Datacenter Optimization Methods for Softwarized Network Services

Luigi Pannocchi[a,*], Sourav Lahiri[b], Silvia Fichera[b], Antonino Artale[b], Tommaso Cucinotta[a,*]

[a]*Scuola Superiore Sant'Anna, Via Moruzzi, 1, 56124 Pisa, Italy*
[b]*Vodafone, Milan, Italy*

Abstract

This paper tackles the problem of optimum Virtual Machine placement, focusing on an industrial use-case dealing with capacity planning for Virtualized Network Functions. The work is framed within an industrial collaboration with the Vodafone network operator, where a particularly important problem is the one of optimum deployment of Softwarized Network Functions within their Virtualized Networking Infrastructure, spanning across several EU countries. The problem is particularly difficult due to the presence of a multitude of placement constraints that are needed in the industrial use-case, including soft affinity constraints, that should be respected only as secondary objective; furthermore, in some EU regions, the size of the problem makes it unfeasible to solve it with traditional MILP-based techniques.

In this work, we review and address limitations of previously proposed heuristics for this kind of problems, and propose a new placement strategy that is shown experimentally to be more effective in dealing with soft affinity constraints. The paper includes an extensive experimental evaluation encompassing a multitude of optimization strategies, applied to a set of problems including both real-world problems that we made available as an open data-set, and additional randomly generated problems mimicking the structure of the original real-world problems.

Keywords: Virtual Machine Placement, Resource Optimization, Genetic Algorithms, Network Function Virtualization

1. Introduction

In the nowadays computing landscape, heavily focused on Cloud and distributed computing scenarios, network services are bearing higher and higher responsibilities for guaranteeing the success of modern and future digital use-cases [1, 2]. Modern

[*]Corresponding author.
Email addresses: `luigi.pannocchi@santannapisa.it` (Luigi Pannocchi), `sourav.lahiri@vodafone.com` (Sourav Lahiri), `silvia.fichera@vodafone.com` (Silvia Fichera), `antonino.artale@vodafone.com` (Antonino Artale), `tommaso.cucinotta@santannapisa.it` (Tommaso Cucinotta)

scenarios, such as Smart Cities, Smart Transportation and self-driving vehicles, Industry 4.0 and smart factory automation, as well as more traditional ones, like Web Services, online gaming, and virtual and augmented reality applications, are increasingly based on high-bandwidth, ultra-reliable, low-latency and mobile communications for their successful implementation and deployment [3]. As a consequence, network operators are updating their networking infrastructures and operations practices, in order to face the new challenges arising from these emerging use-cases, for example with a widespread adoption of 5G architectures in access networks [4]. Their absolutely non-trivial business goal is to design, build, and deploy large-scale networking infrastructures that are characterized by high adaptability and dynamism at an unprecedented level, to be able to cope with the diversity and multitude of workload scenarios that new applications are bringing.

Among the key technologies providing a great degree of flexibility and adaptability in computing and networking infrastructures, we can find *virtualization*, that has recently seen widespread adoption in a number of cases: from system-level virtualization and, more recently, lightweight container-based virtualization, used at large in Cloud Computing infrastructures, to network virtualization and overlays, widely used in access, core, and data-center networks. It does not come as a surprise that modern telecommunication networks are seeing an ever-growing adoption of Cloud Computing principles and paradigms [5], to deploy heavily virtualized computational services within their infrastructure. Indeed, to enhance dynamism and efficiency, network operators are progressively shifting the management of their network functions, from a traditional approach based on *hardware appliances* statically sized for peak-hour operation, to a more dynamic, software-based, and heavily virtualized approach, leveraging Software Defined Networking (SDN) [6] and Network Function Virtualization (NFV) [7, 8, 9]. This allows for: a more automated and dynamic management of network configurations; the implementation of network functions *in software* as Virtual Machines or Containers deployed on general-purpose servers; the use of *horizontal elasticity* typical of the Cloud Computing paradigm, to scale individual Virtual Network Functions (VNFs) so to keep the computational capacity of the deployed functions aligned with the continuously changing conditions of their workload.

In this context, a key problem is the one of an optimum, or near-optimum, deployment of new VNFs within an NFV infrastructure, with a variety of possible objectives, like minimizing the number of used hosts, their overall energy consumption, the expected service latency, or others.

1.1. Paper Contributions

In this paper, we tackle the challenging problem of optimum deployment of VNFs throughout the physical hosts of an NFV infrastructure. We consider the problem as arising from the industrial use-case provided by Vodafone, where VNFs are deployed as sets of Virtual Machines with attached: precise *resource demands*, including CPU, memory, and networking requirements per Host; precise *anti-affinity constraints*, as often needed to deal with reliability and fault-tolerance requirements; *affinity constraints*, often useful to reduce the service interaction latency, and improve the performance experienced in the use of end-to-end network function service chains; and others. In our use-case, the primary objective of the optimization is set as minimizing the number of active hosts needed for a given deployment, whereas the affinity constraints are inherently *soft*, meaning that their satisfaction is desirable but they could be broken, to reduce the deployment size.

We consider three different approaches to tackle the just mentioned problem, each exhibiting different properties:

1. a formulation based on a Mixed Linear Programming (MILP) optimization program, that can be solved exactly with a standard solver, but exhibiting high (and sometimes prohibitive) solving times;
2. a faster heuristic solver based on a traditional Genetic Approach;
3. a Policy-based heuristic solver that is optimized using a Genetic Algorithm, inspired by the work in [10].

Execution speed, optimality guarantees, or capability to adapt to different problem's flavors, can play a role when selecting an algorithm. From a practical point of view, the context in which these problems are solved may change, and this entails a preference for one approach with respect to one another. The first two methods were also the subject of our previous work in [11]. This is extended in the present paper, where we: provide more extended implementation details; and introduce for the first time the 3rd heuristic mentioned above, developed as a multi-dimensional generalization of the work in [10], showing its capability to deal with soft affinity constraints; provide a more extended and insightful evaluation of the three solving strategies, thanks also to the help of a new, more extended, data-set, synthetically built to mimick the main characteristics of our available open data-set with real-world problems.

1.2. Paper Organization

The paper is organized as follows. Section 2 describes the problem of optimum placement of VNFs for NFV, as arising in the Vodafone context, clarifying what are the features coming into play and what are the goals of the work. Section 3 provides an overview of prior related works on the subject in the research literature, highlighting interesting research directions. Section 4 details how the mentioned problem can be tackled using a classical optimization approach, leveraging a formulation based on Mixed-Integer Linear Programming (MILP), that can be solved with a standard MILP solver. Section 5 presents in detail a heuristic solver based on a Genetic Approach, that appeared already in [11]. Section 6 introduces the new Policy-based heuristic solver, which was developed to address the limitations of the other two approaches. Section 7 presents an extended experimental comparison of the three approaches, comparing their advantages and disadvantages. Finally, Section 8 provides a few concluding remarks, along with a sketch of possible future directions for further research on the topic.

2. Scenario

This section describes the NFV capacity planning problem at Vodafone, with the goal to introduce and motivate the problem, explicitly identifying what are the variables at play and the objectives that can be involved in the task.

The problem under consideration has been described in our previous work [11] being extended in this paper. There, we presented the general architecture and prototype implementation, deployed within the Google Cloud Engine, of the software realized to tackle the optimal placement of VNFs within the Vodafone NFV infrastructure, managed through the VMWare vRealize Operations management software[1], under affinity and anti-affinity constraints, with the goal of minimizing the number of used hosts. This is framed within a wider set of tools being experimented at Vodafone, leveraging ML/AI techniques for infrastructure management, as detailed in [12].

The mentioned capacity planning problem can be tackled in different ways, as described in the different solution strategies considered later in Sections 4 to 6.

In reference to Fig. 1, when laying down a Virtual Network Infrastructure, the final goal is to deploy a set of Virtual Network Functions. Each VNF component is in turn composed of a set of Virtual Machines implementing the VNF-specific functionality. These VMs have to be deployed on physical servers, also referred to as Hosts, which constitute the physical infrastructure.

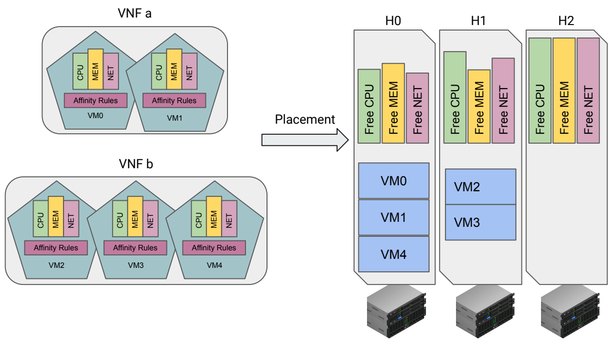

Figure 1: Deployment of Network Functions, previously implemented using custom Hardware, as Virtual Network Functions on a Physical Infrastructure composed by general-purpose servers.

Note that, while this paper focuses on VM placement, a similar approach is being applied within the operator also for tackling the problem of optimum placement of Containerized Network Functions (CNFs) [11]. In general, it is commonplace to use both VM-based and Container-based deployments in NFV infrastructures. VMs provide a higher degree of isolation, security, and compatibility, whilst Containers are known to exhibit lower overheads and better performance. Our case-study, extending the work presented in [11] and based on the open data-set released with that paper, focuses on VM-based placement of VNFs, albeit the approach is easily generalizable to the context of optimum placement of CNFs.

2.1. Resource Requirements

Each VNF component, to be deployed as a VM, is characterized by a multitude of resource requirements, including: the number of vCPUs; amount of memory (RAM);

[1]More information at: `https://docs.vmware.com/en/vRealize-Operations/index.html`.

disk size for the local disk, as well as for remotely mounted volumes, along with bandwidth requirements for accessing them; network requirements for horizontal (East/West) and vertical (North/South) traffic; and others. However, the investigation carried out in this paper is limited to considering only the 3 requirements of vCPUs, RAM memory, and (aggregate) network bandwidth. In our considered NFV use-case, these are the primary factors impacting the placement decisions, whilst others can be neglected. However, the techniques presented in the following sections can easily be extended with additional resource types, such as disk space or others. Clearly, depending on the specific case, such extensions could lead to computationally expensive problems. For example, a considered variant of the problem, albeit not detailed here for the sake of brevity, is the one considering also NUMA Nodes and NUMA (anti-)affinity constraints. These lead easily to doubling the size of the problem for the MILP implementation.

In the scenario considered here, it is assumed that all the VMs of the same VNF component have the same resource requirements. Having multiple identical VMs for a VNF component is justified, for example, as it eases the design of a load-balancer spreading the traffic among them, or for reliability purposes where multiple identical VMs are kept in sync for a better resiliency to faults.

In the placement problem, the aggregate resource requirements of the VMs deployed onto a host should normally not exceed the corresponding availability of resources on the physical hosts, to avoid overcommiting them. Actually, part of the physical resources are already subtracted from the per-host availability, as they are dedicated to the management of the infrastructure. However, for some of the clusters managed in the considered use-case, it is admissible to overcommit resources. This is possible for VNFs that are not performance critical, and for which it is evident from the collected monitoring data that the effective resource utilization is significantly lower than the "official" per-VM requirements. Therefore, in our problem, we consider an over-commitment factor for the various resources.

2.2. Structural Requirements

Given the complexity of a Network Infrastructure, the placement of the VNF components has to consider also their relative location (for some of them). For example, putting VMs on the same host is beneficial in terms of achievable communication bandwidth and latency. On the contrary, making sure that VMs are placed on different hosts is useful from a performance viewpoint, to mitigate noisy neighborhood problems; or, it becomes mandatory from a reliability viewpoint, to implement fault-tolerant policies where a single host failure cannot take down all the VMs of a service at the same time. Therefore, among the considered requirements of VNF components, we have also "structural requirements", such as affinity constraints that control how the VMs should be allocated with respect to one another. Constraints involving the VMs of a single VNF are called "auto-affinity" rules, whilst those involving VMs from different VNFs are called "cross-affinity" rules.

Another important aspect of the planning task is related to the sectioning of the infrastructure. There are physical, logistic, and managerial limits that constitute constraints for the placement. For example, it is necessary to consider limits coming both from the physical infrastructure (e.g., available space in server rooms, data-center network cabling) and from the software used to manage the virtualized infrastructure.

Therefore, in our use-case, hosts need to be organized in separate clusters, where each cluster is characterized by a maximum number of physical hosts and individual VNF components to be placed have to fit within a single cluster (they cannot be split across them).

2.3. Objectives

In the above subsections, we described the constraints that characterize a feasible solution to the placement problem. In this subsection, we focus on the optimization objective to be pursued by the placement strategy.

In this investigation, the focus is on *minimizing the number of hosts* needed to deploy a given VNF workload. This has the benefit of leading to the minimization of the associated costs, as well as minimizing the energy footprint of the deployed workload.

However, as mentioned above, in our considered problem we have also additional factors. The need for partitioning the given VNF workload into separate clusters introduces a *secondary objective* related to the minimization of the number of clusters obtained during the optimization process. Additionally, the presence of soft constraints, that might be relaxed to avoid increasing the required number of physical hosts, introduces an *additional (tertiary) objective*, related to the degree of violation of the soft constraints in place. Indeed, it is likely that the presence of additional structural constraints leads to an increase in the number of hosts necessary to deploy the system.

