An analysis of Genetic Algorithms to support the management of edge computing infrastructures

Claudia Canali
Dept. of Engineering "Enzo Ferrari"
University of Modena
and Reggio Emilia
Modena, Italy
claudia.canali@unimore.it

Riccardo Lancellotti
Dept. of Engineering "Enzo Ferrari"
University of Modena
and Reggio Emilia
Modena, Italy
riccardo.lancellotti@unimore.it

Riccardo Mescoli
Dept. of Engineering "Enzo Ferrari"
University of Modena
and Reggio Emilia
Modena, Italy
riccardo.mescoli@unimore.it

***Abstract*—Edge computing is a novel paradigm aiming to push computation as close as possible to the data sources and to the end users. This paradigm is extremely important in areas such as the mobile applications and IoT. Key characteristic of edge computing are the limited computational resource of the edge nodes and the presence of non-negligible network delays that can affect the performance. Applications are typically designed as a set of inter-operating micro-services, where each service must be placed on node in order to minimize network latency, while balancing the load distribution to on the nodes to avoid the violation the the Service Level Agreements. An additional goal is to minimize energy consumption, meaning that the number of powered-on edge nodes must be kept as low as possible. For these reasons, the problem of micro-service placement over an edge computing infrastructure is complex and must be solved by means of heuristics that must reach suitable solutions under a wide set of operating conditions. We propose a mechanism to solve the placement problem based on genetic algorithms to solve the micro-service placement problem and we analyze the behavior of such heuristic for a wide set of problem characteristics. The proposed algorithm can automatically identify the subset of edge nodes to use and can allocate micro-services to reduce network delays and balance the load. Our experiments demonstrate that the proposed GA is a viable tool to allocate micro-services in edge infrastructures as it can find feasible solutions with a limited number of generations and can consistently minimize the power consumption of the edge infrastructure.**

I. Introduction

The success of the edge computing paradigm lies in its ability to push computation as close as possible to the clients. This is vital for some classes of applications such as latency-sensitive tasks where the network delay to and from a cloud data center is not acceptable. Another class of applications that can benefit form the edge computing approach is the class of data-intensive and distributed tasks as in the case of IoT where a huge number of sensors produce a unprecedented mass of data that must be filtered and aggregated before being used for more refined data analyses. Another common characteristic of modern software is that they do not rely on a classic monolithic architecture but are composed of micro-services that crate a pipeline to process client requests and data. The combination of many micro-services composing an application and a distributed architecture of edge nodes determines the need to map these micro-services over the infrastructure. This mapping problem must satisfy different constraints such as avoiding overload on the edge nodes (that are typically limited from a computational resource point of view); reducing network-related delays and, more generally, guaranteeing a response time below a give Service Level Agreement (SLA); reducing the energy consumption of the infrastructure, turning off under-utilized edge nodes.

The result is a multi-objective optimization problem that may exhibit non-linearity and non-convexty. The nature of the problem prevents the use of classical optimization techniques for scalability reasons and must relay on heuristics to find a suitable approximation of the optimal solution. Evolutionary algorithms have been proposed in literature as a viable solution to tackle this type of problems. However, most heuristics rely on a set of parameters that should be properly tuned to ensure that the problem can be solved effectively and efficiently. In this paper we analyze the use of Genetic Algorithms supporting the placement of micro-services on an edge computing infrastructure. We can summarize the main contributions of our research as:

- We define formally an optimization problem for the placement of micro-services on a distributed architecture of edge nodes. Our problem definition include (1) an energy model to measure energy consumption of the infrastructure; (2) a Queuing theory-based performance model to predict the response time of applications in the presence of multiply co-located micro-services contending the computational resources of the edge node; the performance model captures also the impact of network-related delays in the execution of the micro-services; (3) a set of constraints including SLA requirements on application response time;
- We define an heuristic based on Genetic Algorithms to solve these problems. We define the chromosome structure and the main genetic operators used in the GA tailoring the solution to the problem and introducing a mechanism to automatically determine how many and which nodes should be turned on or off.
- We thoroughly test our algorithms with respect to (1) model parameters, and (2) different operating scenarios,

to evaluate weather our proposal can provide stable and scalable performance in terms of solving problem instances.

The remaining part of this paper is structured as follows. Sec. II discusses the related work on the use of optimization heuristics in problems similar to the one we are considering. Sec. III provides a formal definition of the problem including energy and performance models. Sec. IV discusses the proposed heuristic based on Genetic Algorithms. Sec. V analyze the performance and the stability of the proposed heuristic. Finally, Sec. VI provides some concluding remarks and discusses the future works.

II. Related work

The scientific literature presents various mechanisms for service placement over the geographically distributed nodes of an edge computing infrastructure. Many of them typically start with the simplifying assumption that an IoT application comprises a single micro-service [1]–[3].

More interestingly and realistically, more recent studies have considered IoT applications modeled as micro-service chains for deployment over edge nodes, proposing both distributed and centralized solutions. Among proposals of fully distributed approaches, we mention the study in [4], where authors sought to optimize energy consumption and communication costs through a game-theoretic approximation method. The work in [5] introduced a cooperative scheme that allowed edge nodes to determine the optimal distribution and processing of requests among themselves, enhancing response time.

