model we consider that the generic fog node $j \in \mathcal{F}$ can process a data packet from a sensor with an average service time equal to $1/\mu_j$. The fog nodes send their output to one or more cloud data centers where additional processing and data storage is carried out.

Since the problem of managing a cloud data center has been widely discussed in literature [14], we do not consider the details of the internal management of the cloud resources but we focus instead on the issue of mapping the sensors over the fog nodes. In particular, we aim to model the QoS of the fog infrastructure in terms of response time, taking into account the following two contributions for the infrastructure performance:

- Network-based latency. This delay is due to the communication between the sensors and the fog nodes. We denote the latency as $\delta_{i,j}$ where i is a sensor and j is a fog node.
- Processing time. The processing time on the fog node can be modeled using the queuing theory. It depends on $1/\mu_j$, the service time of a packet of data from a sensor on fog node j , and on the incoming data rate λ_j that is the sum of the data rates λ_i of the sensors i that are sending data to the fog node j .

The reader may refer to Table 1 where we summarize the symbols used in the model.

3.2 Optimization model

In the considered fog scenario, we aim to map the data flows from the sensors on the fog nodes. Hence, we introduce an optimization problem where the decision variable represent this mapping as a matrix of boolean values $X = \{x_{i,j}, i \in \mathcal{S}, j \in \mathcal{F}\}$. Considering the values on the matrix, $x_{i,j} = 1$ if sensor i sends data to fog node j , while $x_{i,j} = 0$ if this data exchange does not occur. To support some stateful pre-processing (even a simple moving window average function would fall in this category), we assume that all the data of a sensor should be sent to the same fog node. Hence, for a given value of i , there is only one value of j such that $x_{i,j} = 1$.

We now introduce a formal model for the previously-introduced optimization problem. The basis is similar to the request allocation problem in a distributed infrastructure, such as the allocation of VMs on a cloud data center [9, 12, 14]. As previously pointed out, we rely on a decision variable represented as a matrix of boolean values X . The values in the matrix determine if a sensor i is sending data to fog node j . Furthermore, we introduce an objective function and a set of constraints as follows:

Table 1: Notation.

Symbol Meaning/Role	
Decision variables	
$x_{i,j}$	Sending data flow from sensor i to fog node j
Model parameters	
\mathcal{S}	Set of sensors
\mathcal{F}	Set of Fog nodes
λ_i	Outcoming data rate from sensor i
λ_j	Incoming data rate at fog node j
$1/\mu_j$	Service time at fog node j
$\delta_{i,j}$	Communication latency between sensor i to fog node j
Model variables	
i	Index of a sensor
j	Index of a Fog node

$$\min Obj(X) = Obj_{net}(X) + Obj_{proc}(X) \quad (1.1)$$

where:

$$Obj_{net}(X) = \sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{F}} x_{i,j} \cdot \delta_{i,j} \quad (1.2)$$

$$Obj_{proc}(X) = \sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{F}} x_{i,j} \cdot \frac{1}{\mu_j - \lambda_j} \quad (1.3)$$

$$\lambda_j = \sum_{i \in \mathcal{S}} x_{i,j} \cdot \lambda_i \quad \forall j \in \mathcal{F}, \quad (1.4)$$

subject to:

$$\sum_{j \in \mathcal{F}} x_{i,j} = 1 \quad \forall i \in \mathcal{S}, \quad (1.5)$$

$$\lambda_j < \mu_j \quad \forall j \in \mathcal{F}, \quad (1.6)$$

$$x_{i,j} = \{0, 1\}, \quad \forall i \in \mathcal{S}, j \in \mathcal{F}, \quad (1.7)$$

In the problem formalization, the objective function 1.1 is composed by two components (Obj_{net} and Obj_{proc} , respectively) that represent the total (and hence the average) latency and processing time, respectively. The computation of the latency component is rather straightforward and aims to capture the communication delay in a geographically distributed infrastructure based on the latency values $\delta_{i,j}$, as detailed in Equation 1.2. The processing time used for the component Obj_{proc} of the objective function is detailed in Equation 1.3. The definition is consistent with other papers that focus on a

distributed cloud infrastructure such as [3]. In particular, the processing time is derived from Little's result applied to a M/G/1 system and takes into account the average arrival frequency λ_j and the processing rate μ_j of each fog node j . Equation 1.4 defines the incoming load λ_j on each fog node j as a function of the mapping of sensors in the infrastructure.

The objective function is combined with a set of constraints. In particular, constraint 1.5 means that, for each sensor i , its data is sent to one and only one fog node. Constraint 1.6 guarantees that, for each fog node j , we avoid a congestion situation, that is we need to avoid the case where, for a generic node j , the load λ_j is higher than the processing capability μ_j . Finally, constraint 1.7 describes the boolean nature of the decision variables $x_{i,j}$.

4 Heuristic algorithm

To solve the optimization problem defined in the previous section we consider an evolutionary inspired heuristic based on the Genetic Algorithms (GAs), with the aim to evaluate its effectiveness in solving the problem by comparing the heuristic performance with the one of commercial solvers.

The main idea behind GAs is to operate on a *population of individuals*, where each individual represents a possible solution of the problem. The solution is encoded in a *chromosome* that defines the individual and the chromosome is composed by a fixed number of *genes* that represent the single parameters characterizing a specific solution of the problem.

A population of individuals is typically initialized randomly. A *fitness function*, that describes the objective function of the optimization problem is applied to each individual. The evolution of population through a set of *generations* aims at improving the fitness of the population using the following main operators:

Mutation is a modification of a single or a group of genes in a chromosome describing the individual of the population. Figure 2 presents an example of such operator where the i^{th} gene of the rightmost individual in the K^{th} generation undergoes a mutation. The main parameter of this operator is the probability of selecting an individual to perform a mutation on one of its genes. In the sensitivity analysis in Section 5.4, we will refer to this probability as P_{mut} .