Therefore, in our use-case we end-up considering a multi-objective problem, where the goal of the optimization is to minimize the primary objective, the number of hosts, then to minimize the additional secondary objectives, i.e., clustering the workload into the minimum number of clusters, and, eventually, minimize the number of affinity constraints that are possibly broken in the placement.

As explained in the next section, these 3 objectives can be added-up into a single objective function, where a careful choice of the weights multiplying each sub-objective can lead to a variety of policies, in the optimization. In the investigation that follows for the considered NFV use-case, the highest priority is assigned to reducing the number of hosts, with the secondary objective of minimizing the number of used clusters, and finally minimizing the number of broken soft affinity constraints, keeping the affinity requirements as a “good-to-have” property, rather than a constraint.

In Section 4, we introduce a precise formalization of the considered VM placement problem, as discussed above.

3. Related Work

In this section, we provide an overview of key related research works that are closely related to the problem and approach investigated in the present paper.

Telecommunication companies welcomed the shift towards NFV, thanks to the many advantages it can provide. Under the new architecture, it is possible to optimize the infrastructure, pursuing a trade-off between the costs and the quality of service. Similarly to what happens in Cloud Computing, NFV infrastructures have the ability to decide on VM and Container placement, and to move them when needed, trying to minimize the service downtimes, maximizing availability, and respecting the service-level agreements (SLAs) possibly in place with the customers, while complying with security regulations.

An outline of the taxonomy of the VM placement techniques that appeared in the Cloud Computing research literature can be found in [13]. In that work, the authors highlight the complexity of the Virtual Machine Placement problem looking at the different research directions and problem formulation that have been proposed in recent years.

Wanting to get some references in the context of NFV, in [14], an overview of VM and container placement strategies studied in the literature can be found, with a specific focus on NFV placement problems. A comprehensive survey of resource allocation techniques specifically applied to NFV data centers can be found in [15]. The panorama that can be framed, putting together all these survey works, is showing up as a multifaced problem that other researchers are sectioning and observing from different points of view when solving for their specific goals.

For example, in the more general context of Cloud Computing data centers, authors in [16] propose a framework for efficient resource management, where, by anticipating the resource utilization of servers, it is possible to apply informed placement and migration decisions throughout the data center, leading to improvements in load-balancing and power consumption.

Different flavors of the baseline placement problem could include some extra features, like the task of balancing the load, as shown by Xu et al. [17]. In their case, the surveyed research works dealing with the workload distribution among servers to avoid imbalanced resource utilization, which in turn could lead to performance degradation. One specific example of load balancing could be found in the work of Hieu et al. [18], where they come up with an algorithm that is able to place Virtual Machines on servers balancing the utilization of multiple resource types.

When considering the performance of the infrastructure more explicitly, and going into the details of ”what is running”, the problem formulation starts embedding awareness of the client workload. Among the indicators that should be looked at when allocating VMs, are the performance of the applications and the presence of Service Level Agreements with the customers. In this respect, some works propose an application-aware placement algorithm. One example of this type of approach is given by the work of Gupta et al. [19], where, in the context of High-Performance Computing (HPC), the placement problem is tackled by keeping an eye on the performance of the applications to avoid breaking possible SLAs.

Given the networking nature of datacenters, many works deal with the problem of allocating the workload paying attention to latency and bandwidth. Similar to the “application awareness”, in this case, allocation algorithms are equipped with “network awareness”, as done by Alicherry et al. in [20]. In that work, the authors considered the geographical distribution of datacenters and the network link properties when dealing with client requests for cloud-computing resources. It is possible also to find works dealing with the deployment of the infrastructure to improve the service capability, for example by increasing the volume of manageable traffic as done by Ma et al. [21]

Once the problem is framed, and the goals are clear, the next step that is needed is the selection of the solution approach. Also in this respect, the panorama is very heterogeneous and reflects the multitude of practical applications that orbit around the topic of Cloud Computing.

Some of the approaches that have been proposed leverage stochastic analysis that helps deal with resource uncertainty. Zhou et al. [22] developed a stochastic meta-heuristic algorithm to perform VM placement, optimizing for energy cost, when the demands are not deterministic. Another example of a probabilistic approach is given by the works of Konstanteli et al. [23, 24]. In their case, they were allocating services on virtualized resources by trying to use an efficient heuristic that uses probabilistic information about service demands to increase the allocation capability.

It is not always the case that workload allocation is modeled as a stochastic system. Indeed, in many cases, the problems are tackled by using a more deterministic approach, for example, by casting the requirements into a classical optimization problem and solving it with a MILP solver, or other approaches. One example is given by Cucinotta et al. [25] in their work on data center optimization, where the authors solved the problem of efficiently managing computing and networking resources by solving a BLP problem, or the more recent work in [26], where a logic programming approach was proposed to tackle this kind of problems. This works for small-size problems, but alternatives are needed for larger problems, which constitute many real-world scenarios.

Some authors [27] considered also the problem of VM placement with affinity constraints, following the same rationale used in our work. However, they excluded the MILP approach as viable due to the computational burden and they relied on custom algorithms to pack the VMs on servers.

Still avoiding the MILP approach but supporting optional affinity constraints, the work of Dow [28] is worth mentioning. It presented a method to efficiently address the VM placement problem, proposing an optimized algorithm able to handle problems in the range of 100 hosts, relying on the assumption that Virtual Machines characteristics are not infinitely variable in resource consumption, but instead belonging to a limited set of few different types. In our case, this approach won't fit very well due to the variability of the VMs characteristics.

As in many works reported above, in case the problem starts growing too much in size, authors lean towards specific heuristics that perform well in their specific case. These heuristics require skills and experience to be found and tuned properly. This constitutes an interesting spot for techniques embedding Artificial Intelligence. There are authors who experimented with Machine Learning (ML), or precisely Reinforcement Learning (RL) techniques, to find good placement strategies. One example of such works is given by Long et al [29], where the authors used Reinforcement Learning to allocate VMs and optimize for energy consumption. In [30], the authors deal with the VM placement problem, while optimizing for energy usage and considering QoS, using a Reinforcement Learning algorithm supported by fuzzy logic.

Huang et.al [31] used a Deep Reinforcement Learning algorithm to tackle the problem of online job scheduling for containerized applications. The authors justify their approach, apart from providing better performance, as a valid method to avoid tedious manual work on ad-hoc simple heuristics.

Also, Genetic Algorithms have been employed to help deploy the NFV components on the infrastructure as in the work of Khoshkholghi et al. [32], where a genetic-based algorithm was used to deploy service chains.

Stepping back from the specific NFV case of study, the baseline problem of bin packing was tackled by Asta et al. [10] and they set up an interesting framework based on policy search, where the placement actions were determined by a score matrix obtained by genetic optimization. Their work was limited to just 1D bin packing, but it was a valid proof of concept for further studies.

In this work, we deal with a deterministic problem, and the problem instances lead us to try different approaches that include MILP formulation and GA-enhanced heuristics. Precisely, in one of the used approches, which constitutes the novel algorithm proposed in this paper, the idea described in the paper [10] is extended and adapted for the VNF placement problem.

Our approach is specifically tailored to the needs of the industrial NFV capacity planning problem at Vodafone, like the presence of soft affinity constraints alongside hard anti-affinity ones, and the other requirements detailed in Section 2. There are works that consider the affinity and anti-affinity properties of workload placement for the cloud as in [27]. Still, they do not consider the possibility of making the affinity constraints optional.

We added the possibility to make constraints optional, however trying to satisfy them with a best-effort approach, by modeling the cost as a modular function as was done by Unuvar et.al [33] when trying to split the workload between the public and private cloud.

4. Classical Optimization Approach

The optimal allocation problem is well-known in the industrial landscape, and there are many well-documented solution strategies. One of the most successful techniques consists in casting the placement problem into a Linear Programming instance that can be solved by readily available solvers. In general, the modeling often introduces Integer variables, leading to the Mixed Integer version of Linear Programming, which is computationally expensive to solve. The archetype for this type of problems is the *Bin Packing problem*, where the goal is to minimize the number of bins to allocate a set of objects of different sizes. The *Bin Packing problem* is known to be computationally NP-hard.

The success of such an approach in real applications comes from the ease of mapping the requirements and performance indexes in the MILP format. The deriving mathematical representation makes the problem easy to share, modify, and reformulate. Moreover, the technology behind MILPs is mature and a lot of technicalities and low-level implementation details are handled silently by solvers, leaving the user more time to focus on the high-level goal and relieve him of unrelated maintenance duties. Such solvers give also optimality guarantees when proposing a solution, hinting at how much further extended calculation could be worth.

However, MILPs tend to become computationally prohibitive when the problem grows in size. This comes from the combinatorial nature of the solving algorithm. When exploring different solutions, it could be useful to have a responsive tool that returns an outcome in a reasonable time. Waiting for hours or days could steer users to look for something else. As shown in the previous work, in the pool of problems coming from the Vodafone Infrastructure there were cases where reaching the global optimum could take many hours and that was considered unattainable. Further considerations can be made also thinking about the possible extensions to the current placement problem. Given the requirements set by Vodafone regarding the solving time, albeit possible, it could be hard to add further layers of placement hieararchy. For example, naively handling NUMA affinity constraints would duplicate (or quadruplicate) the number of variables, since the placement action needs to consider multiple NUMA nodes similarly to how hosts are considered.

Still, the properties of the MILP approach make it worth being considered, and there are workarounds to make it more applicable in practice. Just to mention a simple one, that was implemented after the previous study, there is the possibility to combine the MILP guaranteed solving routine with heuristics that are more efficient in finding a feasible solution quickly. Currently, we used the GA-based Heuristic developed in the previous work to "warm start" the MILP solver. In order not to add too much delay, the GA was used with a reduced population. The idea was to return something feasible, hopefully, slightly optimized, however quickly, and then leave the MILP solver to continue optimizing from that. In the experimental evaluation made in this research work, this was the implementation used for the MILP.

4.1. Main Definitions

Referring to Sec. 2, the set of N_v Virtual Network Functions, that are going to be deployed, are represented here by a set

$$V = \{1, 2, \dots N_v\}. \tag{1}$$

For each VNF component $i \in V$, the set of Virtual Machines implementing the VNF-specific functionality is called I_i. The demands of a VNF $i \in V$ in terms CPU, Memory, and Network Bandwidth are expressed here by D_i^{CPU}, D_i^{MEM} and D_i^{NET} respectively. To simplify the notation, we assume, without loss of generality, that Virtual Machines from different Virtual Network Function components have all different indexes: $\forall i_1, i_2 \in V, i_1 \neq i_2 \implies I_{i_1} \cap I_{i_2} = \emptyset$.

In the following, the set of Hosts, on which the VMs are going to be placed, is represented by $\mathcal{H} = \{1, 2, \dots, N_h\}$, where N_h is the max number of Hosts that the infrastructure can contain.

The placement of a single Virtual Machine j on a host h, is modeled with the Boolean variables $\left\{x_{j,h}\right\}$ as

$$x_{j,h} = \begin{cases} 1 & \text{if VM } j \text{ on host } h \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

4.2. Resource Requirements

The placement solution should be feasible, that is, the infrastructure resources should be sufficient to run the workload demanded by the VNFs. The availability of resources on each Host $h \in \mathcal{H}$ is given by L_h^{CPU}, L_h^{MEM} and L_h^{NET}. In this study, we consider the case of homogenous resources among all the Hosts, but it could be possible to extend to heterogenous Hosts at the expense of computational complexity. This way, the selection of the best hardware for the task could be carried out by the solver itself. For example, it could be possible to augment the problem, by considering multiple versions of each Host *h*, having different resource capabilities, and then adding the constraint that only one can be used for the placement. The placement constraints, due to the availability of limited resources on each host, can be described as follows:

- CPU

$$\forall h \in \mathcal{H} : \sum_{i \in V} D_i^{CPU} \left(\sum_{j \in I_i} x_{j,h} \right) \leq L_h^{CPU}; \tag{3}$$

- Memory

$$\forall h \in \mathcal{H} : \sum_{i \in V} D_i^{MEM} \left(\sum_{j \in I_i} x_{j,h} \right) \leq L_h^{MEM}; \tag{4}$$

- Network Bandwidth

$$\forall h \in \mathcal{H} : \sum_{i \in V} D_i^{NET} \left(\sum_{j \in I_i} x_{j,h} \right) \leq L_h^{NET}. \tag{5}$$

4.3. Structural Constraints

Structural constraints control how the VMs should be placed with respect to one another.

A trivial structural constraint is that each VM must be placed on exactly one host. This is justified by the fact that VM cannot be shared between Hosts, that all the VMs should be placed, and that it's not possible to place twice the same VM. This can be expressed as a set of linear equalities:

$$\forall i \in V, \forall j \in I_i, \sum_{h \in \mathcal{H}} x_{j,h} = 1. \tag{6}$$

In general, in case the problem should require a "best effort" placement, where the request consists of placing as much as possible on a limited amount of hardware, not sufficient for a full deployment, this type of constraint could be relaxed. However, this was not the case for the problem solved in this work.

