Journal of Network and Systems Management (2023) 31:81
https://doi.org/10.1007/s10922-023-09771-y

Intelligent and Resource-Conserving Service Function Chain (SFC) Embedding

Panteleimon Rodis[1] · **Panagiotis Papadimitriou**[1]

Received: 14 December 2022 / Revised: 4 August 2023 / Accepted: 16 August 2023

Abstract
Network Function Virtualization (NFV) opens us great opportunities for network processing with higher resource efficiency and flexibility. In this respect, there is an increasing need for intelligent orchestration mechanisms, such that NFV can exploit its potential and live up to its promise. Genetic algorithms have emerged as a promising alternative to the proliferation of heuristic and exact methods for the Service Function Chain (SFC) embedding problem. To this end, we design and evaluate a genetic algorithm (GA), which computes efficient embeddings with runtimes on par with approximate methods. We present a GA model as state-space search in order to clarify the design choices of a GA. Our proposed GA utilizes a heuristic for the generation of the initial population, with the aim of directing the search towards the solution. Given the sensitivity of GAs on their various parameters, we introduce a parameter adjustment framework for GA fine-tuning. A comparative evaluation among a range of GA variants with diverse features sheds light on the impact of these features on SFC embedding efficiency. The GA variant that stands out is further benchmarked against a baseline greedy algorithm and a state-of-the-art heuristic. Our evaluation results indicate that the GA yields notable gains in terms of request acceptance and resource efficiency.

Keywords NFV · Resource orchestration · Genetic algorithms · Artificial Intelligence

✉ Panteleimon Rodis
rodis@uom.edu.gr

Panagiotis Papadimitriou
papadimitriou@uom.edu.gr

[1] Department of Applied Informatics, University of Macedonia, Egnatia 156, 54636 Thessaloniki, Greece

1 Introduction

Network Function Virtulization (NFV) decouples network functions from the underlying specialized devices, known as middleboxes, which are commonly deployed in enterprise and telco networks [1–3]. As such, network functions, such as firewalls, proxies, network address translation (NAT), intrusion detection, and redundancy elimination, can be deployed using software that runs on virtualized commodity servers [4]. Such virtualized network functions (VNFs) can be either executed on the network operator's premises or can be leased from cloud providers, in the form of Network Function-as-a-Service [2, 5–8]. In this respect, NFV enhances flexibility and resource efficiency compared to middleboxes, whereas it also spurs innovation by lowering the barrier for introducing new functionality into the network.

VNFs are commonly orchestrated in bundles, known as Service Function Chains (SFCs) [5, 9]. A SFC is essentially a sequence of VNFs and is expressed in the form of a VNF-graph. The latter, besides VNF connectivity, commonly encompasses resource demands (*e.g.,* computing demands for VNFs and bandwidth demands for the graph edges), which may either be directly specified by the client or may be derived from service Key Performance Indicators (KPIs).

A key requirement for the deployment of VNFs, either on private datacenters or on public cloud infrastructures (usually termed as NFV infrastructure—NFVI), is VNF-graph embedding, *i.e.,* the assignment of VNFs and the corresponding VNF-graph edges onto the respective NFVI counterparts (*i.e.,* servers and network links). The general case of this problem (*i.e.,* topology embedding) is known to be NP-hard [10] and not efficiently solvable, even when polynomial boundaries [11] or restrictions [12] are applied on the problem parameters. The problem complexity is retained even for special types of substrate network topologies [13]. VNF-graph embedding can be seen as a special case of topology embedding [14], at which the request graphs are directed (as opposed to virtual network embedding, where the virtual network graphs are commonly undirected).

VNF-graph mapping optimization has been mainly tackled using heuristics [6, 8, 14, 15] and exact methods [5, 7, 9, 16–20]. Heuristics and greedy algorithms generate embeddings with typically low solver run-times, but usually at the expense of a considerable optimality gap. On the other hand, exact methods may achieve near-optimal solutions; however, their inherent computational complexity introduces significant scalability limitations. As such, exact methods constitute a feasible approach only to small-scale embeddings.

Considering these limitations of heuristic and exact methods, Artificial intelligence (AI) and Machine Learning (ML) have recently drawn significant attention as viable alternatives to the solution of NP-hard problems [21, 22]. Numerous AI/ML-assisted methods have been particularly developed for the VNF-graph embedding problem [23]. Within the domain of AI, we particularly focus on genetic algorithms (GA), which demonstrate high potential for generating efficient solutions for this class of problems. Yet, their efficiency is not well understood, since they have been rarely employed for VNF-graph embedding.

In this respect, we study GAs as an alternative approach to the VNF-graph embedding problem, extending our previous work in [24]. In particular, we have extended our previous work as follows: (i) we analyze the behavior of GAs using a model based on state space search, (ii) we discuss in more detail the dynamic parameter adjustment of our proposed GA, (iii) we discuss related work on the application of GAs in the SFC embedding problem, (iv) we study the impact of a range of GA features on SFC embedding efficiency by comparing among a set of GA variants, and (iv) we benchmark the most efficient GA variant against a state-of-the-art method (*i.e.,* BACON [25]), uncovering notable gains in terms of resource efficiency.

In more detail, the main contributions of our work are the following:

- The operation of GAs is defined by many parameters; thereby, in order to maintain efficiency and adaptability to different problem configurations, it is necessary to adjust these parameters to each particular problem instance. To this end, we have developed a dynamic parameter adjustment procedure, which is executed in parallel with the SFC embedding, empowering the proposed GA to adapt to different problem configurations.
- In order to analyse the behavior of GAs, we have developed a modeling framework for their functionality, which is based on state space search. As such, we are in position to study the efficiency of GAs on computational problems, such as SFC embedding, and also perform the adjustment of the GA parameters. This GA model is critical for the aforementioned dynamic parameter adjustment.
- We propose a GA that incorporates innovative features, such as dynamic parameter adjustment, the structure of procedures, and a heuristic-based approach to the generation of the initial population. The latter effectively directs the state space search towards the solution, augmenting the GA. All these features combined empower the proposed GA[1] to confront the computational hardness of the SFC embedding problem.
- We scrutinize the behavior of GAs in terms of SFC embedding, by comparing a range of GA variants that differ in the features that they employ. This comparison uncovers various insights regarding the impact of GA features on SFC embedding efficiency. We further quantify the gains of the most sophisticated GA variant (*i.e., GA-PAGA*) against a baseline greedy algorithm and a state-of-the-art heuristic [25].

The remainder of the paper is organized as follows. In Sect. 2, we present the network and request models, and formulate the problem at hand. Section 3 provides background information on GAs. In Sect. 4, we present a modeling framework for the functionality and efficiency of genetic algorithms. In Sect. 5, we discuss in detail our solution for the VNF-graph embedding problem, based on GAs. In Sect. 6, we elaborate on the dynamic adjustment of the parameters for our proposed GA. In Sect. 7, we study the efficiency of selected GA variants and compare our solution

[1] The source code of our implementation is available in [26].

with state-of-the-art based on simulation results on structured topologies. Section 8 provides an overview of related work. Finally, Sect. 9 highlights our conclusions.

2 Problem Description

In this section, we discuss and formulate the SFC embedding problem. More specifically, we commence with a high-level description of the problem at hand (Sect. 2.1); subsequently, we introduce a network and request model (Sect. 2.2), followed by the problem formulation (Sect. 2.3).

2.1 SFC Embedding

Network service deployment on top of virtualized infrastructures raises the need for the placement of VNFs onto servers. In fact, a network service may encompass a number of communicating VNFs. In this respect, the communication requirements are commonly expressed in the form of SFC graphs, which represent both the associated computing and communication demands, as illustrated in Fig. 1. As such, the VNF-graph vertices (*i.e.,* VNFs) and edges should be assigned to their respective counterparts of the substrate network graph. This so-called SFC embedding problem requires the coordination of node and link mapping in order to generate efficient solutions. Solutions that decouple these two steps usually yield considerable sub-optimality, whereas they can also lead more often to request rejections. Figure 1 illustrates an example of SFC embedding onto a two-layer fat-tree topology.

Different SFC embedding optimization objectives (*e.g.,* VNF co-location, load balancing, energy efficiency) have been pursued by the various existing approaches (*e.g.,* [5, 11–13, 27–30]). However, considering that a NVFI is practically an edge or core cloud datacenter, its operator would seek to minimize the embedding footprint, which, in turn, reduces the communication cost leading to bandwidth conservation. This is more critical at the inter-rack level, especially for datacenter topologies (*e.g.,* fat-trees) that are oversubscribed. Therefore, embedding solutions that co-locate communicating VNFs within the same rack are more preferable.

In particular, we consider the online SFC embedding problem, at which SFC requests arrive and are processed one by one. The problem formulation presented in the following is tailored to online embedding and seeks the optimization of the mapping of a VNF-graph, independently of other SFC requests.

2.2 Network and Request Model

2.2.1 Network Model

For the substrate network that hosts the virtualized VNFs, we define the graph $G_s(V_s, E_s)$, where V_s denotes the set of compute nodes (*e.g.,* servers) at which VNFs can be hosted, whereas the set E_s represents the network links. In every substrate node z, we assign value r_z which denotes its residual capacity. Every substrate link

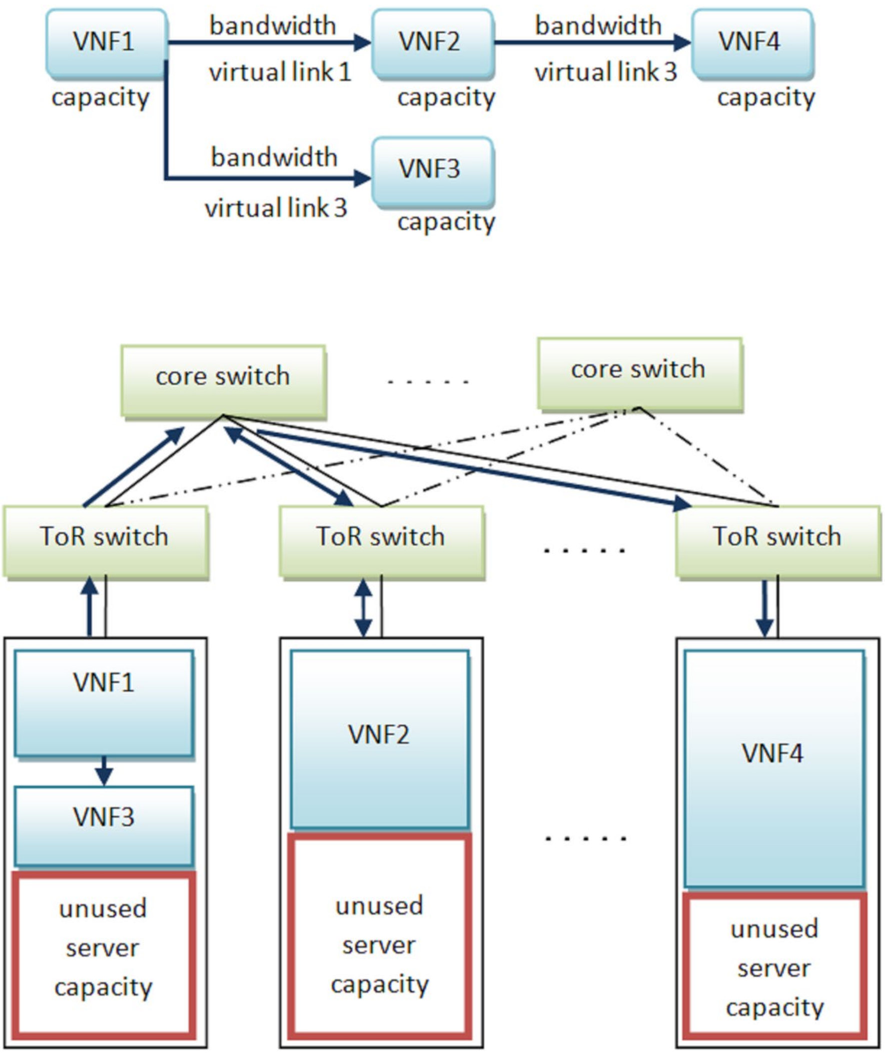

Fig. 1 SFC embedding onto a two-layer fat-tree substrate network topology

$l_{u,z}$ within E_s that connects nodes u, z is associated with b_u, z which denotes the available bandwidth of the link. In addition, for each pair of substrate nodes u, z, we define the path hop-count expressed by $h_{u,z}$.

2.2.2 Request Model

Each embedding request consists of the VNF-graph, modeled by $G_v(V_v, E_v)$. Each node $n \in V_v$ denotes a VNF, whereas associated value d^n expresses its computing demand. We further express each edge between VNFs i and j of the VNF-graph as

$e^{i,j}$. Each edge of the VNF-graph is associated with a bandwidth demand denoted by $d^{i,j}$.