On the other hand, research efforts devoted to investigate micro-service placement in edge environments mainly include centralized approaches, able to achieve high performance in solving the multi-objective placement problem over the whole distributed infrastructure. A centralized approach was explored in [6], where the authors formulated a mixed-integer non-linear programming model aimed at minimizing application completion time by simultaneously addressing task placement and scheduling.

Deng et al. [7] address the scheduling problem of applications based micro-services, aiming to minimize deployment costs while adhering to resource constraints and ensuring the expected response time of requested services. This placement issue is formulated as an Integer Linear Programming (ILP) model. Given the likely NP-hard nature of the micro-service placement problem, exact optimization solutions become impractical for large-scale instances. Consequently, to manage placement complexity effectively, many researchers propose optimal/sub-optimal solutions to multi-objective placement problems, being the Genetic Algorithms and the Particle Swarm Optimization the preferred approaches [8]–[10].

In [8] authors propose a solution based on Particle Swarm Optimization aimed at maximizing the satisfaction of multiple QoS parameters in a context characterized by limited edge resources. Guerrero *et al.* [9] compare three evolutionary algorithms, Weighted Sum Genetic Algorithm (WSGA), non-dominated sorting genetic algorithm (NSGA-II) and multi-objective evolutionary algorithm based on decomposition (MOEA/D), for solving service placement focusing on optimizing latency, service spread and use of resources. In [10] an improved Genetic Algorithm is proposed for micro-service placement with the goal of costs minimization. In [11] an elitism-based genetic algorithm (EGA) is proposed to guarantee QoS of IoT applications in manufacturing environments.

A problem adopting a similar model has been proposed in [12]. However, in this study we enhance the problem definition adding a multi-objective approach with an energy model that was not present in the previous study. Furthermore, we re-define the GA model with a different chromosome structure and different genetic operators to cope with the on/off status of edge node and to enable the search for the optimal subset of edge node to maintain active.

III. Problem definition

A. Problem overview

We now introduce a formal model for the the central problem of our research. To simplify the understanding of the model, we summarize the symbols used in Tab. I

TABLE I
Notation and parameters for the proposed model.

Model parameters	
$\mathcal{M}$	Set of micro-services
$\mathcal{M}_a$	Set of micro-services composing application a
$\mathcal{E}$	Set of edge nodes
$\mathcal{A}$	Set of applications
λ_m	Request rate to micro-service m
λ_e	Request rate to edge node e
λ_a	Request rate to application a
Λ	Global request rate
S_m	Average service time for micro-service m
σ_m	Standard deviation of S_m
C_e	Computational power of edge node e
W_e	Average waiting time on edge node e
S_e	Average service time on edge node e
σ_e	Standard deviation of S_e
R_a	Average response time for application a
T_a^{SLA}	SLA requirement of application a
$o_{m,n}$	execution order of micro-services m, n
$\delta_{e,f}$	network delay between nodes e and f
P_e^0	Power consumption of node e when idle
P_e^M	Maximum power consumption of node e
Model indices	
e	An edge node
a	An application
m	A micro-service
t	A time slot
Status variables	
$x_{m,e}$	Allocation of micro-service m to edge e
y_e	Use of edge node e

We consider a collection of applications, each consisting of a series of micro-services. Let a represent an application

within the set $\mathcal{A}$ of applications. We assume to have a set of applications that are composed by a sequence of micro-services. Let a be a generic application belonging to a set $\mathcal{A}$ of applications. The application consists of a sequence of micro-services $m \in \mathcal{M}_a$ that need to be executed. The application receives requests at a rate of λ_a, and the user-perceived response time is R_a. We assume that the Service Level Agreement for each application a mandates that the average response time must be $R_a \leq T_a^{SLA}$.

Each micro-service $m \in \mathcal{M}$ within an application is defined by its average service time S_m and the standard deviation of its execution time, σ_m.

Micro-services applications are deployed across an infrastructure of edge nodes $e \in \mathcal{E}$. The computational capacity of each fog node provides a speedup C_e, resulting in an average service time for micro-service m of S_m/P_e when executed on edge node e.

In this paper, we assume that the power consumption of an edge node can be represented by a linear model, where the power usage varies from P_e^0 (the power consumption when the edge node e is idle) to P_e^M (the power consumption when the node reaches 100% utilization).

We characterize the micro-service placement problem as a multi-objective optimization with two distinct objectives:

- Minimize power consumption of the infrastructure
- Minimize average execution for each application modeled as a chain of micro-services

The placement of micro-services on nodes is represented by a matrix $x_{m,e}$, where $x_{m,e} = 1$ indicates that micro-service m is hosted on node e, and $x_{m,e} = 0$ indicates that the micro-service is hosted elsewhere. Nodes can be switched on $y_e = 1$ or off $y_e = 0$. Edge nodes that are turned off will not host any micro-services, meaning $x_{m,e} = 0 \forall m \in \mathcal{M}$.

