Routing-Led Algorithm for Large-Scale VNF Placement Problems

Joseph Billingsley, *Student Member, IEEE*, Ke Li, *Member, IEEE*, Wang Miao, Geyong Min, Nektarios Georgalas

***Abstract*—Modern data centers contain thousands of servers making them major consumers of electricity. To minimize their environmental impact, it is critical that we use their resources efficiently. In this paper we study how to discover the optimal placement of virtual network functions in large scale data centers. We propose a novel parallel metaheuristic, fast heuristic objective functions of the QoS and new memory efficient data structures for large networks. We further identify a simple, fast heuristic that can produce competitive solutions to very large problem instances. Using these new concepts, we are able to find high quality solutions for data centres with up to 64,000 servers.**

***Index Terms*—Network Function Virtualization, Multi-Objective Optimization, Routing-Led VNF Placement.**

I. Introduction

Modern communication services have and continue to have an outsized impact on the way that society works [1], travels [2], socializes [3] and engages with politics [4], government [5] and other individuals [6]. But this does not come without a cost. These services are also major consumers of electricity, with communications technology as a whole expected to be responsible for 21% of the worlds total electricity usage by 2030 [7].

If current trends continue, by 2030 data centers will likely be the largest single contributor of carbon emissions among communications infrastructure [7]. Over the previous decade, extensive work was conducted to reduce the energy consumed by overhead, such as heat management and energy provisioning, in data centers [8]. Through these efforts, the energy total consumption of data centers remained constant in the United States [9] and doubled in the European union [10] from 2010 - 2020, despite a forecasted 10$\times$ increase in data center traffic [11]. However, in recent years, progress in this direction has slowed, as improvements to overhead efficiency have reached the point of diminishing returns [12].

Recent research found that the efficiency of a data center could be improved by a further 10% to 40% by maximizing the utilization of existing computing equipment [10], [13]. However, existing communication services make extensive use of specialized hardware which can only be utilized for a single task. Specifically, communication services are typically constructed using sequences of purpose built computing equipment known as middleboxes or physical network functions. Each middlebox can perform a single task with high efficiency, but being physical components, the number and location of these middleboxes must be specified well in advance of their usage. A recent alternative are Virtual Network Functions (VNFs). VNFs use software running on virtual machines to perform the same tasks as middleboxes. Using VNFs, multiple network functions can be provided by the same server, maximizing the utilization of hardware which results in less energy being needed to provide the same quality of service (QoS). In addition, the resources used by VNFs can be moved and scaled to meet traffic demands without over or under allocating resources.

Although VNFs allow increased utilization of servers, it remains an open problem how these VNFs should be assigned to servers. The VNF Placement Problem (VNFPP) is the task of automatically determining the placement of VNFs in a data center so as to provide high quality services with low energy consumption. The VNFPP presents many challenges. First, it is widely acknowledged that VNFPP is a NP hard problem [14]–[16] for which there are no known solutions that can find an exact solution in reasonable time. Second, whilst the QoS objectives and energy consumption are both well understood, they are challenging to model accurately and efficiently. Finally, due to the large numbers of servers and VNFs in a data center the total search space is very large whilst the feasible search space is relatively small [17] which makes finding feasible solutions challenging, and high quality solutions even more so.

In prior work [17], we proposed a combined metaheuristic and model solution with a genotype-phenotype mapping that found good solutions to the VNFPP despite these challenges and which could solve 8x larger problems than the state of the art. However, the proposed algorithm was only able to solve problems for one type of network topology. More recently, we proposed a modification of our original algorithm that allowed the algorithm to work on arbitrary network topologies but this approach had limited scalability [18]. In this paper we build on our earlier work to make further improvements and resolve the scalability issues. In particular, this paper combines our generalized genotype-phenotype mapping with the following innovations:

1) A new parallel metaheuristic that uses decomposition and local search to efficiently search the large solution space.
2) Efficient objective functions that balance model accuracy against evaluation time.
3) Memory efficient data structures that significantly reduce the memory consumption of the metaheuristic

Using these innovations, our solution is the first to solve

J. Billingsley, K. Li, W. Miao and G. Min are with the Department of Computer Science, University of Exeter, North Park Road, Exeter, EX4 4QF, UK (e-mail: {j.billingsley, k.li, wang.miao, g.min}@exeter.ac.uk)

N. Georgalas is with the Research and Innovation, British Telecom, Martlesham, UK (e-mail: nektarios.georgalas@bt.com)

the VNFPP for ∼65,000 servers. This represents a further 4x improvement over our earlier solution, and a 16x improvement over the preceding state of the art.

The remainder of this paper is organized as follows. Section II reviews the current work on the VNFPP. Section III provides a formal definition of the VNFPP. Section IV describes the parallel metaheuristic and operators considered in this work including the memory efficient data structures. Section V discusses common objective functions for the VNFPP. In Section VI we test the effectiveness of each component of the algorithm and evaluate our solution on very large problem instances. Finally, Section VII concludes this paper and outlines some potential future directions.

II. Related Works

This section provides a pragmatic overview of some important developments on the VNFPP. Whilst there is a wealth of work on the problem, most solutions have limitations that make them impractical for real world application. Existing solutions tend to fall into two groups:

- Work that proposes solutions for realistic versions of the problem but which cannot scale to large problems.
- Work that finds a scalable solution to a simplified problem that may not be useful in practice.

A. Realistic but Not Scalable

Several works that use realistic models of the data center in the VNFPP also use exact methods to optimise them. Exact methods guarantee optimal solutions but also have an exponential worst case time complexity for NP-Hard problems such as the VNFPP [19] and as a result, can only solve problems with tens of servers, making them impractical for real world problems. Further, exact methods typically require a linear objective function whilst real measurements of the QoS show it to be non-linear [20], [21]. In order to use a realistic model with exact methods, researchers commonly use piecewise linearization to linearize the measures of QoS. For example, Addis et al. [22] used linear programming and piecewise linearization to solve two VNFPPs: one where waiting time is modelled as a convex piece-wise linear function of the sum arrival rate, and another where the latency is a constant when it is below a certain threshold. This work considered up to 15 servers and solved two objectives, by first minimizing the max link utilization and then minimizing the number of cores used. Gao et al. [23] extended this work to allow one VNF per service to alter the traffic rate leaving the VNF. In [24], Baumgartner et al. place a small number of VNFs across a geographically distributed network. They model the processing, transmission and queueing delays and linearize them with piecewise linearization. Oljira et al. [25] used the same techniques as [24] for modeling and optimization and additionally considered the virtualization overhead when calculating the latency at each VNF. Both of these works considered a problem where there were only 28 locations a VNFs could be placed. Similarly, Jemaa et al. [26] only allowed VNFs to be placed in two locations: a resource constrained cloudlet data center near the user and an unconstrained cloud data center. VNFs were assigned to a data center to optimise for a combination of latency and cloudlet and cloud utilization. Agarwal et al. [27] also use exact methods to solve a small problem instance with 3 servers. They aim to minimize the ratio of the actual latency to the max latency for each service. Marotta et al. [28] use a series of heuristics and exact methods to solve problem instances with 10s of servers. They first assign VNFs to servers to minimize energy and inter-server traffic. Then a heuristic makes the placement robust to changes in arrival rate and finally exact methods are used to find routes between VNFs to form services that minimize the energy consumption whilst meeting latency constraints.

B. Scalable but Unrealistic

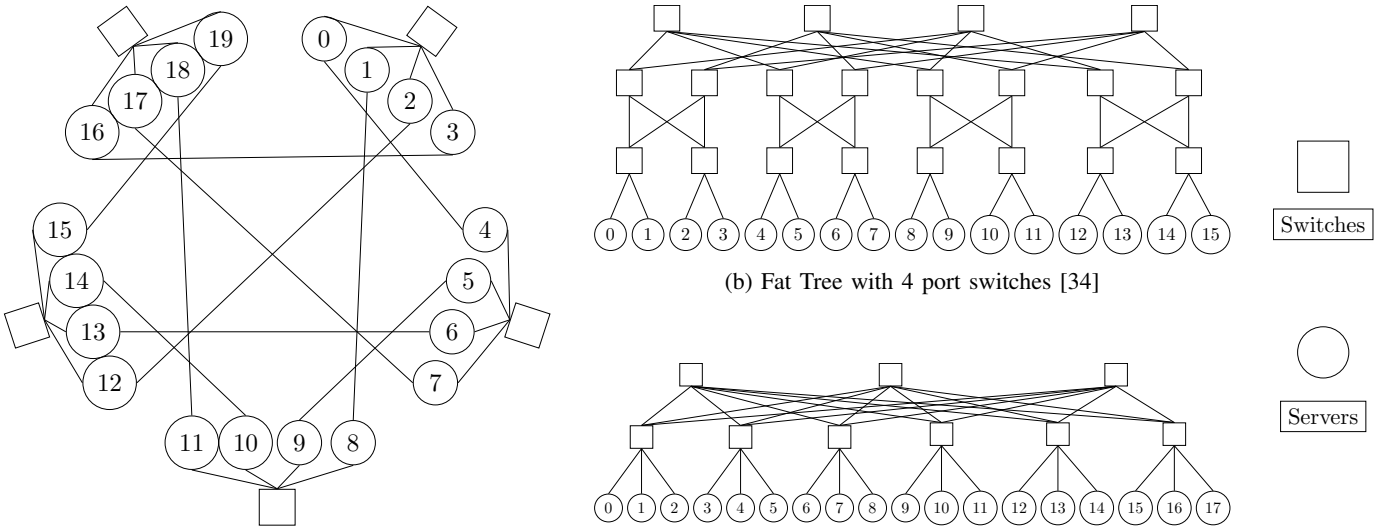
The other class of impractical solutions use heuristic or metaheuristic algorithms to provide scalable solutions but that solve oversimplified versions of the VNFPP. This research can be subdivided into solutions that use heuristic objective functions, solutions which use unrealistic models of the QoS, and solutions that do not utilise a model at all.

1) Heuristic Model: Several works do not consider the QoS or energy consumption and instead optimise for simpler models that are easier to construct and solve. These works do not consider whether these heuristics are suitable metrics compared to an accurate model and hence may not actually be optimizing for the intended objectives and hence may produce subpar solutions. For example, we have previously shown that if common heuristic models are used for the VNFPP the solutions are significantly less diverse than those found by accurate models [17].

Kuo et al. [29] aim to maximize the number of placed services by efficiently reusing network functions. They first preprocess each service with ILP to find a structure that allows for the most VNF reuse. Then they use dynamic programming to place VNFs, aiming to minimize the consumed resources. Guo et al. [30] also reuse VNFs but aim to minimize bandwidth and placement costs. They first preprocess the topology to find the most influential nodes according to the Katz centrality. Shareable VNFs can only be placed on these nodes increasing the chance they are reused. Finally they place VNFs using a markov decision process with lower costs for reusing VNFs. Qi et al. [31] use greedy search to minimize the energy and bandwidth used by the solution. To increase the speed of the search they only considered nodes that were within some number of hops from the current node. Rankothge et al. [32] use a genetic algorithm and custom operators to place VNFs, aiming to minimize link utilization and the number of servers used whilst maximizing the number of links used.

2) Unrealistic Model: Other works do consider the QoS but do so in a way that is inconsistent with real world evidence. Most commonly, these works assume that the waiting time at a network component is constant however in practice the waiting time depends on the utilization of the component [20], [21], [25].

Many works assume there are constant waiting times at all components in the network. Often, these works use greedy heuristics to solve the simplified problem. For example,



(a) Dcell$_1$ with 4 port switches [33]

(b) Fat Tree with 4 port switches [34]

(c) Leaf-Spine with 6 port switches [35]

Fig. 1: Common data center topologies

Alameddine et al. [36] proposed greedy, random and tabu heuristics to place VNFs such that tasks are completed before certain deadlines. Vizarreta et al. [37] also assumed constant waiting times but additionally fixed the starting and ending components for each service. They first found the lowest cost route between the starting and ending components that satisfied latency and robustness constraints and then modified the path until it could accommodate each VNF of the service. Qu et al. [38] used the same assumptions and a similar technique. Their algorithm first attempts to place VNFs on the shortest path between the starting and ending components of each service. If the path cannot accommodate all VNFs then nearby components are also considered.

