Optimum VM Placement for NFV Infrastructures

Tommaso Cucinotta, Luigi Pannocchi, Filippo Galli
Scuola Superiore Sant'Anna, Pisa, Italy
{firstname.lastname}@santannapisa.it

Silvia Fichera, Sourav Lahiri, Antonino Artale
Vodafone, Milan, Italy
{firstname.lastname}@vodafone.com

***Abstract*—This paper constitutes an industrial experience report about the use of data center optimization strategies for softwarized network services within the Vodafone resource management unit for the management of virtualized network infrastructures. The problem of optimum virtual machine placement as needed in the network operator context is detailed, and different solving strategies are proposed and discussed, including heuristics based on genetic optimization. Also, experimental results are presented that compare these strategies with one another from the standpoint of optimality and execution times, using a data-set made of some of the real problems that had to be solved in the past few years by Vodafone, in order to optimize its capacity planning decisions. The presented experimental results highlight that an optimum solver leads to excessively high computation times for large problems, whereas simple heuristics may exhibit significant loss in optimality at reduced computation times. Genetic optimization, on the other hand, constitutes a very interesting trade-off between these two extremes. The data-set used for the provided results is published under an open data license, for possible reuse in future research works on the topic.**

***Index Terms*—VM placement, NFV, Optimization, Networking**

I. Introduction and Related Work

In the last decade, the world of telecommunications and networking has undergone a deep transformation, with the introduction of new network paradigms [1] such as software-defined networking (SDN) [2] and network function virtualization (NFV) [3]. These were introduced to build novel networking infrastructures with higher flexibility in management of physical resources and enhanced ability to dynamically reconfigure their capabilities according to the instantaneous – continuously changing – conditions of the network traffic. This transformation is putting the foundations to tackle the new challenges brought about by novel high-bandwidth and ultra-reliable low-latency communications as needed in mobility scenarios supporting modern and future use-case domains, such as factory automation and Industry 4.0, smart cities, automotive and transportation.

Therefore, Telco companies and network operators had to rethink, redesign and rebuild their own networking infrastructures, and their associated daily operations practices [4], [5]. In this scenario, end-to-end network services are realized with a set of network functions (i.e., virtual network functions - VNFs) deployed in form of either virtual machines (VMs) or Containerized Network Functions (CNFs) within private cloud data centers of the company. These are managed according to cloud computing principles, such as flexible management of the physical resources and dynamic provisioning based on continuous workload tracking to apply horizontal elasticity to a number of scalable VNFs.

The VNF allocation problem is very important for network operators [6] because it is at the heart of fundamental trade-offs that need to be sought between the expected performance of the deployed services and the available budget for expanding the infrastructure over time. Indeed, an optimal allocation of VNFs allows for both reducing capital expenditure (CapEx) for the data centers (DCs), and optimizing the quality of service (QoS) perceived by clients of the deployed services.

In general, in a 5G scenario, it is possible to orchestrate softwarized network functions both in the form of VMs and containers. When comparing these, traditional VMs provide a higher degree of isolation and security, and a richer functionality set, whilst containers exhibit lower overheads, relying on a single operating system kernel for a whole physical machine, resulting in a less bloated software stack. However, traditional machine virtualization has been adopted for longer in cloud computing, resulting in a more mature and consolidated set of tools and industrial practises for the management of large physical infrastructures, and many of their performance shortcomings are nowadays well addressed with para-virtualization and hardware acceleration. Therefore, large industrial infrastructures still rely on VMs, while the recent advent of well-designed and easy-to-use container management frameworks such as Docker and Kubernetes[1] is leading to a progressive shift towards CNFs, especially for ultra low-latency services.

Paper Contribution: In this paper, we consider the problem of optimum placement of VMs within the Vodafone NFV infrastructure in Europe, a.k.a., European Vodafone Network Virtual Infrastructure (NVI). This is composed of 12 Operating Companies (OpCos), each managing one or more DCs in which VNFs (so all the VMs that compose a VNF) have to be spawned according to a planning file called *virtual bill of materials* (vBOM) and the listed constraints.

Related Work: The problem of optimum placement of services in distributed infrastructures has been widely studied in the research literature. For example, a taxonomy of VM placement techniques in Cloud Computing can be found in [7], while [8] constitutes a useful survey of load-balancing algorithms to be used along with various VM placement strategies. Some authors [9] studied specifically the problem of minimizing the number of used hosts while balancing the workload

[1]More information is available at https://kubernetes.io/.

across the used hosts; others studied the problem in the context of federated cloud providers [10], or focused on probabilistic formulations of the problem [11], sometimes in the more general context of service-oriented computing [12], or focused on optimized placement for HPC-alike workloads [13] in cloud infrastructures, designing a customized scheduler that is topology-aware and hardware-aware, and can limit noisy neighbors problem in deployed VMs.