Throughout the paper, we refer to Table I as a summary for the notation used in the model.

B. Energy model

We assume that the power consumption P_e of edge node e depends solely on its utilization, following a linear function that correlates utilization with power consumption.

$$P_e = y_e P_e^0 + (P_e^M - P_e^0)\lambda_e S_e \tag{1}$$

The energy consumption of edge node e, as described by Eq. (1), is influenced by two main factors: a baseline power when the node is active (i.e., $y_e = 1$, and an additional component that grows with the node utilization, represented by $\lambda_e S_e$.

It's important to note that, in order to minimize power consumption, and assuming the edge infrastructure consists of homogeneous nodes (or that $P_e^M - P_e^0 \ll P_e^0$), the problem simplifies to minimizing the number of active edge nodes, represented by $\sum y_e$.

C. Performance model

We base the model for the micro.service performance on the model proposed in [12].

Let's concentrate on a single active edge node. To model this edge node using queuing theory, we need to account for the service process at server e, which will follow a mixture of distributions corresponding to the hosted micro-services. This mixture will have an average service time S_e and a standard deviation σ_e, defined as follows:

$$S_e = \frac{1}{C_e} \cdot \sum_{m \in \mathcal{M}} x_{m,e} \frac{\lambda_m}{\lambda_e} S_m \tag{2}$$

$$\sigma_e = \sqrt{\left(\frac{1}{C_e^2} \cdot \sum_{m \in \mathcal{M}} x_{m,e} \frac{\lambda_m}{\lambda_e} (S_m^2 + \sigma_m^2) \right) - S_e^2} \tag{3}$$

Where the arrival request rate at node e, denoted as λ_e, is defined by the equation: $\lambda_e = \sum_{m \in \mathcal{M}} x_{m,e} \lambda_m$.

Assuming the arrival process adheres to a Poisson distribution, we can utilize the Pollaczek-Khinchin formula to characterize the waiting time W_e in an M/G/1 system.

$$W_e = \frac{S_e^2 + \sigma_e^2}{2} \cdot \frac{\lambda_e}{1 - \lambda_e S_e} \tag{4}$$

Previous research [12] has shown that this model remains accurate when either multiple micro-services are hosted on the same edge node or when the coefficient of variation (the ratio of standard deviation to mean) of the micro-services' service time is approximately 1.

We can hence express the response time of an application a as the total of: (1) the service times of the individual micro-services, (2) the waiting time at the edge nodes where the micro-services are hosted, and (3) the network delays. This result can be formalized as follows:

$$R_a = \sum_{\substack{m \in \mathcal{M}_a \\ e \in \mathcal{E}}} x_{m,e} \left(W_e + \frac{S_m}{P_e} \right) + \\ + \sum_{\substack{m,n \in \mathcal{M}_a \\ e,f \in \mathcal{E}}} o_{m,n} \cdot x_{m,e} \cdot x_{n,f} \cdot \delta_{e,f} \tag{5}$$

where $o_{m,n} = 1 \iff m \lessdot n$, that is when service m is invoked just before n in the service chain describing the application.

D. Problem formulation

Based on the previously outlined model for energy and performance of edge infrastructure, we can formulate the problem of statically placing micro-services on the infrastructure as follows:

$$\min \mathrm{En} = \sum_{e \in \mathcal{E}} P_e \tag{6}$$

$$\min \mathrm{Perf} = \sum_{a \in \mathcal{A}} \frac{\lambda_a}{\Lambda} R_a \tag{7}$$

subject to:

$$\sum_{e \in \mathcal{E}} x_{m,e} = 1 \quad \forall m \in \mathcal{M}, \tag{8}$$

$$\lambda_e S_e \leq y_e(1-\epsilon) \quad \forall e \in \mathcal{E}, \tag{9}$$

$$R_a \leq T_a^{SLA} \quad \forall a \in \mathcal{A}, \tag{10}$$

$$x_{m,e}, y_e = \{0,1\}, \quad \forall m \in \mathcal{M}, e \in \mathcal{E}, \tag{11}$$

The decision variables of the problem are $x_{m,e}$ describing the micro-service placement and y_e defining the power state of the edge nodes.

Eq. (6) represents the primary objective function, focusing on reducing the infrastructure energy consumption. The power consumption, denoted as P_e, is specified in equation (1). The secondary objective function, which is optimized provided the power consumption remains minimal, concerns the average response time of applications, weighted by their invocation frequency. The response time, R_a, is derived from the performance model in equation (5). These two objectives can be merged into a single objective function, $\mathrm{Obj} = \mathrm{En} + \epsilon \cdot \mathrm{Perf}$, where ϵ is a small constant indicating that the secondary goal is considered only if the energy consumption is minimized.