Other non-greedy heuristics also presuppose constant waiting times. These include Hawilo et al. [39], who proposed a heuristic named BACON that places latency sensitive VNFs on components with high betweeness centralities and low latency neighbors. Manias et al. [40] later extended this work, using a decision tree to learn the outputs of BACON. More recently they extended this work to use particle swarm optimization to select the model hyperparameters [41]. Roy et al. [42] use ant colony optimization to minimize the overall latency where the ants heuristically favoured nodes with low constant waiting times. Bari et al. [43] used dynamic programming to place each service separately aiming to minimize energy and placement costs whilst considering latency constraints.

Using the same constant waiting time assumption, some researchers have combined linear programming with other methods to reduce the number of possibilities the solver must consider. Alleg et al. [44] consider a problem with branching services and used a heuristic to match servers and VNFs by their degree in the network topology and service chain respectively. The solver is then only allowed to place VNFs on those nodes in order to meet latency constraints. Luizelli et al. [15] used variable neighborhood search (VNS) to selects subsets of VNFs that can be changed whilst the others variables remain fixed. The new problem is then solved whilst meeting latency constraints using linear programming. A new subset of VNFs are selected for the next iteration and the process is repeated. Pei et al. [45] used linear programming to find the exact solution to a set of problems to form a training set and then used supervised learning to learn a model that can place VNFs.

3) Ignores Model: Some researchers recognize the relevance of accurate QoS models but only use it to evaluate the solutions discovered by a heuristic. This provides additional information to the decision maker but does not change the quality of the solutions. Chua et al. [46] place VNFs of each service in the first server that can accommodate it, but restrict the maximum capacity of each server until all available capacity has been used. They then use an queueing model to inform the end user of the expected latency. Similarly, Zhang et al. [47] propose a best fit decreasing method of placing VNFs and use an queueing model to evaluate the solution.

C. Realistic and Scalable

Finally, there are some works that meet both of the main criteria but which may be improved upon. Some works use more accurate but still imperfect models with known issues. For example, several works disregard packet loss when constructing their model. Whilst this greatly simplifies the model construction it also misrepresents the problem as packet loss impacts on service latency and energy consumption [17]. Gouareb [48] et al. use this type of model to calculate the latency, and present linear programming and heuristic solutions that minimize the latency. The heuristic greedily assigns VNFs with the largest requirements to the server with the most capacity and then scales VNFs horizontally (more instances) or vertically (more resources per VNF) to meet demand. Leivadeas et al. [49] use the same form of model with

Tabu search to jointly minimize energy and latency. Qu et al. [50] use the same model to minimize cost whilst considering latency constraints. They propose a heuristic that attempts to reallocate resources on overloaded nodes to services which are violating constraints and if necessary to migrate services so as to avoid overloaded nodes.

In our earlier work on this problem [18], we proposed a routing-led multi-objective genetic algorithm that used an accurate queueing model of a data center. This earlier work could scale to problems with up to ∼16,000 servers. On larger problems, the time and memory requirements of the algorithm are impractical.

III. Problem Formulation

In this section, we formally define the VNFPP and its constraints. First, we describe a high level overview of the problem and then provide a formal problem statement.

The aim of the VNFPP is to find a solution that maximizes the QoS of each service and minimizes the total energy consumption. A service is formed by directing traffic through VNFs in a specific order where each VNF of the service is placed in the data center. Each VNF requires a certain amount of resources, and can be assigned to any server that has sufficient resources available. The data center consists of a large number of servers with finite computational resources. These servers can communicate using the network topology, a set of switches that interconnect all servers (see Fig. 1), to provide services that require more resources than one server can provide. Hence a service can be constructed by assigning VNFs to servers with sufficient capacity and describing paths that connect them using the network topology.

We can now formally define the VNFPP. First, we define some core terminology and then we define the objectives and constraints of the problem. $\mathcal{S}$ is the set of services that must be placed and $\mathcal{V}$ is the set of all VNFs. A service is a sequence of VNFs: $s \in S$, $s = \{s_1, s_2, ..., s_n\}$. The network topology is represented as a graph $\mathcal{G} = (\mathcal{C}, \mathcal{L})$, where $\mathcal{C}$ denotes the set of data center components and $\mathcal{L}$ denotes the set of links connecting those components. A route is a sequence of data center components where R^s is the set of paths for service s, R^s_i is the ith path of the service and $R^s_{i,j}$ is the jth component of the path. Finally, $|\cdot|$ gives the cardinality of a set or a sequence.

The resource and sequencing constraints of the problem can be formalized as follows:

1) Sequential components in a route must be connected by an edge:
$$(R^s_{i,j}, R^s_{i,j+1}) \in \mathcal{L} \tag{1}$$
2) The sum resources required by the VNFs assigned to a server cannot exceed the maximum capacity of the server:
$$\sum_{v \in A_{c_{\mathsf{sr}}}} C_v < C^{\mathsf{sr}} \tag{2}$$
where $A_{c_{\mathsf{sr}}}$ is the set of VNFs v assigned to server c_{sr}, C_v is the resources required by the VNF v and C^{sr} is the total resources available on a server.
3) All VNFs must appear in the route and in the order defined by the service:
$$\pi^{R^s}_{s_i} \neq \emptyset \tag{3}$$
$$\pi^{R_i}_{s_i} < \pi^{R_i}_{s_{i+1}} \tag{4}$$
where $\pi^{R_i}_{S_i}$ is the index of the VNF s_i in route R_i.

We consider a three-objective VNFPP including two critical metrics of the QoS, latency and packet loss, and the total energy consumption. As service latency, packet loss and energy consumption can conflict [17] we formulate the VNFPP as a multi-objective optimization problem. Further, as the number of services in a data center could number in the thousands it is not practical to treat each service quality metric as a separate objective due to the curse of dimensionality. Instead we aim to minimize the aggregate latency and packet loss and the total energy consumption of the data center. Formally, these objectives are defined as:

- The *total energy consumption* $\mathbf{E}$.
- The *mean latency* of the services $\mathbf{L}$:
$$\mathbf{L} = \sum_{s \in S} W_s / |S|, \tag{5}$$
where W_s is the expected latency of the service $s \in S$.
- The *mean packet loss* of the services $\mathbf{P}$:
$$\mathbf{P} = \sum_{s \in S} \mathbb{P}^d_s / |S|, \tag{6}$$
where $\mathbb{P}^d_s$ is the packet loss probability of the service s.

The exact energy consumption and latency and packet loss of each service for a solution can be found using tools such as discrete event simulation [51]. However, VNFPP solvers rarely use exact measurements as procuring them can be time consuming. Commonly, accurate models or heuristics of the objective functions are used during the optimization process. Alternative objective functions are discussed in detail in Section V.

IV. Improved EMO Algorithm for the VNFPP

In this section we propose a new algorithm for the VNFPP that can find high quality solutions to large scale VNFPPs. There are three key components of our algorithm:

1) A novel parallel metaheuristic that efficiently searches the VNFPP solution space.
2) A genotype-phenotype solution representation that incorporates domain knowledge to improve expected solution quality.
3) A novel initialization operator that ensures a diverse initial pool of feasible solutions.

Fig. 2 provides an overview of how these components work together. In the first stage, *preprocessing*, the algorithm decomposes the multi-objective problem into subproblems and generates data structures that aid in efficient optimization. In the second stage, *optimization*, the algorithm finds solutions to each subproblem in parallel.

The remainder of this section discusses each component of the algorithm in detail. First we discuss the core optimization

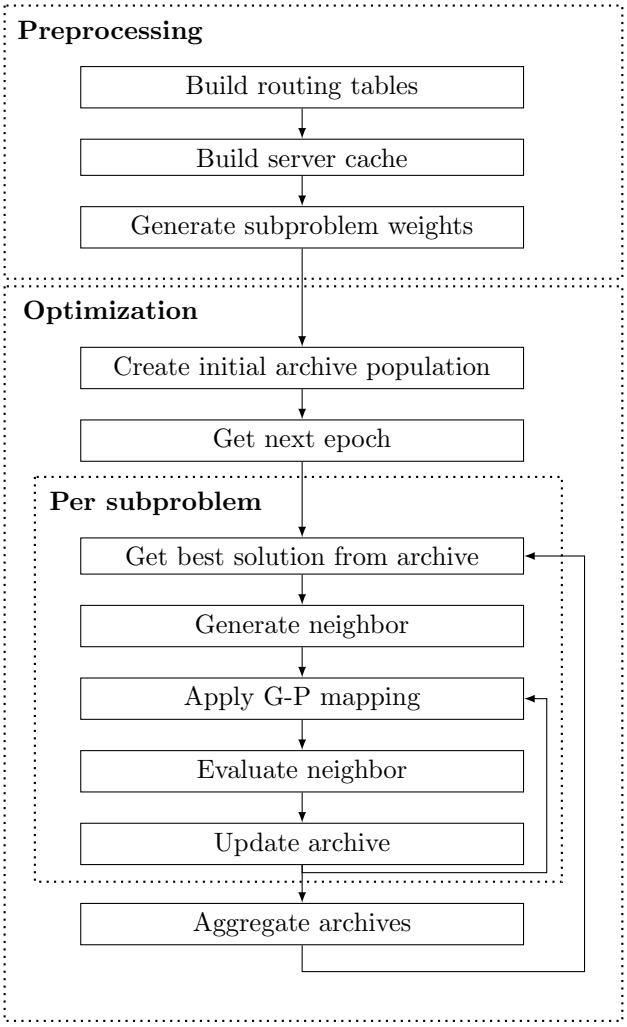


Fig. 2: A high level overview of our proposed algorithm.

Algorithm 1: Our proposed decomposition based parallel multi-objective local search algorithm.

Data: Subproblems per epoch N^s, Number of processes L, Weight vectors $W = \{W^1, ..., W^N\}$, Max evaluations T, Number of objectives M

$N^E \leftarrow \lceil |W| / N^s \rceil$ // Number of epochs
$T^W \leftarrow (T - |A|)/|W|$ // Evaluations per weight
/* Evaluate initial solutions */
$A' \leftarrow$ INITIALIZE **for** $s \in A'$ **do**
  $s_{obj} \leftarrow$ EVALUATE(s)
$A \leftarrow$ Non-dominated pop. of A'
for $e \leftarrow 0$ **to** N^E **do**
  $i_f \leftarrow e \cdot N^s$ // Index of first weight
  $i_l \leftarrow \min(i_f + N^s, |W|)$ // Index of last weight
  for each ℓ *in* $\{W_{i_f}, ...W_{i_l}\}$ **do in parallel**
    $A_\ell \leftarrow \emptyset$ // Process archive
    // Best solution found for weight
    $s \leftarrow \arg\max_{x \in A} g(x \mid W_\ell)$
    $A_\ell \leftarrow A_\ell \cup s$
    $t \leftarrow 0$ // Evaluation counter
    while $t < T^W$ **do**
      /* Get neighbouring solution */
      $s^n \leftarrow$ GENNEIGHBOUR(s)
      $s^n_{obj} \leftarrow$ EVALUATE(s^n)
      $t \leftarrow t + 1$
      /* Update process archive */
      $A_\ell \leftarrow$ Non-dominated pop. of $A_\ell \cup s^n$
      /* Accept improving solutions */
      if $g(s^n|W_\ell) < g(s|W_\ell)$ **then**
        $s \leftarrow s^n$
  $A \leftarrow$ Non-dominated pop. of $(\bigcup_\ell^L A_\ell)$

algorithm and the ways it improves upon from existing parallel metaheuristics in Section IV-A. Next we describe the tailored solution representation we use to aid the search process in Section IV-B. Finally, in Section IV-C we discuss the initialization and local search operators used in this work.