A number of authors focused also on the specific problem of VNF placement within NFV infrastructures, in which the data-intensive nature of the hosted software components requires to properly consider network-awareness in the problem formulation [14]–[16]. Often, the problem is formulated in terms of a Mixed-Integer Linear Programming (MILP) optimization problem, or even a Boolean Linear Programming (BLP) one [15], solved using standard MILP solvers. Some authors [17] also proposed relatively easy but practical mechanisms for VNF placement that can be readily used in the well-known OpenStack[2].

Recently, some authors [18] proposed also to use reinforcement learning (RL) to realize an energy-aware VM placement strategy for cloud data centers, while others mixed RL with fuzzy logic [19]. Others proposed [20] to use Genetic Algorithms (GA) to optimize the placement of VNF service chains.

Finally, it is worth to mention that, VMs or VNFs co-located on the same physical servers may undergo increased levels of temporal interference causing performance instability and degradation, especially when adopting aggressive consolidation strategies. Therefore, some authors [21], [22] studied the placement problem in presence of additional mechanisms to control and mitigate the degree of performance degradation, such as using techniques at the operating system or hypervisor level, coming from the realm of real-time systems.

Contributions: This paper follows up on the above mentioned research, comparing a few approaches for optimum VNF placement, and precisely: *i)* a traditional BLP-based tackling of the optimization problem which is solved exactly using the CPLEX commercial solver; *ii)* a much faster heuristic algorithm providing a quick solution to the problem that is often far from optimality; *iii)* a novel solver based on a Genetic Algorithm (GA), which achieves interesting trade-offs between these two. The paper discusses results obtained applying these solvers to real problem instances that arose in the context of the Vodafone NVI management and operations, focusing on the achievable trade-offs between accuracy and the needed computational complexity. With respect to other cited works, our contribution includes also a more specific problem modeling required to tackle the real-world application demands in the context of NVI. The data-set used for evaluations, i.e., the set of optimum placement problems used for obtaining the results in Section V, are published with an open data license model, and made available for general reuse by researchers investigating on similar topic through their future research.

[2]More information at: https://www.openstack.org/.

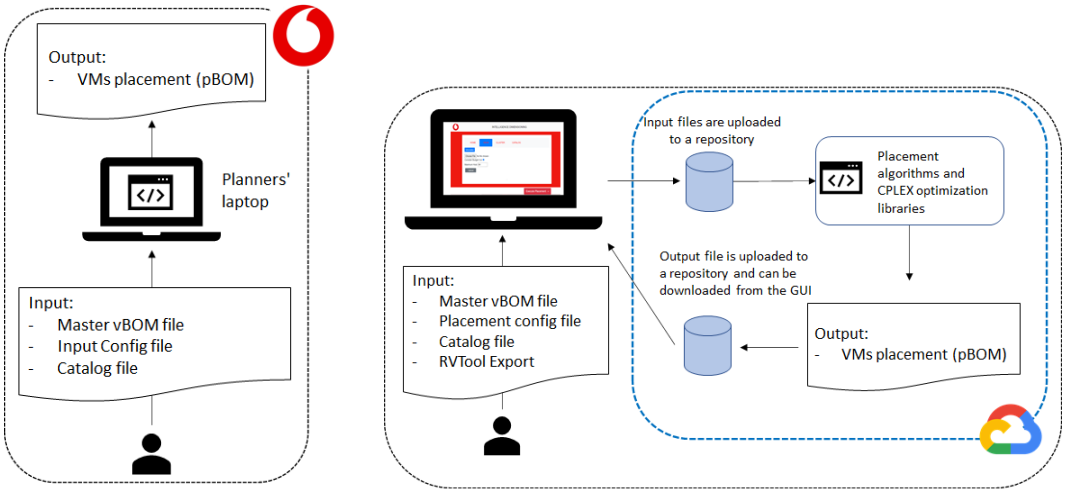
Paper Overview: The paper is structured as follows: in Section II, we introduce the scenario motivating the problem focus; in Section III, we explain the vBOM structure and its placement constraints; in Section IV, we present the possible algorithms that can be used to solve our problem; in Section V, we provide an experimental comparison among the different introduced approaches; finally, in Section VI, we present a few conclusions and ideas for future research on the topic.