Crossover is a merge of two individuals by exchanging part of their chromosomes. Figure 2, again, provides an example of this operator applied to the two individuals composing the population at the K^{th} generation. In particular, in Figure 2 the child individual is characterized by a chromosome containing the genes from c_0 to c_{i-1} from the rightmost parent and the genes from c_i to c_S from the leftmost parent. The main parameter of this operator concerns the selection of the parents. In the sensitivity analysis in Section 5.4, we will refer to the probability of selecting an individual for a crossover operation as P_{cross} .

Selection concerns the criteria used to decide if an individual is passed from the K^{th} generation to the next. The typical approach in this case is to apply the fitness function to every individual (including new individuals generated through mutation and crossover) and to consider a probability of being selected for the next generation

that is proportional to the fitness value. The selection mechanism ensures that the population size remains stable over the generations.

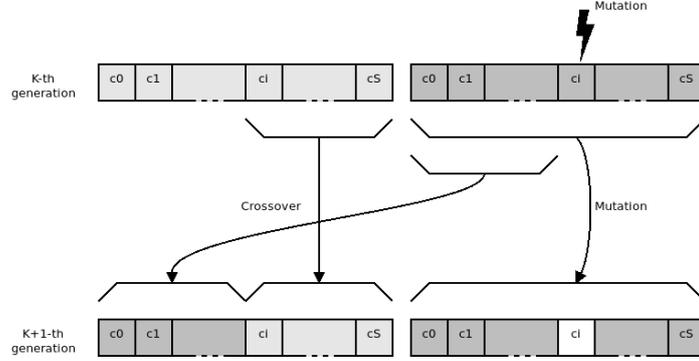


Fig. 2: Examples of genetic algorithms operators

When applying a GAs approach to the problem of mapping sensors over the fog nodes of a distributed architecture, we must encode a solution as a gene. In particular, we aim to formalize the relationship between the model in Section 3.2 and the GA chromosome encoding. Hence, we define a chromosome as a set of S genes, where $S = |\mathcal{S}|$ is the number of sensors. Each gene is an integer number from 1 to F , where $F = |\mathcal{F}|$ is the number of fog nodes in our infrastructure. The generic i^{th} gene in a chromosome c_i can be defined as: $c_i = \{j : x_{i,j} = 1\}$. Due to constraint 1.5 in the optimization model, we know that only one fog node will receive data from sensor i , so we have a unique mapping between a solution of the problem expressed using the decision variable $x_{i,j}$ and the GA-based representation of a solution. As we can map each chromosome into a solution of the original optimization problem, we can use the objective function 1.1 as the basis for fitness function of our problem. Constraints 1.5 and 1.7 are automatically satisfied by our encoding of the chromosomes. The only constraint we have to explicitly take into account is constraint 1.6 about the fog node overload. As embedding the notion of unacceptable solution in a genetic algorithm may hinder the ability of the heuristic to converge towards a solution, we prefer to insert this information into the fitness function, in such a way that the individual providing a solution where one or more fog nodes are overloaded is characterized by a high penalty and is unlikely to enter in the subsequent generation.

Multiple optimization algorithms have been considered before adopting the choice of a genetic algorithm. On one hand, greedy heuristics tend to provide performance that heavily depends on the inherent nature of the problem. For example, the non-linear objective function may hinder the application of some greedy approaches, while the number of sensors that may be supported by each fog node may have significant impact on the performance of branch and bound heuristics. As we aim at providing a general and flexible approach to tackle this problem, we prefer to focus on meta-heuristics that

are supposed to be better adaptable to a wider set of problem instances [4]. Among these solution, we focus on evolutionary programming in general and on genetic algorithms in particular as this class of heuristics has been proven a viable option in similar problems such as the problem of allocating VMs on a cloud infrastructure [20].

5 Experimental results

5.1 Experimental testbed

To evaluate the performance of the proposed solution, we consider a realistic fog computing scenario where geographically distributed sensors produce data flows to be mapped over a set of fog nodes, which are nodes with limited computational power and devoted to tasks such as aggregation and filtering of the received data; then, the pre-processed data are sent to the cloud data center.

To evaluate our proposal in a realistic scenario, we modeled the geographic distribution of all the components of the system according to the real topology of the small city of Modena in Italy (counting almost 180.000 inhabitants). Our reference use case is a traffic monitoring application where the wireless sensors are located on the main streets of the city and collect data about the number of cars passing on the street, their speed and other traffic related measures together with environmental quality indicators (an example of this application can be found in the Trafair Project [16]). Figure 3 shows the map of sensor nodes, fog nodes and cloud data center for the considered smart city scenario. To build the map of sensors, we collected a list of the main streets in Modena and we geo-referenced them. We assume that in each main street we have at least one sensor producing data. We selected a group of 6 buildings hosting the offices of the municipality and we use them as the location of the fog nodes – this assumption is consistent with the current trend of interconnecting the main public building of each city with high bandwidth links. Our final scenario is composed of 89 sensors and 6 fog nodes. The interconnection between fog nodes and sensors is characterized by a delay that we model using the euclidean distance between the nodes. The average delay is in the order of 10 ms, that is consistent with a geographic network. Finally, we assume that the cloud data center is co-located with the actual location of the municipality data center.