The following constraints define the boundaries of the optimization problem solution space:

- Eq. (8) ensures that each micro-service is assigned to one and only one fog node;
- Eq. (9) ensures that each fog node does not experience an overload (where $\lambda_e S_e$ represents the load on edge node e). The constraint requires that for nodes that are turned off (i.e., when $y_e = 0$), the load must be zero. For active nodes the load must stay strictly below 1 (the formulation uses an arbitrarily small value ϵ to this aim);
- Eq. (10) represents the SLA requirement for each application. Following the approach in the literature [13], we define T_a^{SLA} as $K \cdot \sum_{m \in \mathcal{M}_a} S_m$, with K set to 10;
- Eq. (11), models that the decision variables $x_{m,e}$ and y_e are Boolean in nature.

IV. Genetic Algorithm solver

A. Genetic Algorithm overview

Genetic Algorithms (GAs) are a subclass of meta-heuristic optimization algorithms, which belong of the broader class of Evolutionary Algorithms (EAs). GAs are closely inspired by the principles of natural selection and the field of genetics at large. Specifically, GAs operate on a population of potential solutions, referred to as individuals. Every individual k encoded using a sequence $C^k = \{c_l^k\}$, named "chromosome" (in some cases the encoding could involve multiple chromosomes, making the genome and the chromosome two clearly distinct concepts), where each element c_l^k, known as a "gene", corresponds to a specific characteristic of the associated solution.

The algorithm progresses through an iterative process involving successive generations of individuals. The process starts with a population of randomly generated individuals, each of which is scored by the algorithm using a fitness function which corresponds to the objective function of the optimization problem itself. Starting from these initial conditions, the population undergoes a process of evolution that lasts a determined amount of cycles or until a specific condition is met, aiming to improve the overall fitness score of the population. The main tools adopted to perform the evolution process are the following:

- *Mutation* operator: It performs a random change to the chromosome of a specific individual by altering one or more genes. The main purpose of the operator is to facilitate the exploration of new areas of the solution space by introducing novel information to the existing genetic pool. This process helps reducing the likelihood of the algorithm converging prematurely to a local maximum of the fitness function.
- *Crossover* operator: Given two parent chromosomes it creates new individuals by merging complementary sections of the parents into a new set of "offspring chromosomes". This operation allows successful genes or combinations of genes to propagate throughout the population over successive generations.
- Selection operator: Given a set population of individuals from the K^{th} generation it determines which individual will be passed to the next $(K+1)^{th}$ generation. The selection criterion is typically based on the fitness score of each chromosome and it involves a random extraction process, where the likelihood of an individual to be picked related to its fitness. Selection mechanisms with such characteristics ensure that individuals with higher fitness scores have a better chance of contributing to the subsequent generations.

The way the operators just described are applied is determined by another important characteristic of GAs, the *evolutionary strategy*, which defines in detail how the operators are ordered and combined during the evolutionary process.

B. Chromosome structure

In order to exhaustively describe every potential solution using a compact chromosomal encoding, we designed a structure where the information encoded in each gene is a combination of its specific value and the values codified by the genes in its proximity, with the whole chromosome being a description of the allocation of each micro-service $m \in \mathcal{M}$ to a specific edge node $e \in \mathcal{E}$ in the network.

More in detail, each gene $c_l^k \in C^k$ can take any value $c \in \mathcal{C} = \{c | c \in \mathbf{N}, 0 \leq c < (|\mathcal{M}| + |\mathcal{E}|)\}$ where values in the subset $\mathcal{C}_\mathcal{M} = \{c_m | c_m \in \mathcal{C}, c_m < |\mathcal{M}|\}$ represent a micro-service $m \in \mathcal{M}$ and values $c_e \in \mathcal{C}_\mathcal{E} = \mathcal{C} \setminus \mathcal{C}_\mathcal{M}$ represent an edge node $e \in \mathcal{E}$. Then, to decode the solution represented by a chromosome, any gene $c_{l_m}^k = c_m \in \mathcal{C}_\mathcal{M}$ must be interpreted

in relation to the values of the preceding genes by identifying the gene $c^k_{l^*}$ with $l^* = max_l(\{l|c^k_{l_e} = c_e \in \mathcal{C}_\mathcal{E}, l_e < l_m\})$ (the gene representing an edge node closest to $c^k_{l_m}$ from the left in the sequence), which identifies the edge node for which $x_{m,e} = 1$ in the solution.

A consequence of this structured representation of the solution is that, for it to represent a valid solution, every chromosome must have $c^k_0 = c_e \in C_\mathcal{E}$.

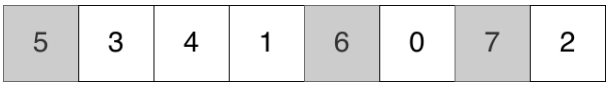

Fig. 1. Gene example

An example of a chromosome is provide din Fig. 1 where 5 micro-services represented with numbers from 0 to 4 are placed over three edge nodes (the corresponding genes are marked with gray color and have a value between 5, corresponding to edge node 0, and 7, corresponding to edge node 2. Edge node 0 hosts micro-services 3, 4, and 1; edge node 1 hosts micro-service 0, and edge node 2 hosts micro-service 2.

Every individual of the randomly generated population adopted at generation $K = 0$ is a sequence of edge nodes and micro-services where the first element always represents an edge node and has a random value in $\mathcal{C}_\mathcal{E}$ and the rest of the sequence is a random arrangement of every other element of $|\mathcal{C}|$.