II. Scenario and Problem Statement

The goal of this work is to propose effective solutions for the placement problem in the context of an industrial Network Virtualized Infrastructure (NVI), like the Vodafone one. The components to be placed are provided in the form of a vBOM table, containing the list of VMs implementing the VNF, along with their nominal resource requirements in terms of required virtual CPU cores, volatile and storage memory, and networking throughput. Additionally, there are also other constraints that encode the structure of the NVI. Indeed, to address robustness and performance demands of a NVI, a number of affinity and anti-affinity constraints have to be satisfied while considering the placement problem.

Currently, the placement of each VNF is done by the Vodafone planners from the NVI capacity planning and management team using a commercial solver. The objective of the placement tool is to place the list of VMs defined in the input vBOM file into a set of blades, occupying the minimum possible number of blades in the process, without overloading the underlying physical resources, or, sometimes, under a controlled over-provisioning policy. The tool currently uses a heuristic greedy algorithm for VM placement that performs a single placement at a time, performing essentially a first-fit placement of the VMs in the order provided by the vBOM specification. The current scenario is represented in Figure 1(a). The planner runs the tool, which takes three files as input: *i)* the master vBOM, *ii)* a problem configuration file detailing options to be used during the optimization (see Section III), and *iii)* a "catalog" configuration file describing the characteristics of the new hardware to be considered.

In the new scenario, depicted in Figure 1(b), the capacity optimizer is deployed within the Google Cloud Platform [23], which is hosting various components of the solution: a web GUI, a set of back-end solvers, including the CPLEX [24]-based and the GA-based optimizers as described below in Section IV. The back-end solvers take as input, in addition to the files mentioned above, also *iv)* the RVTool export representing the current physical status of blades occupied in the data centers under consideration. The web GUI allows for uploading a set of different problem configurations, launching the solvers, and retrieving the computed solutions, which are stored in a dedicated repository. The application supports multiple users concurrently using the system and launching different optimizations simultaneously. The new architecture allows for solving more placement problems simultaneously and store the output results in the cloud. Section III provides



(a) Current scenario in which the planner runs the placement tool in her own PC.

(b) Intelligent dimensioning scenario where the planner is supported by the GCP and the placement is evaluated with placement algorithms running in the Cloud.

Fig. 1: As-is vs proposed scenario.

details on the vBOM placement solving strategies implemented at the moment in the new solver.

III. vBOM Placement Problem

In this section the problem to be solved and its components will be described formally. In particular, we describe the modeling problem that is characterized by the objective function, the resource requirements (i.e., CPU cores, memory, and network bandwidth), and the structural constraints.

We consider a VNF deployment plan including a set V of N_v VNF components to be deployed: $V = \{1, 2, \dots N_v\}$. In turn, each VNF component $i \in V$ is characterized by a set I_i of VMs implementing the functionality, all with homogeneous resource requirements. These can either constitute multiple instances over which the traffic for the VNF component is spread by a load-balancer, or multiple replicas of the same functionality that are kept in sync with each other, so to handle faults of one or more VMs (e.g., using primary/secondary replication and redundancy schemes). The exact load-balancing and VM replication strategy does not impact on our problem formulation, where we just consider that each VM $j \in I_i$ belonging to the same VNF component $i \in V$ is characterized by the same amount of computational, storage and network bandwidth requirements. In what follows, for ease of notation we assume, without loss of generality, that VMs for different VNF components have all different indexes: $\forall i_1, i_2 \in V, i_1 \neq i_2 \implies I_{i_1} \cap I_{i_2} = \emptyset$. The VMs have to be placed on a set H of N_h physical hosts: $H = \{1, 2, \dots, N_h\}$.

In order to represent the placement of a single VM j on a host h, Boolean variables $\{x_{j,h}\}$ are introduced:

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

The placement problem requires that each VM must be placed on exactly one host. This can be expressed as:

$$\forall i \in V, \forall j \in I_i, \sum_{h \in H} x_{j,h} = 1. \tag{2}$$

In the following, the ingredients necessary to build the optimum placement problem are reported:

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

For the industrial case study proposed in this work the focus is on reducing the number of hosts necessary to fulfil the deployment of the NVI. 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:

$$J_h = \sum_{h \in H} u_h. \tag{3}$$

Specifically, $u_h = \vee_j x_{j,h}$, where $\vee_j$ is the logical or operator applied along the index j.