Concerning the traffic and processing models, we rely on two main parameters to evaluate different conditions: the first metric ρ represents the average load of the system; the second parameter $\delta\mu$ represents the ratio between the average network delay (δ) and the service time ($1/\mu$). Specifically, we define the two parameters as:

$$\rho = \frac{\sum_{i \in \mathcal{S}} \lambda_i}{\sum_{j \in \mathcal{F}} \mu_j} \quad (2)$$

$$\delta\mu = \frac{\sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{F}} \delta_{i,j}}{|\mathcal{F}| |\mathcal{S}|} \cdot \frac{\sum_{j \in \mathcal{F}} \mu_j}{|\mathcal{F}|} \quad (3)$$

In our experiments, we consider several scenarios corresponding to different combinations of these parameters, in order to analyze the performance of the GAs-based solution for the sensor mapping problem.

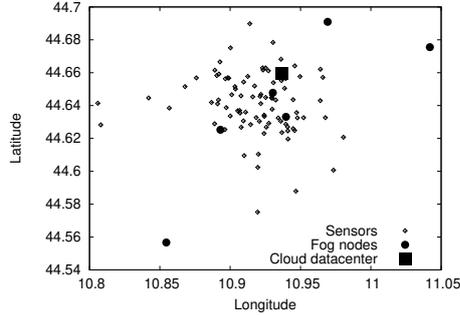


Fig. 3: Smart city scenario [5]

In order to solve the problem of mapping data flows over the sensor nodes, we implemented the optimization model with the AMPL language [2] and then we use the commercial solver K-NITRO [1]. Specifically, the AMPL definition is directly based on the optimization problem discussed in Section 3. Due to the nature of the problem, we were not able to let the solver run until the convergence. Instead, we placed a wall-time limit of 120 minutes, with a 16 core CPU and 16 concurrent threads. Due to this limitation, we also consider a case where we remove the constraint 1.5, thus allowing each data flow to be split over more fog nodes: the best solution achieved in this case represents a theoretical optimal bound for our problem.

The genetic algorithm is implemented using the Distributed Evolutionary Algorithms in Python (DEAP) framework [7] based on the details provided in Section 4. In the evaluation of the genetic algorithm approach, we run the experiments 5 times and we average the main metrics. In particular, for each run of the genetic algorithm, we consider the best achieved solution at each generation. The algorithm maintains a population of 200 individuals and we force a stop of the algorithm after 300 generations. Moreover, the genetic algorithm considers for the main parameters the following default values, that have been selected after some preliminary experiments: mutation probability $P_{mut} = 0.8$ and crossover probability $P_{cross} = 0.5$.

In order to analyze the performance of the evolutionary inspired proposal for the sensor mapping problem, we compare the best solution found by the GA heuristic (Obj^{GA}) with the best solution found by the solver for the walltime limited AMPL problem (Obj^{AMPL}) and with the theoretical optimal bound (Obj^*), considering as the main performance metric the *discrepancy* ϵ between the solutions, as it will be defined in the rest of the section. Furthermore, we evaluate the convergence speed of the GA algorithm, considering as the convergence criteria the case of a fitness value within 2% of the theoretical optimal bound. To this aim, we measure the number of generations needed by the GA heuristic to converge. Finally, we consider also the computation time as a function of the population size.

5.2 Evaluation of Genetic Algorithm performance

The first analysis in our experiments compares the difference between the solution found by the GA and the theoretical optimal bound obtained by the solver in the case the constraint 1.5 is removed. To this aim, we consider as performance metric the discrepancy ϵ^{GA} defined as follows:

$$\epsilon^{GA} = \frac{Obj^{GA} - Obj^*}{Obj^*} \quad (4)$$

In this experiment, we consider several scenarios by varying the values of the parameters ρ and $\delta\mu$. Fig. 4 shows as a heatmap the value of ϵ^{GA} for ρ ranging from 0.2 to 0.9 and $\delta\mu$ ranging from 0.01 to 10. To better understand the results, let us briefly discuss the impact of the considered parameters. For example, a scenario where $\rho = 0.9$ and $\delta\mu = 0.01$ (corresponding to the bottom right corner of Fig. 4) represents a case where network delay is much lower than the average job service time, while the processing demand on the system is high. This means that the scenario is CPU-bound because managing the computational requests is likely to be the main driver to optimize the objective function. On the other hand, a scenario where $\rho = 0.2$ and $\delta\mu = 10$ (top left corner of Fig. 4) is a scenario characterized by a low workload intensity and a network delay comparable with service time of a job, where it becomes important to optimize also the network contribution to the objective function.

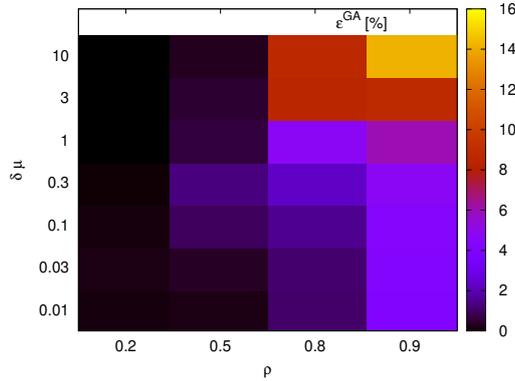


Fig. 4: Performance of GA

In the color coded representation of ϵ^{GA} in the figure, black hues refer to a better performance of the genetic algorithm, while yellow hues correspond to a worse performance. From this comparison, we observe that the value of ρ has a major impact on the performance of the GA heuristic. Indeed, the heatmap clearly shows good performance for low values of ρ (for $\rho = 0.2$, we have ϵ^{GA} below 1% for all the values of $\delta\mu$). On the other hand, when the average load of the system is high ($\rho = 0.9$) the GA algorithm shows a higher discrepancy with respect to the theoretical optimal bound: the risk of

overloading the nodes is higher and the value of the objective function is highly variant with respect to the considered solution. In this case, the value of $\delta\mu$ shows its impact on the final performance: indeed, until the ratio between the average network delay (δ) and the service time ($1/\mu$) is low ($\delta\mu = 0.01$) the discrepancy still remains limited to few percentage points, while for high values of this ratio ($\delta\mu = 10$) it goes up to almost 15%.