A. Novel Parallel Metaheuristic

It is well known that the Pareto front of a problem can be approximated by solving a diverse set of scalar optimization subproblems. For example, given the multi-objective optimization problem:

$$minimize\ F(x) = (f_1(x), ..., f_m(x))^\intercal \tag{7}$$

where $x \in \Omega$ and Ω is the decision space, $F : \Omega \implies R^m$ consists of m real valued objective functions. Provided the objectives in equation (7) contradict each other, no point in Ω minimizes all objectives simultaneously. However, given a weight vector $\lambda = \{\lambda^1, ...\lambda^n\}$ (where $\lambda_i \geq 0 \forall i = 1, \cdots, m$ and $\sum_{i=1}^m \lambda_i = 1$) and a utopian point ((i.e. $z_i^* = min\{f_i(x)|x \in \Omega\}$)),(i.e. $z_i^* = min\{f_i(x)|x \in \Omega\}$) the optimal solution to the Tchebycheff *scalar* optimization problem:

$$minimize\ g^{tc}(x|\lambda, z^*) = \max_{1 \leq i \leq m} \{\lambda_i |f_i(x) - z_i^*|\}, \tag{8}$$

is a Pareto optimal solution to equation (7). Hence, by varying the weight vector λ we can generate a set of subproblems for which the solution is a set of Pareto optimal solutions.

In order to solve large multi-objective problems, we can decompose the problem into a set of scalar subproblems and solve them in parallel. However, there exists a clear tradeoff between communicating information on solutions between subproblems and the work that can be performed in parallel. Decomposition based algorithms such as MOEA/D share solutions between subproblems to facilitate faster convergence but at the cost of limited capacity for parallelization. Alternatively, existing parallel decomposition algorithms such as PPLS/D [52], solve each subproblem in isolation, preventing any inter-problem communication but maximizing the parallelized work.

We propose a new algorithm which utilizes a common archive to allow the degree of communication to be determined

by a parameter. The pseudo-code of our algorithm is listed in Algorithm 1. The algorithm can be split into four key stages:

1) We decompose the problem into L subproblems and group them into 'epochs' where each epoch contains N subproblems (lines 1 - 2).
2) We generate an initial set of solutions (lines 2-3, Section IV-C1) and add the non-dominated ones to the *total archive* (lines 4 - 5).
3) For each subproblem in each epoch, we select the best solution from the total archive as a trial solution (line 14) and then use local search to search for a solution to the subproblem (line 16-25). During this search process, non-dominated solutions are added to the *subproblem archive* (line 22).
4) The subproblem archives are merged into the total archive. The total archive is used in future epochs to select better trial solutions (line 26). By varying the number of subproblems in an epoch, the frequency of communication can be altered.

Our proposed algorithm can be applied to any multi-objective optimization problem however better results for the VNFPP can be achieved with further work. There are three key factors to consider: the choice of scalar objective function, the number of epochs and constraint handling.

1) Scalar objective function: It is well known that decomposition based multi-objective evolutionary algorithms are less effective on problems with disparately scaled objective functions [53]. On large VNFPP problem instances, the disparity between objectives can be very high. A common way to resolve this issue is to normalize each objective, i.e.:

$$\bar{f}_i = \frac{f_i - z_i^*}{z_i^{nad} - z_i^*} \tag{9}$$

where $z_i^{nad} = \left(z_1^{nad}, ..., z_m^{nad}\right)^\intercal$ is the nadir point, (i.e. $z_i^{nad} = max\left\{f_i(x) | x \in \Omega\right\}$). This causes the range of each objective to be in $[0, 1]$.

It is difficult and unnecessary to compute z^{nad} and z^* in advance. In our implementation, we approximate z^{nad} and z^* before each epoch using the highest and lowest objective values found in the total non-dominated archive. For this work we use the normalized Tchebycheff scalar objective function:

$$minimize \max_{1 \leq i \leq m} \left\{ \lambda_i \left| \frac{f_i - z_i}{z_i^{nad} - z_i^*} \right| \right\}. \tag{10}$$

2) Number of epochs: The number of epochs determines how frequently information can be communicated between subproblems but also determines the amount of work that can be performed in parallel. Since fast execution time is a priority in this work, we use the maximum number of processes available in each epoch to maximize the amount of work conducted in parallel.

3) Constraint handling: Finally, whilst Algorithm 1 does not explicitly provide methods for accommodating constrained problems such as the VNFPP, it is trivial to introduce this requirement by modifying which solutions can be accepted into the archive. For this problem, we extended dominance such that feasible solutions dominate infeasible solutions. This is sufficient since the initialization operator typically generates feasible solutions that populate the archive.

Algorithm 2: A non-deterministic breadth first search algorithm.

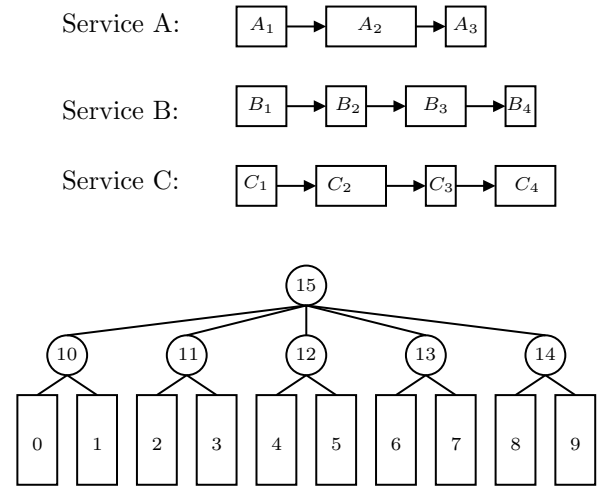
```
   Data: Initial server s, cache size n_max
   Result: Server distance cache C_s
1  C_s ← ∅
2  Q_c ← s // Current horizon
3  Q_n ← ∅ // Next horizon
4  E ← ∅ // Explored nodes
5  while Q_c ≠ ∅ do
6      u ← Get random element of Q_c
7      Q_c ← Q_c u
8      if u is a server then
9          C_s ← C_s ∪ u
10         if |C_s| == n_max then
11             return C_s
12     foreach neighbour v of u do
13         if v ∉ E then
14             Q_n ← Q_n ∪ v
15             E ← E ∪ v
16     if Q_c = ∅ then
17         Q_c ← Q_n
18         Q_n ← ∅
```

B. Genotype-Phenotype Mapping

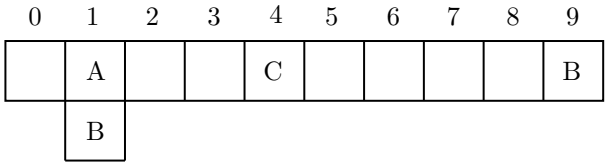
One of the key challenges when designing an EA for real-world problems is selecting the right solution representation. For complex problems such as the VNFPP, it can be beneficial to introduce domain knowledge into the solution representation to improve the expected solution quality. One way this can be performed is using a *genotype-phenotype* (G-P) solution representation. A G-P solution representation defines a 'genotype' solution representation (Ω') and a G-P mapping function to convert it into the true solution space, f:$\Omega' \implies \Omega$. Domain knowledge can be introduced into the genotype and in the G-P mapping function.

In this work we use a G-P solution representation to improve the probability of generating a feasible solution and to minimize the distance between sequential VNFs. The genotype is a list of servers where each server can contain any number of *service instances* - an indicator that a service should start from that location. Fig. 3 illustrates how the G-P mapping is performed. The mapping algorithm iterates over each service instance, and maps each VNF of the service to the nearest server that can accommodate it (Fig. 3 iii). Once all VNFs have been placed, the mapping then finds the set of shortest paths between the VNFs of the service and assigns them to the solution (Fig. 3 iv).

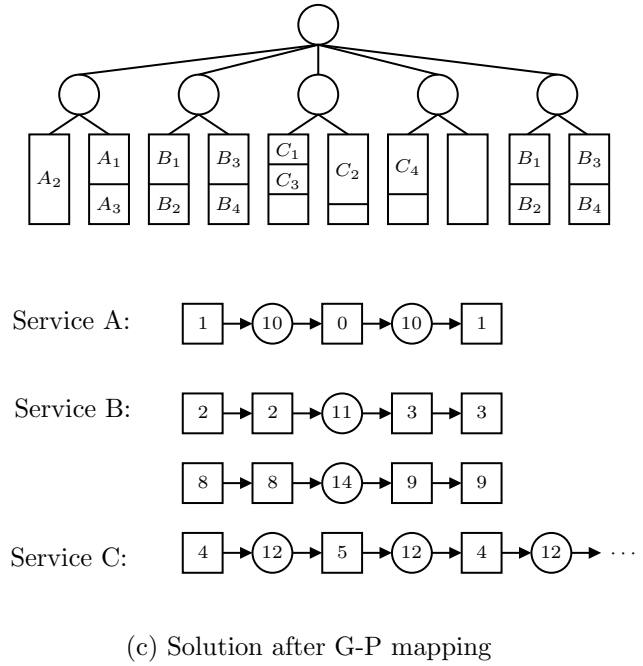
The G-P mapping function uses two preconstructed data structures to run efficiently: a set of routing tables for each network component and a set of distance tables for each server.



(a) Problem formulation



(b) Genotype solution representation



(c) Solution after G-P mapping

Fig. 3: An example of the components of the solution representation.

- *Distance tables* are used to find the next nearest server with sufficient capacity to place a VNF. Each distance tables simply list the other servers in the data center in increasing order of distance.
- *Forwarding tables* are used to find the shortest routes between two servers. Each node in the graph has a forwarding table which can be consulted to find the next steps towards a server.

Naive implementations of these constructs do not scale well to large problems since the memory required scales with the number of servers and switches [18]. In this work we propose compressed versions of both data structures for use on large problems.

1) Distance table compression: The distance tables can be constructed using breadth first search (BFS) which will encounter every other server in the data center in increasing order of distance. In a naive implementation, each distance table will store a reference to every other server, for a $O(n^2)$ total memory complexity across all distance tables. However, since the mapping procedure will only consult the table until it finds a suitable server, each table only needs to contain as many servers as are likely to be accessed.

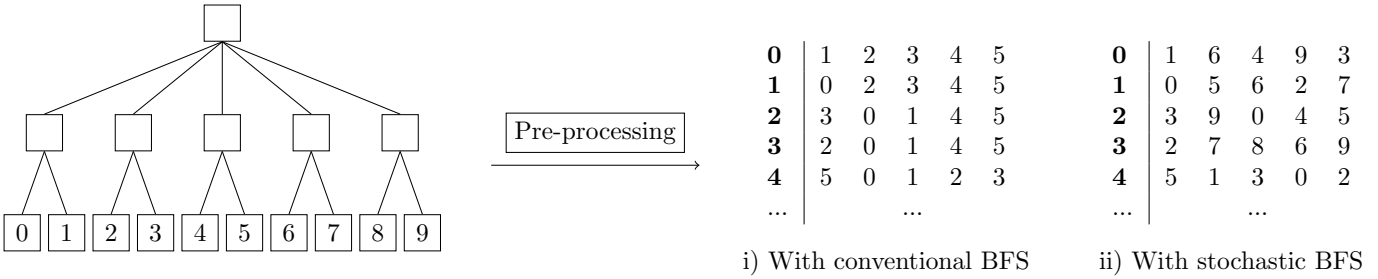
The compressed version of the table can be crafted by simply halting the search early. However, if the distance tables do not contain all servers, a 'cache miss' can occur if the mapping procedure does not find a suitable server in the distance table. As illustrated in Fig. 4, deterministic methods of identifying nearby servers such as BFS can accentuate this issue as a small number of servers will be overrepresented on average.

We resolve this issue using a non-deterministic BFS (Algorithm 2). The non-deterministic BFS modifies the well known BFS by exploring the neighbors of the current frontier in a random order. Since nodes that are encountered earlier are still explored earlier, the nearest nodes to a server are still discovered first. However, all servers which are the same distance away are equally likely to appear in the distance table.