2.3 Problem Formulation

Let graphs $G_s(V_s, E_s)$ and $G_v(V_v, E_v)$ model the substrate network and SFC request, respectively, as defined earlier. We define the mapping of the nodes in V_v to the nodes of subset $V'_s \subseteq V_s$ as:

$$\forall n \exists z \rightarrow n \in V_v \wedge z \in V'_s \wedge x_z^n$$

where x_z^n denotes whether node n is assigned to z (*i.e.,* a value of 1 implies that there is an assignment). Furthermore, we define the mappings of virtual edges as the mappings of the nodes in E_v to the corresponding nodes in E_s, such that:

$$\forall e^{i,j} \exists u, z \rightarrow e^{i,j} \in E_v \wedge z \in E_s \wedge u \in E_s \wedge x_u^i \wedge x_z^j$$

where $e^{i,j}$ is the virtual edge connecting nodes i,j of G_v, while u and z are the corresponding nodes of G_s. The efficiency of the mapping is, therefore, defined as:

$$C_m = \sum_{z \in V_s} \sum_{n \in V_v} (r_z - d^n) + \sum_{u,z \in V_s} \sum_{i,j \in V_v} [(min(b_{u,z}) \times h_{u,z}) - d^{i,j}]$$

Equation C_m corresponds to the fitness function of the genetic algorithm and essentially quantifies the resource efficiency in terms of CPU and bandwidth consumption. More specifically, the first term represents the resource efficiency associated with the whole range of substrate nodes. The second term captures the efficiency with respect to bandwidth allocation across all substrate links. In this respect, we define $min(b_{u,z})$ as the minimum available bandwidth across the links between substrate nodes u, z. The multiplication of this minimum bandwidth by the hop count gives the available bandwidth along each path used for the mapping of VNF-graph edges. Note that the minimization of C_m is strongly correlated with VNF co-location, since a high degree of co-location imposes a reduction in the path hop-count, and, thereby, in the second term of C_m.

The SFC embedding problem can be formulated as an embedding cost minimization problem for any incoming SFC request, *i.e.,* a SFC mapping is sought that minimizes the CPU and bandwidth consumption in the substrate network. Thereby, the SFC embedding problem is formulated as:

$$\text{Minimize} \sum_{n \in V_v} \sum_{z \in V_s} x_z^n d^n + a \sum_{i,j \in E_v} \sum_{u,z \in E_s} f_{u,z}^{i,j}$$

subject to:

$$\sum_{z \in V_s} x_z^n = 1 \quad \forall n \in V_v \tag{1}$$

Table 1 Notations

Symbol	Description
G_s	Substrate network graph representation
V_s	Substrate nodes representing servers
E_s	Substrate network links
$l_{u,z}$	Substrate link connecting substrate nodes u and z
G_v	VNF-graph
V_v	Virtual nodes representing VNFs
E_v	Virtual links
$e^{i,j}$	Virtual edge e connecting nodes i and j
$f^{i,j}_{u,z}$	Traffic flow on substrate link $l_{u,z}$ for the bandwidth demand of the virtual edge $e^{i,j}$
x^n_z	Virtual node n is assigned to substrate node z
$h_{u,z}$	Hop-count between nodes u, z
r_z	Residual computing capacity of substrate node z
d^n	CPU demand of virtual node n
b_u, z	Available bandwidth of link between substrate nodes u, z
d^i, j	Bandwidth demand of virtual edge $e_{i,j}$

$$\sum_{i,j \in V_v} f^{i,j}_{u,z} \le b_{u,z} \quad \forall u, z \in V_s \tag{2}$$

$$\sum f^{i,j}_{u,z} - \sum f^{i,j}_{z,u} = d^{i,j}(x^i_u - x^j_z) \\ i \neq j, \forall i,j \in V_v, u \neq z, \forall u, z \in V_s \tag{3}$$

$$\sum_{n \in V_v} x^n_z d^n \le r_z \quad \forall z \in V_s \tag{4}$$

$$x^n_z \in \{0, 1\} \quad \forall n \in V_v, \forall z \in V_s \tag{5}$$

$$f^{i,j}_{u,z} \ge 0 \quad \forall i,j \in V_v, \forall u, z \in V_s \tag{6}$$

The first term of the objective function minimizes CPU consumption, whereas the second terms seeks the minimization of bandwidth consumption. $f^{i,j}_{u,z}$ denotes the amount of traffic flow on the substrate link (u, z) for the bandwidth demand between the VNFs i and j. In addition, a comprises a normalization weight that acts as a balancing factor between the computational and bandwidth cost.

Next, we explain the various constraints that appear in the problem formulation. Constraint (1) ensures that each VNF is assigned exactly to one substrate node. Constraint (2) enforces a capacity limit on substrate links. Constraint (3) implies flow conservation, *i.e.,* the summation of incoming and outgoing flows of a substrate node must be zero. Condition (4) ensures that the CPU demands of the assigned VNFs do not exceed the residual CPU capacity of the corresponding substrate nodes. Lastly, constraint (5) enforces the binary domain constraints for variable x_z^n, whereas condition (6) enforces the causality of the flows $f_{u,z}^{i,j}$. Table 1 provides a list of all notations.

3 Background on Genetic Algorithms

Genetic algorithms are optimization techniques inspired by Darwin's theories on the evolution of species [31]. They provide efficient solutions in computationally hard problems, such as combinatorial NP-hard problems [32]. The generated solutions may yield a degree of sub-optimality, although they may still be efficient and acceptable [33]. Initially, a genetic algorithm generates a population of possible solutions. The population is usually generated randomly, but may also be the product of a heuristic procedure [34]. Our proposed solution employs both methods. The members of the population are called *chromosomes*; this name implies that the functionality of the algorithm simulates biological procedures. Every chromosome is a string that encodes a possible solution and every symbol of the string is called *gene*. A crucial factor for the operation of the algorithm is the fitness function, which defines the criterion for the margin between the solution encoded in a chromosome and the desired solution. The algorithm iteratively executes the following procedures.

Selection. This step simulates the procedure of natural selection, at which the stronger members of a population survive in the next generation while the weakest members do not survive.

Crossover. During the procedure of crossover, two chromosomes exchange parts of their genetic material that are randomly chosen and generate offspring that represents different solution than their parents.

Mutation. The procedure of mutation refers to the random change of the value of a gene, similar to the biological notion of mutation. Mutation may generate solutions that are not produced by crossover, thereby, directing the search in different parts of the search space.

Each round of sequential execution of the procedures is called *generation*. Eventually, the population becomes homogeneous converging to a strong solution. This outcome stems from the fact that the chromosomes of higher fitness prevail through the procedure of selection over the weaker chromosomes. The genes that are primarily responsible for the high fitness are spread through crossover to a large part of the next generation population. A genetic algorithm is terminated when the desired solution has been computed or when the population has become homogeneous and converges to a solution.

A particular limitation of genetic algorithms is the premature convergence of the population to some local optimum, *i.e.,* failing to compute the global optimum which commonly comprises the objective of the algorithm. This behavior occurs when the search for the optimal solution is confined within a subset of the state space, where the desired solution is not included. The GA model that we introduce in the following section lets us clarify why the desired mapping is unreachable under certain conditions for problems, such as the SFC embedding.

4 Genetic Algorithm Modeling

In this section, we model GA as a state-space search. State space is a method of representing a problem by defining all the possible states at which the problem can fall into, *i.e.,* all the possible solutions of the problem. Therefore, the solution of the problem lays in its state space. It can be identified by searching the state space and it is reasonable to claim that by studying the state space of a problem, we can draw important conclusions about the problem. Modeling an algorithm as state-space search empowers us to analyze how it converges to the final solution across the entire solution space.

Such analysis is critical for understanding the functionality and behavior of a GA. Without loss of generality, we consider the case of the *simple genetic algorithm*; that is an algorithm without variations from the typical design of the GAs. The literature includes lots of variations of the typical simple GA that improve certain features of the algorithm, based on the special requirements of the problem under consideration in each case. These are heuristics used for the population generation and methods that prevent the premature convergence of the population. These variations adhere to the same principles and employ the same genetic procedures. Therefore, the modeling of a typical GA can be also applied to these variations without loss of generality.

State space, also known as solution space in some disciplines, is modeled as a directed graph $C(S, A)$ where:

- S is the set of nodes that correspond to all the states of the space that may be considered as possible solutions to the problem (or else all the configurations of the problem). Every state can be encoded as a chromosome in the case of a GA.
- A represents the set of edges that denote the possible actions that may be applied during a search on S; these are the possible transitions between states. Each action is applied to two states and defines the transition of the search between these states. Any two nodes may be connected by more than one edge, if there are more than one ways of transition between the states. In the particular case of the GA, transition between two states may occur by the procedures of crossover and mutation that are modeled by distinct edges in C. The genetic procedures define the following corresponding actions:
  - Mutation actions connect states that can result from the procedure of mutation. In this respect, for edge $e_{a,b}$, state b is the product of the mutation of a.
  - Crossover actions generate offspring states from parents. For edge $e_{a,b}$, state b is the output of crossover applied to state a. In this respect, state b inherits part of its genotype from a. For instance, the first n bits in the description of a may be identical to the first n bits in the description of b.

The actions are probabilistic reflecting the probabilistic nature of the genetic procedures. The search in the space is defined by the sets of the initial and the goal states:

- Set $P \subset S$ denotes the set of initial states that form the initial population of the algorithm.
- Set $F \subset S$ is the set of goal states, *i.e.,* the states that satisfy the objective of the algorithm which is evaluated by the fitness function f.

Let us label every $s \in S$ with its fitness f_s. The current population defined in set $T \subset S$ at any generation evolves into the new population defined in set T' using the actions on the edges and the probabilities of executing these actions. The selection procedure removes the low-fitness nodes from the population.

Without loss of generality, we model this operation as the probabilistic transition of the population from lower-labeled nodes to higher-labeled nodes in C. This model captures the ability of the genetic algorithms to generate effective results. Based on this consideration, when the consecutive transitions in C lead to nodes with high label, all the members of the population, which may use actions to move to these states, have high probability of performing this transition. This is an explanation for the convergence of the population to some local or global optimum.

A goal state is reached if and only if $\exists g \rightarrow g \in F \wedge g \in T$. This implies that given P and S, reaching goal state $g \in F$ is feasible, if and only if there is path from node $j \in P$ to the node that denotes g. The probability of effectively reaching g is determined by the probabilities of performing the actions on the path.

In order to model the behavior of a sophisticated algorithm as a state space search, we have to determine how this search exploits the properties of the state space so as to perform an efficient and not random search. These properties are found in the structure of the state space and the relationships among the states. The elements of a state space may share structural relationships, which we discuss in the context of the SFC embedding problem.

Definition 1 (*Structural relationships*) The elements of set $S' \subset S$ have structural relationships, if and only if their graph representations share common elements, *i.e.,* nodes and edges. For any $a, b \in S'$, there are corresponding graphs A and B, such that $A = K + G$ and $B = D + G$, where G is a common subgraph to all graphs in S'.

In the SFC embedding problem, the states can be represented as subgraphs of the substrate network representation, which denote the mapping of the SFC graph to the substrate resources. We illustrate such an example in Fig. 2. Both mappings share common nodes that host the VNFs of the SFC and common substrate links.

Definition 2 (*Breadth of influence*) Structural relationships among states may be global or local depending on their breadth of influence. Global relationships are in effect in the whole space S. Let E be a global relationship defined in S. For every

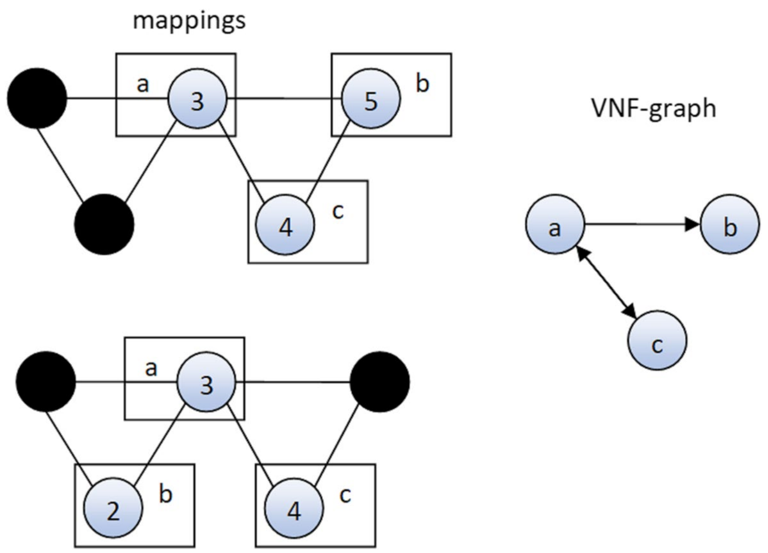

Fig. 2 Example of mappings with structural relationships in nodes 3, 4 and the edge that connects them

$S' \subset S$, the relationship E between any $a, b \in S'$ is defined and evaluated as true. Local relationships are in effect in a subset of the state space. Let H be a local relationship defined in $S_b \subset S$. There is a set $S_a \subset S$ in which H is not defined and not evaluated as true, where $S_a = S - S_b$.

The importance of structural relationships in graph problems solved by GAs is shown in the next lemma.