C. Evolutionary strategy

To choice of evolutionary strategy adopted for the experimentation of this work fell on the *simple strategy* described in [12]. The characteristic of the strategy is that, contrary to other popular methods, such as the "Mu Plus Lambda strategy $(\mu + \lambda)$" [12], it first performs a selection operation over the population of the previous generation, with likelihood of extraction proportional to the individual's fitness score. This approach generates an initial offspring population where fitter individuals are more likely to be represented with multiple replicas and worse performing individuals have a high chance of being completely excluded from the evolutionary process. Once this initial offspring has been obtained, the newly sampled individuals undergo a process where they can be mutated or take part in a crossover operation with predetermined probabilities P_{mut} and P_{cx} respectively. The individuals resulting from mutation and crossover operations replace the initial individuals used as a basis for the operations themselves, thus obtaining the final offspring that will be used as the population for the successive $(K + 1)^{th}$ generation.

D. Chromosome normalization

After any operation affecting the structure of a chromosome there is a tangible possibility for any of the edge nodes encoded in the sequence to suddenly get emptied of any micro-service. This can happen in any case where there is a gene $c^k_l = c_e \in \mathcal{C}_\mathcal{E}$ followed by $c^k_{l+1} = c_m \in \mathcal{C}_\mathcal{M}$ that swaps its value to $c^k_{l+1} = c_e \in \mathcal{C}_\mathcal{E}$, thus not leaving any gene encoding a micro-service in between, making the node encoded by c^k_l without any allocated micro-service. Such chromosome is functionally identical to a chromosome where the gene c^k_l is missing.

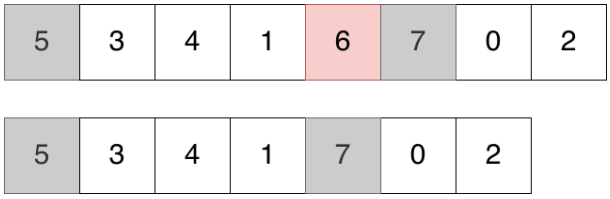

Fig. 2. Normalization example

The *chromosome normalization* process is precisely the process of removing the first gene in sequences where two edge nodes appear consecutively. An example of such process is provided in Fig. 2: in the upper chromosome the edge node 1 (the gene contains the value of 6 and is highlighted with a pink color) is not hosting any micro-service and can be safely be removed from the chromosome. The normalized chromosome is shown in the lower part of the figure.

This operation brings multiple advantages, the most relevant being the removal of unnecessary information from the encoded solution and the introduction of an incentive to converge towards simpler (typically more energy-efficient) solutions. It is worth to note, however, that such choice requires a careful modification of the other genetic operators (crossover in particular) to cope with chromosome in the genetic pool that can have different lengths.

E. Mutation Operator

Among the many mutation operators known in literature, such as: Uniform integer, random variable, directed, etc., the method used for the experimentation presented is a *shuffle mutation operator*, which, with probability P_{mut} for each chromosome, takes a random gene of the sequence and swaps its value with the value of one of the following genes in the sequence.

It's important to note that, given the structure explained in IV-B, a swap between the value of the first gene of the sequence and a gene with a value in $\mathcal{C}_\mathcal{M}$ would turn the sequence into an encoding of an invalid solution. To avoid this issue, in case the first gene is picked by the operator, the swap will consider only genes with value in $\mathcal{C}_\mathcal{E}$ to make sure the sequence still represents a valid solution.

After the mutation process is done, the resulting chromosome is normalized as described in IV-D.

F. Crossover Operator

The implemented crossover operator is a modified *ordered crossover* method [12] adapted to handle the peculiarities of chromosome structure defined in IV-B and the possibility of parent sequences of different length, consequence of the normalization process described in IV-D.

The process involves two parent chromosomes C^k_{p1}, C^k_{p2} that will be combined in order to generate an equal amount of child

sequences C^k_{c1}, C^k_{c2}, and can be expressed with the following steps:

1) Initialize the two child sequences C^k_{c1}, C^k_{c2} as "empty" sequences of length $|C^k_{p1}|$ and $|C^k_{p2}|$ respectively.
2) Generate two random indexes l_{st}, l_{end} such that: $1 \leq l_{st} < l_{end} < min(\{|C^k_{p1}|, |C^k_{p2}|\})$ These indexes will identify a contiguous sub-sequence of each parent that will be directly inherited by one of the children.
3) The child sequence C^k_{c1} (C^k_{c2} resp.) directly inherits from parent C^k_{p1} (C^k_{p2} resp.) both the first gene of (ensures it represents an edge node) and the sub-sequence identified at step 2 at the same positions they had in the parent sequence.
4) Fill in order by index the "empty" genes of C^k_{c1} (C^k_{c2} resp.) by extracting from parent C^k_{p2} (C^k_{p1} resp.) all gene values not already present in the destination child (the values are extracted in the same order they appear in the source parent), in case the values that can be extracted from the parent exceed the free positions in the child, the remaining values are appended at the end of the child sequence, making it longer.
5) Similarly to step 4 the remaining "empty" genes of C^k_{c1} (C^k_{c2} resp.) are filled with the values from the same parent sequence they inherited from at step 3, using the same logic as step 4 in case of an excess of new values to add from the parent.
6) The resulting child sequences undergo the normalization process described in IV-D.