2) Forwarding table compression: We create the forwarding tables using a modified version of the initialization process of the equal-cost multi-path (ECMP) routing protocol. In the first step, a server broadcasts a message listing its ID and the distance the message has travelled so far. Upon receiving a message, if the component has received a message for that server with a lower distance it will discard it. Otherwise it will record the origin of the message as one of the next steps to the server, increment the distance and rebroadcast the message. Once all network components have received a message from all servers, the algorithm terminates.

Due to the large number of servers and switches in a data center, it is infeasible to store uncompressed forwarding tables in memory. As data centers must support high numbers of servers each port must facilitate access to many servers. Hence, we can often aggregate several rules which list the same hops together into a single rule to save memory.

In this work, we aggregate rules that pertain to servers with sequential IDs. Each rule specifies the range of servers it encompasses and the next hop for which any server in that



i) With conventional BFS ii) With stochastic BFS

Fig. 4: The results of generating the server cache with a conventional BFS vs a stochastic BFS

range should take. We apply these compression rules for each forwarding table after each server broadcast occurs.

The memory saving from this approach depends on the network topology. In the worst case, this approach has the same worst case memory complexity as the naive implementation. In practice however, we find it results in significant memory savings (see Section VI).

C. Operators

Finally we discuss the design of the operators used in this work. Our proposed metaheuristic requires two operators: an initialization operator that provides seed solutions to the subproblems and a neighbor generation operator that generates a solution in the neighborhood of a given solution and is an integral part of the local search procedure.

1) Initialization: Good initial solutions can significantly improve the final quality of the solutions to single objective problems [54]–[56] such as the subproblems in our algorithm. Since the weights are uniformly distributed over the objective space, it is important that the archive contains a diverse population. We designed an initialization operator that produces diverse initial solutions for the first archive in order to seed the search process with good initial solutions.

To ensure the population is diverse, we determine the minimum and maximum number of service instances the data center can accommodate and then generate solutions uniformly over this space. A feasible solution has at least one instance of each VNF, hence the minimum capacity that will be used is the sum of the size of all VNFs,

$$M^{\mathsf{min}} = \sum_{v \in \mathcal{V}} C_v \tag{11}$$

Similarly, a solution cannot place more VNFs than there is capacity for, hence the maximum capacity is the total capacity of the data center,

$$MD^{\mathsf{max}} = n \cdot C^s \tag{12}$$

Next, we determine how much capacity each solution should be allocated. For a population with n individuals, we permit the ith individual to use at most $i/n\%$ of the total capacity. The number of instances of each service is determined by:

$$N_v^i = \left(M^{\mathsf{min}} \ / \ M^{\mathsf{max}} - 1 \right) \cdot \frac{i}{n} + 1. \tag{13}$$

where the whole number part determines the guaranteed number of each instance whilst the fractional part determines the probability an additional instance is placed. This ensures that expected number of service instances meets the target capacity.

Once the number of instances of each service have been calculated, the service instances are distributed uniformly at random over the initial solution.

2) Neighbor Generation: The local search procedure aims to find improving solutions by exploring the immediate neighborhood of a solution. The neighborhood of the VNFPP contains changes in the number and placement of service instances. To explore this space, our neighbor generation function has an equal probability to add or remove a service instance or to move a service instances to a new server.

V. Practical Objective Functions for Large Scale VNFPPs

Since the objective function will be used many times per run, a suitable objective function is critical. Two types of objective function are often used on the VNFPP: accurate models and heuristic models. Accurate models can find accurate estimates of the objective functions by modelling how a typical packet moves through the data center. Heuristic models instead use fast heuristics which correlate with the objective functions, e.g. a common heuristic for latency is the average path length. Heuristic models typically evaluate solutions faster than accurate models but existing proposals are ineffective on multi-objective VNFPPs [17]. In this work we evaluate models of both types in order to better understand the tradeoffs between model efficiency and effectiveness.

A. Accurate models

Accurate models of the VNFPP typically use queueing theory to model the average flow of packets through each component in the data center. A queueing model (QM) will calculate the expected number of packets arriving at each component in the data center based on some assumptions. Next, the component waiting time (W_c), packet drop probability ($\mathbb{P}_c^d$) and utilization (U_c) can be determined. These metrics can then be used to calculate the average QoS for each service.

Specifically, the average latency is the sum of the expected waiting time, calculated by:

$$W_s = \sum_{i=1}^{|R^s|} W_{R_i^s} \cdot P_{R_i^s}, \tag{14}$$

where $W_{R_i^s}$ is the average latency for the path R_i^s. It is calculated as the sum of the waiting time at each network component:

$$W_{R_i^s} = \sum_{c \in R_i^s} W_c. \tag{15}$$

Similarly, the packet loss is the probability a packet does not complete the service which is calculated with the expected packet loss over each path:

$$\mathbb{P}_s^d = \sum_{i=1}^{|R^s|} \mathbb{P}_{R_i^s}^d \cdot \mathbb{P}_{R_i^s}, \tag{16}$$

where $\mathbb{P}_{R_i^s}^d$ is the probability that a packet is dropped on the path R_i^s which is calculated as:

$$\mathbb{P}_{R_i^s}^d = 1 - \prod_{c \in R_i^s} \left(1 - \mathbb{P}_c^d\right). \tag{17}$$

Similarly, the expected energy consumption is the sum energy consumption of each component. In this work, we use a three state model of energy consumption where a component can be either `off`, `busy` or `idle`. A component is off if it will not be used, otherwise it is either busy if it is currently servicing packets or idle if it is waiting for packets to arrive. A component uses different amounts of electricity in each state. The total energy consumption of a data center is the sum of the energy consumed by all its components:

$$E_C = \sum_{c \in \mathcal{C} \setminus \mathcal{C}^{\mathsf{vm}}} U_c \cdot E^A + (1 - U_c) \cdot E^I, \tag{18}$$

where $\mathcal{C}^{\mathsf{vm}}$ is the set of VMs and U_c is the active period of the data center component c. To calculate U_c, we need to consider both single- and multiple-queue devices. The active period of a queue is given by [57]:

$$\overline{U}_c = \begin{cases} 0, & \text{if } \lambda = 0 \\ \frac{1-\rho}{1-\rho^{K+1}}, & \text{if } \lambda \leq \mu \\ \frac{1}{K+1}, & \text{otherwise} \end{cases}. \tag{19}$$

Physical switches can be modeled with a single-queue for their buffers. Hence the active period of a switch U_c is equal to the active period of its queue:

$$U_{c_{\mathsf{sw}} \in \mathcal{C}^{\mathsf{sw}}} = \overline{U}_{c_{\mathsf{sw}}}, \tag{20}$$

where $\mathcal{C}^{\mathsf{sw}}$ is the set of switches. A server has multiple buffers: one for the virtual switch and one for each VNF. The server is `idle` when no packets are being processed at any of its buffers. Thus, the utilization of a server is calculated as:

$$U_{c_{\mathsf{sr}} \in \mathcal{C}^{\mathsf{sr}}} = 1 - \left(1 - \overline{U}_{c_{\mathsf{sr}}^{\mathsf{vs}}}\right) \cdot \prod_{c_{\mathsf{v}} \in \mathcal{A}_{csr}} \left(1 - \overline{U}_{c_{\mathsf{v}}}\right), \tag{21}$$

where $\mathcal{C}^{\mathsf{sr}}$ is the set of servers and $c_{\mathsf{sr}}^{\mathsf{vs}}$ is the virtual switch of the server.

QMs can be distinguished by the assumptions that they make. For the VNFPP, most QMs will assume that traffic arrives according to a Poisson distribution and that they a packet is serviced and leaves the queue according to an exponential distribution. The key distinguishing factor is whether the model assumes bounded or unbounded length queues.

1) Unbounded Queues: Unbounded QMs assume that the queue can grow to be infinitely long. This assumption is unrealistic and exhibits two inaccuracies when used in practice. First, the QM will report that the component packet drop probability $P_c^d = 0$, irregardless of the arrival rate. Second, if the arrival rate exceeds the service rate at any component, the expected length and waiting time of the queue will tend towards infinity. As a result, any service that visits the component will have infinite latency and the solution will be infeasible. In practice, as the arrival rate approaches the service rate, the packet drop probability increases, limiting the maximum length of the queue. Despite these inaccuracies, unbounded QMs are widely used on the VNFPP [48]–[50].

The component utilization, waiting time and packet loss for an unbounded queue can be calculated using standard queueing formula [57]. The component waiting time is given by:

$$W_c = \begin{cases} \frac{1}{\mu_c - \lambda_c}, & \text{if } \lambda_c > \mu_c \\ \infty, & \text{otherwise.} \end{cases} \tag{22}$$

The component utilization is given by:

$$U_c = \frac{\lambda_c}{\mu_c}, \tag{23}$$

and the packet loss $P_c^d = 0$.

2) Bounded Queues: More realistic models acknowledge the presence of packet loss in the network. In the case of bounded queues, each queue has a finite maximum length. If a packet arrives whilst the queue is full, it is dropped and the packet is lost. Hence in a bounded QM, the traffic rate leaving the component will be less than the arrival rate. Additionally, the average queue length grows as the queue utilization increases and so the greater the arrival rate relative to the service rate, the greater the expected packet loss.

Packet loss can introduce complex interactions between components that are difficult to model. If one component has a high utilization, other components that appear later on the same service path will have a lower arrival rate as a result. If the same component is visited multiple times on the same path, the arrival rate at the component becomes dependent on its own packet loss. The majority of works that use bounded queueing models ignore these interactions and instead calculate the arrival rate at the start of the system before the interactions can occur [28], [46]. Recently, we proposed a new method which can be used to accurately calculate the arrival rate at each component [17].

Regardless of how the arrival rate is deduced, the component utilization, waiting time and packet loss can be calculated using standard queueing formula for bounded queues [57]. The component waiting time is given by:

$$W_c = \overline{N} / \hat{\lambda}_c, \tag{24}$$

where $\hat{\lambda}_c$ is the effective arrival rate for the component c, $\hat{\lambda}_c = \lambda_c \cdot \left(1 - P_c^d\right)$, and $\overline{N}$ is the expected queue length at the component c [57]:

$$\overline{N} = \begin{cases} \frac{\rho[1-(N^M+1)\rho^{N^M}+N^M\rho^{N^M+1}]}{(1-\rho)(1-\rho^{N^M+1})}, & \text{if } \lambda \neq \mu \\ N^M/2, & \text{otherwise.} \end{cases} \tag{25}$$

where $\rho = \lambda_c/\mu_c$ and N^M is the maximum queue length. The packet loss P_c^d is given by:

$$P_c^d = \begin{cases} \frac{(1-\rho)\rho^{N^M}}{1-\rho^{N^M+1}}, & \text{if } \lambda \neq \mu \\ \frac{1}{N^M+1}, & \text{otherwise.} \end{cases}, \tag{26}$$

And the component utilization, U_c is given by:

$$U_c = \frac{1-\rho_c}{1-(\rho_c)^{N^M}} \tag{27}$$

where $\rho_c = \lambda_c/\mu_c$.

B. Heuristic models

Heuristic models do not calculate the objectives directly but instead optimize metrics about the solution which are expected to correlate with the objectives. Whilst they are often used as a fast replacement for accurate models, existing heuristic models do not capture the tradeoffs between the objectives, resulting in a loss of diversity on multi-objective VNFPPs [17]. In this paper, we propose a new heuristic model which uses the ratio between the arrival rate and service rate, $\rho_c = \lambda_c/\mu_c$, as a heuristic for the latency and packet loss.

Intuitively, each objective is minimized by reducing the queue length, which depends on the ratio ρ_c. The conflict between objectives occurs because adding more VNFs will distribute traffic over more components, reducing the arrival rate, but also increasing the number of components that may be idle. Hence whilst adding additional VNFs will always reduce the latency and packet loss, it may increase the total energy consumption.