Lemma 1 *Any consecutive states in path h of graph C which denotes state space S share structural relationships.*

Proof By definition, path h in C connects states that are formed by the genetic procedures of crossover and mutation. During crossover, chromosome c is formed by combining structural elements of chromosomes a and b. Then, a and b have structural relationships with c. During mutation, chromosome d' is formed by the random variation of genes of chromosome d per generation; then d' and d share common elements. As a consequence, for every state s in h, there is at least one state s' also in h, which shares common elements with s. □

We notice that, after some generations, the population that is generated may be completely altered and may not contain any common chromosomes with the initial population; it is possible that the members of the initial and the final population will not share structural relationships. For any two consecutive generations m and $m + 1$ though, the members of their populations share structural relationships as the chromosomes of $m + 1$ are products of the population of m under the genetic procedures. In conclusion, any two consecutive nodes on path h denote states that share structural relationships, but this is not necessarily in effect in more distant nodes on *h*. Lemma 1 and implies the following corollary:

Corollary 1 *Reaching a goal state is feasible for a genetic algorithm, if and only if there is state $p \in P$ and state $g \in F$ between which there is sequence H of nodes, such that consecutive nodes h_i and h_{i+1} in H have a structural relationship.*

In order to identify the hardness of problems in the field of computational complexity, we group them in classes of problems with similar characteristics and study their behavior. We follow this concept and further examine two classes of problems with similar characteristics and shed light into the efficiency of GAs. The two classes of problems are defined using general schemas, namely *n-subgraph* and *k-subgraph*. These schemas are formulated as languages. All the languages that describe problems in one of these classes are in compliance with the corresponding schema.

In the remainder of this section, we formulate decision problems and states as languages. Then language L_D describes the problem while language L_S its states. For the decision problem described by language L_D, there is language L_S, such that every state in state space S of L_D belongs to L_S and the problem is resolved as true if $\exists s \in S \rightarrow s \in L_D$.

We particularly care about function problems that can be efficiently solved by GAs. As such, we define them on the corresponding decision problem and language L under the general definition:

L: find s in S so that $s \in L_D$

Next, we define as languages two classes of problems related to SFC embedding and study their hardness when computed by GAs.

The n-subgraph class

n-subgraph = $\{G, n, w \mid$ There is n-node subgraph N in n-node graph G of maximum degree $\Delta(G) > 0$, where property w is valid.$\}$

The states of the state spaces of the instances of the problems that belong to the *n-subgraph* class belong to the following language:

n-subgraph$_S = \{N \mid N$ is an n-node graph $N\}$

The state space for each instance of problem in the class consists of all the possible graphs that may be generated using the n-nodes of G and all the combinations of its edges. The demand for $\Delta(G) > 0$ excludes problem instances of edgeless graphs.

The number of states in S can be reduced if we consider a bounded number of edges for N. The states have global structural relationships. The elements of the state space are all graphs of the form $G(V, E)$, where V is common to all. This constitutes a global structural relationship.

In Sect. 6, we discuss the problem of *Parameter Adjustment* that belongs to this class. We particularly utilize parameter adjustment for the optimization of the operation of our genetic algorithm. Another well-known problem that belongs in this class is the minimum spanning tree problem [35], at which we seek to find a spanning tree N of minimum weight in weighted graph G. Property w is valid if N forms a tree and it is minimum in G. The number of edges in N is bounded to $n - 1$.

For each problem in the class where the number of edges in N is bounded, the cardinality l of its state space S equals to $l = \binom{e}{m}$, where e is the number of edges in G and m the number of edges of N.

The k-subgraph class

k-subgraph $= \{G, n, k, w \mid$ There is k-node subgraph K in n-node graph G of maximum degree $\Delta(G) > 0$, where property w is valid and $n \geq k\}$

For the state space S of any problem in this class, the states within are defined as follows:

k-subgraph$_S = \{K \mid K$ is a k-node graph$\}$

Each state space consists of the descriptions of all the possible k-node subgraphs of G. Each problem in the class can by sufficiently defined through G and w.

The SFC embedding problem belongs to this class. Its state space consists of all the k-node sub-graphs of the state space. Property w is valid, if the set of k nodes that will host a VNF-graph is within constraints, is connected and has the minimum sum of weights of all connected k-node subgraphs of G, where weighted graph G models the substrate network. In order to simplify our analysis, in this section and without loss of generality, we consider a restricted case of the SFC embedding problem, at which the VNFs are hosted on exactly k nodes. In the general case of the problem that is studied in this paper, the VNFs are hosted at most on k nodes, favoring VNF co-location. The elements of its state space belong to the following language:

SFC-E$_S = \{G_k \mid G_k$ is a k-node graph$\}$

As an example of another problem that belongs to this class, we refer to the widely studied problem of CLIQUE, which aims at determining the existence of a k-node complete subgraph in graph G. In CLIQUE, w is satisfied if K is a complete k-node graph.

We further describe the relationships among the elements of the state space for the problems in the *k-subgraph* class. The cardinality l of S equals to $l = \binom{n}{k}$. There are structural relationships of local breadth among the elements of S. Each edge of G (or each pair of nodes) participates in the formation of $l_e = \binom{n-2}{k-2}$ subgraphs of k nodes as a common structural element. Each node is a common structural element in the formation of $l_v = \binom{n-1}{k-1}$ subgraphs.

Both structural relationships have local breadth in S, as each edge or node is common only to a subset of S. If there were global relationships, there would be common elements for all the states in S, which is not a valid argument. As we show in Theorem 2, there are states in S that are distinct under the following definition:

Definition 3 (*Distinct states*) States a, b are distinct, if and only if they correspond to distinct subgraphs $G_a(V_a, E_a)$, $Gb(V_b, E_b)$, so that $V_a \cap V_b = \emptyset$ and $E_a \cap E_b = \emptyset$.

There are numerous GAs that deal efficiently with problems in the *n-subgraph* class [36, 37]. Instead, problems in the *k-subgraph* class are considered computationally harder [38]. Note that this assertion is mainly based on empirical evidence. In this respect, we provide a more elaborate explanation based on theoretical grounds.

The main reason behind the use of a GA for the aforementioned problems is the development of solvers with low run-time, *i.e.,* much lower compared to an exhaustive search of the state space. To this end, the initial population needs to be relatively small, since a large initial population would lead to a complexity on par with the exhaustive search. For the problems in *k-subgraph* class, we show in the rest of this section that for a relatively small population $P \subset S$, there is subset S' of the state space, the elements of which have no structural relationships with the elements in P. As such, for a relatively large S', there is low probability of reaching a goal state in S'.

Theorem 2 *In state space S of a problem described by language L in k-subgraph class, there are subsets S_1, $S_2 \subset S$, such that any states $s \in S_1$ and $m \in S_2$ are distinct.*

Proof Let i_1 and i_2 be two instances of a problem described by language L in *k-subgraph* class that are defined by graphs G_1, G_2 with respective state spaces S_1, S_2. Let us now form problem instance i_3 in graph G_3 that is generated by connecting the nodes of G_1 and G_2 with arbitrary placed edges. The state space of i_3 can be expressed as the union $S_3 = S_1 \cup S_2 \cup A$, where A is the set of states defined in subgraphs of G_3 that consist of nodes from both graphs G_1 and G_2. It is clear that any states s and m are distinct, if $s \in S_1$ and $m \in S_2$. This is a general property of the problems in *k-subgraph* class. □

In the next theorems, we examine the effectiveness of crossover and mutation in exploring S.

Theorem 3 *Most states in space S of the problem described by L in k-subgraph class are infeasible to be generated by applying crossovers on a population g of relatively small size.*

Proof Following the arguments on the proof of Theorem 2 and Corollary 1, a population generated by the elements of graph G_1 is not feasible to generate the states in G_2 using crossover. Then for a small population g, subgraph G_1 is smaller in size than G_2. This implies that the states that are feasible to be generated by g are less than the states that cannot be generated. □

Theorem 4 *For subsets of distinct states S_1 and S_2 of state space S of a problem described by L in k-subgraph class, a population that denotes states in S_1 has low probability of generating states in S_2 through mutation.*

Proof Every chromosome after mutation has probability $Pr = |S_2|/|S|$ of generating a chromosome that represents a state in S_2. From Theorem 2, we derive that $S = S_1 \cup S_2 \cup A$ and $|A| = \binom{n}{k} - \binom{n_1}{k} - \binom{n_2}{k}$, where n, n_1 and n_2 are the nodes that form S, S_1, S_2 respectively and $S = S_1 + S_2$. These imply that $|A| > |S_2|$ and $|S_2|$ is significanlty smaller than $|S|$. As such, probability Pr is low. □

Theorems 3 and 4 imply that finding the global optimum on a state space is hard for problems in the *k-subgraph* class. Under certain conditions, a goal state will probably not be reached. The demand for efficiency imposes the generation and maintenance of relatively small populations throughout the operation of a GA, thereby, reaching a goal state is uncertain. These observations dictate our choices (*i.e.,* heuristic population generation, multiple execution of the GA in different populations) in the design of the proposed GA, as explained in Sect. 5.

The arguments used in the proofs of the theorems are not valid for problems in the *n-subgraph* class. The states of any instance of a problem within the *n-subgraph* class that is defined in graph G include all the nodes of G and this structural relationship is global. In the general case, the problems within the *k-subgraph* class are harder to solve using GAs compared to the *n-subgraph* class.

5 Genetic Algorithm

During the development of GAs, various critical issues are raised that are inherent to the nature of GAs. In order to address them, it is often necessary to apply variations on the typical design of GAs [34, 39]. In the following, we discuss the variations that we apply.

Maintenance of efficient solutions. In the common approach, the procedures of crossover and mutation modify the genotype of the population without preserving the initial chromosomes. This design may fail to preserve an efficient solution on the population that may take part in mutated or crossover procedures.

This issue is resolved by including in the same population both the parents and their offspring; similar to biological systems where populations consist of a mixture of parents and offspring. During mutation, we include both the original and the mutated chromosomes in the population. This technique increases the size of the population during crossover and mutation. Then the selection procedure reduces the population size to its default size. The same reasoning satisfies the need to maintain the best solution computed during each generation as a candidate output of the algorithm.

Premature convergence to local optima. A significant issue concerning GAs, in general, is the convergence to an undesired solution. This stems from the restriction of the search in a part of the state space that contains only local optima. In this case, the population becomes homogeneous, before converging to the optimal solution.

The technique developed for the avoidance of premature convergence, in our work, is based on the concept of competition that takes place in stages and in sets of competitors. The competitors that prevail in each set form the groups of competitors that will compete in the next stage of the competition. The winners of the

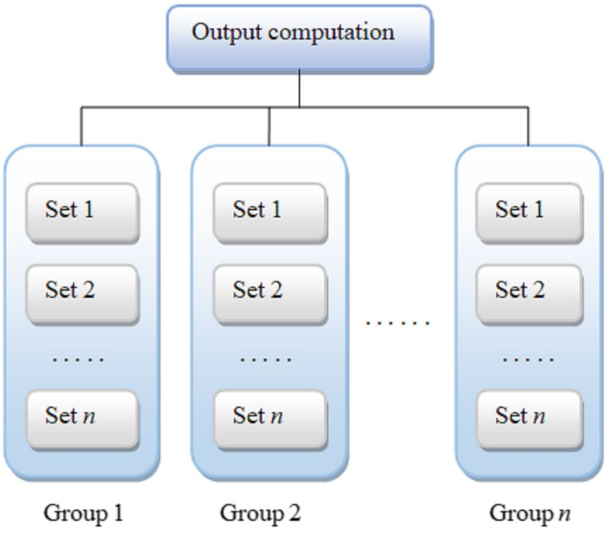

Fig. 3 Structure of procedures

group stage are nominated in the last stage of the competition, which designates the stronger competitor as the winner of the competition.

The GA is executed in n groups of n sets of chromosomes. In each set, there are p chromosomes that constitute the population for the algorithm, which is executed for g generations. The best solution generated in each set is promoted to form the population of the group that it belongs. Subsequently, in each group, the algorithm is executed over the population generated in the sets. The final output is computed by executing the algorithm on the population generated on the groups. This technique develops a structure of procedures that is sketched in Fig. 3.

In this method, the algorithm combines in the group stage the local optima discovered in the sets, avoiding the confinement within a single local optimum. We explain the benefits of this technique based on the theory developed in theorems 3 and 4. In the i-th execution of the algorithm, the subset S_i of the state space S is explored, which differs from the subset S_j explored by the j-th executions of the algorithm. Also, the subset $S - S_i$ that is inaccessible by the initial population of i also differs from the subset $S - S_j$ of j. Then for large S executions, i and j generate distinct mappings. Computing in the group stage the mappings generated from subsets S_i and S_j enables the generation of the combined best solutions found in subset $S_c = S_i \cup S_j$, resulting in a more advanced exploration of S. Set S_c includes mappings that are generated partially from each of the two subsets S_i and S_j achieving a broader exploration of S. Simulation results demonstrating the advantages of this technique are presented in Sect. 7.3.

A computational benefit of this approach is that computations in sets and groups can be executed in parallel, thereby, significantly reducing solver run-time in multi-core systems. We have used a multi-threaded implementation in order to enable parallelism. The computation of each group is sequential while the computations of the sets in each of the groups run in parallel, reducing the the run-time that is required for the sets to produce the population that will be used in the groups.

Chromosome representation. The mappings of the VNF-graph elements to the substrate network resources and the chromosomes are represented by an array H of length equal to the order of the VNF-graph. In position i of H, we place the index of substrate node $z \in V_s$, which hosts the node $i \in V_v$.