A brief example of the crossover process can be seen in Fig. 3 and Fig. 4, with the first picture showing a pair of parent sequences, along with the selected sub-sequences (delimited by two dotted lines), and the second representing the pair of child sequences obtained at the end of the process. It is important to note the second child in Fig. 4 ended up with an empty edge node at the end of the sequence. That gene will be subject to removal during the normalization process at step 6.

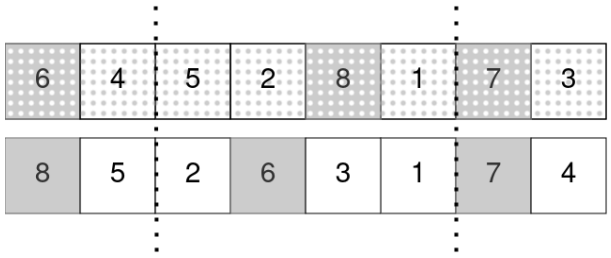

Fig. 3. Parent sub-sequence selection

G. Selection Operator

The selection of the individuals from generation K^{th} to use to generate the population for $(K+1)^{th}$ generation of the algorithm is performed using a tournament selection operator with tournament size $TS = 7$, that is a typical value for this operator [12].

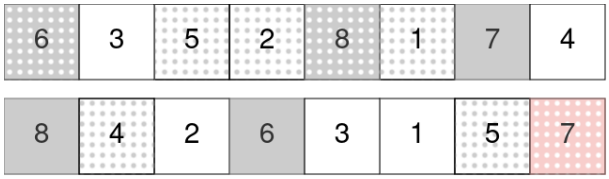

Fig. 4. Resulting child sequences (the second child has a gene up for deletion)

The way the process works is by generating each new individual by randomly extracting TS individuals from the old population and keeping only the only with the highest fitness score among those extracted. As described in IV-C the newly generated individuals will then undergo the processes of mutation and crossover with probability P_{mut} and P_{cx} respectively.

V. Experimental results

The experimentation performed has the aim to put to the test the adaptability and robustness of the proposed optimization algorithm. Each scenario tested has undergone 30 experimentation runs over networks with different, randomly generated delays, a population of 600 randomly initialized individuals and a maximum number of generations of 600. Results are provided in the form of average value and standard deviation.

The analysis of the experimental results is divided in a first set of tests (Sec. V-A) aiming to tune the algorithm parameter and to evaluate how much algorithm parameters affect the algorithm performance over multiple scenarios; a second set of experiments (Sec. V-B) evaluate the stability of the algorithm performance depending on problem characteristics.

The main performance metrics used in the paper concerns the two objective functions (that is energy consumption and response time) as well as the number of generation in the GA required to converge. In particular we consider that the algorithm convergence is reached when the least significant objective function (that is response time) has reached a value within 1% of the best value achieved throughout the whole experimental run.

A. Genetic algorithm parameter tuning

The first experimentation step is the tuning of two fundamental parameters of the genetic algorithm, the mutation probability P_{mut} and crossover probability P_{cx}, with the aim to identify a pair of values that provided satisfying performances on a wide variety of problem sizes (problem size is is measured as the amount of edge nodes in the system $|\mathcal{E}|$). We evaluate the convergence speed for problems sizes $|\mathcal{E}| \in \{10, 15, 20\}$. For each problem size we consider different values of mutation and crossover probabilities $P_{mut}, P_{cx} \in \{0.1, 0.2, \ldots, 0.9\}$. All the problems tested shared some relevant parameters: the amount of applications $|\mathcal{A}| = 9$, with each application composed of $|\mathcal{M}_a| = 10 \quad \forall a \in \mathcal{A}$ and the system utilization defined as $\overline{\rho} = \frac{\sum_{a \in \mathcal{A}} \lambda_a \sum_{m \in \mathcal{M}_\dashv} S_m}{\sum_{e \in \mathcal{E}} C_e} = 0.6$.

Analyzing the variation in convergence of the three problems in relation to P_{mut}, shown in Fig. 5, we find that the

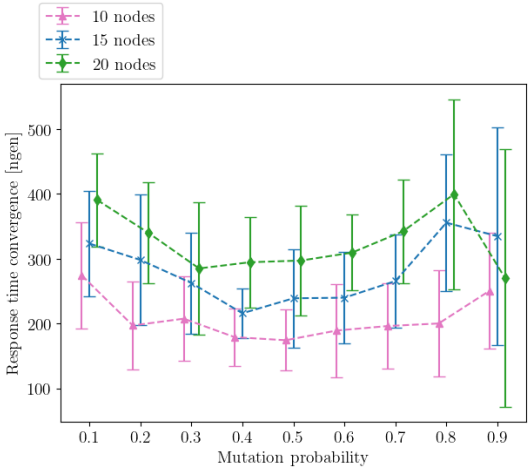

Fig. 5. Convergence speed *vs.* P_{mut}

$P^*_{mut} = 0.4$ provides the best results for every considered problem size and is therefore the value used in the subsequent analyses.