Using this understanding, we propose a heuristic model that uses the average value of ρ_c of components on each path as a heuristic for the latency and packet loss objectives. To calculate the objectives, we first determine the arrival rate using an unbounded queueing model. Although this model is not accurate, the arrival rate at each component in this model will correlate with the true arrival rate. For the heuristic, the first objective is to minimize the average service utilization, where the service utilization for each service U_s is the expected utilization over the paths:

$$U_s = \sum_{i=1}^{|R^s|} U_{R_i^s} \cdot P_{R_i^s}, \tag{28}$$

and the service utilization $U_{R_i^s}$ is calculated with:

$$U_{R_i^s} = \sum_{c \in R_i^s} \rho_c. \tag{29}$$

As the energy consumption is a conflicting objective, we also require the optimization algorithm to minimize the total energy consumption using equation (18).

Finally, other heuristic functions have also been proposed in the literature:

- Constant waiting time or packet loss (CWTPL): As in [37] and [39], this model assumes that the waiting time at each data center component is a constant. In addition, we also keeps the packet loss probability at each component as a constant. Based on these assumptions, we can evaluate the latency and packet loss for each service and use the accurate bounded model to determine the the energy consumption. All these constitute a three-objective problem that aims to minimize the average latency, packet loss and total energy consumption.
- Resource utilization (RU): As in [30] and [31], this model assumes that the waiting time is a function of the CPU demand and the CPU capacity of each VM. In addition, the demand is assumed to determine the packet loss probability as well. Based on these assumptions, we evaluate the latency for each service and apply the accurate bounded model to determine the the energy consumption. All these constitute a two-objective problem that aims to minimize the average latency (and by extension the packet loss) and the total energy consumption.
- Path length and used servers (PLUS): This model uses the percentage of used servers to measure the energy consumption (e.g., [32], [58], [59]) and the length of routes for each service as a measure of service latency, packet loss or quality (e.g., [15], [44]). All these constitute a two-objective problem that aims to minimize the path length and the number of used servers.

VI. Empirical Study

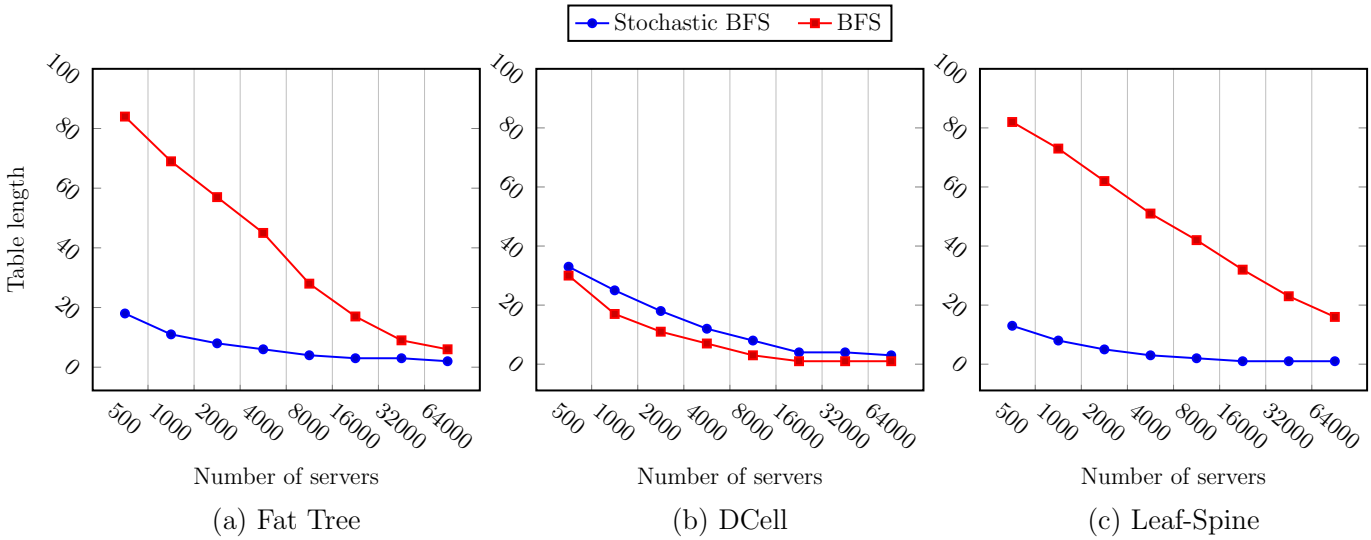
We seek to answer the following five research questions (RQs) through our experimental evaluation.

- **RQ1**: Are our proposed data structures practical for large scale VNFPPs?
- **RQ2**: Does our new initialization operator improve upon our previous operator?
- **RQ3**: How does our proposed metaheuristic compare against other parallel and synchronous MOEAs?
- **RQ4**: Does our proposed heuristic objective function improve on existing objective functions?
- **RQ5**: What are the largest problem instances we can consider and what are the current limiting factors?

For each research question, we consider three common network topologies: DCell [33], Fat Tree [34], and Leaf-Spine [35], illustrated in Fig. 1. Where applicable, we calculate the HV of populations of solutions to judge the convergence and diversity of the population. Due to the disparity in the objectives, in each of these tests we first normalize the objective values of all solutions as described in the appendix. The parameter settings for each algorithm are based on the respective authors' recommendations and are also listed in the appendix.

A. Distance Table Compression Evaluation

1) Methods: We first aim to determine the minimum practical size of the distance tables. We use monte-carlo methods to find a minimum acceptable size of the distance tables N^T. Smaller settings of N^T will grant larger memory savings but also reduce the probability of finding a feasible placement for all VNFs of all services in a solution. We defined an acceptable setting of N^T as one which will place all service instances for 99.9% of a diverse set of solutions where each



(a) Fat Tree (b) DCell (c) Leaf-Spine

Fig. 5: The minimum setting of N^T that was required to reach 99.9% reliability.

TABLE I: Category and actual number of servers for each network topology

Number of servers	Topology		
	Fat Tree	Leaf-Spine	DCell
500	423	512	420
1000	1024	968	930
2000	2000	2048	1806
4000	3456	4050	3192
8000	8192	7938	8190
16000	16000	15842	17556
32000	35152	31752	33306
64000	65536	64082	57840

solution requires at least 90% data center utilization. Since it is unlikely that a data center will reach this level of utilization, this ensures that the parameter setting is robust against all but the most extreme situations. Similarly, the 99.9% threshold ensures that on average most initial solutions in a population will be feasible, reducing the risk of a bias towards solutions with low numbers of VNFs.

We considered three topologies at increasingly large scales. Due to the structure of each network topology, it is not possible for all topologies to have the same number of servers at each scale. The exact number of servers for each scale are listed in Table I.

For each data center topology and scale, we defined 100 problem instances and generated 1000 solutions for each problem. For both the stochastic BFS and the deterministic BFS, we found the minimum acceptable setting of N^T in the range $0, 1, \cdots, 100$ (i.e. for $99900/100000$ solutions, all service instances must be placed). Note that lower settings of N^T directly correlate with lower memory consumption e.g., $N^T = 0.2$ indicates an 80% memory saving compared to the naive method.

2) Results: Fig. 5 shows the minimum acceptable setting of α for each method and for each topology at each scale. These figures show that the memory requirements of the distance tables can be greatly reduced by utilizing a low setting of N^T.

Notably, lower settings of N^T can be tolerated on larger graphs. On the smallest topologies, we found that at least 70% of the information contained in the naive routing tables was not required. On the largest topologies, typically 99% of the information is not required. Further, we find that across topologies of each scale, the memory requirements remain fairly constant. This suggests that the *relative percentage* of servers in each distance table is less important than the *number* of servers. It is plausible that there exists a constant value for the size of each distance table for an acceptable feasible solution probability. Given the memory requirements of each distance table can be made satisfactorily small with the existing approach, we leave further exploration of this outcome to future work.

Additionally, we found that our proposed stochastic BFS typically allowed for lower settings of N^T than the traditional deterministic BFS. The DCell topology is a special case where the deterministic BFS outperformed the stochastic BFS, however we note that this does not indicate that the deterministic BFS is preferable. Our implementation of the deterministic BFS first explores the servers which reside under a separate switch. Since the DCell topology is symmetrical, this ensures that each server appears the same number of times, provided that a small enough setting of N^T is used. If a larger size were used, or if the deterministic BFS were instead to explore the local switch first, the non-deterministic BFS would be preferable since in the deterministic BFS a small number of servers would be overrepresented. In this way, the deterministic BFS is only preferable for specific instances that can only be guaranteed to occur when the topology is known. Hence it is not appropriate for arbitrary graphs.

In partial response to RQ1, the effectiveness of our proposed

TABLE II: The percentage of memory saved by using our compressed routing table.

Number of servers	Topology		
	Fat Tree	Leaf-Spine	DCell
500	98.38	98.63	24.68
1000	99.28	99.26	31.00
2000	99.61	99.65	36.53
4000	99.77	99.82	41.26
8000	99.90	99.91	48.84
16000	99.95	99.95	54.67
32000	99.98	99.98	59.31
64000	99.99	99.99	63.10

distance table compression algorithm enables us to consider far larger problem instances of all topologies.

B. Routing Table Compression Evaluation

1) Methods: Without compression, the routing tables required for the mapping process can require impractical amounts of memory. To determine whether compression can reduce the memory requirements to a practical level, we constructed routing tables with and without compression for each topology and at different scales.

2) Results: Fig. 6 shows the amount of memory that the routing tables of each topology required with and without compression. The figure shows how the uncompressed routing tables are impractical for large data centers and also that our proposed compression technique uses significantly less memory than the naive approach.

The routing table compression was most effective on hierarchical data centers, e.g. the Fat Tree and Leaf-Spine topologies. We found that hierarchical topologies are particularly amenable to compression. When all servers communicate with other servers via a switch, all paths for a server can be compressed into 2 or 3 rows (the servers with IDs less than the current server, the current server, and the servers with IDs after the current server). Similarly, the hierarchy of the topology ensures that each port of a switch will allow access to either a single server or a cluster of sequential servers which can be compressed into one row. As a consequence, the compressed routing tables used $\sim$3 orders of magnitude less memory than the uncompressed routing tables.

The compressed DCell routing tables also required significantly less memory, specifically 44.92% less memory on average (see Table II), but still required on the order of 10 - 100 GBs of memory to store the larger topologies. This is because of two related factors. First, the DCell topology uses direct connections between servers. As a consequence, each server requires at least one row for each port. Further, the direct server connections mean that although many servers may be accessed via the same component, the IDs of these servers are only contiguous for short stretches, and hence our compression is less effective. Despite these limitations, the compressed DCell topology still uses significantly fewer rows. Second, each row in our compressed formulation requires 50% more memory. Specifically, the naive implementation must store one ID for the destination server and one ID for the next physical component. The compressed implementation must store an additional ID since it requires two IDs to denote the start and end destination. The combination of these two factors causes the compression to be less effective on the DCell topology.

TABLE III: Median wall clock time (s) as a multiple of the median wall clock time of our proposed algorithm averaged over all topologies.

Algorithm	Number of servers					
	500	1000	2000	4000	8000	16000
NSGA-II	5.00	4.38	4.05	3.80	3.32	3.13
IBEA	4.44	4.03	3.82	3.59	3.19	3.01
MOEA/D	6.81	6.08	5.65	5.35	4.71	4.44
PPLS/D	0.94	0.95	0.94	0.96	0.97	0.99

To conclude our response to RQ1, it is clear that the proposed routing table compression algorithm enables us to consider larger problem instances. However, it is disproportionately more effective on hierarchical topologies.

C. Comparison of Alternative Initialization Operators

1) Methods: To answer RQ2, we compare the effectiveness of our proposed improved initialization operator against an initialization operator we proposed in earlier work [17]. To evaluate each operator we generated 30 problem instances for each topology and for each data center scale. For each problem we calculated the HV of the initial population.