Formally the structural constraints, which could be required to model the specific placement demands, are formalized as

- *Auto Anti-Affinity constraint (AAF)*: The Virtual Machines belonging to VNFs with this type of constraint should be placed on separated hosts. If the set of VNFs that should satisfy this constraint is represented by $V^{AAF} \subset V$, the constraint can be written formally as

$$\forall i \in V^{AAF}, \forall h \in \mathcal{H} : \sum_{j \in I_i} x_{j,h} \leq 1 \tag{7}$$

- *Auto Affinity constraint (AFF)*: Indicating by $V^{AFF} \subset V$ the set of VNFs with the AFF property, the constraint requires each VNF to have all of its VMs placed on the same host:
$$\forall i \in V^{AFF},\ \forall h \in \mathcal{H} : \sum_{j \in I_i} x_{j,h} \in \{0, |I_i|\}, \tag{8}$$
where $|\cdot|$ denotes the set cardinality operation. This constraint can be reformulated in a directly implementable form by forcing all the variables referring to the VMs in set I_i to be equal:
$$\forall i \in V^{AFF},\ \forall h \in \mathcal{H}, j \in I_i : x_{j,h} = x_{j_0,h}, \tag{9}$$
where j_0 is the first index of the set I_i.

- *Cross Anti-Affinity constraint (CAAF)*: Given a set of VNFs, it requires that any VM of one of the VNFs should not be placed on a host occupied by any of the VMs of the other VNFs. Calling by V^{CAAF} the set of (i_α, i_β) couples of VNFs undergoing this constraint, the formalization is given by:
$$x_{j_\alpha,h} + x_{j_\beta,h} \le 1, \tag{10a}$$
$$\forall h \in \mathcal{H}, \tag{10b}$$
$$\forall (i_\alpha, i_\beta) \in V^{CAAF}, \tag{10c}$$
$$\forall j_\alpha \in V_{i_\alpha},\ \forall j_\beta \in V_{i_\beta} \tag{10d}$$

- *Cross Affinity constraint (CAFF)*: Given a set of VNFs, every VM of every VNF in the set needs to be co-located on the same host. Calling by $V^{CAFF} \subset V$ the set of tuples undergoing the constraint, the formalization is given by:
$$\sum_{i \in V^{CAFF}, j \in I_i} x_{j,h} \in \{0, |I_{CAFF}|\}, \tag{11}$$
where $|I_{CAFF}| = \sum_{i \in V^{CAFF}} |I_i|$, $\forall h \in \mathcal{H}$. The same approach used for the implementation of the simpler Auto Affinity in Eq.9 can be used here.

- *Master-Slave constraint (AMS)*: Virtual Machines designated to the VNF with AMS should be subdivided into two equal groups such that no VM from one group is co-located with a VM of the other group. The AMS constraint is reduced to a Cross Anti-Affinity constraint, for which the definition was given in Eq. 10.

- *Cluster constraint*: this optional constraint specifies that the physical infrastructure is logically partitioned in clusters of the same given size, and that, for every VNF, the VM components need to be partitioned across such an infrastructure in a way that ensures that entire VNF components are placed entirely within one of the available clusters, i.e., that the VMs of a VNF component cannot be placed across two or more clusters. The way the constraint has been implemented is by encoding this as a sort of Auto Affinity Constraint.

4.4. Objective

In order to formalize the placement problem within an optimization framework, it is necessary to specify the "objective" function to be optimized.

Precisely, after the discussion from Sec. 2, it is clear that we need a "multi-objective" one. The formal definition includes the following components:

1. Primary Objective
   As far as the main goal is concerned, using a Boolean variable u_h to represent whether a host h has been used in the placement, the cost function is given by the following expression:
   $$J_h = \sum_{h \in \mathcal{H}} u_h. \tag{12}$$
   Specifically, $u_h = \vee_j x_{j,h}$, where $\vee_j$ is the logical *or* operator applied along the index j. The logical *or* operator $\vee_j$ over a set J can be implemented in a linear form by a set of constraints as follows
   $$u_h \geq x_{j,h}, \forall j \in J, \tag{13a}$$
   $$u_h \leq \sum_j x_{j,h}. \tag{13b}$$
   The first inequality guarantees that if one of the $x_{j,h}$ is 1, then the u_h is also 1, whilst the second inequality forces the u_h to be zero when all the $x_{j,h}$ are zero.
2. Secondary Objective
   Modeling the affinity constraints as "soft" can be done by defining some new variables $\xi_{i,h}$ as
   $$\forall i \in V^{AFF},\ h \in \mathcal{H},\ \xi_{i,h} = \vee_{j \in I_i} x_{j,h}. \tag{14}$$
   These variables represent whether there is at least one VM belonging to VNF i with affinity constraints on host h. Using these variables it is possible to define a cost for breaking the affinity constraint, such as an Affinity Penalty, as
   $$J_{aff} = \sum_{i \in V^{AFF}} \left(\sum_{h \in \mathcal{H}} (\xi_{i,h} - 1) \right). \tag{15}$$
   In the case of met affinity, the VMs will be allocated on a single Host, so only one $\xi_{i,h}$ will be 1 and the cost expressed in Eq. 15 will be equal to 0. Otherwise, the Affinity Penalty will be greater than 0 and the optimizer will try to make it as low as possible. In the implementation made up to this point, only the number of Hosts overused is taken into account and not the distribution pattern of the affine VMs on them. For example, in the case of 5 VMs with affinity constraints allocated on 2 Hosts, there is no Penalty difference in allocating 4 of them on the first Host and the remaining VM on the other, or 3 VMs on one Host and 2 VMs on the other. The same approach is valid for the Cross Affinity constraints, where this time the set of Virtual Machines' indexes used to compute the logical OR is composed by joining the Virtual Machines from the set of Virtual Network Functions involved in each constraint.

Finally, the optimization objective for the full problem can be defined as a combination of the previous components:

$$J = K_h J_h + K_{aff} J_{aff}, \tag{16}$$

where K_h, K_{aff} are tuning gains used to make the multiobjective problem separable. Considering the requirements, this means that the max achievable Affinity Penalty $K_{aff} J_{aff}$ should never be more than the increase of the objective determined by adding an extra Host. This choice comes directly from the requirements in the considered industrial scenario, where the reduction of the number of used hosts was given higher priority than meeting the soft affinity constraints, thus avoiding the need for a full Pareto Optimality analysis.

Extending this model to consider more objectives is possible by adding more components to the cost, following the same approach to make them separable and avoid Pareto Optimality issues. Doing so, the range of the optimization objective would increase, making the work of the solver harder.

5. GA-based Heuristic

Another strategy that was explored during the industrial collaboration is the employment of Genetic Algorithms in the context of VM placement. As shown in the previous work, this approach was selected as a trade-off among the simpler Heuristic solvers and the heavy-weight MILP solver. Indeed, they tend to share a simpler modeling phase and fast execution time with the Heuristic solvers, but they are more robust in terms of convergence towards a minimum.

5.1. Implementation Details

The first step in the implementation of a Genetic Algorithm is the selection of the features that encode a solution. Following the modeling of the problem presented in Section 2, it would come naturally to encode the placement of the VMs by a vector where every entry i gives the Host on which the i-th VM has been placed. However, the design choice that was made in the previous work was to indirectly refer to such a placement solution by acting on the order in which the VMs are going to be placed by a First Fit algorithm. First Fit algorithms are very efficient in finding feasible placement solutions, and what matters is the order in which the objects to be placed are drawn. A posteriori, knowing an optimal solution, it's possible to find the placement order such that a First Fit algorithm leads to that. Following this idea, the goal was to have an algorithm that could search for a good placement sequence, adapting to the specific problem instance.

The encoding of a solution, or "chromosome", was a vector of VMs' indexes s representing the placement sequence by a First Fit. Considering a problem with N_{vms} VMs, it is possible to formalize the chromosome s as the permutation of the VMs indexes:

$$s \in \mathbb{N}^{N_{vms}}, \tag{17}$$

$$\text{such that } s_i \neq s_j \ \forall i, j \in [1, N_{vms}], \tag{18}$$

$$i < j \implies VM_{s_i} \text{ placed before } VM_{s_j}. \tag{19}$$

The Genetic Algorithm works by propagating and modifying an initial population of different candidate solutions, favoring the survival of better candidates with respect to a “fit function”. Selecting the “fit function” is the second important step in implementing a GA. In the original problem, solved in the previous work, the fit function consisted of the number of hosts used to allocate all the VMs. The adaptation made in this work extended that to include the affinity, leading to the same expression from Eq.16.

The Genetic Algorithm is implemented as an iterative procedure applied to a population of chromosomes. A schematic representation of the main blocks is reported in Fig. 2 The solving routine starts with the random generation of a population P_0 of N_s candidate solutions. In our specific case, the P_0 is obtained by performing a First Fit allocation randomly scrambling the order in which the VMs are placed N_s times. Once P_0 is generated, the algorithm starts the iterative phase. At every iteration k, the current population P_k is modified to explore better candidates as follows:

1. Obtain an extended population P^-_{k+1} by adding new candidate solutions to P_k;
2. Compute the fit value for each candidate of P^-_{k+1};
3. Generate a P_{k+1} population, sampling the N_s best elements in terms of their fit value from the P^-_{k+1}.

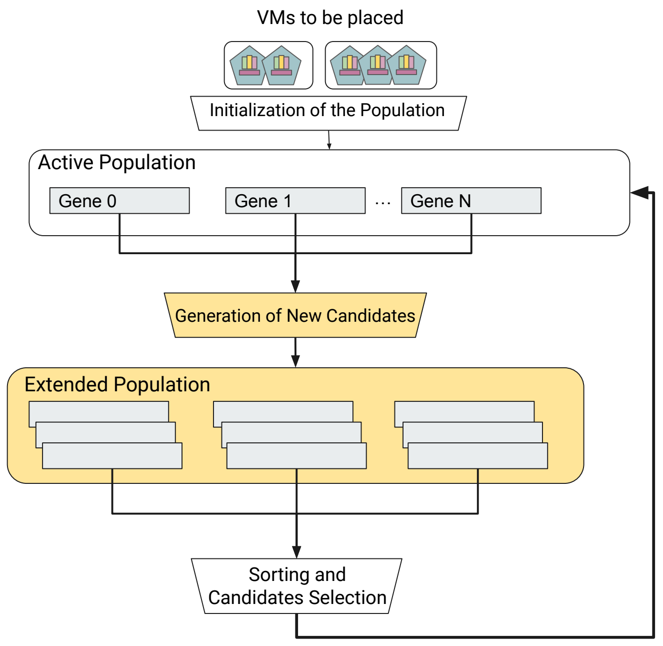
VMs to be placed
Initialization of the Population
Active Population
Gene 0
Gene 1
...
Gene N
Generation of New Candidates
Extended Population
Sorting and Candidates Selection

Figure 2: Schematic representation of a Genetic Algorithm with the main phases.

The generation of new candidates to extend the current population is made by using chromosomes from the original population. Usually, the generation of new candidates is carried out using a cross-over operation, which mixes the genes among two or more parent candidates. However, in this case, the implementation of this step was different, since the encoding approach used in this GA was not optimal for the classical cross-over operation. The way it was implemented was more similar to a mutation-only GA, where starting from a candidate solution, the algorithm searches for a pair of VMs to swap in terms of the order by which they are placed.

At first, after having selected the two associated indexes in the permutation vector, the two VMs are deallocated, together with all the VMs at indexes greater than the first one. These deallocated VMs are later reallocated following the new order generated by swapping the 2 VMs. The partial deallocation and replacement are necessary because the elements of the population should be maintained as valid First Fit placements. Indeed, considering a swap that involves the VMs pair (V_{s_i}, V_{s_j}) with $i < j$, any VM placed after V_{s_i} had the location selected by a First Fit algorithm that took the presence of V_{s_i} into account. After the swap, at step i the V_{s_j} is going to be placed, so the First Fit allocation should be re-evaluated for all the remaining VMs. One example of the swapping mechanism for new chromosome generation is shown in Fig. 3. In the Fig-

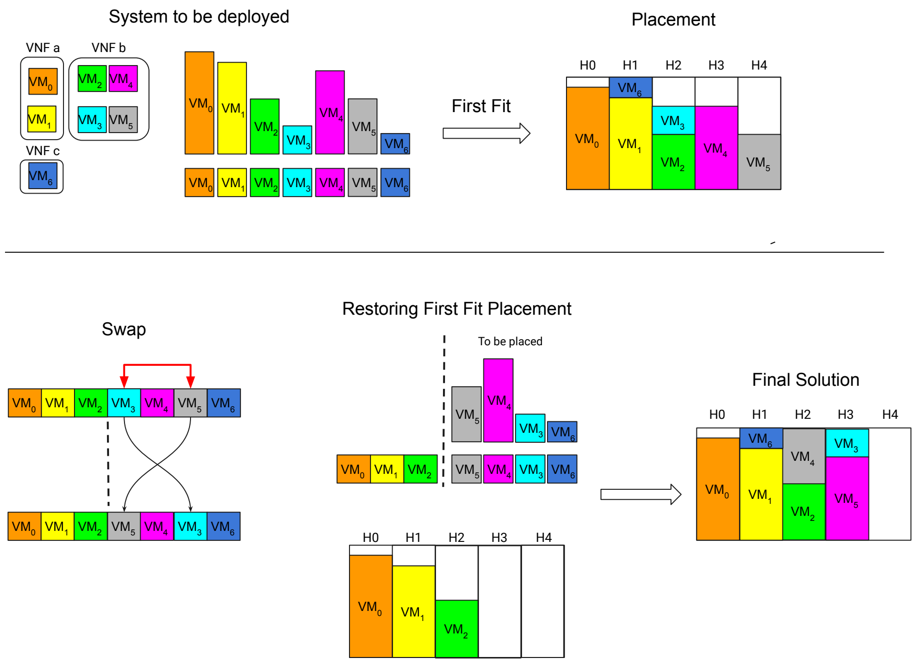

Figure 3: Details on the generation of new candidates using the swap method.

ure there are 7 VMs, belonging to 3 different VNFs that need to be placed. Applying a First-Fit placement to the original sequence it will use 5 Hosts. VM_3 and VM_5 are selected to be swapped and this triggers the re-placement of all the VMs placed after VM_3 in the original sequence. Notice that the part of the chromosome that occurred before VM_3 is not changed. After re-placing the removed VMs, the solution takes 4

Hosts.