We now present an in-depth analysis where we separately measure the discrepancy regarding the individual contributions of the two main components of the objective function, related to the total network latency Obj_{net} and to processing time Obj_{proc} (defined in equations 1.2 and 1.3, respectively) with respect to the corresponding optimal values. To this aim, we define the discrepancies ϵ_{net}^{GA} and ϵ_{proc}^{GA} defined as follows:

$$\epsilon_{net}^{GA} = \frac{Obj_{net}^{GA} - Obj_{net}^*}{Obj_{net}^*} \quad (5)$$

$$\epsilon_{proc}^{GA} = \frac{Obj_{proc}^{GA} - Obj_{proc}^*}{Obj_{proc}^*} \quad (6)$$

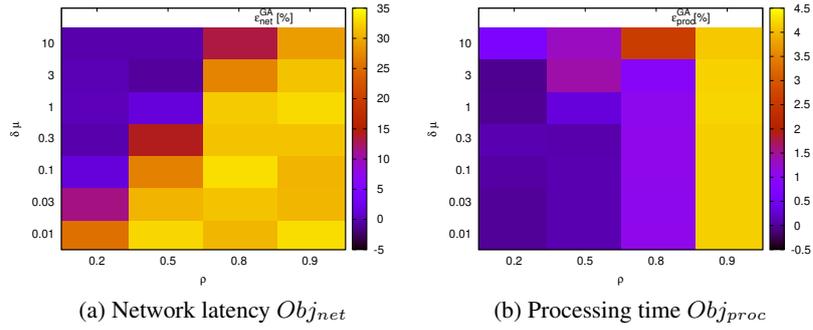


Fig. 5: Components of the objective function

Fig. 5 shows as heatmaps the values of ϵ_{net}^{GA} and ϵ_{proc}^{GA} for the considered scenarios with varying ρ and $\delta\mu$. Both the figures confirm that the difficulty to achieve a solution close to the optimum for high values of ρ . Furthermore, we observe high discrepancies (between 30% and 35%) regarding the network latency in the bottom right part of Fig. 5a. The reason of this result is that the network latency contribution to the overall object function is very small when $\delta\mu$ is low with respect to the processing time, hence the network latency component is not optimized, thus showing high discrepancies with respect to the corresponding optimal value.

To have a confirmation of the observed result, we directly measure the weight of the contribution of the two main components (network latency and processing time) to the overall value of the objective function in the case of the solution corresponding to the theoretical optimal bound. To this aim, we define W_{net} and W_{proc} as follows:

$$W_{net} = \frac{Obj_{net}^*}{Obj^*} \quad (7)$$

$$W_{proc} = \frac{Obj_{proc}^*}{Obj^*} \quad (8)$$

Fig. 6 show as heatmaps the weight of the two components expressed as percentages of the total value of the objective function, for varying values of ρ and $\delta\mu$. In the figures, black hues refer to a low percentage of the component, while yellow hues correspond to a higher weight.

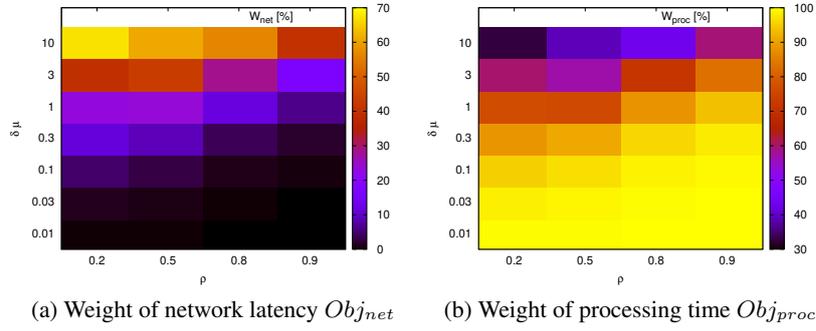


Fig. 6: Weight of components of objective function

Fig. 6a confirms the motivation of the previous result, showing how small the contribution of the network latency can get with respect the overall objective function for low values of $\delta\mu$: as show in the lower part of the heatmap, for $\delta\mu = 0.01$ the weight of latency is close to 1% of the objective function. On the other hand, Fig. 6b shows that the weight of the processing time contribution always never decreases below the 30% in the considered scenarios, reaching values between 90% and 100% in all the cases with $\delta\mu$ lower than 0.3.

We now evaluate the performance of the solution of the AMPL model obtained by the solver (Obj^{AMPL}) with respect to the theoretical optimal bound (Obj^*) and to the GA heuristic (Obj^{GA}). To this aim, we consider ϵ^{AMPL} and $\epsilon^{GA-AMPL}$ defined as follows:

$$\epsilon^{AMPL} = \frac{Obj^{AMPL} - Obj^*}{Obj^*} \quad (9)$$

$$\epsilon^{GA-AMPL} = \frac{Obj^{GA} - Obj^{AMPL}}{Obj^{AMPL}} \quad (10)$$

Fig. 7a compares the performance of the solver with the theoretical optimal bound in the form of a heatmap. We observe that for the majority of the scenarios identified

by the considered values of ρ and $\delta\mu$ the discrepancy ϵ^{AMPL} is quite low (below 7%), while it significantly increases up to almost 40% for very high average system load ($\rho = 0.9$) and $\delta\mu \leq 1$. The high discrepancy is due to the non-linear nature of the objective function, and in particular to the presence of several local minima that the solver is not able to overcome within the time limitation of 120 minutes. This clearly evidences that, in case of very high average system load, also the solver is not able to guarantee good performance due to the fact that the risk of overloading the fog nodes is high and the value of the objective function is highly variant with respect to the considered solution.