2) Results: It is clear from Fig. 7 and Fig. 8, that our proposed solution produces more evenly distributed individuals than the prior initialization operator. These figures also make it evident that the significantly improved diversity of our improved initialization operator has not come at the cost of solution quality. The prior initialization operator tended to cluster solutions in specific regions since it could only produce solutions where each service instance had the same number of instances. In contrast, our improved initialization operator can vary the numbers of service instances in order to reach a target capacity. This allows for more variance in the energy consumption and average latency and packet loss across solutions.

D. Comparison with Other Approaches

1) Methods: To answer RQ3, we compare the effectiveness of our proposed algorithm with four state-of-the-art peer MOEAs. Specifically, we compare our algorithm against a similar parallel metaheuristic, PPLS/D [52], and three important MOEA algorithms (NSGA-II [60], IBEA [61], MOEA/D [53]).

PPLS/D is a decomposition algorithm which uses local search to solve each subproblem in parallel. However, unlike our proposed algorithm, PPLS/D requires one thread per subproblem and hence its effectiveness as an optimization algorithm is dependent on the hardware it is executed on.

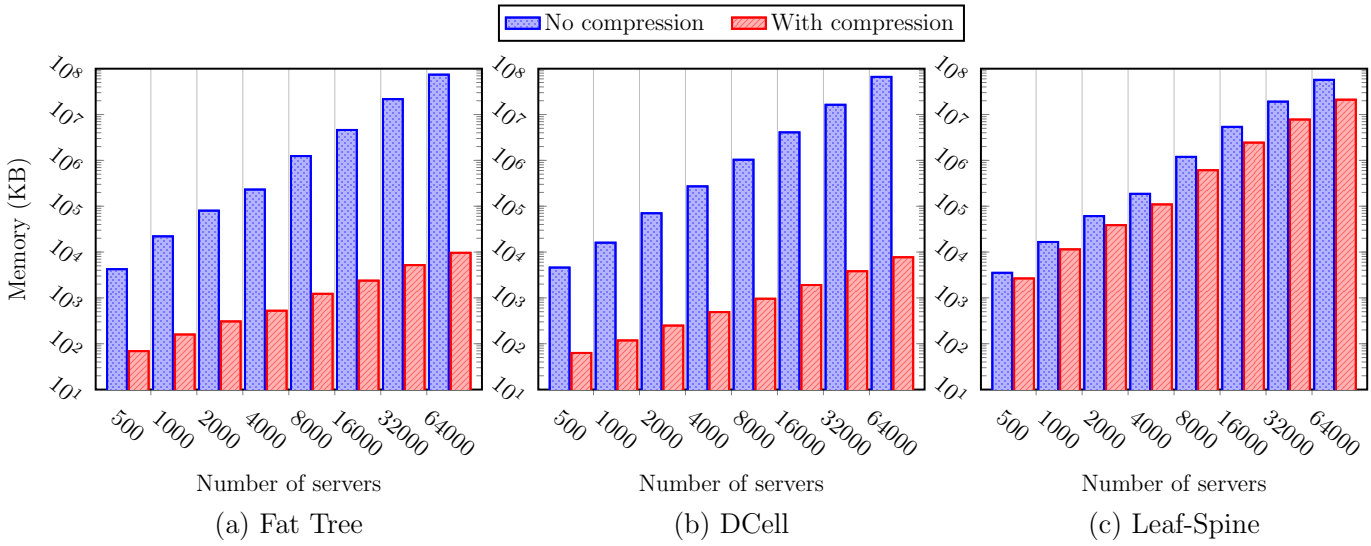


Fig. 6: The comparative memory consumption of the routing tables with and without our proposed compression strategy.

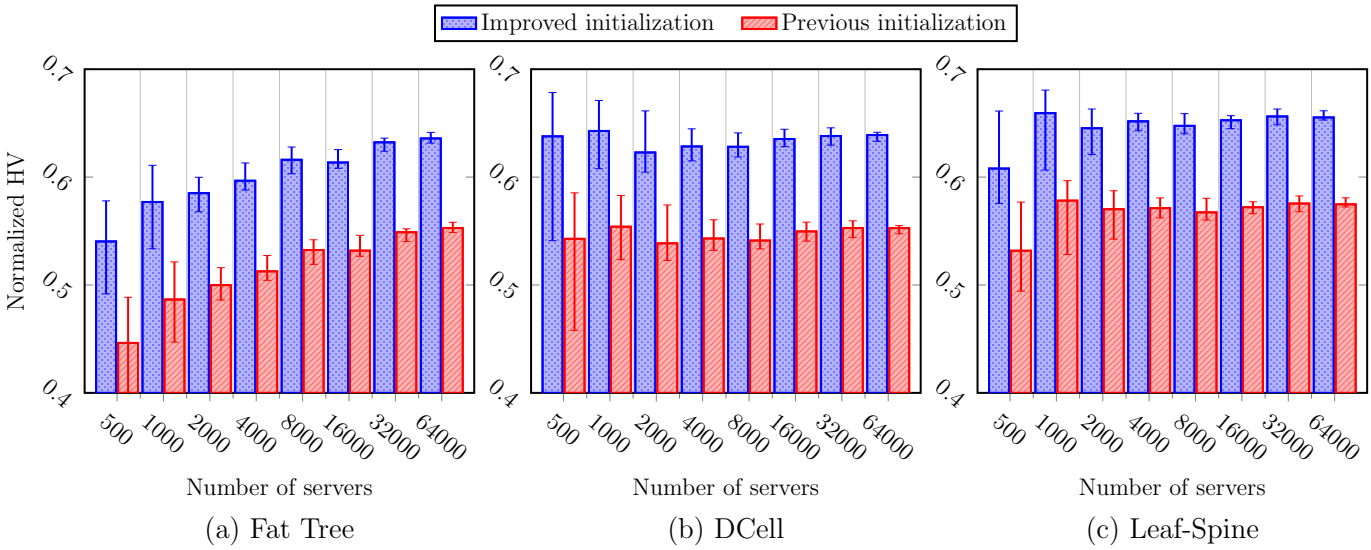


Fig. 7: Median and quartile plots of the hypervolume of the initial population using each operator.

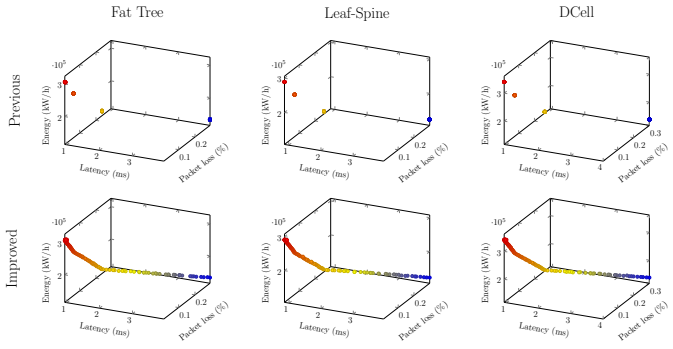


Fig. 8: An illustrative example of the non-dominated archives of each initialization operator on a problem instance with 16,000 servers.

PPLS/D maintains a population of unexplored solutions and an archive of non-dominated solutions for each subproblem. First, an initial solution is added to the unexplored population of each subproblem. Next, for each subproblem the algorithm selects the best solution from the unexplored population and generates a set of neighboring solutions. A neighboring solution is added to the unexplored population and the non-dominated archive if it is: 1) closer to the subproblem that generated it than any other subproblem, as measured by the angle between the solution and the subproblems and 2) is either a better solution to the subproblem than the current best solution or if it is not dominated by any solutions in the non-dominated set of the subproblem. Once the stopping condition is met, the algorithm aggregates the non-dominated archives from each subproblem and returns the overall non-dominated solutions.

PPLS/D was designed for unconstrained optimization problems, so require modification to fit the VNFPP. In our vari-

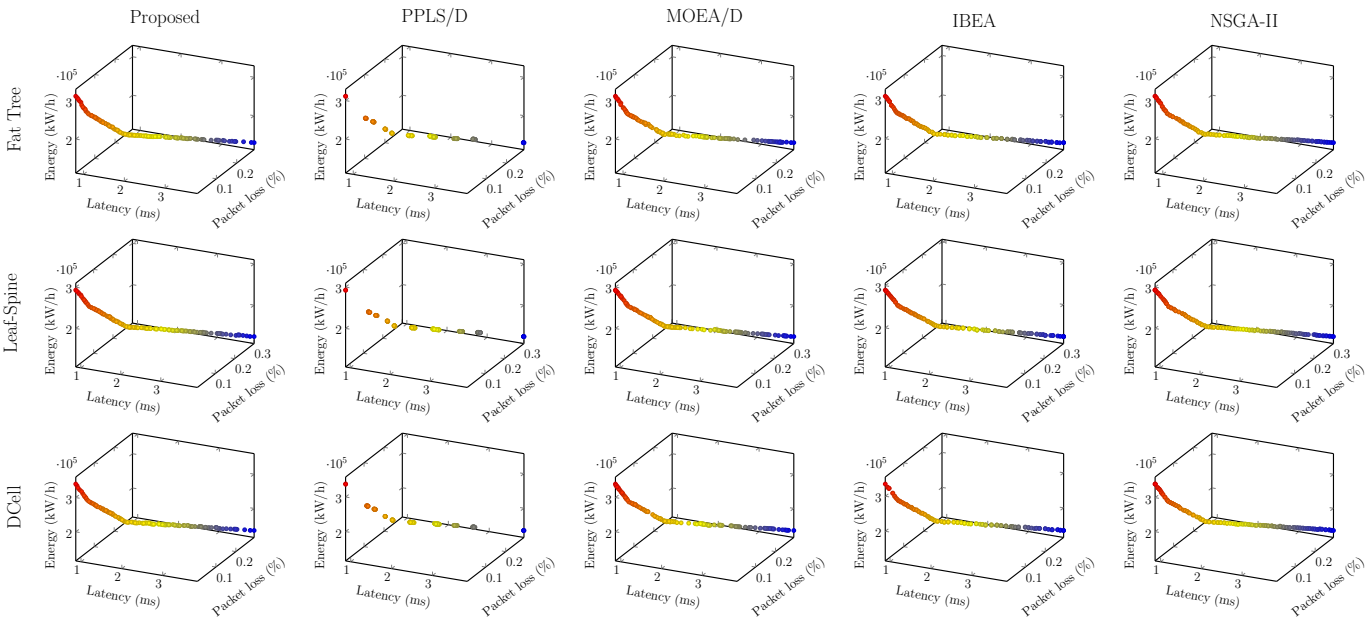
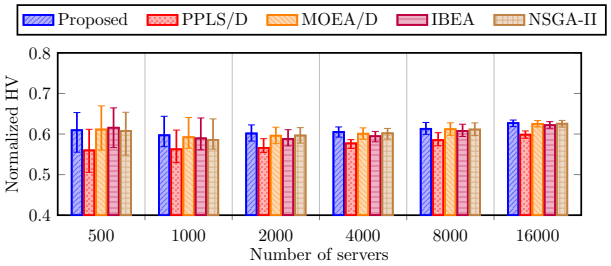
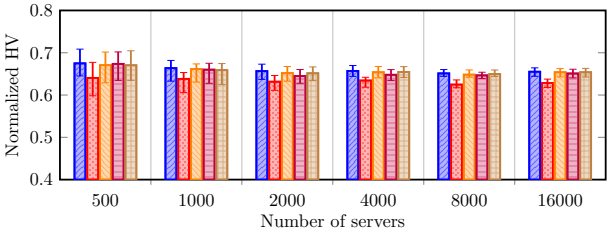


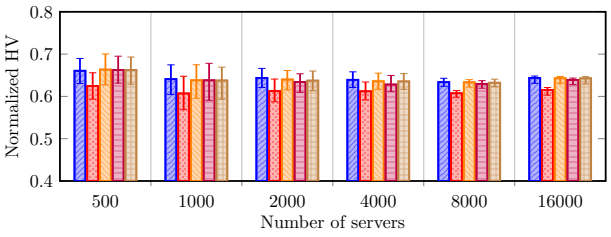
Fig. 9: An illustrative example of the non-dominated archives of each algorithm on a problem instance with 16,000 servers.