Initial population generation. The operation of the GA consists of the generation of an initial population of chromosomes and the iterative execution of the genetic procedures, as shown in Algorithm 1. There are two approaches for the generation of the initial population, *i.e.,* the random and the heuristic generation of its chromosomes. In the former approach, the chromosomes represent a set of possible solutions chosen randomly from the state space (or else solution space) of the problem. In the latter approach, a heuristic is employed to form the population and direct the search in specific parts of the state space. We employ both methods; the largest part of the initial population is generated heuristically and, subsequently, for the remaining part of the population we apply random generation. The heuristic generation of the population improves the fitness of the output of the GA, as it favors solutions that minimize inter-rack traffic.

For the population generation, the heuristic only considers the servers that have sufficient capacity for accommodating the minimum VNF demand; that is subset $V'_s \subseteq V_s$. As such, the algorithm avoids the computation of chromosomes that are incapable of generating valid solutions and restricts the computation of the state space to states that are more likely to constitute valid mappings. Then for every $s \in V'_s$, the heuristic generates a chromosome that maps all the VNFs of the chain to s. Even if this mapping is not valid, during crossover and mutation valid mappings are generated that favor the co-location of multiple VNFs on the same server, directing state space search to parts of the state space that meet the minimum traffic demand. The rest of the population consists of randomly generated mappings of the VNFs to the nodes in V'_s. In case that the size of the population is smaller than the size of V'_s, the ratio of heuristically to randomly generated chromosomes is set to 0.8. The generated initial population also exhibits high diversity, which advances its operation. In theory, the initial population of a genetic algorithm should exhibit high diversity, which usually implies that it covers a wide range of the state space.

The use of this heuristic directs the state space search to states that favor the mapping of communicating VNFs on the same servers, alleviating the communication cost and exploiting the computational capacity of the servers more efficiently. In addition, states that include substrate nodes that are not capable of hosting any VNF are excluded from computation, reducing the size of the state space to by computed.

The execution of the genetic procedures in each chromosome depends on its fitness, computed by fitness function C_m using input graph G_s, which is a representation of the substrate network on its current computational load and traffic. In order to determine the number of times the genetic procedures will be executed in a GA, we define the parameter *generations*. For the formation of sets and groups and their computation, we also use the parameter supergenerations. Specifically, this expresses the number of times that the GA will be executed in the sets and groups that will eventually generate the embedding. The thresholds of mutation and crossover probabilities along with a random number generator specify whether the two procedures are executed in every chromosome. These parameters along with the size of the population, the size of each VNF-graph and the current network state affect the operation and the efficiency of the algorithm. These parameters are inserted into the algorithm as the setup on which the algorithm will be applied.

Algorithm 1 Genetic Algorithm

```
Input: graphs $G_s$, $G_v$, $generations$, $supergenerations$, $setup$
Output: best chromosome
1: run init1()

2: Procedure init1()
3: for $t = 0$ to $supergenerations$ do
4:     for $s = 0$ to $supergenerations$ in parallel do
5:         run thread {
6:                 generate $population1$
7:                 run init2($population1$)
8:                 store best chromosome in $population2$
9:         }
10:     end for
11:     run init2($population2$)
12:     store best chromosome in $population3$
13: end for
14: run init2($population3$)
15: return best chromosome

16: Procedure init2($population$)
17: compute $deviation$ in $population$
18: if $deviation > 0$ then
19:     for $i = 0$ to $generations$ do
20:         run crossover($pop$)
21:         run mutation($pop$)
22:         run selection($pop$)
23:     end for
24: end if
25: return best chromosome
```

In the following, we elaborate on the three genetic procedures.

Selection. The selection procedure, described in Algorithm 2, is executed only if the population is not homogeneous; otherwise, its execution is pointless. Whereas the procedures of crossover and mutation increase the population in the current generation, selection maintains a constant size for the population of the next generation. We maintain a population of constant size in order to control the complexity of the algorithm. Homogeneity is evaluated by computing the deviation of the population fitness. If the population is deemed to be heterogeneous, the fitness of each chromosome is computed and if its value is lower than c, the chromosome is added in the future population. This occurs for $c = f_{min} + dev \times q$, where f_{min} is the value of the strongest chromosome of the current population, dev the deviation of the fitness values of the population, and q is a randomly generated value.

Algorithm 2 Selection

```
Input: population $pop$, $newpop\_maxsize$
Output: new population $newpop$
1:  compute $deviation$ in $population$
2:  if $deviation > 0$ then
3:      while $newpop.size < newpop\_maxsize$ do
4:          generate random value $q$
5:          $c =$(minimum $pop$ fitness)$+deviation \times q$
6:          for $i = 0$ to $pop$.size do
7:              if $pop[i]$.fitness$< c$ then
8:                  add $pop[i]$ to $newpop$
9:              end if
10:         end for
11:     end while
12: end if
13: return $newpop$
```

Crossover. This procedure generates new chromosomes that are added in the current population. For every pair of sequentially chosen chromosomes (*e.g.,* i, j), we generate a random value q. If q is smaller than the crossover probability, the two chromosomes produce two offsprings. Given a random number h, chromosome i copies its first h genes to the first offspring and the rest to the second one. Likewise, chromosome j copies its first h genes to the second offspring and the rest to the first one (see Algorithm 3).

Algorithm 3 Crossover

```
Input: population $pop$, crossover probability $cr$
Output: new population $newpop$
1: $c_l$ =chromosome length
2: for $i = 0$ to $pop$.size-1 with step 2 do
3:     add $pop[i]$, $pop[i+1]$ to $newpop$
4:     generate random value $q \in [0, 1]$
5:     if $q \leq cr$ then
6:         generate empty chromosomes $temp1$, $temp2$
7:         generate random value $h \in [0, c_l)$
8:         add genes 0 to $h$ of $pop[i]$ in $temp1$
9:         add genes $h+1$ to $c_l$ of $pop[i+1]$ in $temp1$
10:        add genes 0 to h of $pop[i+1]$ in $temp2$
11:        add genes $h+1$ to $c_l$ of $pop[i]$ in $temp2$
12:        add $temp1$, $temp2$ to $newpop$
13:    end if
14: end for
15: return $newpop$
```

Mutation. This procedure is sequentially executed in every chromosome. If a randomly generated value q_1 is smaller than the mutation probability, a copy of the chromosome is generated and mutated. A gene of the copied chromosome is randomly chosen and a random value is assigned to it. The mutated chromosome is added to the population, as described in Algorithm 4.

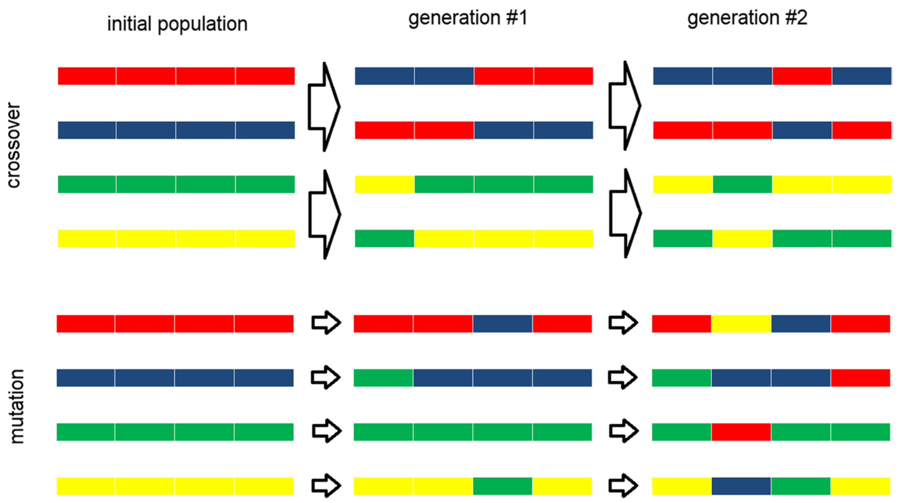

Fig. 4 During crossover, pairs of chromosomes exchange parts of their genetic codes. Mutation regards the random variation of randomly selected genes in every chromosome

A simplified example of the procedures of crossover and mutation is illustrated in Fig. 4.

Algorithm 4 Mutation

```
Input: population $pop$, mutation probability $mut$
Output: new population $newpop$
 1: $c_l$ =chromosome length
 2: for $i = 0$ to $pop$.size do
 3:     add $pop[i]$ to $newpop$
 4:     generate random value $q_1 \in [0,\ 1]$
 5:     if $q_1 \leq mut$ then
 6:         generate random values $q_2 \in [0, c_l)$, $q_3 \in [0, |V_s|)$
 7:         place $q_3$ in gene $q_2$ of $pop[i]$
 8:         add $pop[i]$ to $newpop$
 9:     end if
10: end for
11: return $newpop$
```

6 Dynamic Parameter Adjustment

Identifying the optimal values for the parameters in the GA setup entails a significant challenge and it is subject to the problem that the GA is applied. Merely increasing the values of these parameters will increase the complexity of the algorithm, while decreasing these values will affect its efficiency. As such, the quest for optimality is crucial. A common approach is to identify the appropriate values through extensive experimentation. Instead, we rely on a computational solution that enables adaptability across different topologies and different instances of the problem.

6.1 Parameter Adjustment as an Optimization Problem

Parameter adjustment is always problem-oriented. In this respect, we have to take under consideration the special properties of each problem instance on which the genetic algorithm is applied. In particular, we approach parameter adjustment as an optimization problem and rely on GAs for its solution. In this section we analyse the efficiency of the parameter adjustment problem and prove that it is in the *n-subgraph* class and therefore it is computationally tractable. In this analysis, we use the same notation and language-oriented descriptions as in Sect. 4.

Let function $f_g(Q, i)$ produce the output of GA g on input i using setup Q. We seek to determine the minimum values for the elements of Q that maximize the output of f_g. Minimization of the values in Q implies minimum computational burden, while maximizing f_g implies maximizing the effectiveness of the algorithm. It is reasonable to apply bounded values in the elements of Q so as to maintain efficiency and avoid undesired arbitrary large values. As such, we define the Parameter Adjustment (PA) problem, as follows:

PA: Given i, a, b find minimum Q which maximizes $f_g(Q, i)$ where $\forall q \in Q \rightarrow q \in [a, b]$

Theorem 5 *PA is in n-subgraph class.*

Proof We show that PA is reducible to a problem within the *n-subgraph* class. Consider graph $G(V, E)$ where V consists of n weighted nodes to which we assign all the values in $[a, b]$ and the zero value; in node n_0. For every edge $q(n_0, n_z)$ that we place in G, we assign to parameter q of Q the weight of node n_z. Thereby, PA is reduced in finding the optimal placement of the edges of the form $q(n_0, n_z)$ that will construct G in order to achieve the objective of PA. Graph G is an n-node graph with edges equal to the number of parameters in Q and satisfies property w, as it is described by the objective of PA and its construction. It is then a problem within the *n-subgraph* class. □

Based on the model presented in Sect. 4, the PA problem can be approached efficiently using a GA. Another advantage is that f_g is an increasing function with respect to each parameter. By increasing each parameter alone the genetic algorithm will explore a larger part of the state space, since it will either generate more chromosomes or perform more iterations of the genetic procedures and, thereby, increase its efficiency. This property implies that PA is a computationally tractable problem.

6.2 Genetic Algorithm for Parameter Adjustment

We have designed and implemented a *Parameter Adjustment Genetic Algorithm* (PAGA) in order to identify near-optimal values during the execution of the embedding algorithm. PAGA is shown in Algorithm 1. It differs from the SFC embedding version in its population, which consists of candidate parameter setups for the embedding algorithm. The fitness of each chromosome in PAGA is computed by running an instance of the embedding algorithm on the setup that the PAGA chromosome represents in order to determine its efficiency. The efficiency is defined by function C_m, similar to the SFC embedding algorithm.

The functionality of PAGA is determined by the specification of its operational parameters. These comprise the setup on which PAGA will operate and the range of its output values. Starting from the latter, the range of values that PAGA produces for the embedding algorithm parameters will affect its performance and should be bounded in order to ensure efficiency. The range of output values in our evaluation (Sect. 7.1) is defined as follows:

- Population size: 440 - 490
- Generations: 40 - 60
- Supergenerations: 4 - 6
- Crossover probability: 0.10 - 1.00
- Mutation probability: 0.10–1.00

We have determined empirically that the PAGA may function effectively on a setup of 30 chromosomes in its population, 30 generations, 2 supergenerations and for crossover and mutation probabilities of 0.85 and 0.05, respectively. The parameter adjustment procedure is dynamic to the operation of the network, *i.e.,* the output is not computed using static data. The overall procedure is described in Fig. 5. The advantage of this approach is that the embedding algorithm may be adapted to any network topology using inputs at real-time.

PAGA is executed in parallel with the embedding algorithm in order to compute an efficient setup for each SFC embedding problem configuration. The setup that corresponds to a specific problem configuration is stored in a database and used on demand. For the sake of simplicity, we define an abstract and restricted description of the problem configuration based on two parameters. More precisely, the configuration is defined as a 2-tuple (v, q), where $v = V_v$ is the size of the VNF-graph, whereas q represents a classification determined by the ratio of the CPU demands of the VNFs currently embedded over the total CPU capacity of the substrate nodes. More precisely, we define m classes and $q = \lfloor m \times (c_{cur}/c_{max}) \rfloor$, where c_{cur} denotes the CPU demands of VNFs and c_{max} is the CPU capacity of the substrate nodes. In our implementation, we set $m = 4$. For each combination of (v, q), a single setup is stored in the database.