On the other hand, Fig. 7b, comparing the performance of the solver and of the proposed GA heuristic, shows that the GA achieves solutions very similar to the solver ($\epsilon^{GA-AMPL}$ close to 0%) for the majority of the scenarios, while differences can be observed for high values of average system load ($\rho \geq 0.8$). In these cases, the behavior of the GA heuristic shows significant differences. When the system is processing bound ($\delta\mu \leq 1$) the GA algorithm tends to perform much better than the solver, with the discrepancy $\epsilon^{GA-AMPL}$ reaching negative values close to -25% for $\rho = 0.9$. This is an important results showing that the GA is not only able to reach an efficient solution even in presence of a complex problem with integer programming and a non-linear objective function, but can also outperform the solver in a challenging case of highly loaded system. On the other hand, in the top right part of the heatmap ($\rho \geq 0.8$ and $\delta\mu = 10$) the GA shows worse performance than the solver, with a discrepancy $\epsilon^{GA-AMPL}$ between 7% and 15%. In this case, where the system is network bounded and with a high average load, finding an optimal solution would require to explore a wider space of solutions that the GA cannot explore being limited to 300 generations in our experiments.

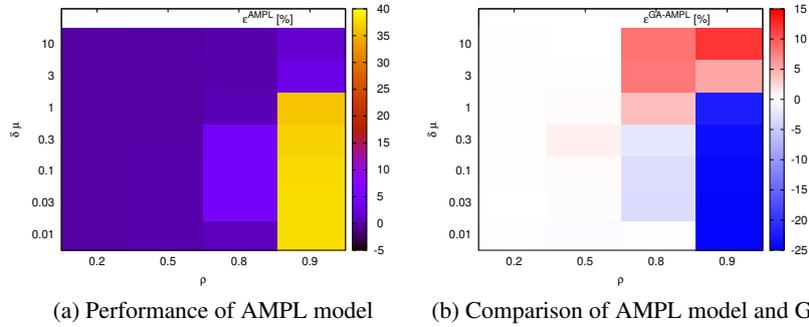


Fig. 7: Performance of AMPL model

5.3 Convergence analysis of GAs

A critical analysis concerns the impact of the number of generations on the performance of the GA heuristic, in particular on its capability to reach convergence. To carry out

this analysis (and the following sensitivity analysis), we select two specific intermediate scenarios corresponding to points of interest: the first scenario, characterized by $\rho = 0.5$ and $\delta\mu = 1$, represents a case of intermediate average system load and where the two main contributions (network latency and processing time) of the objective function have a similar weight; the second scenario, characterized by $\rho = 0.8$ and $\delta\mu = 0.3$, represents a processing bound case with a high average system load where finding a good solution for mapping data flows over the fog nodes reveals more challenging, as shown by the previous results. The motivation of this choice can be supported by the graph in Fig.8, showing the behavior of the Obj_{proc} component of the objective function for different values of ρ . Specifically, we consider the theoretical curve of Obj_{proc} for the case of one fog node and $1/\mu = 1$. The graph confirms how the value of $\rho = 0.8$ represents a point where the problem is ill-conditioned: due to the high risk of overload in the fog nodes, little variations in ρ can cause significant oscillations of the objective function. On the other hand, for $\rho = 0.5$ this risk is low, as shown by the slopes of the tangents to the load curve in the corresponding points.

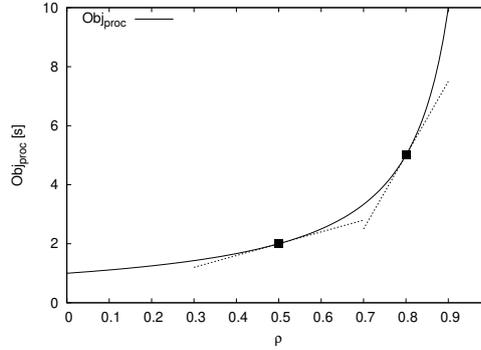


Fig. 8: Load curve of Obj_{proc}

In the following convergence analysis we consider the previously introduced discrepancy ϵ^{GA} between the GA and the theoretical optimal bound. The value of ϵ^{GA} is measured at every generation for the GA (and compared with the final optimal bound). This allows us to evaluate whether the population is converging over the generations to an optimum.

Fig. 9 presents the results of the analysis. Specifically, we consider the evolution of ϵ^{GA} for the two considered scenarios through 300 generations of the GA. The graph shows also an horizontal line at the value of 2%: we consider that the GA has reached convergence when $\epsilon \leq 2\%$ and we consider the generation when this condition is verified as a metric to measure how fast the algorithm is able to find a suitable solution.