2) Resource Requirements: In the following, the demands of a VNF $i \in V$ in terms CPU, Memory and Network Bandwidth are expressed by D_i^{CPU}, D_i^{MEM} and D_i^{NET} respectively. Similarly, the availability of resources on each Host $h \in H$ is given by L_h^{CPU}, L_h^{MEM} and L_h^{NET}. The placement constraints, due to the availability of limited resources on each host, can be described as follows:

- CPU

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

- Memory

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

- Network Bandwidth

$$\forall h \in H : \sum_{i \in V} D_i^{NET} \left(\sum_{j \in I_i} x_{j,h} \right) \leq L_h^{NET}. \quad (6)$$

3) Structural Constraints: Structural constraints control how the VMs should be placed with respect to one another. Constraints referring to the VMs of a single VNF component are called "self-affinity" rules, while constraints extending to VMs belonging to different VNF components are called "cross-affinity" rules.

Formally, the constraints to be considered in the industrial context under consideration are of the following types:

- *Self 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

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

- *Self 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:

$$\forall i \in V^{AFF}, \forall h \in H : \sum_{j \in I_i} x_{j,h} \in \{0, |I_i|\}, \quad (8)$$

  where $|\cdot|$ denotes the set cardinality operation.
- *Cross Anti-Affinity constraint (CAAF)*: Given a set of VNF, 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:

$$x_{j_\alpha,h} + x_{j_\beta,h} \leq 1, \quad (9a)$$
$$\forall h \in H, \quad (9b)$$
$$\forall (i_\alpha, i_\beta) \in V^{CAAF}, \quad (9c)$$
$$\forall j_\alpha \in V_{i_\alpha}, \forall j_\beta \in V_{i_\beta} \quad (9d)$$

- *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:

$$\sum_{i \in V^{CAFF}, j \in I_i} x_{j,h} \in \{0, |I_{CAFF}|\}, \quad (10)$$

  where $|I_{CAFF}| = \sum_{i \in V^{CAFF}} |I_i|, \forall h \in H$.
- *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. 9.
- *Cluster constraint*: this optional constraint specifies that the physical infrastructure is logically partitioned in clusters of the same given size, normally 64 hosts, and that a relatively large vBOM with several VNF components needs 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; this partitioning/splitting of large vBOMs into clusters was traditionally done manually by placement experts, but the CPLEX-based and the GA-based optimizers we realized are capable of performing these operations automatically, optimizing the target objective function.

Regarding the self-affinity and cross-affinity constraints, in addition to the "hard" interpretation of them as detailed above, we also allow for a "soft" interpretation of them. This means that soft affinity constraints become a nice-to-have property in the final solution, but they can be sacrificed to achieve reduced numbers of hosts. Therefore, the solver can be configured to replace soft affinity constraints with an additional term in the objective function that allows for achieving, as a secondary objective, the minimization of the number of affinity constraints that are broken in the solution.

IV. Solvers

Optimal Placement is usually achieved by leveraging classical optimization techniques, such as Mixed Integer Linear Programming (MILP). Specifically, given the nature of the problem, it is a Boolean Linear Programming (BLP). These type of approaches are characterized by the possibility to give guarantees on the solution quality, but they are also known to be quite slow and computationally prohibitive. For this reason, there are efforts to find alternative solutions employing heuristic approaches, such as Machine Learning (ML) or Computational Intelligence, which can provide a good trade-off between optimality and computational performance. In this work the placement problem will be tackled with the classical MILP approach and then a comparison with an alternative approach that employs Genetic Optimization is proposed.

A. Standard MILP solvers

Standard MILP solvers, such as open-source ones like *GLPK*[3] and *Coin-OR CBC*[4], or commercial ones like *CPLEX*[5] by *IBM* or *Gurobi*[6], allow for tackling the placement problem by formulating it as a Mixed Integer Linear Programming (MILP) optimization problem. MILP solvers allow for expressing a problem using their custom interfaces and APIs. However, these tools usually have also a command-line solving tool that takes as input a MILP problem in a standard `lp` format, a textual algebraic format that is human-readable, thus being also a useful debugging tool.

[3]https://www.gnu.org/software/glpk/
[4]https://www.coin-or.org/Cbc/
[5]https://www.ibm.com/analytics/cplex-optimizer
[6]https://www.gurobi.com/

Implementation: In this work, the problem formulation used with the MILP approach is based directly on the linear formulation shown in Section III. As far as the implementation is concerned, we realized a tool capable of using the CBC and CPLEX solvers, encoding the problem in the `lp` file format and invoking the command-line solver provided by the tools.

B. Heuristic Solver

Heuristic solvers are simple and computationally efficient. With respect to a standard MILP solver, that require to cast everything in a linear form, a heuristic solver may be simpler to realize. Given their nature of being lightweight and easy to implement, they are quite useful in practical applications, but they lack optimality guarantees since they can easily get stuck in local minima.