(a) Fat Tree



(b) Leaf-Spine



(c) DCell

Fig. 10: Median and quartile plots of the hypervolume of each algorithm.

ant, an infeasible solution is accepted into the unexplored population only if there is not a more feasible solutions in the population. The original implementation of PPLS/D also explores all neighbors of each solution in the final population. Given the very large neighborhoods present in the VNFPP, this step was removed.

To evaluate each algorithm we generated 30 problem instances for each topology and for several data center scales. Since NSGA-II and IBEA do not utilize an external population, they can only discover a fixed number of solutions. To ensure that the other algorithms do not have an unfair advantage, we find a subset of solutions from each archive the same size as the population size of IBEA and NSGA-II. A common method of reducing the size of a non-dominated archive is to use a clustering algorithm and then to select a representative solution for each cluster [62]. In this work we use k-means clustering [63] to generate the same number of clusters as the population size of IBEA and NSGA-II, and then select the nearest non-dominated solution to the centroid of each cluster from the archive population.

For each problem we recorded the wall-clock execution time and the HV of the final population or subset. We allowed each algorithm to run until they had performed 12,000 evaluations. All tests were run on a 8 core/16 thread CPU at 2.6GHz. All parallel algorithms were implemented using the parallelization library Rayon[1].

2) Results: A key benefit of parallelizing is its potential to reduce the execution time. Table III shows that the parallel algorithms require multiple times less execution time than comparable sequential metaheuristics. Specifically, our algorithm has a median wall clock time between 3.01 - 6.81× faster than IBEA, NSGA-II and MOEA/D. In this regard, our proposed algorithm significantly out performs state of the art, serial MOEAs.

Further, Fig. 10 shows that the fast execution time of our proposed algorithm does not come at the cost of solution qual-

[1]https://github.com/rayon-rs/rayon

ity. From Fig. 9, we can see that all algorithms tended to find similar populations of solutions. In this regard our proposed algorithm performs as well as other state of the art MOEAs on the VNFPP. Critically, the high quality solutions of most algorithms appear to be a consequence of the initialization operator. It is clear from Fig. 8 and Fig. 10 that the metaheuristics were only able to make small improvements over the solutions found by the improved initialization operator. As a result, any algorithm that maintains a diverse population of solutions will be fairly effective.

In contrast, PPLS/D suffers from two issues related to population diversity. First, the algorithm has specific rules about which solutions can be included in the total archive which causes it to disregard much of the initial population. Second, we can see from Fig. 9 that the algorithm is not effectively exploring solutions away from the subproblems and hence cannot recover the information it has lost. This hints at a potential drawback of the local search approach: since the neighborhood of each solution is very large, it is only possible to search a small region around the initial solution using local search. Since the number of subproblems in PPLS/D is limited by the number of available threads, PPLS/D only explores a small region of the search space and hence represents a small region of the Pareto front. As our algorithm considers initial solutions in the initial archive and, as is it is not limited by specific properties of the hardware it is executed on, it does not encounter these issues.

Overall, it is clear that our algorithm is more efficient than the alternative serial MOEAs considered, and ensures higher quality solutions than a comparable parallel MOEA. However, it is not clear that the philosophy behind our proposed algorithm (i.e. favouring local search over global search) is beneficial for the VNFPP.

E. Efficient Alternative Models

1) Methods: An efficient and effective model that aligns with our objective functions is critical for the overall performance of our algorithm. To answer RQ4, we compare the performance of our proposed utilization model against each of the other objective functions introduced in Section V (i.e. accurate, M/M/1/K, M/M/1, PLUS, RU, and CNST). Due to the efficiency and effectiveness of our proposed algorithm, we use it as the optimization algorithm for each test.

To evaluate each model we generated 30 problem instances for each topology and for several data center scales. We used our proposed algorithm to solve each problem for each model. We allowed the algorithm to run until it had performed 12,000 evaluations. For each problem we recorded the wall-clock execution time and calculated the HV of the final population.

2) Results: Fig. 12 shows that our proposed heuristic underperforms in comparison to both accurate models in terms of the HV, but it significantly outperforms all alternative heuristics. This indicates that our heuristic captures *most* of the information on the tradeoffs between objectives. The discrepancy is likely because the utilization of a component changes *linearly* with changes to the arrival rate, whilst the latency and packet loss change *non-linearly*. As a result, the utilization heuristic would consider two possible paths with the same sum utilization as equal, even if one path utilizes an overloaded component with a very high arrival rate and hence high latency and packet loss. The larger discrepancy on small problems reflects the optimization algorithms ability to identify and rectify this situation easier when there are fewer variables to consider.

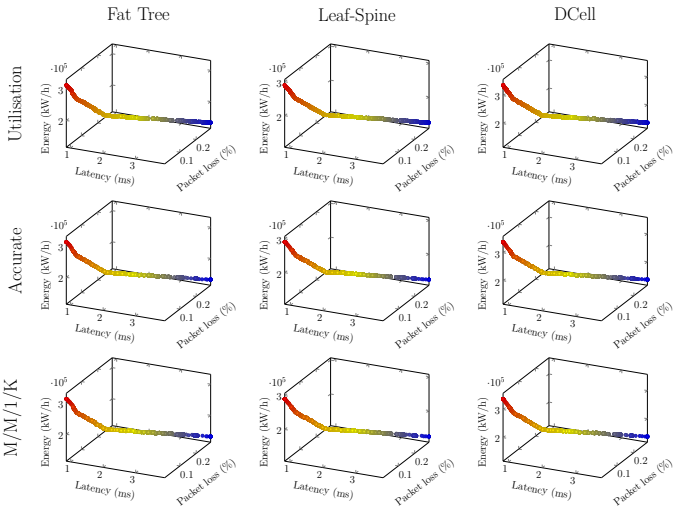


Fig. 11: An illustrative example of the non-dominated solutions found by our proposed heuristic and both accurate models for problems with 16,000 servers.

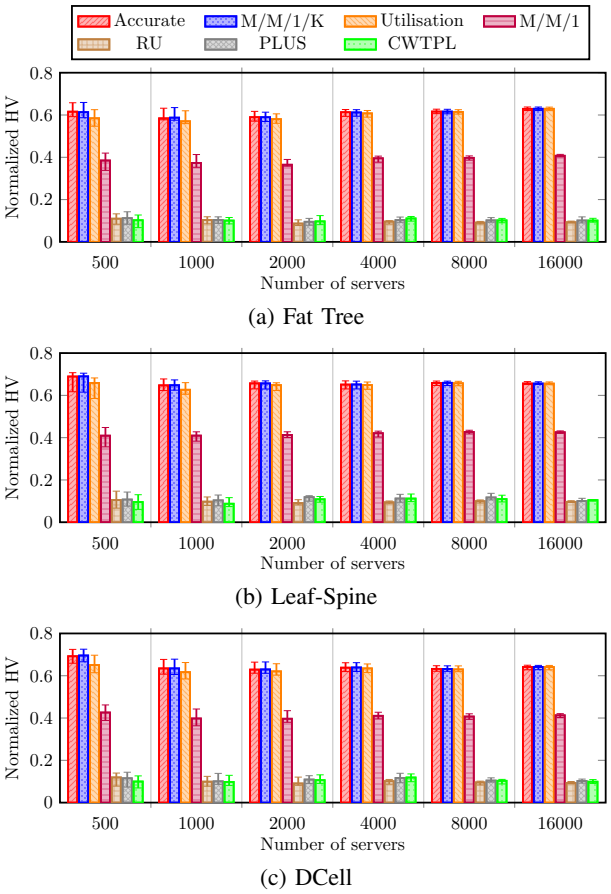
Despite this, Table IV shows that the utilization heuristic is typically orders of magnitude faster than both accurate models and also outperforms some existing heuristic approaches. Specifically, the mean wall clock execution time of our heuristic is 53.33 - 4.37$\times$ faster than the accurate model and 20 - 2.23$\times$ faster than the M/M/1/K model. Notably, these results are significantly affected by the DCell topology where the execution times are closer.

The superior performance of our proposed heuristic in terms of execution time is due to multiple factors. Principally, the heuristic leads to fewer operations overall. Clearly, the heuristic requires fewer operations than more accurate models which will directly lead to performance improvements. Indirectly, since the heuristic forms a two objective problem, it is faster to determine whether one solution dominates another. Further, since the objective space is smaller, the non-dominated archive will contain fewer solutions on average, accelerating the process of calculating the non-dominated set.

Overall, the utilization heuristic presents a viable trade off between wall clock execution time and solution quality. Whilst some of the other heuristic approaches require less time still, the impact on solution quality and diversity is too severe. The proposed heuristic is likely better suited for larger problem instances where the execution time of more accurate models could be prohibitively time consuming.

F. Applications to Large Data Centers

1) Methods: Finally, to fully understand the scalability of our algorithm, we combined the most effective components as



(a) Fat Tree

(b) Leaf-Spine

(c) DCell

Fig. 12: The median and quartiles of the normalized HV using our proposed algorithm and each model.

TABLE IV: Median wall clock time (s) as a multiple of the median wall clock time of our proposed model averaged over all topologies.

Model	Number of servers					
	500	1000	2000	4000	8000	16000
Accurate	53.33	13.33	12.50	8.88	6.08	4.37
M/M/1/K	20.00	5.33	5.07	3.80	2.80	2.23
M/M/1	1.00	0.40	0.57	0.65	0.59	0.52
RU	10.00	2.00	2.00	1.45	1.04	0.79
PLUS	1.00	0.10	0.04	0.20	0.21	0.25
CWTPL	10.00	2.67	2.43	1.60	1.13	0.85

measured in previous tests and applied the resultant algorithm to increasingly large problem instances. Specifically, we integrated the utilization model, with our proposed operators and compression techniques into our proposed algorithm. Our aim is to identify which, if any, of these components prevent the algorithm from being applied to arbitrarily large VNFPPs.

We generated 30 problem instances for each topology and for increasingly large data center scales. Since solving a large problem instance can become time consuming, each data center scale contains approximately twice as many servers as the previous scale (i.e. 500, 1000, 2000, . . .). We allowed the algorithm to run until it had performed 12,000 evaluations. We terminated testing when it was no longer practical to solve problem instances. For each problem we recorded the wall-clock execution time and calculated the HV of the final population.

2) Results: Fig. 13 demonstrates the high scalability of our algorithm. Specifically, our proposed algorithm can solve problems with up to 64,000 servers. We found that the routing table construction step prevented our algorithm from scaling to larger scale problems. The BFS based routing table construction algorithm has a low theoretical quadratic time complexity that scales with the number of components. In practice, a quadratic time complexity is unsuitable for very large topologies with hundreds of thousands of physical components. By way of illustration, we applied the routing construction algorithm to the first 1/10th of the servers in a DCell network topology with ∼128,000 servers. The construction of these routing tables took ∼124 hours (∼5 days). By extrapolation, we can calculate the full construction of the routing tables would require at least 7 weeks to complete. Notably, unless otherwise resolved, this limitation applies to other optimization algorithms, including heuristics, that place VNFs in arbitrary networks.

As Fig. 14 shows, the other components of the algorithm demonstrated promising results on large scale topologies. Once preprocessing was complete, the algorithm execution time required on the order of seconds to minutes for 500-32000 servers. The largest data center topologies had a median time complexity of between ∼27 minutes (Leaf-Spine, 1631 seconds) and ∼70 minutes (DCell, 4225 seconds).

It is interesting to note that the average execution time of the algorithm is significantly higher on the DCell topology. This could be explained by differences in the availability of cached resources. Whilst the data center is small, a larger proportion of the required memory can be held in a cache. However, on larger problems, *cache prediction* will become more important. Cache prediction algorithms aim to retrieve data before it is requested. One assumption in cache prediction is the principle of spatial locality: the assumption that future memory accesses are likely to occur near to recent ones [64]. Since the DCell network topology allows servers with distant IDs to communicate, this assumption will be violated, and memory accesses are less likely to use a cache. In contrast, on the Fat Tree and Leaf-Spine topologies, nearby servers have similar IDs and hence are stored nearby in memory.