Comparing the two curves we observe a different behavior. On one hand, for the curve characterized by $\delta\mu = 1$ we have a clear descending trend. On the other hand, for the $\delta\mu = 0.3$, the value of ϵ^{GA} is very low even with few generations and remains quite stable over the generations. The reason for this behavior can be explained con-

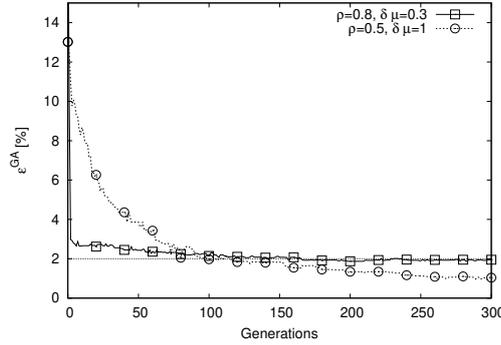


Fig. 9: Convergence analysis

sidering the nature of the problem, where the objective function depends on two main contributions: the processing time (that depends mainly on the ability of the algorithm to distribute fairly the sensors among the fog nodes) and the network latency (that depends on the ability of the algorithm to map the sensors on the closest fog node). When $\delta\mu = 0.3$, the impact of the second contribution is quite low, so, any solution that provides a good level of load sharing will be very close to the optimum. As the genetic algorithm initializes the chromosomes with a random solution, it is likely to have one or more individuals right from the first generation that provide good performance: this explains the more stable values of ϵ^{GA} over the generations. On the other hand, when $\delta\mu = 1$ the two contributions to the objective function are similar: hence, the genetic algorithm must explore a wider space of solutions before finding good individuals, and this requires more generations before reaching convergence.

As a final observation, we note that for both the scenarios, convergence is quite fast, with the objective function almost reaching the optimal value in little more than 75 generations. This result is interesting because it means that the genetic algorithm is able to explore the solution space in a small amount of time, reaching the proximity of the optimum (even if the actual optimum value may require more generations to be found). In terms of execution time, the time for the genetic algorithm to reach a value within 2% of the optimum is in the order of 15 seconds.

5.4 Sensitivity to mutation and crossover probability

As a further important analysis, we carry out a sensitivity of the genetic algorithm with respect to the two probabilities that define the evolution of the population (the mutation probability P_{mut} and the crossover probability P_{cross}) to understand whether the capability of the genetic algorithm to rapidly reach optimal solutions just occurs for a properly tuned algorithm or if the property of fast convergence is stable.

The first analysis evaluates the impact of the mutation probability P_{mut} in the two considered scenarios. The results are shown in Figure 10, reporting the value of the discrepancy ϵ^{GA} between the GA best fitness and the theoretical optimal bound as a function of the mutation probability. The graph also shows the number of generations

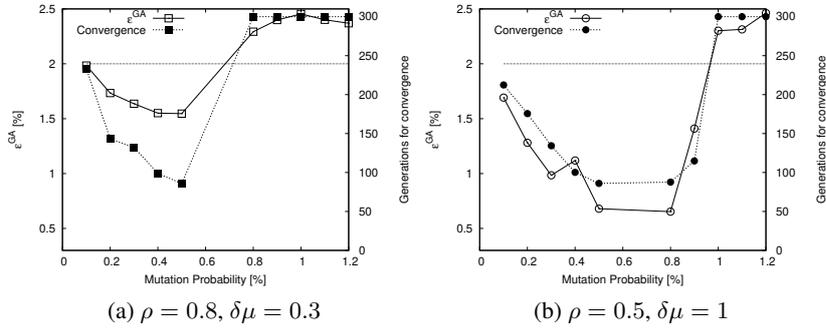


Fig. 10: Sensitivity to mutation probability

necessary to reach the convergence, that means a value of ϵ^{GA} below 2%, represented through the horizontal line. For both scenarios, we observe U-shaped curves, where both low values and high values of P_{mut} result in the algorithm reaching higher value of ϵ^{GA} . In particular, the major challenges are encountered for high values of P_{mut} , when the GA is not able to reach convergence. On the other hand, values in the range $0.4\% \leq P_{mut} \leq 0.8\%$ provide good performance. This behavior is explained considering the two-fold impact of mutations. On one hand, a low value of P_{mut} hinders the ability to explore the solutions space by creating variations in the genetic pool. On the other hand, an higher mutation rate may simply reduce the ability of the algorithm to converge, because the population keeps changing too rapidly and good genes cannot be passed through the generations. If we now compare the results for the two different scenarios, we observe that they lead to a similar message, with the only difference of having a smaller range of intermediate P_{mut} values giving good performance. However, this is consistent with the major challenges posed by the scenario with high system load ($\rho = 0.8$).