The selection of the VMs pair can be made by looking at some specific properties, such as selecting from Hosts that are not completely full, but in the original implementation, the search for a pair was purely random.

The focus of the previous work was to find a trade-off between solution optimality and computation time. On one side we had the MILP approach that provided optimality guarantees but was slow to return a solution, on the other side there was a simple Heuristic, which could provide a feasible solution very quickly, but it was often sub-optimal. Wanting to propose a GA-based Heuristic solver as a balanced solution some expedients were necessary. Indeed, GAs can become computationally heavy due to the high number of attempts they need to make when searching for the best candidates. For this reason, an early termination check was introduced to speed up returning solutions. The implementation of this early stop procedure was based on a value representing the confidence of the algorithm to be near an optimal solution. The idea came from the MILP concept of the optimality gap, which quantifies the distance between the current solution and the best one obtained via relaxation. In the case of Heuristic solvers, it is not trivial to assess how far the current solution is from the optimum, and some estimation is necessary. In our implementation of the GA, the idea was that if the population becomes too uniform in terms of fit value it could be a sign that the solver found a good solution. In practice, the algorithm monitors the maximum spread of the fit function among all the population and stops the iterations when the spread is under a given configurable threshold.

The algorithm has some hyper-parameters that can be tuned, such as the size of the population N_s, the number of new candidates generated from each candidate of the old population, the number of iterations to be applied to the population, and the threshold for the early stopping feature.

6. Policy-based Heuristic

This section covers a different approach: the “Policy-based Heuristic” developed for the problem proposed by Vodafone. The motivation fueling this step comes from the limitations observed in the previous study. The details on the comparisons and comments on the outcomes will be the objective of the next section, whereas in this one the focus is on the description of the approach.

Solving problems relying on heuristics is a very common approach since it can quickly provide acceptable solutions. This is true, provided that they are designed by tailoring them to the specific problem. Manually finding the heuristic that performs better in a given case study requires experience and can be daunting. Often, it’s a matter of finding which aspect is more relevant to produce an optimal solution. From another point of view, a heuristic translates into assigning priorities to the possible actions and then selecting the one that comes up first. In the context of this work, an action is the placement of a VM on a specific host. For example, in the heuristic approach described in the previous work, constituting the baseline for the GA implemented there, the priority was encoded as the Host ID, and the actions placing on hosts with lower IDs had higher priority. Another possible option could be ranking the hosts by free available resources, that is, the feasible host with the least amount of available resources is selected first or vice versa. Different priority rules, looking at different system properties, translate in different algorithms, such as “first fit”, “best fit”, “worst fit”, etc..

The idea behind the proposal of the Policy-based Heuristic is to develop a routine that automatically adapts to the problem by finding a heuristic that, using information about the current status of the system, leads to a good placement. The adaptation is achieved by leveraging Genetic Optimization.

6.1. Modeling the Heuristics

As it has been done for the other approaches, it is necessary to define a model for the system and the information that can be handled by the algorithm. The system configuration, that is, the current occupancy of the hosts, provides all the desired information to make decisions. This is going to be represented as a set of placement actions A, that is, a set of tuples (v, h), where v is a VM from the set $\mathcal{V}$ and h is the hosting Host. For example, using as a reference the final placement shown in Fig. 3, the final state would be

$$\begin{aligned} A = [& \\ & (v_0, h_0), \\ & (v_1, h_1), (v_6, h_1), \\ & (v_2, h_2), (v_4, h_2), \\ & (v_3, h_3), (v_5, h_3) \\ ]. & \end{aligned}$$

Considering the specific problem this work deals with, useful information, obtainable from the current placement configuration, which can hint at a good move, can be in the form of free resources on the hosts, requirements of the new coming Virtual Machine, or affinity scores quantifying the distribution of the components. The algorithm is going to work iteratively, placing one Virtual Machine at a time using that information. This means that, at the arrival of a new Virtual Machine v, for every feasible placement action that consists of selecting a feasible hosting host h, there will be a priority score $\sigma(v, h, \mathcal{V}, A)$, dependent on the current state of the system, quantifying how good that action would be.

A natural implementation of this priority score assignment is a multi-dimensional lookup-table. Precisely, in this work, the heuristic has been encoded with a multi-dimensional matrix of positive integers priority values $\mathcal{P}$, where the coordinates are quantities related to the information extracted from the system configuration and the $(vm, host)$ action pair. The quantities selected as coordinates of the policy matrix have been discretized in a number of finite intervals. The number of intervals, that determine the size of the search grid, is a hyperparameter of the model and can be selected by the user. That is, calling N_{u_i} the number of intervals for the input coordinate i, the matrix will lay in $\mathbb{N}^{N_{u_0} \times N_{u_1} \times \ldots N_{u_k}}$, where k is the number of quantities selected to be the input for the policy. This multi-dimensional matrix has been called the Policy Matrix. A represenatation of a 2-dimensions Policy Matrix and coordinate computation example is shown in figure 4, where 2 actions (v_0, h_0) and (v_0, h_1) are characterized by the VM cpu requirement and the free CPU available on the hosts. The discretization of the coordinates leads to the selection of 2 different tiles in the matrix associated to two different priority scores.

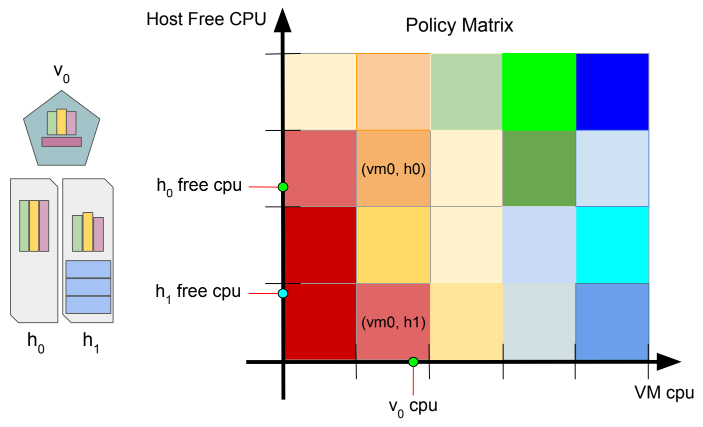

Figure 4: Example of 2D Policy Matrix and score evaluation with 2 actions.

The process of action selection is exemplified in Fig. 5, where a VM *vm*0 can be placed on three hosts. In the Figure, the Policy Matrix is only 2D for ease of representation, and the different colors represent different values for every cell, similar to a heatmap. As explained, the decision process is carried out by considering the configuration of the system for every possible action, finding the associated score by looking at the Policy Matrix and selecting the one that ranks better (lower score value).

As far as our case-study is concerned, the quantities selected to be used as inputs to the algorithm are the following:

- Host's Free CPU;
- VM's requested CPU;
- Affinity Score;
- Anti-Affinity Score.
- Standard Deviation of the VM sizes (CPUs) on the Host.

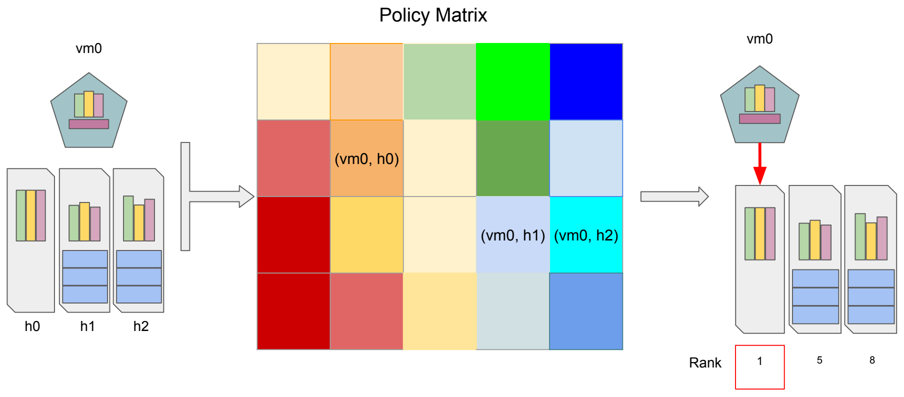

Figure 5: Iterative placement of VMs on Host using a policy matrix that can rank feasible actions to perform selection: the VM $vm0$ can be placed on three hosts, and after evaluating all three possible actions $(vm0, h_i)$ the best one is selected.

The selection of the previous quantities has been driven by practical experience in the field, by looking at what constitutes a more limiting constraint for the placement problem and what is more related to the properties that someone seeks to optimize. In principle, it is possible to add any number of other variables, but this would make the computation more intense, without bringing a tangible improvement to the final result. In those cases where it is needed to add more dimensions to the Policy Matrix, it is possible to counter-balance the size increase, by making the quantization more coarse on some of them. Considering the real placement problem provided by Vodafone, it turned out that other resources, such as memory and network bandwidth, resulted less relevant in shaping the placement solution. It is anyway important to point out that, even if some quantities are neglected during the score computation, the algorithm will always take into account all the constraints, and only feasible allocations are considered.

Apart from the quantities that involve the CPU, which are quite simple to interpret, the ones related to affinity and anti-affinity deserve more explanation. When there are affinity demands, it is expected that a good placement algorithm will place affine VMs together as much as possible. In order to make the heuristic sensitive to this request, an affinity score has been defined and included among the input variables. For every action pair, the score is computed by counting how many VMs, affine to the one that is going to be placed, are on the target host. It is worth noticing that the computation of this score is based only on the current configuration of the placement, that is, the part of the system that has been already placed, and the single action pair. With this information available, an algorithm has the ability to give higher priority to hosts that have a higher score, driving towards a better affinity.

The score related to the anti-affinity is slightly different. Whilst the previous score can be computed based on the current, partial, placement, without considering what needs to be placed next, for the anti-affinity case, the computation requires considering the property of all the VMs that are going to be placed. Precisely, when computing the score for an action that involves a VM with anti-affinity, the score is evaluated by counting the number of VMs anti-affine with it that have already been placed in the system and comparing that number to the total. This score was thought to be informative for a placement logic, since the impact of placing a VM with many anti-affine VMs depends on how many other anti-affine VMs are still to be placed. For example, in case the set of still-to-be-placed anti-affine VMs is empty, then no impact should be expected on the remaining VMs. On the contrary, placing a VM that has a lot of anty-affine pairs waiting to be placed could require some considerations, since the selected Host will become forbidden to them.

The information regarding the Standard Deviation of the VM allocated on a Host is useful for judging the heterogeneity of the placement. It could be sensible to think that having available a set of different sizes to choose from could help pack more items inside a container. On the contrary, placing all the items of a given size in a single container could make them scarce in the future when they could be needed to fill a specific gap. Following this idea, it has been decided to add that quantity such that the Genetic Algorithm could have the choice to integrate that into the policy.

As far as the optimization task is concerned, the Fitness Function used to evaluate the Population of the GA is selected to reflect the goal of the placement problem in Sec. 2.3 and correspond to the same function used for the other two methods reported in 16, apart from some scaling factor.

6.2. Algorithm

Wanting to formalize the routine, it is possible to consider the placement made with the Policy-based Heuristic as a two steps process:

1. a *Forward Pass* (Alg. 1), where the learned policies are applied to allocate the VNFs workload;
2. a *Search Phase* (Alg. 2), where the policies are improved using Genetic Algorithm techniques.

As previously stated, the goal of the algorithm is to find a good policy $\mathcal{P}^*$, that is, a policy that determines an allocation satisfying the desired objectives described in Sec. 2.3. Clearly, the search for a good policy will also lead to a placement solution for the proposed problem. Casting this into Genetic Algorithm vocabulary, the policy matrices $\mathcal{P}$s constitute the chromosomes, whilst the solution cost will be the Fitness value.

Forward Pass. The *Forward Pass* is described in Alg. 1. The algorithm starts with the set of VMs that need to be placed and the current population of solutions, that is the set of current Policies. For every policy, the algorithm is going to accomplish the allocation of all the VMs in $\mathcal{V}$.

The order with which the VMs should be placed can be left unaltered, or it can be shuffled. We decided for the latter to find policies that are robust to the placement order and avoid adapting to a specific sequence. If a specific policy survives along the epochs, this means that it is able to perform well while placing the VMs in different order of arrival.

For every VM, the algorithm looks at the current configuration of the system A_p and computes the set of feasible Hosts H_v. In case all the hosts in the set A_p are not compatible with the VM, a new empty host is added and then populated with the VM. For each feasible host, the priority score is computed and in the end the best placement action is selected and added to A_p. At the end of the *Forward Pass* the current population is paired with full solutions of the placement problem and is ready to be ranked and selected.