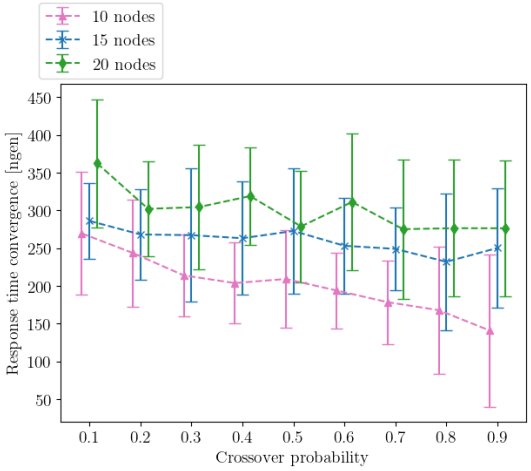

Fig. 6. Power consumption and convergence at the variation of ρ

Focusing on P_{cx} the results seem to depend on the problem size, with small problems converging faster when the crossover probability is high. However, looking at the best response time achieved, we observe that when $P_{cx} \geq 0.7$, the response time (show in Fig. 7) becomes extremely variant, meaning that the final solution is no longer identified correctly. For this reason, in the following experiments we select $P^*_{cx} = 0.5$.

B. Sensitivity to problem characteristics

Having tuned the GA identifying P^*_{mut} and P^*_{cx}, we evaluate the ability of the GA to provide stable results as a function of the problem size. To this aim we generate several problems with increasing number of edge nodes $|\mathcal{E}| \in \{4, 6, 8, \ldots, 20\}$. We increase proportionally also the number of applications so that the overall infrastructure utilization remains constant $\overline{\rho} = 0.6$.

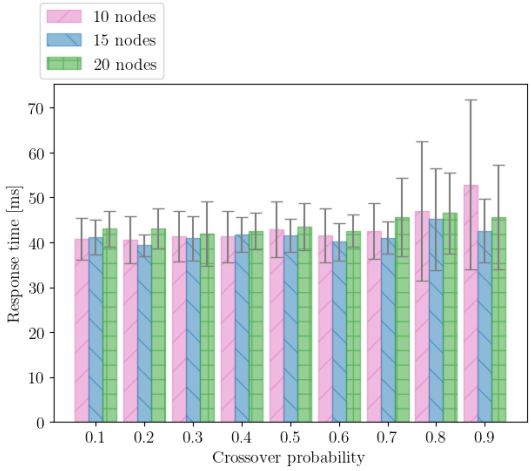

Fig. 7. Power consumption and convergence at the variation of ρ

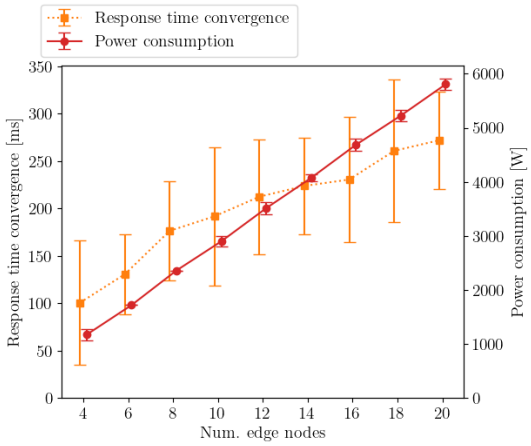

Fig. 8. Power consumption and convergence *vs.* $|\mathcal{E}|$

Fig. 8 shows that the power consumption grows linearly as more nodes and more applications are added to the infrastructure (red line in the graph). The low variance suggests that the algorithm is highly effective in identifying the best solution. Looking at the convergence speed (orange line) we observe that, as the solution space grows, the search for the best solution becomes slower, but the GA can still identify the best solution in few hundreds of generations.

The next analysis focuses on the impact of the infrastructure utilization. For this analysis (and for the subsequent ones), we consider a scenario with $|\mathcal{E}| = 15$, $|\mathcal{A}| = 9$. To evaluate the impact of infrastructure utilization we have that $\overline{\rho} \in \{0.1, 0.2, \ldots, 0.8\}$.

The results, represented in Fig. 9, demonstrated that the algorithm can consistently converge to solutions that are equivalent in power consumption and grows linearly with $\overline{\rho}$. As $\overline{\rho}$ grows, the areas of unfeasible solutions increases, making more difficult to converge, as testified by the higher number of generations required to converge, but this increase remain below 10%.