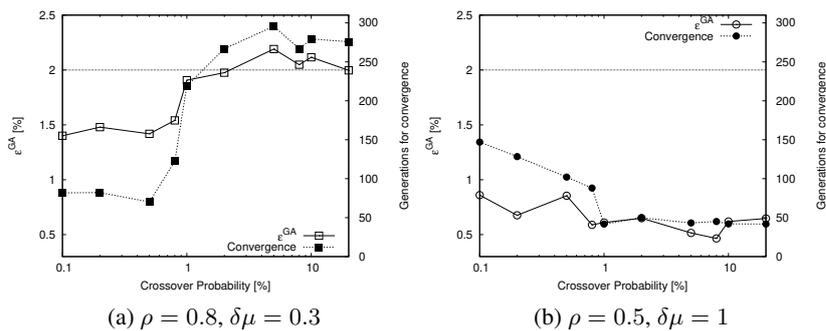


Fig. 11: Sensitivity to crossover probability

The second analysis shown in Fig. 11 evaluates the impact of the probability of selecting an individual for a crossover operation P_{cross} . Again we show both ϵ^{GA} and the number of generations to reach convergence as a function of this parameter. Since we change the crossover probability over a large range of values (from 0.1% to 20%), we use a logarithmic scale for the x -axis. First of all, we observe that also the crossover probability has a major impact on the performance of the GA. However, in this case the results show two very different behaviors for the considered scenarios. In the scenario with $\rho = 0.5$ (Fig. 11b) the algorithm reaches convergence quite fast showing very low values of ϵ^{GA} for every value of P_{cross} . On the other hand, in the scenario with $\rho = 0.8$ (Fig. 11a) the crossover probability has a very significant impact on the GA performance, hindering the capability to converge for high values of P_{cross} . This is due to the fact that with a high value of ρ , little variations in the solutions can cause significant oscillations of the objective function due to the high risk of overload in the fog nodes, as already pointed out in the comment of Fig. 8. Hence, the high crossover probability, that is likely to increase fast changes in the population, may lead to oscillations that hinder the convergence capability. To underline this effect, we analyze the convergence of the GA heuristic in the case with for $\rho = 0.8$ and $\delta\mu = 0.3$ for different values of the crossover probability. The results are shown in Fig. 12. We can clearly observe as different values of P_{cross} lead to completely different behaviors: while for $P_{cross} = 0.8$ the continuous oscillations in the solutions do not allow to converge, for $P_{cross} = 0.5$ the behavior is quite stable and stable to reach convergence, as already noticed in Fig. 9.

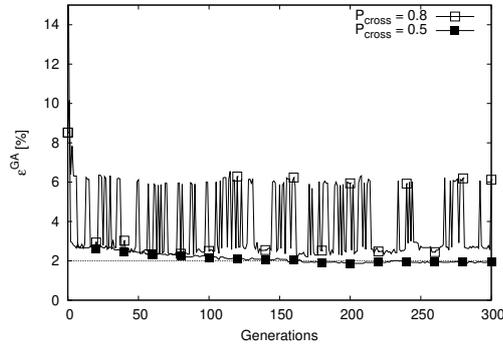


Fig. 12: Convergence analysis for different values of crossover probability

This sensitivity analysis confirms that, when the average system load is very high, reaching an optimal solution is challenging both for the GA heuristic and the solver. Anyway, the GA approach is in many cases able to achieve performance close, or even better, compared to the solver, even in this challenging scenario. However, we need to take into account that in this case the GA heuristic becomes more sensible to its main parameters, such as the crossover probability. Hence, in the most challenging

scenarios, an approach based on specifically designed ad-hoc heuristics can be worth to investigate.

5.5 Sensitivity to population size

As a final analysis, we evaluate how the population size affects the performance of the genetic algorithm. To this aim, we change the population size from 50 to 500 individuals and we measure the execution time, the difference in the objective function ϵ^{GA} , and the number of generations required to reach the convergence.

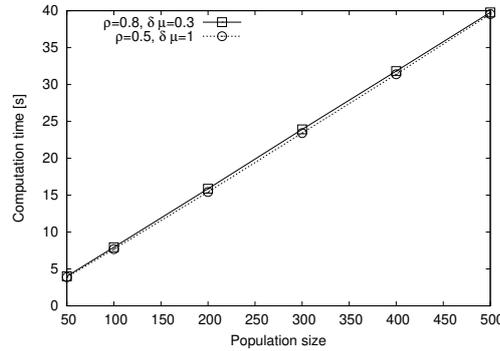


Fig. 13: GA execution time vs. population size

The first significant result, shown in Fig 13, shows the execution time of the genetic algorithm for 300 generations. Specifically, we observe a linear growth in the execution time with respect to the population size for both considered scenarios. This result is expected, as the core of the genetic algorithm iterates, for every generation, over the whole population to evaluate the individuals and for every operation, such as crossover, mutation, and selection for the next generation. The message from these experiments is that every benefit from an increase of the population should be weighted against the additional computational cost we may incur in.

Another interesting result comes from the evolution of ϵ^{GA} and of the number of generations required to reach convergence in the considered scenarios, shown in Figs. 14a and 14b. The two scenarios show similar behaviors, even if for the more challenging case of $\rho = 0.8$ we observe higher discrepancy and number of generations needed for convergence, as expected. In general, the results show that increasing the population provides a benefit as it can both reduce the minimum value of ϵ^{GA} and reduce the number of generations required for the convergence. This can be explained considering that increasing the population allows the algorithm to explore a larger portion of solution space with each generation, thus accelerating the convergence. Furthermore, it is worth to note that the benefit from increasing the population decreases as the population grows, while the execution time grows linearly, as previously discussed.

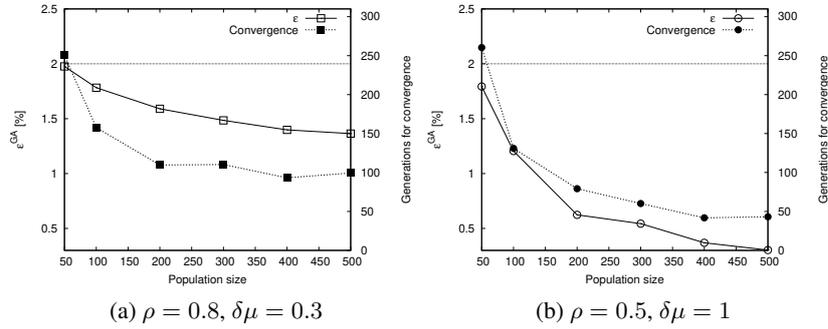


Fig. 14: Sensitivity to population size

For this reason, in our experiments we choose a population of 200 individuals, that is a value close to the knee where the benefit from a larger population decreases rapidly.

6 Conclusions

In the present chapter we discussed the design of a typical sensing application in a smart city scenario. We consider a set of sensors or other smart devices that are distributed in a geographic area and produce a significant amount of data. A traditional cloud-based scenario would send all these data on a centralized data center, with the risk of network congestion and high latency. This is clearly unacceptable for several classes of applications, that require a fast and scalable management of data packets (critical examples are the support for automatic traffic management and autonomous driving or the data collection in a widespread array of sensors). Hence, it is common to move the tasks of data aggregation, filtering and, in general, pre-processing towards the edge of the network, where a layer of computing nodes called *fog nodes* is placed.