1) Implementation: In this work, the implemented a Heuristic Solver based on the the "First Fit" algorithm, where the VMs are allocated, one after the other, in the first feasible spot available on the hosts. The solution feasibility is dictated by the same constraints described in Section III.

C. Genetic Optimization

The Genetic Optimization approach (or genetic algorithm - GA) shares with the Heuristic Solver the simpler modeling phase with respect to the Standard MILP approach, but adds a better heuristic search to improve the convergence properties towards a minimum, overcoming the problem mentioned for the pure Heuristic Solvers.

Implementing a Genetic Optimizer requires selecting the features that encode candidate solutions and the evolution process that leads to the optimum. Following the modeling of the problem presented in Section III, it would come natural 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.

The Genetic Optimizer works by propagating and modifying an initial population of different candidate solutions, favoring the survival of better candidates with respect to a "fit function". In this case, the fit function used to select the best candidates is the number of hosts used to allocate all the VMs.

To increase the probabilities of success, it is necessary to handle a big-enough population of possible candidates. The risk of increased computation time is usually tackled by recurring to simple and inexpensive operations for manipulating the population of candidate solutions.

1) Implementation: The Genetic Optimizer is implemented as an iterative procedure applied to a population of candidate solutions. The optimization 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}.

The generation of new candidates to extend the current solution is made by using solutions from the original population. Precisely, starting from a candidate solution, the algorithm searches for a pair of VMs that can be swapped with respect to the hosts on which they are placed. In the current implementation the search for a pair is completely random. A new solution is then generated, swapping the found pair of VMs and trying to compress the obtained solution by moving the other VMs in the free spaces that could have been created by the swap. The process is visually represented in Figure 2, where a placement on 3 Hosts undergoes the swapping and compression steps, as described in the GA algorithm.

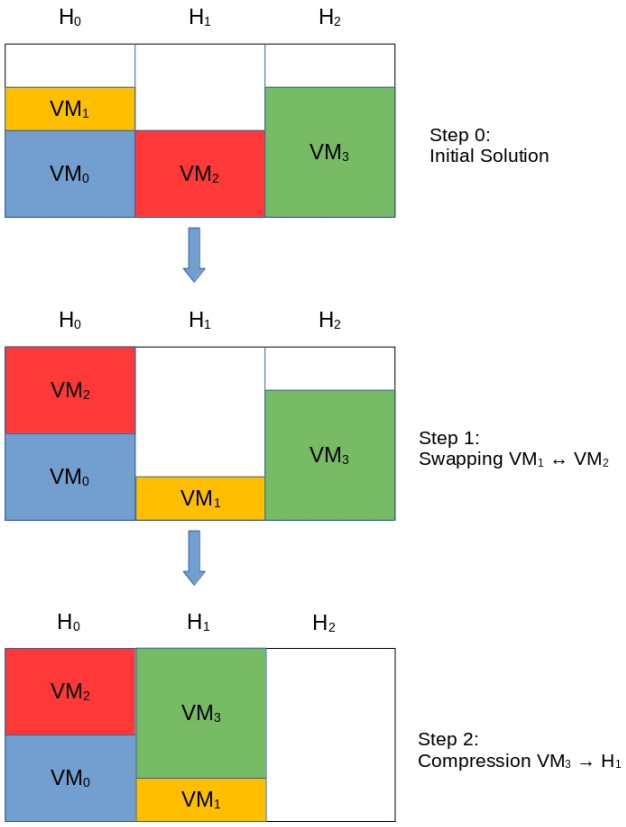


Fig. 2: Generation of new candidates and compression step in the proposed GA implementation.

GA solvers are known to be computationally heavy due to the high number of attempt they make in order to select the best candidates. The implementation proposed in this work allows also a way to early stop the solver in case no improvement is foreseen in the current population. MILP approaches have the possibility to quantify the gap between the current solution and a best one obtained via relaxation and use that to stop the solver earlier. Heuristic solvers do not know how far are from the optimum, for this reason it is necessary to evaluate heuristically whether it is worth continuing exploring

new solutions. In the approach proposed with this work, the idea is 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 max spread of the fit function along 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 on the population and the threshold for the early stopping feature.

V. Experimental results

The proposed approach has been evaluated by using data coming from the planning needs of the Vodafone operator, concerning its NFV infrastructure. Real vBOMs, coming from the Vodafone network, have been collected and the associated placement problems have been solved using the three approaches previously mentioned: Simple Heuristic, MILP and the proposed Genetic Optimization. The vBOMs differ in terms of number of VNFs to be placed, number of VMs implementing them and number of constraints to be satisfied. An open data-set including a number of vBOMs we have optimized, has been made publicly available[7]. The goal of the experiment is to evaluate the performance of the proposed Genetic Optimizer with respect to state-of-the-art solvers, such as an optimum modeling using MILP, and a simplistic greedy heuristic, in solving placement problems.