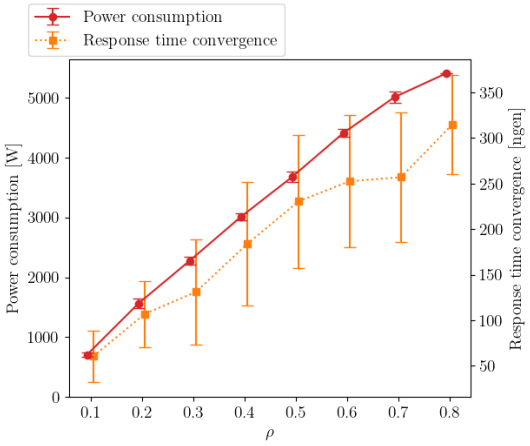

Fig. 9. Power consumption and convergence *vs.* ρ

The last experiment focuses on the behavior of the algorithm in relation to the granularity of the application (number of micro-services per application) and has the objective to highlight what the algorithm effectively prioritizes when allocating the micro-services. In particular, the subjects were problems with the same characteristics, but we change the number fo micro.services for application $|\mathcal{M}_a| \in \{2, 4, \ldots, 20\}$.

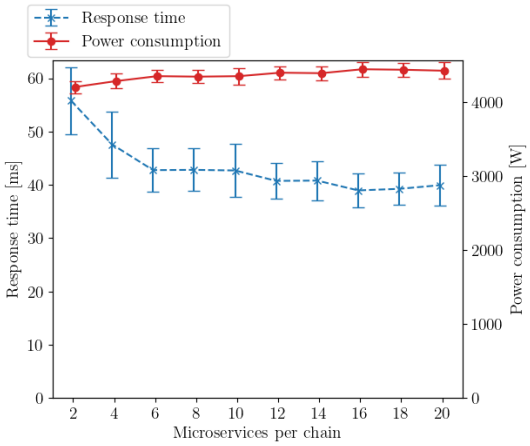

Fig. 10. Power consumption and convergence at the variation of ρ

We observe that power consumption remain more or less constant as the global service time of the application remain the same. However, when $|\mathcal{M}_a|$ is low, that is when the application is composed of just a few large micro-services. It becomes harder to achieve load balancing (because a single micro-service can lead a node in a near-overload condition. On the other hand, when we have a larger number of smaller micro-services, the algorithm is more effective in distributing the load on the edge node, achieving lower response time, thanks to the better load balancing.

VI. Conclusions and Future Work

Throughout this paper we presented an heuristic aiming to help the placement of applications composed by micro-services over a distributed infrastructure of edge nodes. Our proposal is based on a multi-objective optimization problem that considers both energy consumption and performance.

The model is embedded in a heuristics based on a Genetic Algorithm. We define the chromosome structure as well as the genetic operator specifically tailored to our problem representation.

Our test demonstrate that the proposed algorithm is easy to tune and can provide stable performance for a wide range of input problem characteristics. This is just a first step in a research line Our future work will focus on other classes on genetic algorithms, trying to define the Pareto front for the energy/performance trade-off. furthermore, we aim to explore a wider set of problem characteristics such as the impact of network-related delays on the algorithm stability.

References

[1] R. Yu, G. Xue, and X. Zhang, "Application provisioning in fog computing-enabled internet-of-things: A network perspective," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 783–791.

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

[3] 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.

[4] P. Kayal and J. Liebeherr, "Distributed service placement in fog computing: An iterative combinatorial auction approach," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2019, pp. 2145–2156.

[5] Y. Xiao and M. Krunz, "Qoe and power efficiency tradeoff for fog computing networks with fog node cooperation," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, 2017, pp. 1–9.

[6] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, "Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system," *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3702–3712, 2016.

[7] S. Deng, Z. Xiang, J. Taheri, M. A. Khoshkholghi, J. Yin, A. Y. Zomaya, and S. Dustdar, "Optimal application deployment in resource constrained distributed edges," *IEEE Transactions on Mobile Computing*, vol. 20, no. 5, pp. 1907–1923, 2021.

[8] S. Pallewatta, V. Kostakos, and R. Buyya, "QoS-aware placement of microservices-based IoT applications in Fog computing environments," *Future Generation Computer Systems*, vol. 131, pp. 121–136, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X22000206

[9] C. Guerrero, I. Lera, and C. Juiz, "Evaluation and efficiency comparison of evolutionary algorithms for service placement optimization in fog architectures," *Future Gener. Comput. Syst.*, vol. 97, no. C, p. 131–144, Aug 2019.

[10] Z. Ding, S. Wang, and C. Jiang, "Kubernetes-oriented microservice placement with dynamic resource allocation," *IEEE Transactions on Cloud Computing*, vol. 11, no. 2, pp. 1777–1793, 2023.

[11] B. Natesha and R. M. R. Guddeti, "Adopting elitism-based Genetic Algorithm for minimizing multi-objective problems of IoT service placement in fog computing environment," *Journal of Network and Computer Applications*, vol. 178, p. 102972, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804520304239

[12] C. Canali, G. Di Modica, R. Lancellotti, S. Rossi, and D. Scotece, "A validated performance model for micro-services placement in fog systems," *SN Computer Science*, vol. 4, no. 4, 2023.

[13] 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*, pp. 1–1, 2018.