The introduction of the fog layer in an infrastructure motivates our study, that concerns the mapping of data flows from the sensors to the fog nodes. We tackle this problem, providing a formal model as an optimization problem that aims to minimize the average response time experienced in the system taking into account both network latency and processing time. Furthermore, we present an heuristic approach that leverages the evolutionary programming paradigm to solve the problem.

Using a smart city scenario based on a realistic testbed, we validate our proposal. First, we analyze the problem using a solver to explore the impact of the parameters that affect the problem solution. Next, we demonstrate the ability of the proposed heuristic to solve the problem in a fast and scalable way. Finally, we provide a sensitivity analysis to the main parameters of the genetic algorithm to explore the limits of its stability.

References

1. Knitro Website, available at <https://www.artelys.com/solvers/knitro/>, last accessed on 10th Jul 2019

2. AMPL: Streamlined modeling for real optimization (2018), available at <https://ampl.com/>, last accessed on 10th Jul 2019
3. Ardagna, D., Ciavotta, M., Lancellotti, R., Guerriero, M.: A hierarchical receding horizon algorithm for qos-driven control of multi-iaas applications. *IEEE Transactions on Cloud Computing* pp. 1–1 (2018)
4. Binitha, S., Sathya, S.S., et al.: A survey of bio inspired optimization algorithms. *International Journal of Soft Computing and Engineering* 2(2), 137–151 (2012)
5. Canali, C., Lancellotti, R.: 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)
6. Cardellini, V., Grassi, V., Lo Presti, F., Nardelli, M.: Optimal operator placement for distributed stream processing applications. In: *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*. pp. 69–80. DEBS '16, ACM, New York, NY, USA (2016), <http://doi.acm.org/10.1145/2933267.2933312>
7. DEAP: Distributed Evolutionary Algorithms in Python (2018), – <https://deap.readthedocs.io>
8. Deng, R., Lu, R., Lai, C., Luan, T.H., Liang, H.: Optimal Workload Allocation in Fog-Cloud Computing Toward Balanced Delay and Power Consumption. *IEEE Internet of Things Journal* 3(6), 1171–1181 (Dec 2016)
9. Duan, H., Chen, C., Min, G., Wu, Y.: Energy-aware scheduling of virtual machines in heterogeneous cloud computing systems. *Future Generation Computer Systems* 74, 142 – 150 (2017), <http://www.sciencedirect.com/science/article/pii/S0167739X16300292>
10. Karimi, M.B., Isazadeh, A., Rahmani, A.M.: Qos-aware service composition in cloud computing using data mining techniques and genetic algorithm. *J. Supercomput.* 73(4), 1387–1415 (Apr 2017), <https://doi.org/10.1007/s11227-016-1814-8>
11. Liu, J., Li, J., Zhang, L., Dai, F., Zhang, Y., Meng, X., Shen, J.: Secure intelligent traffic light control using fog computing. *Future Generation Computer Systems* 78, 817 – 824 (2018), <http://www.sciencedirect.com/science/article/pii/S0167739X17302157>
12. Noshay, M., Ibrahim, A., Ali, H.: Optimization of live virtual machine migration in cloud computing: A survey and future directions. *Journal of Network and Computer Applications* 110, 1–10 (2018)
13. Sasaki, K., Suzuki, N., Makido, S., Nakao, A.: Vehicle control system coordinated between cloud and mobile edge computing. In: *2016 55th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*. pp. 1122–1127 (Sept 2016)
14. Shojafar, M., Canali, C., Lancellotti, R.: A Computation- and Network-Aware Energy Optimization Model for Virtual Machines Allocation. In: *Proc. of International Conference on Cloud Computing and Services Science (CLOSER 2017)*. Porto, Portugal (Apr 2017)
15. Tang, B., Chen, Z., Hefferman, G., Wei, T., He, H., Yang, Q.: A hierarchical distributed fog computing architecture for big data analysis in smart cities. In: *Proceedings of the ASE BigData & SocialInformatics 2015*. pp. 28:1–28:6. ASE BD&SI '15, ACM, New York, NY, USA (2015), <http://doi.acm.org/10.1145/2818869.2818898>
16. Trafair Project Staff: Forecast of the impact by local emissions at an urban micro scale by the combination of lagrangian modelling and low cost sensing technology: the trafair project. In: *Proc. of 19th International conference on Harmonisation within Atmospheric Dispersion Modelling for Regulatory Purposes*. Bruges, Belgium (June 2019)
17. Wen, Z., Yang, R., Garraghan, P., Lin, T., Xu, J., Rovatsos, M.: Fog orchestration for internet of things services. *IEEE Internet Computing* 21(2), 16–24 (Mar 2017)
18. Yi, S., Li, C., Li, Q.: A survey of fog computing: Concepts, applications and issues. In: *Proceedings of the 2015 Workshop on Mobile Big Data*. pp. 37–42. Mobidata '15, ACM, New York, NY, USA (2015), <http://doi.acm.org/10.1145/2757384.2757397>

19. Yousefpour, A., Ishigaki, G., Jue, J.P.: Fog computing: Towards minimizing delay in the internet of things. In: 2017 IEEE International Conference on Edge Computing (EDGE). pp. 17–24 (June 2017)
20. Yusoh, Z.I.M., Tang, M.: A penalty-based genetic algorithm for the composite saas placement problem in the cloud. In: IEEE Congress on Evolutionary Computation. pp. 1–8 (July 2010)