The MILP solver is realized using CPLEX libraries. In order to avoid very long solving times, a termination procedure was triggered whenever the computation exceeded 600 seconds. We allowed a "gentle" termination, guaranteeing at least a feasible solution and making sure partial solutions are completely evaluated.

The Heuristic Solver implements a First Fit placement that is run only once for each problem.

The GA is run with a population of 50 candidates and a max of 25 iterations. These values have been found after a preliminary test campaign on the dataset to evaluate the exploration/exploitation/solving-time trade-off of the GA. The preliminary tests were also used to check the convergence property of the GA solver, running the biggest problem of the dataset multiple times and verifying the stability of the outcome. For every original candidate 4 new candidates are generated in the swapping phase.

The Heuristic Solver and the GA have been implemented from scratch in Python, leveraging the *Numpy* package to improve the computations with vectorized operations.

The placement problems have been carried out considering a target network infrastructure composed by homogeneous machines, each having the following hardware specifications:

- CPUs: 44;
- RAM: 420 GB;
- Network Bandwidth: 15000 Mbit/s.

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

	Heuristic	MILP	GA
VNF: 220 VM: 1611 AAF: 1475 AFF: 0	209 (2.443)	183 (3892.295)	183 (72.891)
VNF: 122 VM: 780 AAF: 487 AFF: 0	166 (0.85)	153 (629.860)	154 (8.479)
VNF: 47 VM: 294 AAF: 154 AFF: 0	38 (0.044)	34 (11.291)	34 (1.246)
VNF: 34 VM: 140 AAF: 140 AFF: 0	28 (0.034)	26 (1.051)	26 (0.482)
VNF: 43 VM: 144 AAF: 74 AFF: 0	16 (0.032)	15 (2.466)	15 (0.495)
VNF:10 VM: 50 AAF: 50 AFF: 0	17 (0.012)	17 (0.023)	17 (0.108)

TABLE I: Some experimental results: for every test case we report the number of hosts computed by the solvers; in parenthesis the time (seconds) needed to find the solutions.

The experimental evaluation of the different placement approaches has been run on a dedicated server equipped with a *Intel(R) Xeon(R) CPU E5-2640 v4 @2.40GHz* and 64 GB of RAM.

A. Experimental Evaluation

Table I reports some of the results obtained during the experiments and it confirms what was mentioned in the previous sections. As a matter of fact, the Heuristic Solver is definitely the faster solver compared to the other two, but, when the placement problem involves several constraints, it can return sub-optimal solutions. As an example, in the placement problem involving 220 VNFs, the Heuristic Solver reported a number of required hosts that could have been reduced by 12% using the other approaches. The classical MILP approach, while providing the best results together with guarantees regarding the solution optimality, is significantly slower than the other approaches. This was expected, since MILP problems are NP hard. The fact that there are cases where the MILP solving time is greater than the time-limit is due to our "gentle" implementation of the abort procedure, where we preferred to bring to an end the current relaxation step. The proposed Genetic Optimizer, provides solutions that are comparable with the ones returned by MILP, just with a considerable reduction in the time required to obtain them. The lower computation times of the Heuristic and Genetic approaches, were also expected. Indeed, they are leveraging high level information about the problem to be solved to drive efficiently towards feasible solutions, whilst the classical MILP is unaware of the high level description of the problem and just deals with the low-level mathematical formulation.

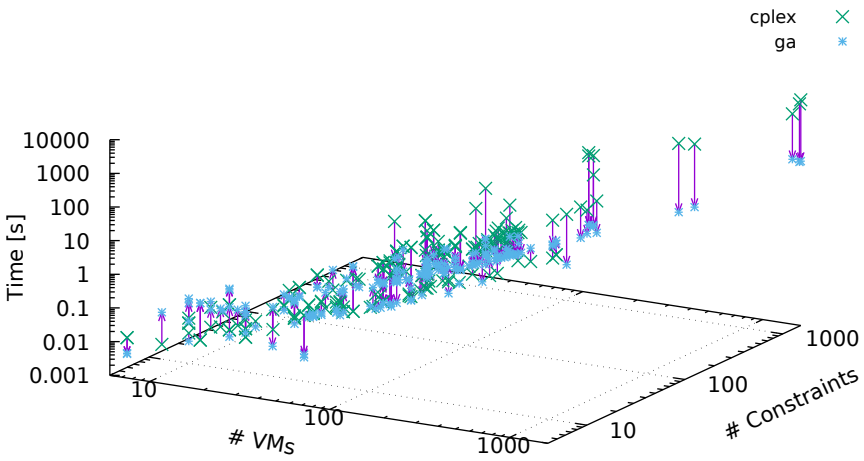


Fig. 3: Comparison of the computation times using the GA and the MILP approaches as a function of the number of VMs and Constraints in a placement problem.