Search Phase. During the *Search Phase*, described in Alg. 2, the current population of policies, which produced the Map of System Configurations $\mathcal{A}$ during the *Forward Pass*, is evaluated in terms of the Fitness Function. In this regard, it is worth explaining the strategy used to "judge" the solutions in the selection. At the end of the *Forward Pass*, every element of the population, that is, the placement policy, is characterized by how well the VMs have been allocated using it. The evaluation metric reflects the number of Hosts used, and the amount of affinity broken after the placement. As explained in the previous sections for the other algorithms, reducing the number of Hosts has a higher priority and there is the risk that the affinity could be overlooked in the evolutionary process. For example, after a few rounds of selection, the surviving population could be composed by elements that perform well in terms of number of Hosts but the affinity is poorly managed. For this reason, to make sure that the population contains policies that are able to place VMs maintaining a good affinity, the selection of the surviving population is accomplished as follows:

- Sort the population by the main Fitness Function, which comes directly from the demands and weights more the number of hosts used, taking the best individuals: this population will be the "*Alpha*" population;

- Sort the remaining individuals by a secondary Fitness function, which weights only the affinity, and take a subset of the bests: this will be the "*Beta*" population.

The sizes of the *Alpha* and *Beta* populations are tunables of the algorithm and they depend on different factors, such as problem size and number of affinity constraints.

In this way, when performing the cross-over operation to generate the new population from the *Alphas*, *Betas* survivors, the information contained in policies leading to good affinity can be mixed with the ones leading to more packed placements, hopefully discovering good trade-off. The current algorihtm implementation makes use of the uniform cross-over.

Algorithm 1: Policy-based Heuristic: *Forward Pass*

```
Data: Set $\mathcal{V}$ representing the VM to be placed.
Data: Set of Policy Matrices $\mathcal{P}$
Result: Map of System Configurations $\mathcal{A}$, one for each policy $p \in \mathcal{P}$, in the
        form of tuples $\{p, [(v, h_v)]\}$

1  $\mathcal{A} \leftarrow \emptyset$;
2  for each Policy $p \in \mathcal{P}$ do
       // Randomize the order of VMs to be placed
3      $\mathcal{V}' \leftarrow randomize(\mathcal{V})$;
4      $A_p \leftarrow \emptyset$;
5      for each VM $v \in \mathcal{V}'$ do
           /* Compute the set of feasible Hosts in the current $A_p$
              for $v$, or select a new empty Host                  */
6          $H_v \leftarrow computeFeasibleHosts(v, A_p)$;
           /* Compute the score on $H_v$                           */
7          $s \leftarrow [\infty, \infty, ..., \infty]$;
8          for each $h \in H_v$ do
9              $s_h \leftarrow \sigma(v, h, \mathcal{V}', A_p, p)$;
           /* Select the best action and update $A_p$              */
10         $h_v \leftarrow argmin_h(s_h)$;
11         $A_p \leftarrow A_p \cup (v, h_v)$;
       // Add the configuration obtained applying $p$ to the
          output set
12     $\mathcal{A} \leftarrow \mathcal{A} \cup \{p, A_p\}$;
   // Final result
13 Output: $\mathcal{A}$;
```

After each round of selections performed in the Policy *Search Phase* the algorithm has a set of promising policies and the respective placement solutions. In practice, instead of a single policy, the algorithm could return a group of policies. Precisely, in the final implementation, the algorithm returns the policies of the last iteration surviving set.

Algorithm 2: Policy-based Heuristic: Policy *Search Phase*

```
   Data: Map of System Configurations $\mathcal{A}$
   Result: New Population $\mathcal{P}$
1  $\mathcal{P} \leftarrow \emptyset$;
   // Compute the Fitness of the current population $\mathcal{A}$
2  $s \leftarrow [\infty, \infty, ..., \infty]$;
3  for each (Policy, Configuration) $\{p, A_p\} \in \mathcal{A}$ do
4      $s_p \leftarrow computeFitness(A_p)$;
   // Extract the Alphas and Betas from the current population
5  $(\alpha,\beta) \leftarrow rankPopulation(s, \mathcal{A})$;
   // Generate the new population
6  $\mathcal{P} \leftarrow generateNewPopulation(\alpha,\beta)$;
   // Final result
7  Output: $\mathcal{P}$;
```

A schematic representation of the full process is reported in Fig. 6.

Fig. 7 shows an example of some statistics of the Fitness Value as a function of the GA epochs obtained while solving a problem in the generated dataset. The presence of a decreasing pattern means that, on average, the genetic material surviving is able to keep placing the set of VMs independently from the order. In the specific case represented in the Figure, the population's best solution was reporting a number of hosts equal to 48, the Avg. stabilized around 49 and the worst cases could give 51-52 hosts.

7. Experimental Evaluation

This section provides results obtained from the experimental evaluation of the approaches introduced above.

7.1. Dataset

The algorithms proposed in this work have been evaluated both on real and artificial problems. Indeed, initially, the goal was to solve problems coming directly from the Vodafone Infrastructure, and this led to the first comparative assessment of performance in our previous work [11]. The data was provided in the form of planning files for the networking services, called *virtual bill of material vBOM*s containing the requirements in terms of VNFs components and their constraints. The content of these files was used to frame the optimization problem described in Section 2. After

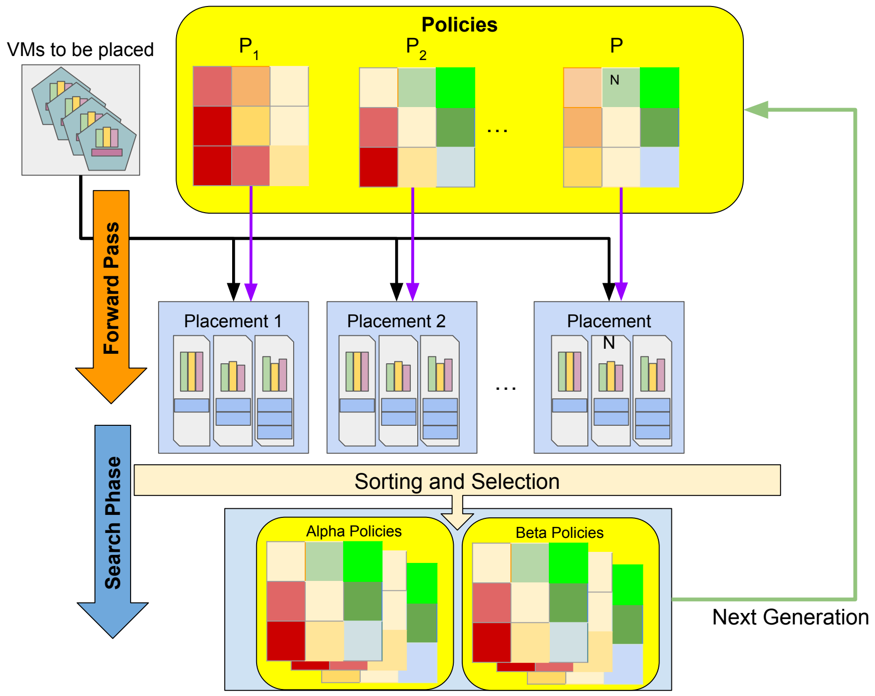

Figure 6: Schematic representation of the Policy-based Heuristic: Given a list of VMs to be placed a population of Policy Matrices is used to perform placements; The obtained solutions are then ranked and a subset of the well-behaving ones is computed; The genetic material in this subset is then used to generate the population for the next iteration.

anonymization and relevant information extraction, the dataset of these original problems was made publicly available[2].

However, despite the fact that this gave us the opportunity to work with real data, it had some limitations. First of all, the number of problems we had access to was limited. Also, the variety of problem features was limited or at least not very well distributed. For example, there were far more small-size problems than large ones. Even if sometimes it is possible, and recommended, to use small problems to assess the properties of algorithms, it is true that bigger problems are more relevant in practice. Among those available problems, only a limited number had affinity constraints, making it difficult to assess how that type of request is dealt with by the proposed algorithms. To confirm our claims, Figs. 9 and 8 show the distribution of the number of Virtual Machines and required resources in the set of problems coming from the Vodafone dataset.

In order to have a better insight into the performance of the proposed approaches, it has been decided to generate further artificial problems to increase the size and coverage of the datasets. Indeed, one of the points of interest stemmed from the previous

[2]`http://retis.sssup.it/~tommaso/papers/ic2e22.php`

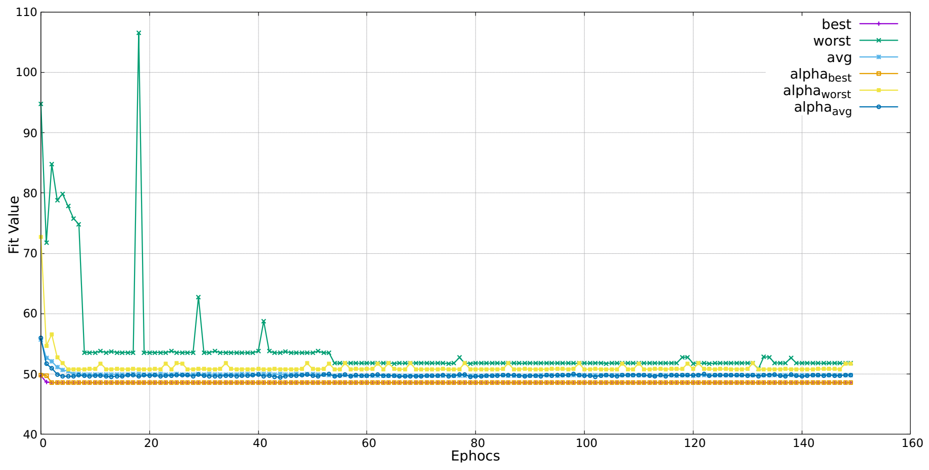

Figure 7: Average, Worst, and Best values of the Fitness Function among the full population and the Alpha population during training.

work was to analyze the relationship between some characteristics of the problems and the performance of the algorithms. In order to be able to spot such relationships, if any, it was necessary to have a way to generate problems of different sizes and possibly with a more uniform distribution of the problem properties.

The problem generation is controlled by a set of chosen parameters, given by the following list:

- number of VNFs;
- max number of VMs that each VNF can have;
- minimum and maximum amount of CPU that can be required by VMs;
- minimum and maximum amount of Memory that can be required by VMs;
- minimum and maximum amount of Network Bandwith that can be required by VMs;
- $p_{auto_{aff}}$ VM's probability of having Intra-Affinity Affinity constraint;
- $p_{auto_{aaf}}$ VM's probability of having Intra-Affinity Anti-Affinity constraint;
- $p_{cross_{aff}}$ VM's probability of having Inter-Affinity Affinity constraints;
- $p_{cross_{aaf}}$ VM's probability of having Inter-Affinity Anti-Affinity constraints;
- Max number of components that can be Inter-Affine with each VNF.

For the sake of simplicity, the random operations used during the generation are based on Uniform Probabilities.

In Figs. 11 and 10 the distribution of the resources and number of Virtual Machines is reported for an artificially generated dataset.

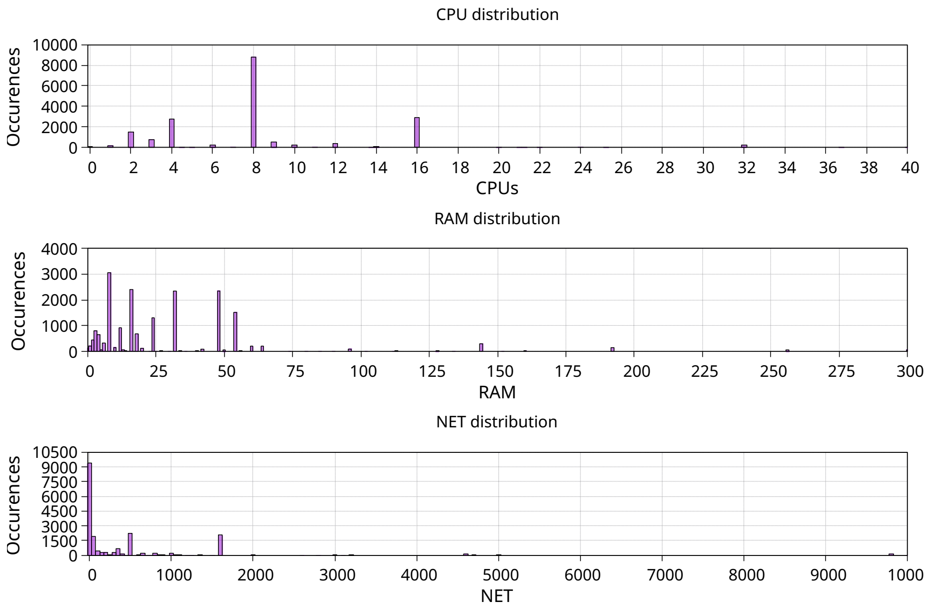

Figure 8: Distribution of the Resources in the Vodafone original dataset.

7.2. Experiments

In the following, the MILP and the GA-based Heuristic were run using the same tuned parameters used in the previous study. Precisely, the GA was run with a population of 50, performing 5 swaps for every element of the population and running for 25 epochs.