PAGA requires the handling of a number of requests by the embedding algorithm in order to adapt to all the possible problem configurations. To generate an efficient embedding procedure, we define a basic setup for initiating the algorithm, which will be used when an optimal setup is not available for an incoming request. As the database will be gradually populated, the embedding algorithm will use the stored setups to handle the new requests with respect to the problem configuration.

The basic setup for the implementation of the embedding algorithm in our simulations (Sect. 7.1) utilizes a population of 440, whereas the generations and supergenerations parameters are set to 40 and 4, respectively. In addition, the crossover and mutation probabilities are set to 0.5 and 0.2, respectively. By this configuration, PAGA strikes a balance between efficiency and solver run-time, and also provides efficient embedding results, until it has converged to the optimal setup.

When a request arrives, the embedding algorithm queries the database for a corresponding setup. If the setup is not found, the embedding algorithm uses the stored setup that better approximates the request and, subsequently, PAGA computes an optimal setup for the request, which is eventually stored in the database for future use. The proximity of each stored setup to the request is computed as the euclidean distance $d(a, b)$, where a is the configuration (v, q) of the request and b the configuration (v', q') of the stored setup. An optimal setup implies that $d(a, c) = 0$.

Gradually, all possible configurations lead to certain stored setups. After that, there is no need to utilize the basic configuration, as well as the PAGA. The operation of the embedding algorithm is then optimized. In our execution environment, PAGA achieves the optimization of the operation of the embedding algorithm at about 500 requests. For these requests, the algorithm, using the basic setup, generates efficient mappings that yield a small and acceptable overhead in

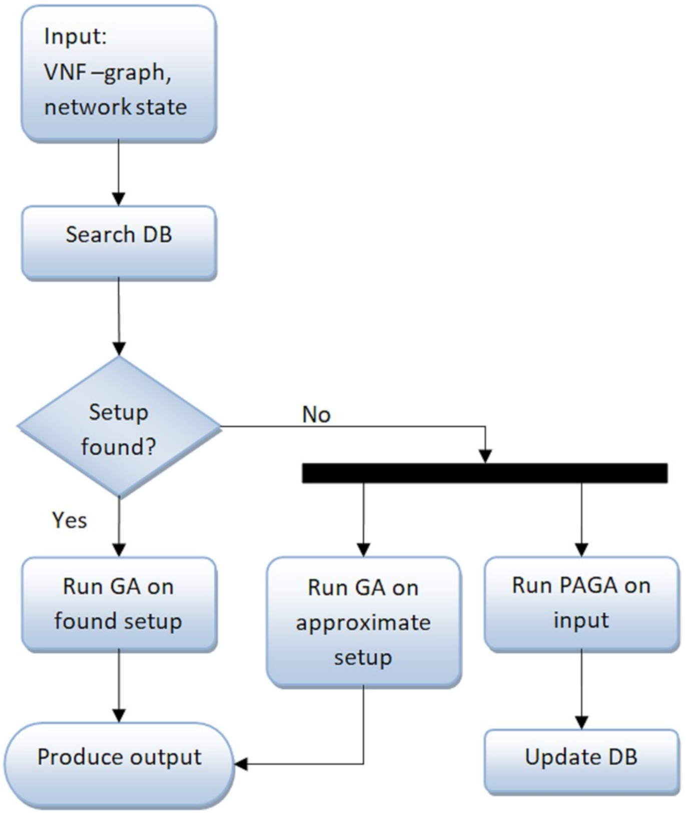

Fig. 5 Flow chart of SFC embedding with PAGA

its operation till its convergence to the optimal setup. In this respect, Fig. 6 illustrates the convergence to an optimal setup.

Table 2 shows an instance of the database generated during the simulation. As stated earlier, in the first column of the table the problem configuration is described by two values; the first one indicates the size of the incoming request v, while the second one the network state classification q. Both the table and the basic setup indicate large values of the mutation probability, which deviate from typical values, *i.e.,* around 0.05. The values for the basic setup are computed using PAGA, as well, and this result is in line with the model developed in Sect. 4. There are states within the state space of the instances of the SFC embedding problem that can not be explored by executing crossover on the initial population. The extensive use of mutation enables the exploration of parts of the state space that cannot be reached via crossover, increasing the efficiency of the generated solutions.

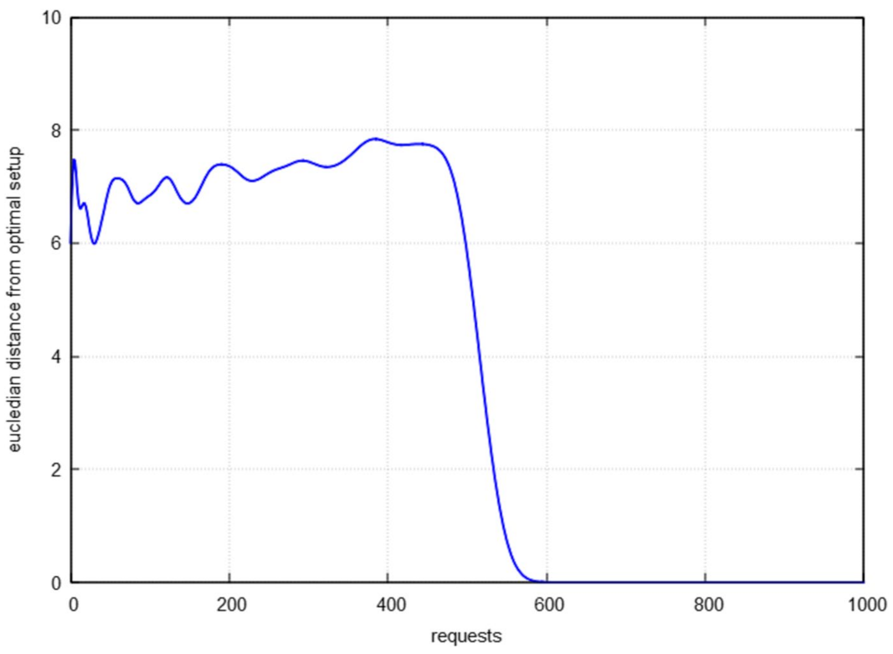

Fig. 6 Deviation from optimal setup

Table 2 Database instance

Problem configuration	Population	Generations	Super-generations	Crossover & mutation probability
5 0	464	58	6	0.91 0.97
5 3	486	56	6	0.63 0.29
9 3	470	50	6	0.77 0.35
8 3	444	48	4	0.90 0.62
7 3	446	56	6	0.73 0.29
6 3	454	48	4	0.96 0.44

7 Evaluation

In this section, we initially present the evaluation environment (Sect. 7.1), followed by the methods of comparison (Sect. 7.2), and the discussion of our evaluation results (Sect. 7.3), where we assess the SFC embedding efficiency of GA variants with different features. In addition, we compare the outstanding GA variant against a baseline greedy algorithm and a state-of-the-art heuristic, *i.e.,* BACON [25].

7.1 Environment

The algorithms and the evaluation environment are implemented in Java 15 and executed on a computer equipped with a 16-core Intel Xeon CPU at 2.1 GHz and 8 GB RAM. The simulations are conducted on a structured network topology. More specifically, the topology corresponds to a 3-layer fat-tree datacenter network, which comprises a common NFVI. The simulated fat-tree topology consists of 12 pods, which encompass 72 racks and 432 servers, in total. Each server is equipped with 8 CPU cores at 2.5 GHz. The capacity of links connecting the servers with the Top-of-the-Rack (ToR) switches is 1 Gbps, whereas the links at the upper layers of the topology have capacity of 10 Gbps. The source code and compiled binaries for the simulation environment are available at [26].

Each VNF-graph request consists of a diverse number of nodes, picked randomly within the range of 5 to 9. In our simulations, the VNF-graphs have tree-based structure and consist of two branches. The CPU demand for each VNF in the request varies between 2 and 6 GHz. Likewise, the bandwidth demands in the VNF-graph vary between 20 and 100 Mbps. The VNF-graph requests are expiring. In particular, every embedded SFC is associated with a lifetime picked randomly within the range $[0, y]$, it determines the number of requests that have to be served before it expires. In order to evaluate resource utilization in conditions of resource saturation, we set $r = 1620$. Upon expiring, the embedded SFCs are removed from the network and all reserved resources are released.

7.2 Comparison Methods

Our proposed GA is compared against a baseline greedy algorithm and a state-of-the-art solution. The greedy method, described in Algorithm 5, sorts the nodes of the substrate network and the VNF-graph in descending order based on their capacities and demands. Subsequently, in every substrate node of the sorted list with the largest available capacity, the algorithm maps sequentially the VNFs with the largest computing demand that can fit into the respective node, while taking into account bandwidth constraints. The algorithm terminates as soon as all VNFs have been mapped. If the mapping is not feasible (*i.e.,* when either the computing or bandwidth demands are not met), the execution of the algorithm is terminated with the rejection of the request. The source code for the greedy algorithm is available at [26].

Algorithm 5 Baseline Greedy Algorithm

```
Input: graphs $G_s$, $G_v$
Output: mapping
1: sort $V_s$ of $G_s$ ,$V_v$ of $G_v$
2: for $i = 0$ to $V_v$.size do
3:     for $u = 0$ to $V_s$.size do
4:         if mapping is incomplete then
5:             if $V_v[i]$ fits in $V_s[u]$ and meets bandwidth demands then
6:                 map $V_v[i]$ to $V_s[u]$
7:             end if
8:         end if
9:     end for
10: end for
11: if mapping is complete then
12:     return mapping
13: else
14:     return rejection
15: end if
```

For the comparison against the state-of-the-art, we rely on the heuristic approach employed by BACON [25]. The rationale behind this comparison method is that BACON comprises a prominent SFC embedding solution that has the same objectives with our proposed GA method, *i.e,* VNF consolidation and minimization of inter-rack traffic.

The BACON heuristic is based on the criticality ranking of the VNFs. The criticality of a VNF is defined as proportional to the interconnections it has with other VNFs in the SFC, *i.e.,* the degree of the corresponding virtual node in the VNF-graph. BACON also considers latency between servers that host interacting VNFs as a criterion of efficiency. In our implementation, we consider the hop-count between servers as a measure for latency. The minimization of hop-count comprises an objective of our problem formulation and, in our simulation environment, this implies the minimization of latency between interacting substrate nodes.

BACON applies a server ranking criterion, namely *Betweeness Centrality* (BC), for choosing the most efficient substrate node for hosting each VNF. In structured topologies, such as fat-trees, the substrate nodes exhibit high availability and homogeneous properties, which results in the same ranking for substrate nodes. In order to alleviate this, in each iteration of the simulation, the BACON heuristic considers only the servers that have sufficient capacity to host the VNF (of the SFC) with the minimum demand. As such, the BACON heuristic ranks the servers in descending order based on BC, and in every step it searches for the server with the highest BC that generates an efficient mapping.

Table 3 GA variants vs. supported features

	Basic parameter setup	PAGA	Initial population with heuristic	Structure of procedures
GA-B	X			
GA-H	X		X	
GA-HS	X		X	X
GA-PAGA		X	X	X

7.3 Evaluation Results

7.3.1 Comparison Among GA Variants

We initially seek to gain insights into the efficiency of GAs and the impact of certain features, when these are incorporated into the design and implementation of a GA. To this end, we compare the efficiency among four GA variants, which differ in terms of the following supported features: (i) generation of the initial population using the heuristic described by Algorithm 5, (ii) structure of procedures of sets and groups, exemplified in Sect. 5, and (iii) dynamic parameter adjustment using PAGA, as explained in Sect. 6. Note that the absence of the dynamic parameter adjustment from a GA variant implies the use of the basic parameter setup (see Sect. 6.2). The features supported by each GA variant are illustrated in Table 3.

Figure 7 depicts the request acceptance rates for these GA variants in simulations conducted with the fat-tree (*i.e.,* structured) topology. Relying on *GA-B* for the generation of the initial population with the basic parameter setup yields low acceptance rates. This stems from the fact that the search of the state space is not directed to any specific areas, and, thereby, the algorithm fails to adapt effectively in the search.

Next, we focus on the comparison between the remaining GA variants, which employ the heuristic and lead to notably higher acceptance rates. A comparison between *GA-H* and *GA-HS* uncovers that the structure of procedures enables a more advanced search of the feasible state space, generating better embeddings with slightly higher acceptance rates. The average runtime for handling each request with the basic parameter setup in our system is about 600 msec (this pertains to the three aforementioned GA variants).

The *GA-PAGA* variant, instead, relies on the most sophisticated parameter setup method (Sect. 6.2), which leads to higher SFC embedding efficiency, as shown in Fig. 7. In particular, this gain stems from the fact that after the adjustment procedure has been completed, the behavior of the network is stable with the highest acceptance rate. This gain in the embedding efficiency of *GA-PAGA* is slightly outweighed by an increase in the solver runtime. More specifically, the average runtime for handling each request is 1.1 sec and is proportional to the size of the VNF-graph and the supergeneration parameter.