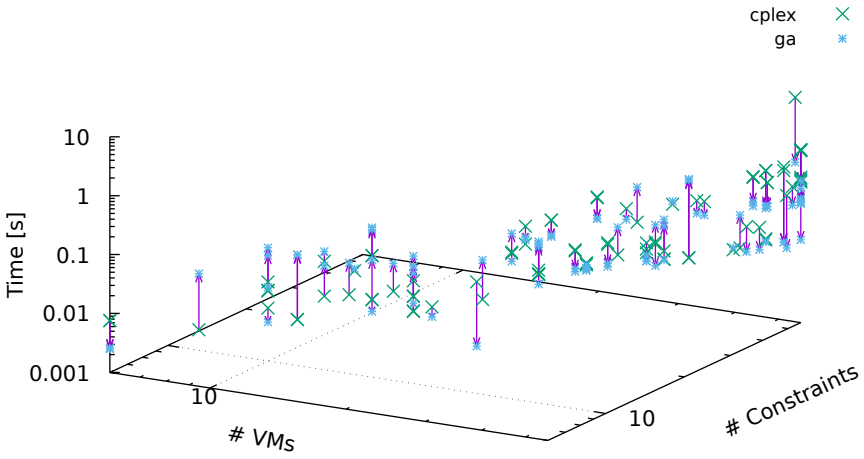


Fig. 4: Detail of the results, showing the behavior of the GA and MILP approaches for small problems.

An interesting observation can be made looking at Fig. 3, where the computation times of the GA and the MILP are plotted together for all the experiments. Indeed, even if the GA provides a good improvement for big problems, for very simple ones, the MILP can be actually faster than the GA. A closer look at this behavior can be given in Fig. 4 which focuses on smaller problems. This is possibly due to the computationally efficient implementation of the MILP. In our case, even if attention has been paid to vectorize part of the computations to leverage optimized Python libraries, our proposed algorithm still suffers from a Python implementation, which is less efficient with respect to what could be achieved with other languages. Apart from this observation, which can be easily tackled with a better implementation, it is interesting to notice how the proposed approach is leading to interesting results with medium-big size problems that occur in industrial applications.

VI. Conclusions and Future Work

This paper formalized the VNF placement problem as being investigated in the context of NFV infrastructure management for the Vodafone network operator. We have compared the execution time and optimality of the solution achieved by using traditional BLP-based solution strategies, with a novel GA-based solver. Results have been presented applying these

tools on real vBOM problems coming from recent deployments performed within the network operator. The data used for the experimentation has been made openly available for download and re-use by other researchers willing to perform further research on the topic.

The presented techniques are being integrated in production code that is used by the Vodafone NVI capacity planning and optimization team for their periodic operations.

Following up on the research being presented in this paper, a number of further investigations are possible. For example, seeking for optimal solutions with the CPLEX-based approach becomes easily cumbersome for particularly large infrastructure sites. Indeed, in some cases, we end-up with a BLP problem that takes days to be solved, even if in the majority of the cases the best solution output after 1 hour is already considered "good enough", as the solver finds an optimality gap below 5%. For large vBOMs, an optimal CPLEX-based approach might be applied on smaller problems, by doing some heuristic partitioning of the problem into smaller ones. Or, we could apply the GA-based solver that drops optimality in favour of a greatly reduced solving time, still managing to find a solution relatively close to the optimal one found in much more time by CPLEX, as shown Section V in this paper. MILP solvers usually benefit from initial guesses, and this could lead to a further interesting evaluation where the GA is used in combination with CPLEX to tackle large problems.

Moreover, the presented GA-based heuristic can be improved and extended, for example adding additional mutations to be applied to the population of solutions within each Genetic Algorithm step, in order to increase the chances to discover solutions with a better objective function.

Furthermore, the set of constraints to be considered is still being evolved within the Vodafone NVI infrastructure, as required for example by high-performance VNFs requiring NUMA-aware placement or specific management of hyper-threading throughout the physical infrastructure. Adding further constraints and case studies will be interesting to investigate how the nature of the problem with its different constraints can impact the solvers performance.