A difference with respect to the previous work is that the MILP was run providing the solver with a warmstart solution found with a short run of the GA-based Heuristic. The first purpose of the warmstart solution was to obtain a reasonably tight upper bound on the number of Hosts necessary to produce feasible solutions. This value is needed to define the size of the $\mathcal{H}$ set in the optimization problem and, thus should be known before running the solver. Without having a warmstart solution, the problem could have been defined using an estimated worst-case value. Although this option is viable, a too-high worst-case value would make the problem unnecessarily harder to solve, giving the MILP approach some unfair disadvantages. The second purpose of the warmstart solution is to help the solver get an initial feasible solution that can be used to speedup the searches on the solution tree. Given the fact that this is just an initial solution, the interest was not focused on getting the best solution, but just obtaining it quickly. For this reason, the GA-based Heuristic was run with a smaller population, performing only 2 swaps, and with a 2 epochs limit. The milp solver was also configured with an optimality gap of $1e-6$ which was required due to the presence of the soft affinity cost, which, depending on the size of the problem, could be quite small. This fixed value is clearly not optimal, and in the future we are planning to implement some automatic hyperparameters tuning, but, currently, we opted for this "one size fits all" for the

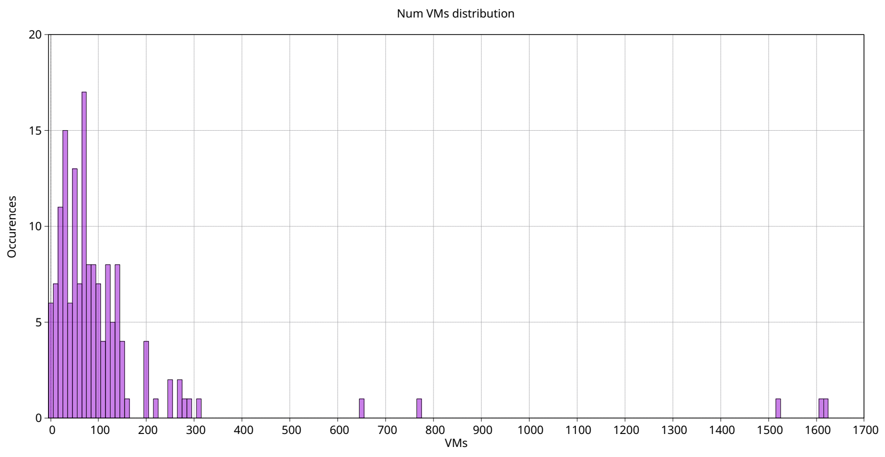

Figure 9: Distribution of the number of Virtual Machines in the Vodafone original dataset.

Vodafone problems.

The Policy-based Heuristic Genetic Optimization was run for 150 epochs, with a population size of 1500, selecting 50 and 10 elements of the population as Alphas and Betas, respectively. The Policy Matrix was choosen to be in $\mathbb{N}^{25\times25\times25\times10\times10}$.

The comparison with respect to the affinity is made using an Affinity Penalty index $\mathcal{P}_{AFF}$ which is computed as follows:

$$\mathcal{P}_{AFF} = \sum_{i \in V^{AFF}} \left(H_i^{AFF} - 1 \right), \tag{20}$$

where V^{AFF} is the set of VNF having Affinity relationships and H_i^{AFF} is the number of Hosts used to allocate the VMs involved in the Affinity relationships of VNF i. For example, if VNF_A with VMs $[vm_{a_0}, vm_{a_1}]$ is affine with VNF_B with VMs $[vm_{b_0}, vm_{b_1}]$, the contribute of VNF_A to the Affinity Penalty is given by the number of Hosts used to allocate $[vm_{a_0}, vm_{a_1}, vm_{b_0}, vm_{b_1}]$ minus 1.

The experiments were run on a dedicated machine having a *Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz* GPU and 64 GB of RAM @2666 MT/s.

7.2.1. Original Dataset

The first phase of the experimental evaluation makes use of the original dataset provided by Vodafone. The goal was to investigate the capability of the previously proposed solution from a different perspective that was neglected in the initial work. Indeed, in the previous work evaluation, the focus was put only on the number of Hosts used to perform the placement, without considering the performance in terms of the soft affinity requirements. However, as explained in Sec. 2, soft affinity, even if not directly affecting the cost of the infrastructure, could be worth considering in cases where network bandwidth and computational throughput of VNFs are taken into account. For this reason, in this work, the original dataset was used to extract also some information regarding the algorithms' outcome in terms of affinity. Running

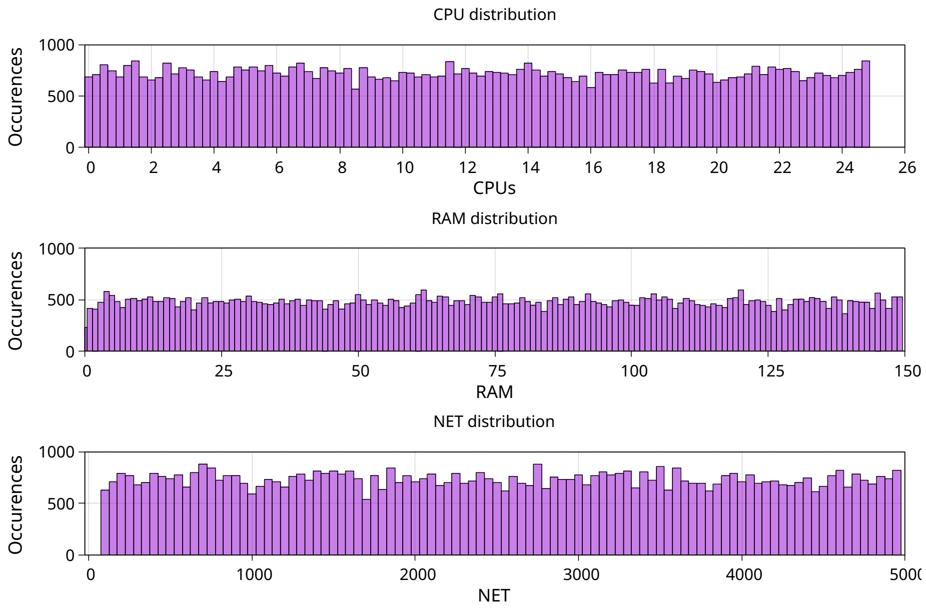

Figure 10: Distribution of the Resources in the artificially generated dataset.

again the experiments and looking at the value of the affinity penalty, shown in Fig. 12, it's possible to see that the previously proposed GA approach tends to be far from optimal. Considering the way the previous GA algorithm was developed, it could have been expected, since the probability that swapping two random VMs, could lead to a new solution with a better affinity score (lower penalty) is likely low. For this reason, even if the previously proposed approach was able to find a good solution in terms of used hosts, comparable with the one returned by the MILP approach, as can be seen in Fig. 12, the inefficiency in accommodating soft affinity requirements led to the development of different approaches. Another observation regarding the original dataset is the fact that only a small subset of problems had the soft affinity constraint involved. This is the reason why in Fig. 12 many experiments are reporting no penalty.

7.2.2. Artificial Dataset

The two algorithms from the previous work and the new one have been evaluated on artificially generated problems having many soft-affinity constraints to verify the capability to handle such requests, together with the baseline goal of minimizing the number of used hosts.

Initially, a set of 40 small-size problems have been generated with the following parameters:

- Number of VNFs = 10;
- Max Number of VMs per VNF = 7;

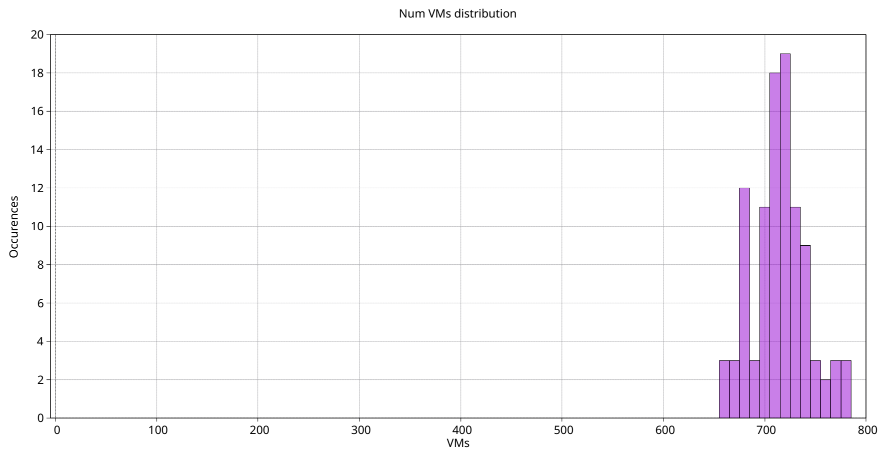

Figure 11: Distribution of the number of Virtual Machines in the artificially generated dataset for large-size problems.

- $CPU \in [0.1, 15]$;
- $RAM \in [0.5, 6]$;
- $Net \in [100, 1000]$;
- $p_{auto_{aff}} = 0.4$;
- $p_{auto_{aaf}} = 0.5$;
- $p_{cross_{aff}} = 0.12$;
- $p_{cross_{aaf}} = 0.15$;
- Max number of VNF in Cross-Affinity = 2.

This led to a dataset where among the 10 VNFs, roughly 40% of them had Auto-Affinity and 15% had Cross-Affinity requirements.

From the results plotted in Fig. 13, we can observe that the number of Hosts is approximately the same and matches the general trend shown on the dataset of the previous work. As far as the deployment cost for the Network Operator is concerned, all three algorithms are proposing nearly equivalent solutions. Also in terms of Affinity Penalty, the artificial dataset confirms what was observed in the real dataset: the GA-based Heuristic is performing worse than the MILP approach. Wanting to compare with the new Policy-based Heuristic, this latter was able to find, apart from a few cases, a better solution with respect to the Genetic Algorithm proposed in the previous work. There are cases where the new Policy-based algorithm seems to perform worse, but at a close inspection that was due to the fact that the placement was made on a smaller number of Hosts. For this reason, to ease the interpretation of the results, Fig. 14 shows the relative improvement of the Affinity Penalty between the old GA

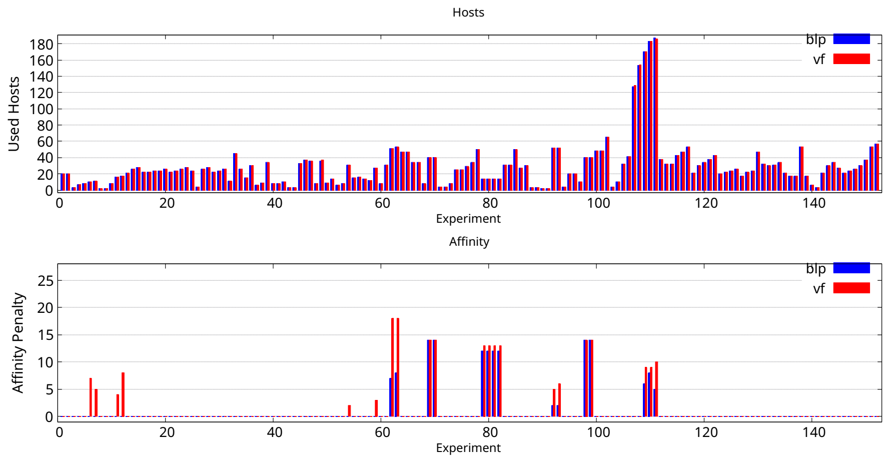

Figure 12: Number of hosts necessary to perform the placement and Affinity Penalty as computed by the placement algorithms on the Vodafone dataset.

and the Policy-based one, along the problems' dataset, and also a scatter plot, where the relative improvement is shown together with the improvement in the number of Hosts used for the placement. As expected, the MILP approach is the one returning the better solutions, except that the computation time is higher. In the case of the small-size dataset, the MILP was able to reach the end of the optimization routine, without triggering the timeout, thus providing a solution that was guaranteed to be optimal. Unfortunately, as we will see in the next examples, this is not always the case.

To evaluate the performance on more realistic sizes, a set of 40 mid-size problems has been generated increasing the number of VNF to 110, and leaving the other generation parameters as in the small-size dataset.

The results in terms of needed hosts to implement the solutions and affinity penalty are shown in Fig. 15.

It is possible to observe that for all the problems the number of hosts used is the same and that the differences are only on the Affinity side. In this respect, the Affinity is generally handled better by the MILP and Policy-based Heuristic. The details on the Affinity Penalty shown in Fig. 16 are in line with what was discovered in the smaller cases. As can be seen there, the new Policy-based Heuristic, except in a few cases, is usually able to find a solution with better affinity. The improvement gap is also noticeable, reaching up to about 50%.