Our evaluation results indicate a more efficient exploration of the state space by our proposed GA-PAGA, in comparison to the other GA versions. This is mainly attributed to the GA tuning using PAGA, as well as the utilization of the structure

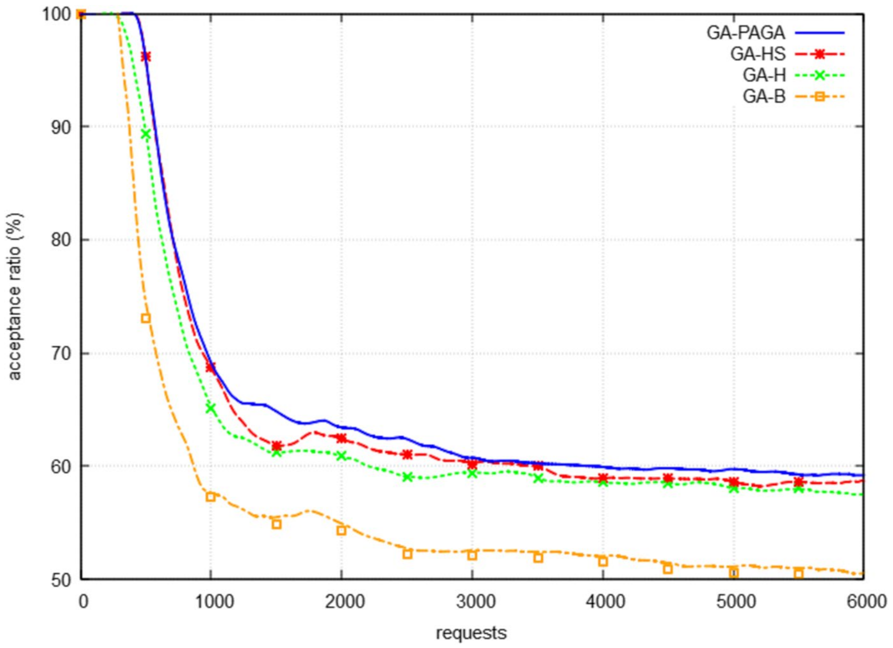

Fig. 7 Request acceptance rates of diverse GA variants

of procedures. In addition, some of the GA variants under consideration (including PAGA) further benefit from the heuristic generation of an initial population that speeds up the convergence into the goal states. These advanced features eventually turn GAs into more efficient computational methods for SFC embedding.

The margins among the GA variants are reflected in the resource utilization, especially when the capacity is saturated. Figure 8 depicts server CPU utilization throughout the entire duration of the simulation. According to this plot, the *GA-PAGA* variant yields more efficient CPU utilization. This plot also corroborates the CPU utilization gains stemming from the advanced features employed by the GA variants *GA-PAGA*, *GA-H*, and *GA-HS*, in contrast to *GA-B* which yields significantly lower efficiency.

7.3.2 Comparison of GA-PAGA with BACON and Greedy

In the following, we focus on the *GA-PAGA* variant, which yields the higher efficiency in terms of acceptance rate and resource utilization among all GA variants. In this respect, we perform a comparison against the greedy algorithm (Algorithm 5) and the BACON heuristic.

We initially compare the three methods in terms of acceptance rate. The simulation results on the fat-tree topology (Fig. 9) indicate that *GA-PAGA* achieves higher acceptance rates than the other methods and converges faster to a steady state. These gains stems from the fact that *GA-PAGA* utilizes the available resources more

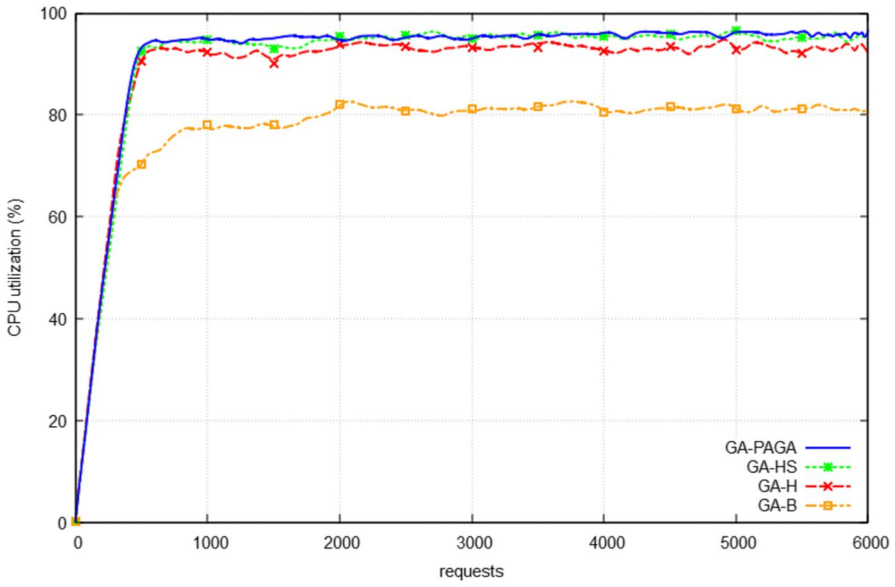

Fig. 8 Server CPU utilization with diverse GA variants

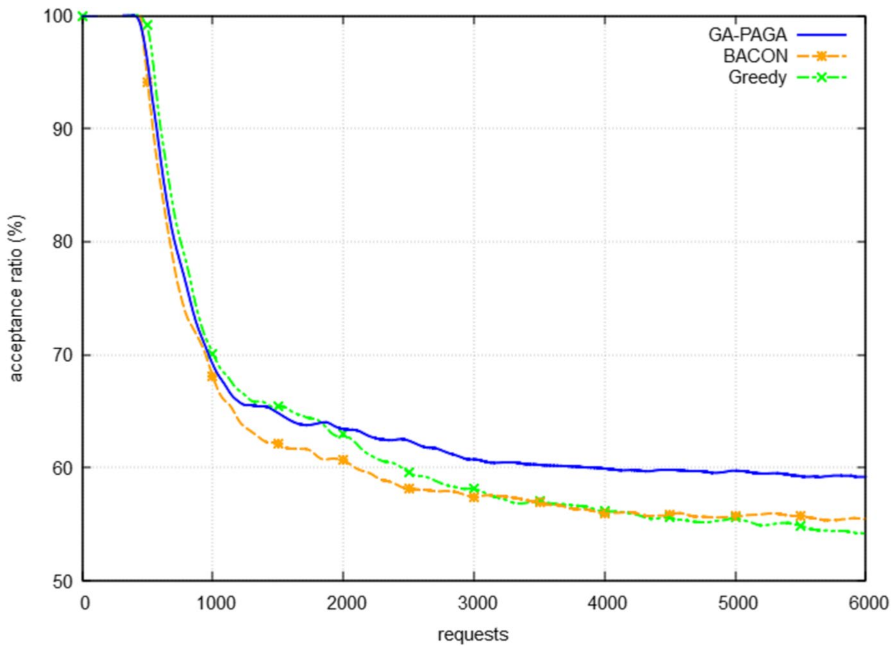

Fig. 9 Request acceptance rates of GA-PAGA, greedy, and BACON

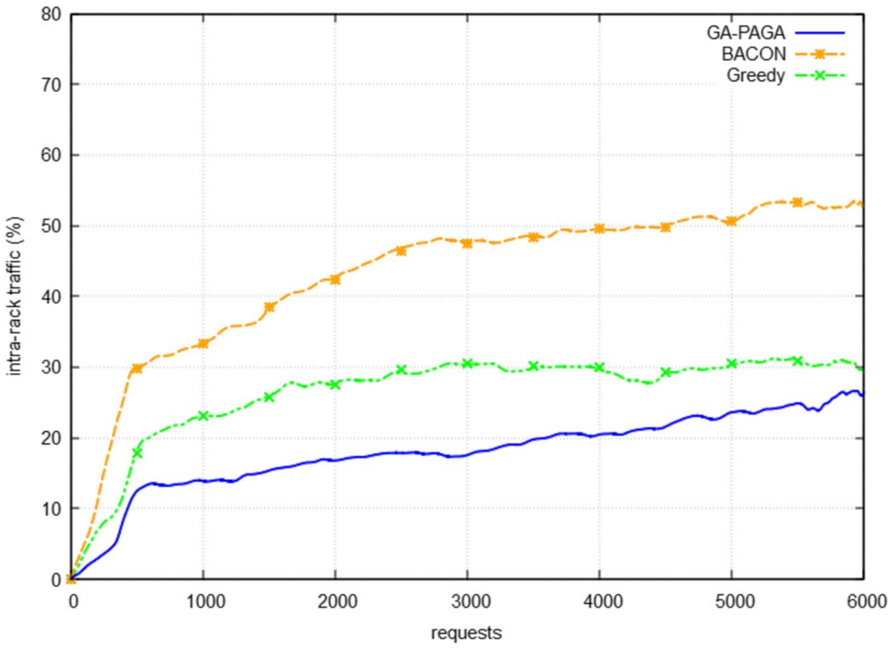

Fig. 10 Intra-rack traffic generated by GA-PAGA, greedy, and BACON

efficiently, generating valid mappings in conditions of saturated bandwidth and capacity by exploiting the state space more effectively.

In the following, we measure the traffic at intra-rack and inter-rack level generated by the embeddings of *GA-PAGA*, greedy, and BACON. Note that *GA-PAGA* and BACON aim at reducing the communication cost among the interacting VNFs (*i.e.,* adjacent VNFs in the SFC graph). To this end, these methods follow different approaches, resulting in different VNF placement strategies. More precisely, *GA-PAGA* strives to place the VNFs of every SFC on the same server in order to eliminate the communication cost among the interacting VNFs. On the other hand, BACON seeks to place interacting VNFs onto the same rack (where its nodes exhibit higher BC), but not necessarily on the same server. High BC is achieved when there is a group of at least three servers on the same rack that may host a subgroup of the VNFs. The greedy algorithm exercises a VNF consolidation strategy, similar to *GA-PAGA*.

Figure 10 illustrates the intra-rack traffic generated by the embedding of SFCs using the three methods under comparison. *GA-PAGA* and the greedy algorithm yield lower volumes of intra-rack traffic, since they achieve a high level of VNF consolidation within the same server. This is not the case for BACON, which generates a substantial amount of traffic within each rack. This stems from the VNF placement strategy explained earlier, which leads to the partitioning of SFC graphs among multiple servers.

According to Fig. 11, *GA-PAGA* further leads to significant bandwidth conservation at the inter-rack level compared to BACON and greedy. This gain of

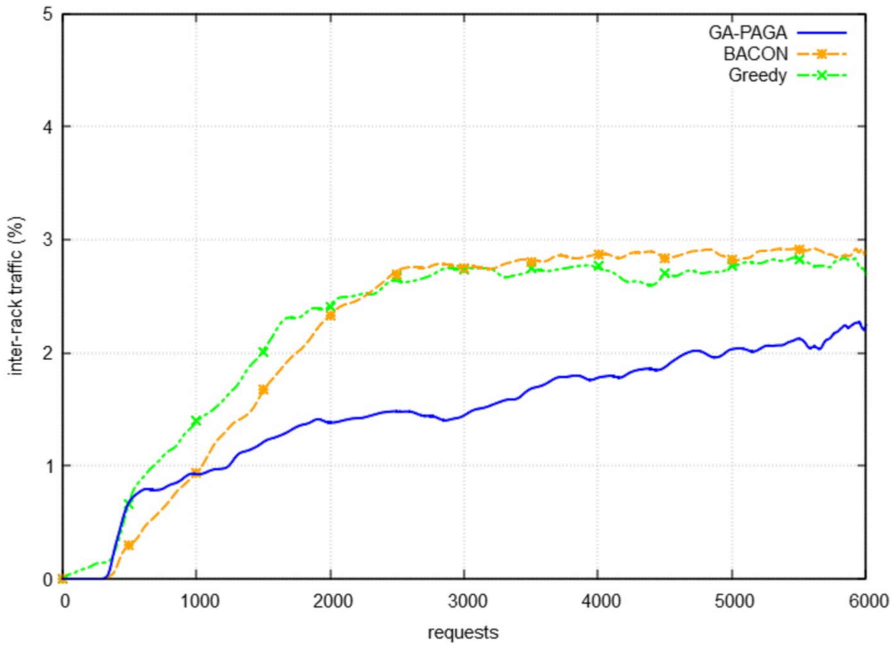

Fig. 11 Inter-rack traffic generated by GA-PAGA, greedy, and BACON

GA-PAGA is more evident for up to 5500 requests. Eventually, the generated inter-rack traffic volumes of all three methods tend to converge, since the utilization of resources within racks enforces the partitioning of SFC graphs across multiple racks. Nevertheless, *GA-PAGA* is more effective in terms of inter-rack traffic reduction with the generation of embeddings that use a minimum number of racks. As shown in Fig. 11, the inter-rack traffic generated by BACON is for most of the time on par with the greedy, as the corresponding curves in the plot converge after 2000 requests.

Figure 12 illustrates the CPU utilization of the servers across the NFVI. *GA-PAGA* yields efficient CPU utilization on par with the greedy and BACON. The latter exhibits a slight advantage in terms of CPU utilization, compared to the other two methods. However, this minor gain cannot outweigh the larger amounts of traffic generated by BACON at the intra- and inter-rack level.