REFERENCES

[1] H. Woesner and D. Verbeiren, "SDN and NFV in telecommunication network migration," in *2015 Fourth European Workshop on Software Defined Networks*. IEEE, Sep. 2015.

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

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

[4] A. Leaf, "Vodafone Completes European Roll-out of VMware Network Virtual Infrastructure to Accelerate Shift to Digital Network," https://news.vmware.com/releases/vodafone-completes-european-roll-out-of-vmware-network-virtual-infrastructure-to-accelerate-shift-to-digital-network, Apr 2020.

[5] R. Honnachari, "Software Define Your Enterprise WAN with Virtual Network Services – A Frost & Sullivan White Paper," https://www.business.att.com/content/dam/attbusiness/reports/frost-and-sullivan-att-flexware-whitepaper.pdf, 2021.

[6] J. Gil Herrera and J. F. Botero, "Resource Allocation in NFV: A Comprehensive Survey," *IEEE Trans. on Netw. and Serv. Manag.*, vol. 13, no. 3, p. 518–532, sep 2016. [Online]. Available: https://doi.org/10.1109/TNSM.2016.2598420

[7] F. L. Pires and B. Barán, "A virtual machine placement taxonomy," in *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May 2015, pp. 159–168.

[8] M. Xu, W. Tian, and R. Buyya, "A survey on load balancing algorithms for virtual machines placement in cloud computing," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 12, p. e4123, 2017, e4123 cpe.4123. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4123

[9] N. T. Hieu, M. D. Francesco, and 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*, ser. CLOUD '14. USA: IEEE Computer Society, 2014, p. 474–481. [Online]. Available: https://doi.org/10.1109/CLOUD.2014.70

[10] K. Konstanteli, T. Cucinotta, K. Psychas, and T. A. Varvarigou, "Elastic Admission Control for Federated Cloud Services," *IEEE Transactions on Cloud Computing*, vol. 2, no. 3, pp. 348–361, Jul. 2014.

[11] J. Zhou, Y. Zhang, L. Sun, S. Zhuang, C. Tang, and J. Sun, "Stochastic virtual machine placement for cloud data centers under resource requirement variations," *IEEE Access*, vol. 7, pp. 174 412–174 424, 2019.

[12] K. Konstanteli, T. Varvarigou, and T. Cucinotta, "Probabilistic admission control for elastic cloud computing," in *2011 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, Dec 2011, pp. 1–4.

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

[14] M. Alicherry and T. Lakshman, "Network aware resource allocation in distributed clouds," in *2012 Proceedings IEEE INFOCOM*, March 2012, pp. 963–971.

[15] T. Cucinotta, D. Lugones, D. Cherubini, and E. Jul, "Data Centre Optimisation Enhanced by Software Defined Networking," in *2014 IEEE 7th International Conference on Cloud Computing*, June 2014, pp. 136–143.

[16] W. Ma, C. Medina, and D. Pan, "Traffic-Aware Placement of NFV Middleboxes," in *2015 IEEE Global Communications Conference (GLOBECOM)*, Dec 2015, pp. 1–6.

[17] S. Oechsner and A. Ripke, "Flexible support of VNF placement functions in OpenStack," in *Proceedings of the 1st IEEE Conference on Network Softwarization (NetSoft)*, April 2015, pp. 1–6.

[18] S. Long, Z. Li, Y. Xing, S. Tian, D. Li, and 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)*, Dec 2020, pp. 223–230.

[19] A. Jumnal and 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)*, Feb 2021, pp. 29–35.

[20] M. A. Khoshkholghi, J. Taheri, D. Bhamare, and A. Kassler, "Optimized service chain placement using genetic algorithm," in *2019 IEEE Conference on Network Softwarization (NetSoft)*, June 2019, pp. 472–479.

[21] S. Xi, C. Li, C. Lu, C. D. Gill, M. Xu, L. T. Phan, I. Lee, and O. Sokolsky, "Rt-open stack: Cpu resource management for real-time cloud computing," in *2015 IEEE 8th International Conference on Cloud Computing*, June 2015, pp. 179–186.

[22] T. Cucinotta, L. Abeni, M. Marinoni, R. Mancini, and C. Vitucci, "Strong temporal isolation among containers in OpenStack for NFV services," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2021. [Online]. Available: https://doi.org/10.1109%2Ftcc.2021.3116183

[23] Cloud Computing Services — Google Cloud. [Online]. Available: https://cloud.google.com/

[24] IBM ILOG CPLEX Optimizer. [Online]. Available: https://www.ibm.com/analytics/cplex-optimizer