One point that requires explanation is when the MILP approach does not perform better than the other two approaches. Considering the properties of a MILP solver, this should not be the case, provided that the solver is allowed to run up to complete exploration of the solution tree. Indeed, in our case, a time limit was set up at 3500 seconds and for such problems, the solver terminated always hitting the time limit. Wanting to consider the solving time of the proposed approaches on these mid-size problems, the results are shown in Fig. 17. As expected, the MILP approach is the one with the longest solving time, and in all cases, the end is given by timeout. In practice,

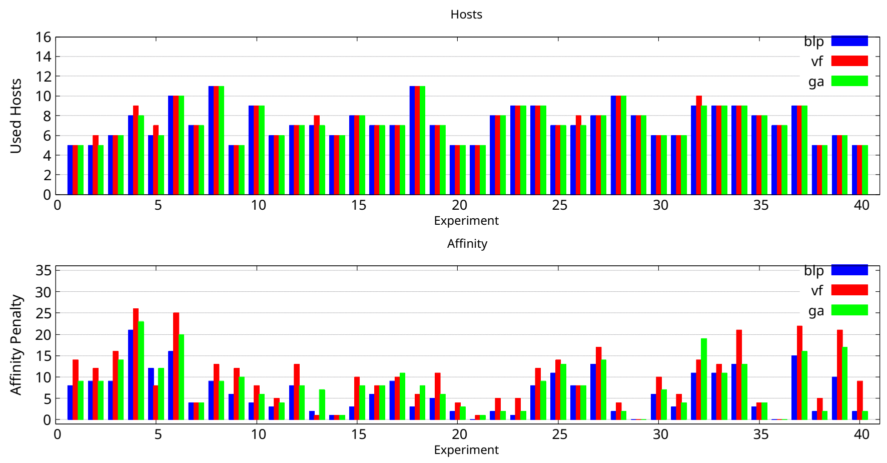

Figure 13: Number of Hosts necessary to perform the placement and Affinity Penalty computed by the placement algorithms on a small-size dataset.

this issue could limit the applicability of the MILP approach to large problems. As a reference, the mid-size MILP problems had between $30,000$ and $50,000$ columns and between $50,000$ and $100,000$ rows. Sometimes it is possible and it is worth waiting hours, or days, for a good solution, sometimes this is not viable. For example, in our industrial experience, the long solving time needed by the MILP was an issue and was the driving force behind the development of the other two approaches. For this reason, it does not come as a surprise if the simple GA-based Heuristic, launched with the same hyperparameters used in the previous work, is the one reporting the lower solving times. The new Policy-based Heuristic is in between the other two, due to the Genetic Optimization routine used to select the Policy.

On average, the mid-size problems had about 430 VMs to place. Considering that Vodafone problems had a few cases with more than 1000 VMs, another set of problems was generated, focusing on a larger problem size. Precisely, a set of 100 large-size problems has been generated with the following parameters:

- Number of VNFs = 180;
- Max Number of VMs per VNF = 7;
- $CPU \in [0.1, 25]$;
- $RAM \in [0.5, 15]$;
- $Net \in [100, 5000]$;
- $p_{auto_{aff}} = 0.4$;
- $p_{auto_{aaf}} = 0.3$;
- $p_{cross_{aff}} = 0.12$;

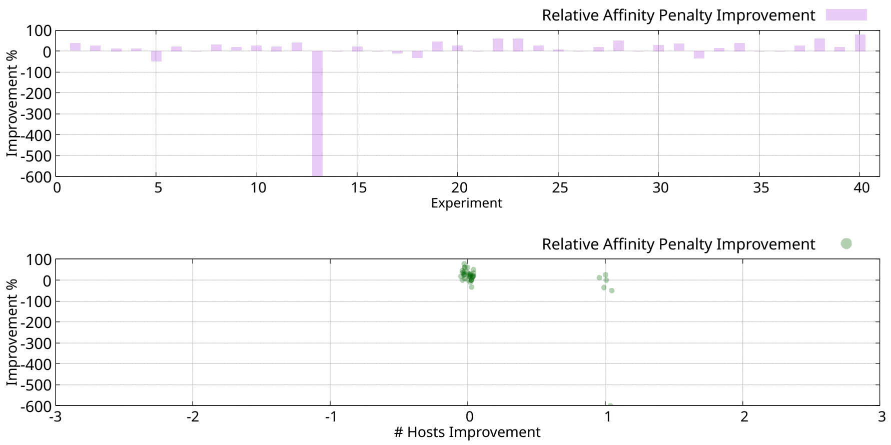

Figure 14: Affinity Penalty Improvement with respect to the GA-based Heuristic for the small-size problem dataset. On the x-axis, the values are integer, and the visible perturbations have been introduced artificially to ease the visualization of the distribution.

- $p_{cross_{aaf}} = 0.15$;
- Max number of VNF in Cross-Affinity = 2.

The 180 VNFs can have up to 7 VMs each and the span of CPU, RAM, and Network requirements have been expanded up to 25 CPUs, 16 GB, and 5000 Mb/sec respectively. The large-size MILP problems had between $140,000$ and $230,000$ columns, and between $250,000$ and $420,000$ rows.

The results on this set of problems show the same trend spotted in the mid-size problems. From Fig. 18, the number of Hosts necessary to implement the solutions is approximately the same, whilst the Affinity Penalty shows a little advantage of the newest Policy-based Heuristic. Clearly, this is just due to the timeout issue of the MILP. As easily spotted from Fig. 19, the MILP was failing to return in the 3500 timeout window. In those cases, it is not rare to find out that the returned solution is the one provided as a warmstart. Without constraining the computation time, the MILP should return the best solution, except for issues coming from approximation and tolerances.

In terms of Affinity Penalty improvement it is possible to notice that, even if the Policy-based Heuristic performs better in most of the cases, the improvement margin is not as big as it was for smaller problems. However, it's also necessary to point out that it was able to perform better in cases where the old GA-based Heuristic was using more Hosts. Indeed, it is usually common to see solutions with a higher number of Hosts reporting better affinity. This is expected since there is more freedom to move the VMs around. This is shown in Fig.20.

8. Conclusions and Future Work

In this paper, we considered the problem of designing data-center optimization strategies in the context of Softwarized Network Services for NFV, as arising from an

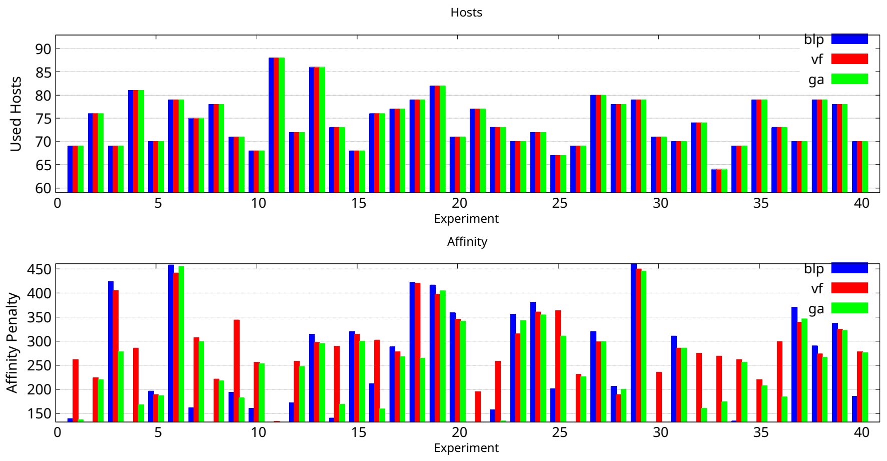

Figure 15: Number of Hosts necessary to perform the placement and Affinity Penalty computed by the placement algorithms on mid-size dataset.

industrial capacity planning use-case in the context of the Vodafone network operator. We investigated three different approaches to tackle the problem: a MILP-based formulation that allows for using a standard MILP solver; a heuristic solver based on a traditional use of genetic optimization; a policy-based heuristic solver that is optimized using a genetic algorithm. We described the 3 techniques, explaining the reasoning behind their adoption, and we provided experimental comparative results among them. The results showed that, depending on the context in terms of optimization goals and time allowance, there is no clear winner among these 3 strategies. Solving a number of real-world placement problems coming from the considered industrial use-case, when the solving time is not an issue, the MILP approach is able to return the best solution with formal optimality guarantees. However, there are cases where the size of the problem is too large, to have an optimal solution in a reasonable time. If a faster solution is required, then heuristics are often realized, and this work reported an attempt to come out with good heuristics empowered with Computational Intelligence methods, such as Genetic Algorithms in different flavors. Presenting an extended set of experiments, it was shown how these latter approaches can achieve a good trade-off between computation time and quality of the found solution. In order to overcome the difficulties in heuristic selection and ease the adaptation capability to new problem requirements, a Policy-based Heuristic has been proposed. The GA-based heuristic algorithm was able to find a good placement policy to allocate the VMs in terms of used physical hosts, but it fell short in dealing with the secondary objective of minimizing the number of broken soft affinity constraints. The policy-based heuristic performed well in the experimental evaluation, and it allowed us to improve the placement in terms of the preserved soft affinity constraints within the solution, with respect to the simpler GA-based heuristic developed in our previous work [11]. In terms of execution time, the new method collocated itself in the middle between the MILP and the old genetic algorithm but still was considered manageable.

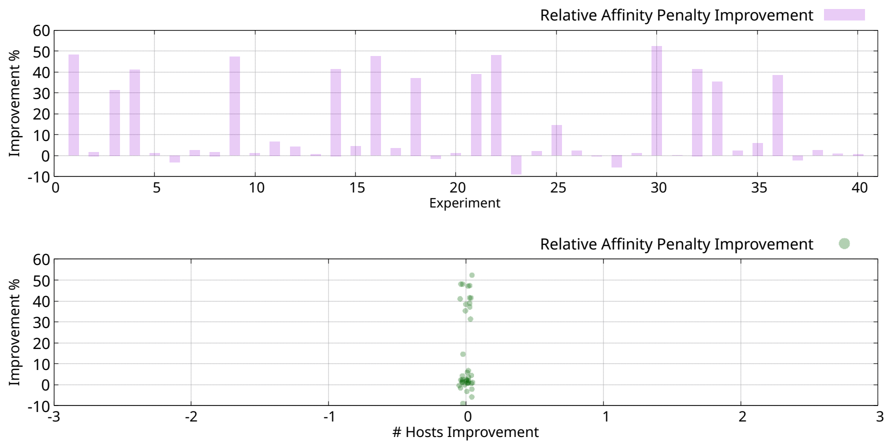

Figure 16: Affinity Penalty Improvement with respect to the GA-based Heuristic for the mid-size problem dataset. On the x-axis, the values are integer, and the visible perturbations have been introduced artificially to ease the visualization of the distribution.

In terms of possible future directions of research on the topic, it might be interesting to generalize the techniques described in this paper to handle the placement strategy of a wider multitude of resource types and structural requirements. For example, increasing the details of the network structure, keeping into account link bandwidth limitations and latencies, would extend the applicability of the proposed approach, e.g., merging with network-aware placement approaches like [25].

Second, it might be interesting to investigate solvers making use of machine learning to improve the solving time of classical MILP problems, as done recently in [34, 35]. Third, the Policy-based approach is suitable for an extension involving learning. Currently, the Policy is "learned" for a given problem instance, and it would be interesting to see the grade of generalization that can be achieved using this approach. It could seem reasonable to expect that similar problems, differing by scale, but keeping some similarity in terms of components and constraint types, could be solved using a similar placement policy. If this were the case, it would be beneficial in terms of computation time since the algorithm could be started from a previous solution and improved on that.

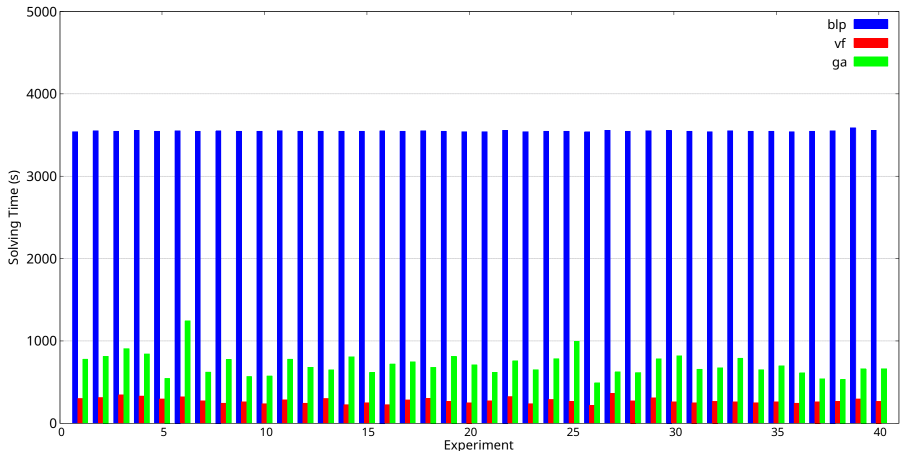

Figure 17: Solving time for the mid-size problem dataset.

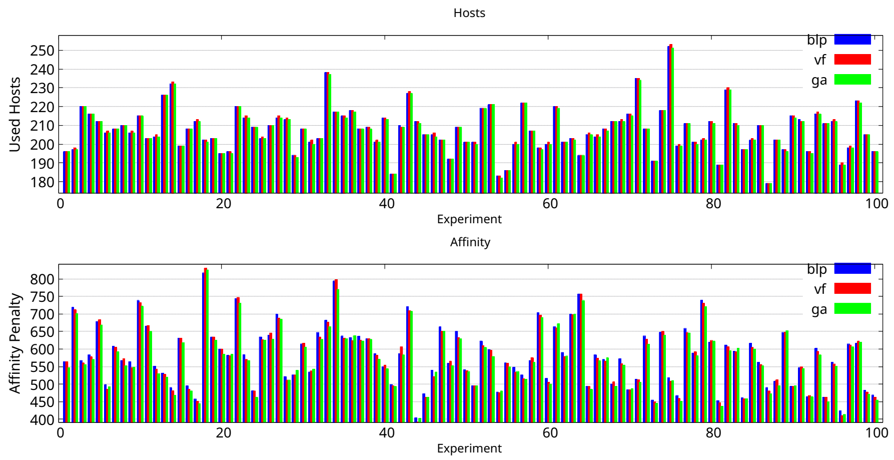

Figure 18: Number of Hosts necessary to perform the placement and Affinity Penalty computed by the placement algorithms on large-size dataset.