In addition to the request acceptance rate, we employ the *Cost-to-Revenue Ratio (CRR)* in order to quantify the efficiency of the SFC embeddings, inline with [40]. To this end, we initially define the *Revenue* ($\mathbb{R}$) of a SFC request, as:

$$\mathbb{R} = \sum_{n \in N_v} d^n + a * \sum_{i,j \in N_v} e^{i,j}$$

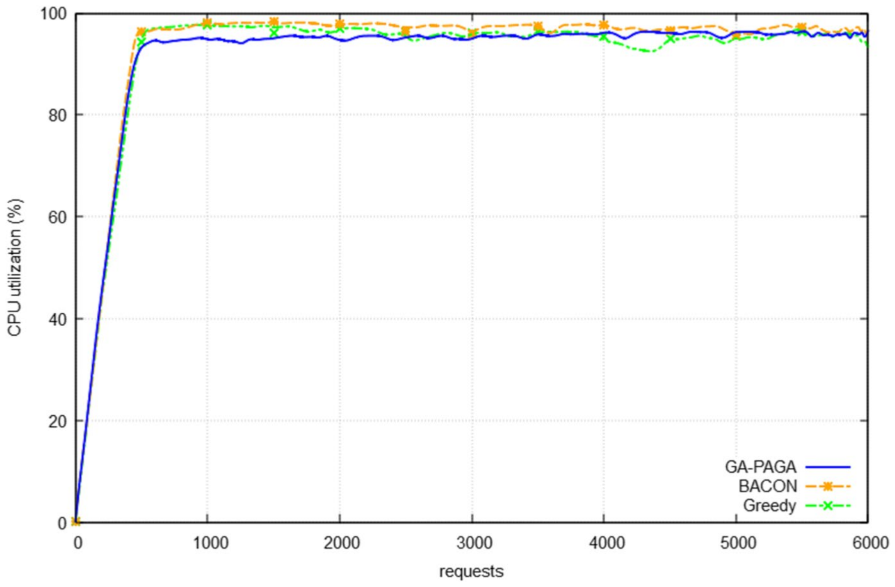

Fig. 12 Server CPU utilization with GA-PAGA, greedy, and BACON

In essence, revenue accumulates all the node and link capacity demands of the SFC request. Furthermore, we define the *Embedding Cost* ($\mathbb{C}$) that essentially accumulates all node and link embedding costs, as follows:

$$\mathbb{C} = \sum_{n \in N_v} d^n + a * \sum_{i,j \in N_v} e^{i,j} * h_{u,z}$$

Note that a is set to 0.5 in order to strike a balance between the two terms in both formulas.

Based on the definitions above, CRR is computed as: $CRR = \mathbb{C} \,/\, \mathbb{R}$. Note that the lower the CRR the better. Practically, CRR is mainly affected by the second term of $\mathbb{C}$, *i.e.,* embeddings with longer hop-counts increase the embedding cost, and, thereby, the CRR. Hence, high CRR values imply a high degree of SFC partitioning among racks.

Figure 13 illustrates the CDF of the CRR for all embedded requests with the three methods. According to this plot, *GA-PAGA* generates embeddings with lower CRR compared to BACON and the greedy. For instance, 80% of the embedded SFCs computed by *GA-PAGA* are associated with a CRR of 1.11 or less. This CRR value indicates a low fragmentation of SFCs among racks and is in accordance with the lower volume of inter-rack traffic generated by *GA-PAGA*.

Lastly, we compare the three SFC embedding methods in terms of solver runtime in our system. The corresponding measurements appear in Table 4. As expected, the greedy algorithm yields the lower runtime, due to its low complexity. *GA-PAGA* exhibits a runtime slightly higher than 1 sec, which is substantially

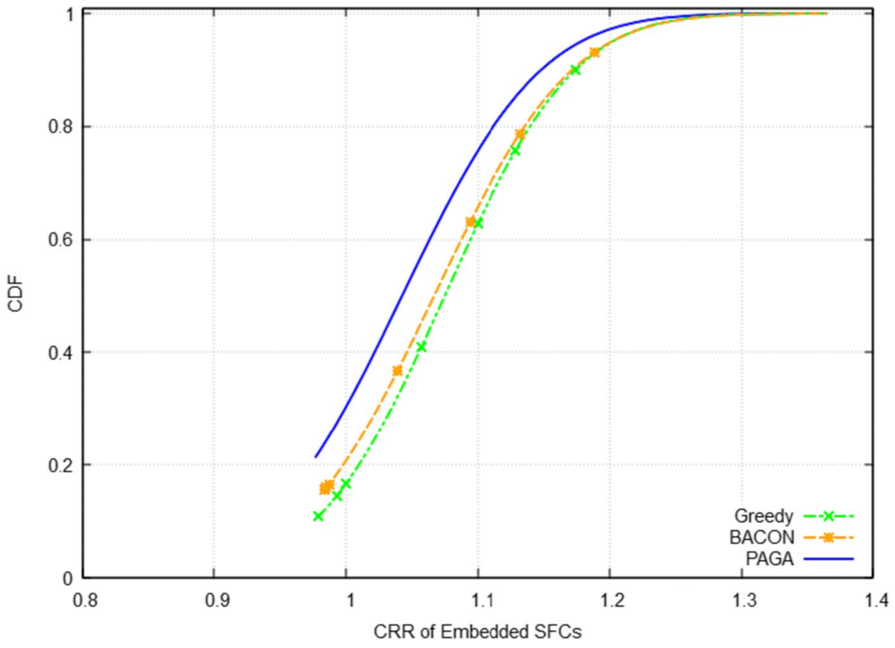

Fig. 13 CDF of Cost-to-Revenue Ratio (CRR) with GA-PAGA, greedy, and BACON

Table 4 Average solver runtime

	Solver runtime
GA-PAGA	1.1 sec
BACON	3.2 sec
Greedy	600 ms

lower compared to the 3.2 sec required by BACON to compute a SFC embedding. The factors that mostly affect the runtime of the *GA-PAGA* are the VNF-graph size and (despite parallelism) the *supergenerations* parameter. Setting *supergenerations* to 2 reduces the runtime of *GA-PAGA* at the level of the greedy.

Overall, *GA-PAGA* compares very favorably against a state-of-the-art method (BACON) and a greedy algorithm. Whereas all three methods exhibit similar (high) levels of CPU utilization, *GA-PAGA* stands out in terms of bandwidth utilization, as its embeddings generate the lower amount of inter-rack traffic. This can allow for a higher level of oversubscription in datacenter network topologies, reducing the expenditure (CapEx) for NFVI operators. Furthermore, if required, *GA-PAGA* can be tuned to generate embeddings in the order of hundreds of msec via parameter adjustment and parallelization.

8 Related Work

We hereby discuss related work on SFC and virtual network embedding (Sect. 8.1) and on the study on the functionality of GAs (Sect. 8.2). We note that most related work relies on machine learning, reinforcement learning, or heuristics, whereas GAs are rarely employed within this scope. We further refer to a range of techniques for the adjustment of GA parameters.

8.1 GAs for Graph Embedding

Authors in [27] present a GA for virtual network embedding. The proposed algorithm is designed, such that it can benefit from its parallel execution on multiple servers. The representation of the chromosomes is complex with variable length, which increases the computational requirements of the algorithm. Furthermore, the evaluation of the algorithm is carried out only for random topologies of relatively small scale. As such, the efficiency of the proposed algorithms in structured topologies (*e.g.,* fat-trees or leaf-spine) is not assessed.

A hybrid GA for virtual machine placement is proposed in [28]. This algorithm is tailored to energy optimization in datacenters. In this algorithm, energy consumption is considered as one of the evaluation parameters of the generated solutions. This approach extends the model of the simple GA by applying a process of repairing invalid solutions and a process of local optimization, enhancing the efficiency of the algorithm. This method is applied and evaluated in small-scale random network topologies.

Another GA for the VNF placement problem is proposed in [29]. VNF chaining is not taken into account; instead, this work addresses the need for VNF scaling. The proposed GA method is shown to be more efficient than an Integer Linear Program (ILP), generating acceptable solutions. The evaluation is limited to small-scale network topologies.

Authors in [41] elaborate on a system that partitions the substrate network into clusters, using the k-medoids method in order to reduce the complexity of the algorithm. A GA and the method of Chemical Reaction Optimization are applied in each cluster separately for virtual network embedding. Both methods perform well in comparison to integer programming. Another related application of GAs is load balancing on virtualized infrastructures [42].

Various techniques for the segmentation of the state space are presented in [43], aiming at a more efficient exploration of the state space. In each technique, an algorithm searches each segment for its optimal solution and, subsequently, the generated solutions from all the segments are combined. The issue with this approach is that a predefined segmentation strategy may not be effective in different network topologies. Such techniques lack the property of adaptability in different instances of the problem, while our approach effectively adapts to different types of network topologies. Comparing clustering and segmentation methods with our technique based on the structure of procedures, we note that the aforementioned methods apply an exploration of the state space defined on a static fashion, whereas our method is dynamic and adaptable to different topologies.

This problem of VNF-graph embedding is addressed in [30] using a GA. The novelty of this approach lies in the addition of a repairer in the algorithm for the correction of invalid solutions. The evaluation of the proposed method is conducted on medium-scale topologies; however, the embedding efficiency is not compared to other existing methods.

Authors in [44] use crossover and mutation operators to avoid population homogeneity by maintaining diversity in the population and delay convergence. A downside of this approach is that it is not efficient for different types of networks. The dynamic parameter adjustment procedure that we have devised does not exhibit this disadvantage.

Using heuristics for the generation of the initial population can direct the search in areas of the state space that are more likely to include an efficient solution, such as in [45]. This is a method that we also employ as described in Sect. 5.

8.2 GA Modeling and Parameter Adjustment

There are various efforts to understand the behavior of GAs. Most popular modeling frameworks include Markov chain modeling [46] and the building blocks hypothesis [47]. Both frameworks exhibit limitations, since there are applications of GAs that they do not cover. In addition, they do not explain the ability of GAs to either efficiently solve (or not) a range of computational problems. Instead, our model captures the behavior of GAs with respect to a class of computational problems that SFC embedding belongs.

Parameter adjustment on GAs is still an open problem, due to the lack of a concrete theoretical framework for their operation [48]. Researchers have adopted empirical methods in order to investigate the optimal parameter values for practical applications of GAs. In most relevant studies, the researchers experiment on problems of static configuration, such as algebraic problems. However, in reality, we often deal with problems the configuration of which is subject to variations during runtime, such as the embedding problem that we study in this work. In our study of parameter adjustment, we take this particular aspect into consideration.

Various studies focus on the optimization of one of the parameters (*e.g.,* population size), while others follow a holistic approach, studying the behavior of the whole set of parameters. We mainly care about the latter, which can lead to more interesting results of practical value. First, we examine studies that aim to draw conclusions about the nature of GAs and then self-adaptation methods integrated on the GAs and executed at run-time.

In [48], the interactions among the parameters of a GA are studied extending previous works on the same direction. The authors study GAs which compute the global optima of a set of algebraic functions. The behavior of each parameter is studied by applying variations on its value and then comparing the results with experiments of the same fashion on the other parameters. The experimental results lack generalization, as they are applied on a specific type of problems and do not lead to any practical solution or to some concrete substantiation on the matter, which could be used in algorithm design.

The investigation of parameter adjustment using a meta-level GA is investigated in [49]. The meta-level algorithm generates a population of possible setups and computes the optimal one. The researchers perform extensive experiments on this design and provide interesting results. In our previous work [24], we utilize a similar method. The downside in both cases is that the methods are applied on static problem instances (*i.e.,* not during runtime), so they lack adaptability on different problem configurations. In the design of PAGA, we overcome these limitations.

The execution in parallel of the same algorithm with different setups is a common approach on self-adaptive GAs. The results from the parallel executions are then compared in order to determine the optimal setup and modify the current parameters of the algorithm. In [50], the optimal setup is determined using reinforcement learning; however, the efficiency of this method is not firmly assessed. In [51], the optimal setup is computed based on the comparison of the outputs from each parallel execution, leading to satisfactory results on problems of static nature.

The parallel execution of the genetic algorithm for the more efficient search of the state space and the avoidance of premature convergence is also used in several studies (*e.g.,* [52]). The major difference between these approaches and the structure of procedures that we employ is that in the aforementioned works the algorithms merely choose the best mapping from the parallel computation outputs, while in our solution the outputs are combined in order to compute a new mapping improving the efficiency of the algorithm.

9 Conclusions

SFC embedding comprises a crucial orchestration aspect for NFV infrastructures. In contrast to various heuristics and exact methods employed to tackle this problem, we investigated the efficiency of AI-assisted embedding, leveraging on genetic algorithms. Our proposed GA-based solution deviates from mainstream GA methods, since it incorporates advanced features, such as the dynamic parameter adjustment depending on the problem configuration and the generation of an initial population using a greedy algorithm. The former achieves the fine-tuning of the GA based on the particular problem instance, whereas the latter augments the GA with the convergence to the desired solution.

Coupling these capabilities with the structure of procedures empowers our most advanced GA variant, *i.e., GA-PAGA*, to confront the computational complexity of VNF-graph embedding and generate efficient solutions. Our evaluation results uncover significant gains for *GA-PAGA* in terms of request acceptance and resource utilization, in comparison to other GA variants that employ a subset of *GA-PAGA's* features. *GA-PAGA* also outperforms a baseline greedy algorithm and a state-of-the-art heuristic with a similar optimization objective. Additional micro-benchmarks indicate that these efficiency gains stem from improved VNF consolidation, which, in turn, yields lower bandwidth consumption.

In future work, we plan to explore the suitability of other methods for AI-assisted resource orchestration across multiple domains, and identify potential gains and trade-offs compared to existing heuristic and exact methods. This problem entails challenging aspects, such as the need for privacy preservation and autonomous decision making within each domain. In this respect, federated learning comprises a promising approach that we will harness.

Author Contributions All authors contributed in the preparation of all the sections of the manuscript.

Funding This research was funded by the European Union's Horizon Europe research and innovation program under grant agreement No. 101070487 (NEPHELE). The publication of the article in Open Access mode was financially supported by HEAL-Link.

Data Availibility Not applicable.

Code Availability The code used for the implementation of the presented algorithms and procedures is available in [26] as an open source project.

Declarations

Conflicts of interest The authors confirm that there are no competing interests between the authors and the organizations of the authors.

Ethical Approval Not applicable.

References

1. ETSI Network Function Virtualization. http://www.etsi.org/technologies-clusters/technologies/nfv
2. Kourtis, M.-A., et al.: T-nova: an open-source mano stack for nfv infrastructures. IEEE Trans. Netw. Serv. Manag. **14**(3), 586–602 (2017)
3. Papathanail, G., Pentelas, A., Fotoglou, I., Papadimitriou, P., Katsaros, K.V., Theodorou, V., Soursos, S., Spatharakis, D., Dimolitsas, I., Avgeris, M., Dechouniotis, D., Papavassiliou, S.: Meson: optimized cross-slice communication for edge computing. IEEE Commun. Mag. **58**(10), 23–28 (2020)
4. Sherry, J., et al.: Making middleboxes someone else's problem: network processing as a cloud service. ACM SIGCOMM Comput. Commun. Rev. **42**(4), 13–24 (2012)
5. Dietrich, D., et al.: Multi-provider service chain embedding with Nestor. IEEE Trans. Netw. Serv. Manag. **14**(1), 91–105 (2017)
6. Abujoda, A., Papadimitriou, P.: Midas: middlebox discovery and selection for on-path flow processing. In: IEEE COMSNETS (2015)

7. Dietrich, D., *et al.*: Network function placement on virtualized cellular cores. In: IEEE COMSNETS, pp. 259–266 (2017)
8. Abujoda, A., Papadimitriou, P.: Distnse: Distributed network service embedding across multiple providers. In: IEEE COMSNETS (2016)
9. Papagianni, C., *et al.*: Rethinking service chain embedding for cellular network slicing. In: IFIP Networking, pp. 1–9 (2018)
10. Chakrabarti, A., Chekuri, C., Gupta, A., Kumar, A.: Approximation algorithms for the unsplittable flow problem. Algorithmica **47**(1), 53–78 (2007)
11. Dräxler, S., Karl, H., Mann, Z.Á.: Jasper: joint optimization of scaling, placement, and routing of virtual network services. IEEE Trans. Netw. Serv. Manag. **15**(3), 946–960 (2018)
12. Gong, L., Jiang, H., Wang, Y., Zhu, Z.: Novel location-constrained virtual network embedding lc-vne algorithms towards integrated node and link mapping. IEEE/ACM Trans. Netw. **24**(6), 3648–3661 (2016)
13. Amaldi, E., Coniglio, S., Koster, A.M., Tieves, M.: On the computational complexity of the virtual network embedding problem. Electron. Notes Discret. Math. **52**, 213–220 (2016)
14. Figiel, A., Kellerhals, L., Niedermeier, R., Rost, M., Schmid, S., Zschoche, P.: Optimal virtual network embeddings for tree topologies. In: Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures. SPAA '21, pp. 221–231 (2021)
15. Colorni, A., Dorigo, M., Maffioli, F., Maniezzo, V., Righini, G., Trubian, M.: Heuristics from nature for hard combinatorial optimization problems. Int. Trans. Oper. Res. **3**(1), 1–21 (1996)
16. Cohen, R., Lewin-Eytan, L., Naor, J.S., Raz, D.: Near optimal placement of virtual network functions. In: IEEE INFOCOM (2015)
17. Basta, A., Kellerer, W., Hoffmann, M., Morper, H.J., Hoffmann, K.: Applying nfv and sdn to lte mobile core gateways, the functions placement problem. In: Proceedings of the 4th Workshop on All Things Cellular, pp. 33–38 (2014)
18. Pentelas, A., Papathanail, G., Fotoglou, I., Papadimitriou, P.: Network service embedding across multiple resource dimensions. IEEE Trans. Netw. Serv. Manag. **18**(1), 209–223 (2021)
19. Harutyunyan, D., Riggio, R.: Flex5g: flexible functional split in 5g networks. IEEE Trans. Netw. Serv. Manag. **15**(3), 961–975 (2018)
20. Riggio, R., Bradai, A., Harutyunyan, D., Rasheed, T., Ahmed, T.: Scheduling wireless virtual networks functions. IEEE Trans. Netw. Serv. Manag. **13**(2), 240–252 (2016)
21. Renzi, C., Leali, F., Cavazzuti, M., Andrisano, A.O.: A review on artificial intelligence applications to the optimal design of dedicated and reconfigurable manufacturing systems. Int. J. Adv. Manuf. Technol. **72**(1–4), 403–418 (2014)
22. Guo, H., Hsu, W.H.: A machine learning approach to algorithm selection for NP-hard optimization problems: a case study on the mpe problem. Annals Operations Res. **156**(1), 61–82 (2007)
23. Zhang, B., Fan, Q., Zhang, X., Fu, Z., Wang, S., Li, J., Xiong, Q.: A survey of vnf forwarding graph embedding in b5g/6g networks. Wirel. Netw. (2021). https://doi.org/10.1007/s11276-021-02741-9
24. Rodis, P., Papadimitriou, P.: Intelligent network service embedding using genetic algorithms. In: 2021 IEEE Symposium on Computers and Communications (ISCC), pp. 1–7 (2021). IEEE
25. Hawilo, H., Jammal, M., Shami, A.: Network function virtualization-aware orchestrator for service function chaining placement in the cloud. IEEE J. Sel. Areas Commun. **37**(3), 643–655 (2019)
26. Rodis, P.: SFC Embedding simulator and algorithms source code. https://rodispantelis.github.io/SFC-Embedding/
27. Lu, Q., Nguyen, K., Huang, C.: Distributed parallel algorithms for online virtual network embedding applications. Int. J. Commun. Syst. **36**, 4325 (2020)
28. Tang, M., Pan, S.: A hybrid genetic algorithm for the energy-efficient virtual machine placement problem in data centers. Neural Process. Lett. **41**(2), 211–221 (2015)
29. Rankothge, W., Le, F., Russo, A., Lobo, J.: Optimizing resource allocation for virtualized network functions in a cloud center using genetic algorithms. IEEE Trans. Netw. Serv. Manag. **14**(2), 343–356 (2017)
30. Pham, T.A.Q., Sanner, J.-M., Morin, C., Hadjadj-Aoul, Y.: Virtual network function-forwarding graph embedding: a genetic algorithm approach. Int. J. Commun. Syst. **33**(10), 4098 (2020)
31. Holland, J.H.: Adaptation in natural and artificial systems, university of Michigan press. Ann Arbor MI **1**(97), 5 (1975)
32. Diveev, A., Bobr, O.: Variational genetic algorithm for np-hard scheduling problem solution. Procedia Comput. Sci. **103**, 52–58 (2017)

33. Green, D., Aleti, A., Garcia, J.: The nature of nature: why nature-inspired algorithms work. Nature-inspired computing and optimization, pp. 1–27. Springer, Berlin (2017)
34. Paul, P.V., Moganarangan, N., Kumar, S.S., Raju, R., Vengattaraman, T., Dhavachelvan, P.: Performance analyses over population seeding techniques of the permutation-coded genetic algorithm: an empirical study based on traveling salesman problems. Appl. Soft Comput. **32**, 383–402 (2015)
35. Graham, R.L., Hell, P.: On the history of the minimum spanning tree problem. Annals Hist. Comput. **7**(1), 43–57 (1985)
36. Moharam, R., Morsy, E.: Genetic algorithms to balanced tree structures in graphs. Swarm Evolut. Comput. **32**, 132–139 (2017)
37. Contreras-Bolton, C., Gatica, G., Barra, C.R., Parada, V.: A multi-operator genetic algorithm for the generalized minimum spanning tree problem. Expert Syst. Appl. **50**, 1–8 (2016)
38. Marchiori, E.: Genetic, iterated and multistart local search for the maximum clique problem. Workshops on applications of evolutionary computation, pp. 112–121. Springer, Berlin (2002)
39. Lim, S.M., Sultan, A.B.M., Sulaiman, M.N., Mustapha, A., Leong, K.Y.: Crossover and mutation operators of genetic algorithms. Int. J. Mach. Learn. Comput. **7**(1), 9–12 (2017)
40. Dietrich, D., Papadimitriou, P.: Policy-compliant virtual network embedding. In: 2014 IFIP Networking Conference, pp. 1–9 (2014). IEEE
41. El Mensoum, I., Wahab, O.A., Kara, N., Edstrom, C.: Musc: a multi-stage service chains embedding approach. J. Netw. Comput. Appl. **159**, 102593 (2020)
42. Lagwal, M., Bhardwaj, N.: Load balancing in cloud computing using genetic algorithm. In: 2017 International Conference on Intelligent Computing and Control Systems (ICICCS), pp. 560–565 (2017). IEEE
43. Squillero, G., Tonda, A.: Divergence of character and premature convergence: a survey of methodologies for promoting diversity in evolutionary optimization. Information Sci. **329**, 782–799 (2016)
44. Hrstka, O., Kučerová, A.: Improvements of real coded genetic algorithms based on differential operators preventing premature convergence. Adva. Eng. Softw. **35**(3–4), 237–246 (2004)
45. Rocha, M., Neves, J.: Preventing premature convergence to local optima in genetic algorithms via random offspring generation. International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, pp. 127–136. Springer, Berlin (1999)
46. Schmitt, L.M.: Theory of genetic algorithms. Theoretical Comput. Sci. **259**(1–2), 1–61 (2001)
47. Sastry, K., Goldberg, D., Kendall, G.: Genetic algorithms. Search Methodol. (2005). https://doi.org/10.1007/0-387-28356-0_4
48. Deb, K., Agrawal, S.: Understanding interactions among genetic algorithm parameters. Found. Genet. Algorithms **5**(5), 265–286 (1999)
49. Sipper, M., Fu, W., Ahuja, K., Moore, J.H.: Investigating the parameter space of evolutionary algorithms. Bio Data Mining **11**(1), 1–14 (2018)
50. Pellerin, E., Pigeon, L., Delisle, S.: Self-adaptive parameters in genetic algorithms. Data mining and knowledge discovery: theory, tools and technology, pp. 53–64. International Society for Optics and Photonics, Bellingham (2004)
51. Tongchim, S., Chongstitvatana, P.: Parallel genetic algorithm with parameter adaptation. Information Process. Lett. **82**(1), 47–54 (2002)
52. Nguyen, K.T.D., Huang, C.: An intelligent parallel algorithm for online virtual network embedding. In: 2019 International Conference on Computer, Information and Telecommunication Systems (CITS), pp. 1–5 (2019)

Panteleimon Rodis is a Ph.D. candidate at the Department of Applied Informatics in the University of Macedonia, Greece. He obtained a B.Sc. in Computer Science in 2011 and a M.Sc. in Engineering of Pervasive Computing Systems in 2020 both from Hellenic Open University. His research interests include the applications of Artificial Intelligence in virtual network embedding and VNF orchestration.

Panagiotis Papadimitriou is an Associate Professor at the department of Applied Informatics in the University of Macedonia, Greece. Before that, he was an Assistant Professor at the Communications

Technology Institute of Leibniz Universität Hannover, Germany, and a member of L3S research center in Hanover. He received a Ph.D. in Electrical and Computer Engineering from Democritus University of Thrace, Greece, in 2008, a M.Sc Information Technology from University of Nottingham, UK, in 2001, and a B.Sc. in Computer Science from University of Crete, Greece, in 2000. He has been a (co-)PI in several EU-funded (*e.g.,* NEPHELE, NECOS, T-NOVA, CONFINE) and nationally-funded projects (*e.g.,* G-Lab VirtuRAMA, MESON). Panagiotis was a recipient of Best Paper Awards at IFIP WWIC 2012, IFIP WWIC 2016, and the runner-up Poster Award at ACM SIGCOMM 2009. He has co-chaired several international conferences and workshops, such as IFIP/IEEE CNSM 2022, IFIP Networking TENSOR 2020–2023, IEEE NetSoft S4SI 2020, IEEE CNSM SR+SFC 2018–19, IFIP WWIC 2016–2017, and INFOCOM SWFAN 2016. Panagiotis is also an Associate Editor of IEEE Transactions on Network and Service Management, and a Senior Member of IEEE. His research activities include (next-generation) Internet architectures, network processing, programmable dataplanes, time-sensitive networking (TSN), and edge computing.