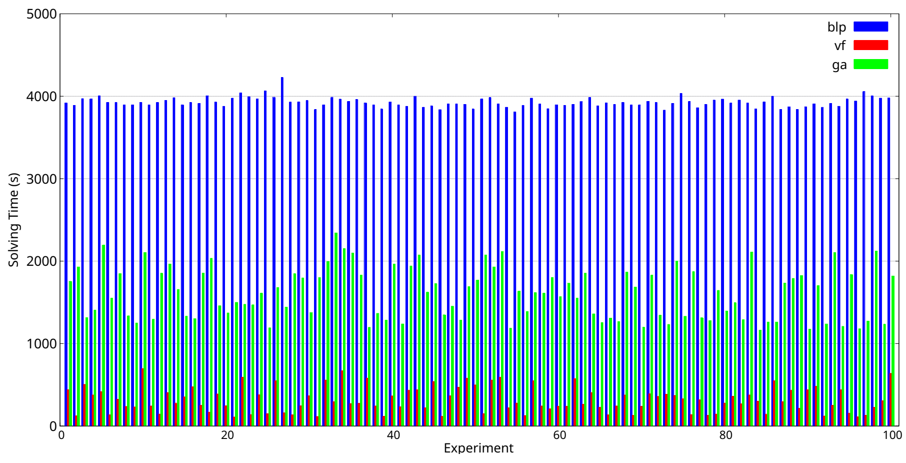

Figure 19: Solving time for the large-size problem dataset.

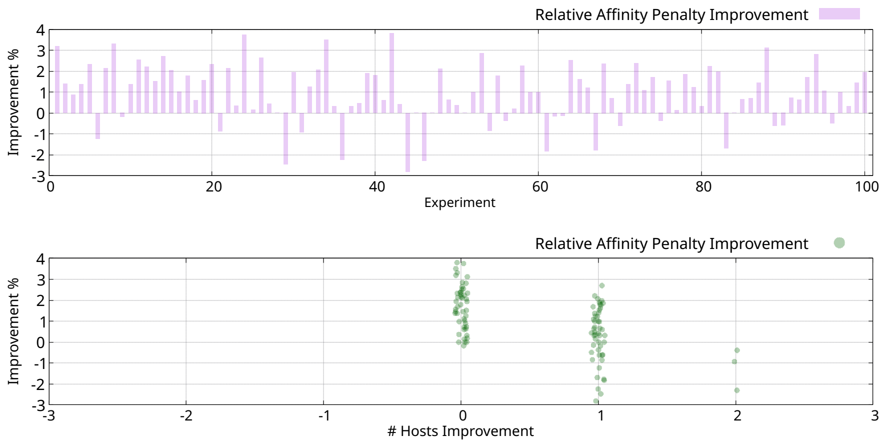

Figure 20: Affinity Penalty Improvement with respect to the GA-based Heuristic for the large-size problem dataset. On the x-axis, the values are integer, and the visible perturbations have been introduced artificially to ease the visualization of the distribution.

References

[1] Q. Liu, J. Gu, J. Yang, Y. Li, D. Sha, M. Xu, I. Shams, M. Yu, C. Yang, Cloud, Edge, and Mobile Computing for Smart Cities, Springer Singapore, Singapore, 2021, pp. 757–795.
URL https://doi.org/10.1007/978-981-15-8983-6_41

[2] P. R. Singh, V. K. Singh, R. Yadav, S. N. Chaurasia, 6g networks for artificial intelligence-enabled smart cities applications: A scoping review, Telematics and Informatics Reports 9 (2023) 100044.
URL https://www.sciencedirect.com/science/article/pii/S277250302300004X

[3] Distributed Cloud Computing and its Impact on the Cabling Infrastructure within a Data Center White Paper, https://www.cisco.com/c/en/us/products/collateral/interfaces-modules/transceiver-modules/white-paper-c11-2399751.html (5 2021).

[4] A. Osseiran, S. Parkvall, P. Persson, A. Zaidi, S. Magnusson, K. Balachandran, 5G wireless access: an overview – Ericsson White Paper 1/28423-FGB1010937, https://www.ericsson.com/4ac666/assets/local/reports-papers/white-papers/whitepaper-5g-wireless-access.pdf (4 2020).

[5] R. Buyya, C. Vecchiola, S. T. Selvi, Chapter 4 - Cloud Computing Architecture, in: R. Buyya, C. Vecchiola, S. T. Selvi (Eds.), Mastering Cloud Computing, Morgan Kaufmann, Boston, 2013, pp. 111–140. doi:https://doi.org/10.1016/B978-0-12-411454-8.00004-8.
URL https://www.sciencedirect.com/science/article/pii/B9780124114548000048

[6] Open Network Foundation (ONF), ONF SDN Evolution, White Paper, ONF (2016).
URL http://www.opennetworking.org/wp-content/uploads/2013/05/TR-535_ONF_SDN_Evolution.pdf

[7] ETSI, Network Functions Virtualisation, White Paper 1, SDN and Openflow World Congress, Darmstadt, Germany (2012).
URL https://portal.etsi.org/NFV/NFV_White_Paper.pdf

[8] ETSI, Network Functions Virtualisation, White Paper 2, SDN and Openflow World Congress, Frankfurt, Germany (2013).
URL http://portal.etsi.org/NFV/NFV_White_Paper2.pdf

[9] ETSI, Network Functions Virtualisation, White Paper 3, SDN and Openflow World Congress, Dusseldorf, Germany (2014).
URL http://portal.etsi.org/NFV/NFV_White_Paper3.pdf

[10] S. Asta, E. Özcan, A. J. Parkes, Champ: Creating heuristics via many parameters for online bin packing, Expert Systems with Applications 63 (2016) 208–221.

doi:https://doi.org/10.1016/j.eswa.2016.07.005.
URL https://www.sciencedirect.com/science/article/pii/S0957417416303499

[11] T. Cucinotta, L. Pannocchi, F. Galli, S. Fichera, S. Lahiri, A. Artale, Optimum VM Placement for NFV Infrastructures, in: 2022 IEEE International Conference on Cloud Engineering (IC2E), IEEE, 2022. doi:10.1109/ic2e55432.2022.00029.
URL https://doi.org/10.1109%2Fic2e55432.2022.00029

[12] S. Fichera, A. Artale, A. Derstepanians, L. Pannocchi, T. Cucinotta, Artificial Intelligence in virtualized networks: a journey, in: F. Falchi, F. Giannotti, A. Monreale, C. Boldrini, S. Rinzivillo, S. Colantonio (Eds.), Proceedings of the Italia Intelligenza Artificiale – Thematic Workshops, co-located with the 3rd CINI National Lab AIIS Conference on Artificial Intelligence (Ital IA 2023), Vol. 3486, CEUR Workshop Proceedings, Pisa, Italy, 2023.

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

[14] W. Attaoui, E. Sabir, H. Elbiaze, M. Guizani, Vnf and cnf placement in 5g: Recent advances and future trends, IEEE Transactions on Network and Service Management 20 (4) (2023) 4698–4733. doi:10.1109/TNSM.2023.3264005.

[15] J. Gil Herrera, J. F. Botero, Resource Allocation in NFV: A Comprehensive Survey, IEEE Trans. on Netw. and Serv. Manag. 13 (3) (2016) 518–532. doi:10.1109/TNSM.2016.2598420.
URL https://doi.org/10.1109/TNSM.2016.2598420

[16] D. Saxena, A. K. Singh, R. Buyya, Op-mlb: An online vm prediction-based multi-objective load balancing framework for resource management at cloud data center, IEEE Transactions on Cloud Computing 10 (4) (2022) 2804–2816. doi:10.1109/TCC.2021.3059096.

[17] M. Xu, W. Tian, R. Buyya, A survey on load balancing algorithms for virtual machines placement in cloud computing, Concurrency and Computation: Practice and Experience 29 (12) (2017) e4123, e4123 cpe.4123. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.4123, doi:https://doi.org/10.1002/cpe.4123.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4123

[18] N. T. Hieu, M. D. Francesco, A. Y. Jääski, A virtual machine placement algorithm for balanced resource utilization in cloud data centers, in: Proceedings of the 2014 IEEE International Conference on Cloud Computing, CLOUD '14, IEEE Computer Society, USA, 2014, p. 474–481. doi:10.1109/CLOUD.2014.70.
URL https://doi.org/10.1109/CLOUD.2014.70

[19] A. Gupta, L. V. Kale, D. Milojicic, P. Faraboschi, S. M. Balle, HPC-Aware VM Placement in Infrastructure Clouds, in: Proceedings of the 2013 IEEE International Conference on Cloud Engineering, IC2E '13, IEEE Computer Society, USA, 2013, pp. 11–20. `doi:10.1109/IC2E.2013.38`.
URL `https://doi.org/10.1109/IC2E.2013.38`

[20] M. Alicherry, T. Lakshman, Network aware resource allocation in distributed clouds, in: 2012 Proceedings IEEE INFOCOM, 2012, pp. 963–971. `doi:10.1109/INFCOM.2012.6195847`.

[21] W. Ma, C. Medina, D. Pan, Traffic-Aware Placement of NFV Middleboxes, in: 2015 IEEE Global Communications Conference (GLOBECOM), 2015, pp. 1–6. `doi:10.1109/GLOCOM.2015.7417851`.

[22] J. Zhou, Y. Zhang, L. Sun, S. Zhuang, C. Tang, J. Sun, Stochastic virtual machine placement for cloud data centers under resource requirement variations, IEEE Access 7 (2019) 174412–174424. `doi:10.1109/ACCESS.2019.2957340`.

[23] K. Konstanteli, T. Varvarigou, T. Cucinotta, Probabilistic admission control for elastic cloud computing, in: 2011 IEEE International Conference on Service-Oriented Computing and Applications (SOCA), 2011, pp. 1–4. `doi:10.1109/SOCA.2011.6166251`.

[24] K. Konstanteli, T. Cucinotta, K. Psychas, T. A. Varvarigou, Elastic Admission Control for Federated Cloud Services, IEEE Transactions on Cloud Computing 2 (3) (2014) 348–361. `doi:10.1109/tcc.2014.2325034`.

[25] T. Cucinotta, D. Lugones, D. Cherubini, E. Jul, Data Centre Optimisation Enhanced by Software Defined Networking, in: 2014 IEEE 7th International Conference on Cloud Computing, 2014, pp. 136–143. `doi:10.1109/CLOUD.2014.28`.

[26] R. Andreoli, S. Forti, L. Pannocchi, T. Cucinotta, A. Brogi, A logic programming approach to vm placement, in: Proceedings of the 14th International Conference on Cloud Computing and Services Science, SCITEPRESS - Science and Technology Publications, 2024. `doi:10.5220/0012729500003711`.
URL `http://dx.doi.org/10.5220/0012729500003711`

[27] J. Chen, Q. He, D. Ye, W. Chen, Y. Xiang, K. Chiew, L. Zhu, Joint affinity aware grouping and virtual machine placement, Microprocessors and Microsystems 52 (2017) 365–380. `doi:https://doi.org/10.1016/j.micpro.2016.12.006`.
URL `https://www.sciencedirect.com/science/article/pii/S0141933116304136`

[28] E. M. Dow, Decomposed multi-objective bin-packing for virtual machine consolidation, PeerJ Computer Science (2016). `doi:https://doi.org/10.7717/peerj-cs.47`.

[29] S. Long, Z. Li, Y. Xing, S. Tian, D. Li, R. Yu, A reinforcement learning-based virtual machine placement strategy in cloud data centers, in: 2020 IEEE 22nd International Conference on High Performance Computing and Communications;

IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2020, pp. 223–230. doi:10.1109/HPCC-SmartCity-DSS50907.2020.00028.

[30] A. Jumnal, S. M. Dilip Kumar, Optimal VM Placement Approach Using Fuzzy Reinforcement Learning for Cloud Data Centers, in: Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV), 2021, pp. 29–35. doi:10.1109/ICICV50876.2021.9388424.

[31] J. Huang, C. Xiao, W. Wu, Rlsk: A job scheduler for federated kubernetes clusters based on reinforcement learning, in: 2020 IEEE International Conference on Cloud Engineering (IC2E), 2020, pp. 116–123. doi:10.1109/IC2E48712.2020.00019.

[32] M. A. Khoshkholghi, J. Taheri, D. Bhamare, A. Kassler, Optimized service chain placement using genetic algorithm, in: 2019 IEEE Conference on Network Softwarization (NetSoft), 2019, pp. 472–479. doi:10.1109/NETSOFT.2019.8806644.

[33] M. Unuvar, M. Steinder, A. N. Tantawi, Hybrid Cloud Placement Algorithm, in: 2014 IEEE 22nd International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, 2014, pp. 197–206. doi:10.1109/MASCOTS.2014.33.

[34] V. Nair, S. Bartunov, F. Gimeno, I. von Glehn, P. Lichocki, I. Lobov, B. O'Donoghue, N. Sonnerat, C. Tjandraatmadja, P. Wang, R. Addanki, T. Hapuarachchi, T. Keck, J. Keeling, P. Kohli, I. Ktena, Y. Li, O. Vinyals, Y. Zwols, Solving mixed integer programs using neural networks (2021). arXiv:2012.13349.

[35] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, A. Lodi, Exact Combinatorial Optimization with Graph Convolutional Neural Networks, Curran Associates Inc., Red Hook, NY, USA, 2019.