Fig. 13 also shows that the initialization operator produces populations that are competitive with those found by our metaheuristic and hence could function as a standalone heuristic. It is clear from the results of earlier tests that the initialization operator is very effective as it is considers both the number of instances (through the improved initialization operator) and their placement (through our genotype-phenotype solution representation). Overall, our metaheuristic produces populations with a higher median HV, but the overall results are not significantly better than the initial population ($p < 0.05$) in any test case. Further, the initialization operator is a simpler algorithm, has a low time complexity and does not need to evaluate solutions. As a result, it is significantly faster on all problem instances. Despite this, since the initialization operator relies

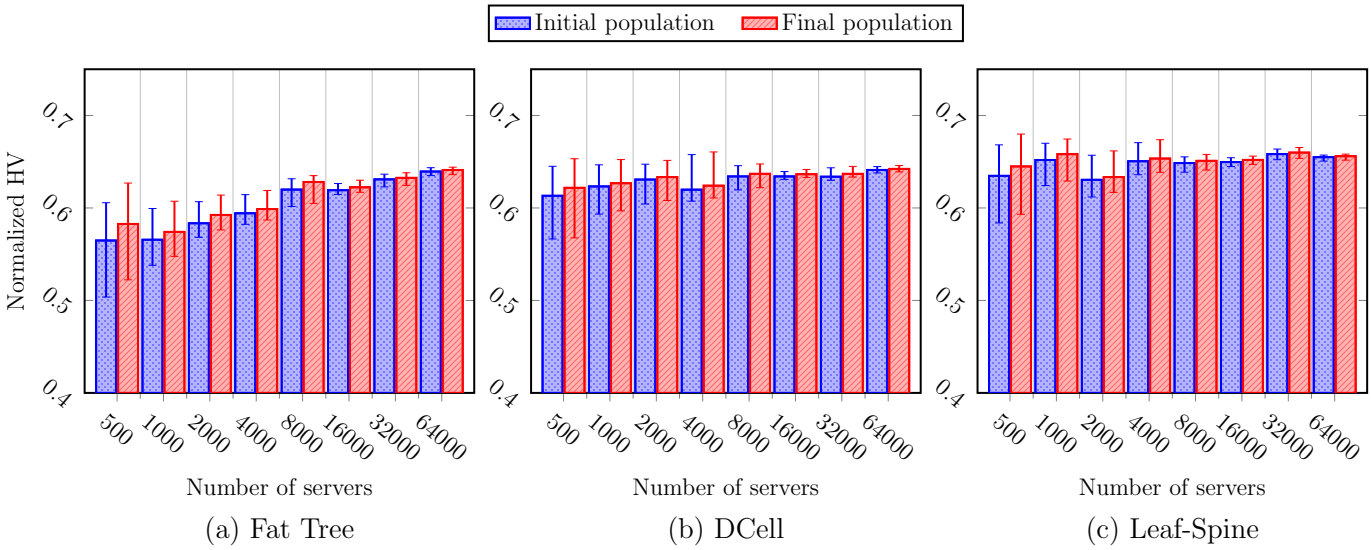


Fig. 13: Hypervolume values for each topology and scale

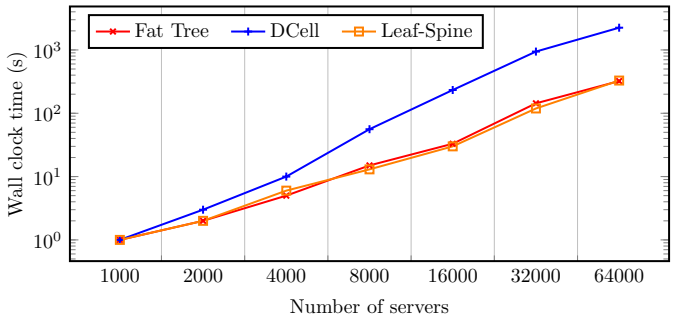


Fig. 14: Wall clock execution time for each topology and scale

on the genotype-phenotype solution representation, it cannot scale to solve larger problem instances than our proposed algorithm.

The good relative performance of the initialization operator can be partly explained by the difficulty of very large problem instances. The largest problem instances in our tests contain on the order of 6000 services. Since the objective functions calculate the mean QoS across all services, a visible improvement to the objective functions requires improving the QoS of many services. However, the optimization algorithm is permitted only 12,000 evaluations to ensure it finishes in an acceptable time. As a result there is little opportunity to find improving solutions for many services. Irregardless, the effectiveness of the initialization operator should not be overlooked as it presents a viable alternative to more complex metaheuristic or exact approaches on very large problem instances.

VII. Conclusions

In this paper we proposed a new MOEA for arbitrary graphs capable of efficiently solving large scale instances of the VNFPP. To this end we proposed:

- A novel solution representation for large scale, arbitrary network topologies.
- A novel local search based, parallel MOEA that significantly reduces the total algorithm execution time whilst achieving greater solution diversity.
- An improved initialization algorithm that ensures high quality, diverse solutions.
- A novel heuristic model that captures sufficient information about the VNFPP whilst greatly reducing the evaluation time.

We evaluated the effectiveness of each of these components and found they made significant improvements over state-of-the-art alternatives. By synthesizing these components we demonstrated that our algorithm could solve problem instances with up to 64,000 servers, a 16× improvement over problems that have been considered in the literature so far. Further, we found that the improved initialization operator combined with the genotype-phenotype solution representation is highly effective on very large problem instances.

Further extensions could be considered in future work.

- Our proposed operators combined produce an effective heuristic for the multi-objective VNFPP. It would be interesting to apply the same principles to the single-objective VNFPP which aim to minimize a cost metric and treat QoS requirements as a constraint.
- A challenging problem that remains unresolved is *service resilience* i.e. the capacity for the data center to continue to provide a service when components may fail. This is an instance of a well known NP-Hard problem, graph partitioning, hence approximation algorithms and heuristics will likely be required.
- Finally, future work could extend our metaheuristic into a dynamic optimization problem such as in [65]. Metaheuristics are well suited to dynamic optimization problems and many algorithmic frameworks have been proposed [66]. This would enable our algorithm to adapt to changing requirements and to present the best set of options at any given moment.

ACKNOWLEDGMENT

K. Li was supported by UKRI Future Leaders Fellowship (Grant No. MR/S017062/1) and Royal Society (Grant No. IEC/NSFC/170243). J. Billingsley was supported by EPSRC Industrial CASE and British Telecom (Grant No. 16000177).

REFERENCES

[1] OECD, "New skills for the digital economy," 2016.

[2] Z. Xiang, V. P. Magnini, and D. R. Fesenmaier, "Information technology and consumer behavior in travel and tourism: Insights from travel planning using the internet," *Journal of Retailing and Consumer Services*, vol. 22, pp. 244–249.

[3] J. A. Bargh and K. Y. McKenna, "The internet and social life," *Annual Review Psychology*, vol. 55, pp. 573–590, 2004.

[4] H. Farrell, "The consequences of the internet for politics," *Annual review of political science*, vol. 15, 2012.

[5] A. Chadwick and C. May, "Interaction between states and citizens in the age of the internet:"e-government" in the united states, britain, and the european union," *Governance*, vol. 16, no. 2, pp. 271–300, 2003.

[6] M. Castells, "The impact of the internet on society: a global perspective," *Ch@nge. 19 Key Essays on How the Internet is Changing our Lives*, pp. 132–133, 2014.

[7] A. Andrae and T. Edler, "On global electricity usage of communication technology: trends to 2030," *Challenges*, vol. 6, no. 1, pp. 117–157, 2015.

[8] M. Avgerinou, P. Bertoldi, and L. Castellazzi, "Trends in data centre energy consumption under the European code of conduct for data centre energy efficiency," *Energies*, vol. 10, no. 1470, pp. 1–18, 2017.

[9] A. Shehabi, S. J. Smith, E. Masanet, and J. Koomey, "Data center growth in the United States: Decoupling the demand for services from electricity use," *Environmental Research Letters*, vol. 13, no. 12, 2018.

[10] N. Dodd, F. Alfieri, M. N. D. O. G. Caldas, L. M.-D. J. Viegand, S. Flucker, R. Tozer, B. Whitehead, and A. W. F. Brocklehurst, "Development of the eu green public procurement (gpp) criteria for data centres," Server Rooms and Cloud Services, Publications Office of the European Union, Tech. Rep., 2020.

[11] "Cisco global cloud index: Forecast and methodology, 2016–2021," Cisco, Tech. Rep., 2018.

[12] "Google data centers: Efficiency," accessed: 30-08-2021. [Online]. Available: www.google.co.uk/about/datacenters/efficiency/

[13] A. Shehabi, S. J. Smith, D. A. Sartor, R. E. Brown, M. Herrlin, J. G. Koomey, E. R. Masanet, N. Horner, I. L. Azevedo, and W. Lintner, "United States data center energy usage report," Tech. Rep., 2016. [Online]. Available: https://eta.lbl.gov/publications/united-states-data-center-energy

[14] R. Cohen, L. Lewin-Eytan, J. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *INFOCOM'15: Proc. of the 2015 IEEE Conference on Computer Communications*, 2015, pp. 1346–1354.

[15] M. C. Luizelli, W. L. da Costa Cordeiro, L. S. Buriol, and L. P. Gaspary, "A fix-and-optimize approach for efficient and large scale virtual network function placement and chaining," *Comput. Commun.*, vol. 102, pp. 67–77, 2017.

[16] Y. Sang, B. Ji, G. R. Gupta, X. Du, and L. Ye, "Provably efficient algorithms for joint placement and allocation of virtual network functions," in *INFOCOM'17: Proc. of the 2017 IEEE Conference on Computer Communications*, 2017, pp. 1–9.

[17] J. Billingsley, K. Li, W. Miao, G. Min, and N. Georgalas, "Evolutionary multi-objective virtual network function placement: A formal model and effective algorithms," manuscript under review.

[18] ——, "Routing-led placement of vnfs in arbitrary networks," in *CEC'20: Congress on Evolutionary Computation*. IEEE, 2020, pp. 1–8.

[19] D. Landa-Silva, "Design of modern heuristics," *Genet. Program. Evolvable Mach.*, vol. 14, no. 1, pp. 119–121, 2013.

[20] "Impact of the intel data plane development kit (intel dpdk) on packet throughput in virtualized network elements," Intel, Tech. Rep., 2013.

[21] "Packet processing performance of virtualized platforms with linux* and intel architecture," Intel, Tech. Rep., 2013.

[22] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *CloudNet'15: IEEE International Conference on Cloud Networking*. IEEE, 2015, pp. 171–177.

[23] M. Gao, B. Addis, M. Bouet, and S. Secci, "Optimal orchestration of virtual network functions," *Comput. Networks*, vol. 142, pp. 108–127, 2018.

[24] A. Baumgartner, V. S. Reddy, and T. Bauschert, "Combined virtual mobile core network function placement and topology optimization with latency bounds," in *EWSDN'15: European Workshop on Software Defined Networks*, 2015, pp. 97–102.

[25] D. B. Oljira, K. Grinnemo, J. Taheri, and A. Brunström, "A model for qos-aware VNF placement and provisioning," in *NFV-SDN'17: IEEE Conference on Network Function Virtualization and Software Defined Networks*. IEEE, 2017, pp. 1–7.

[26] F. B. Jemaa, G. Pujolle, and M. Pariente, "Analytical models for qos-driven VNF placement and provisioning in wireless carrier cloud," in *MSWiM'16: International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2016, pp. 148–155.

[27] S. Agarwal, F. Malandrino, C. Chiasserini, and S. De, "Joint VNF placement and CPU allocation in 5g," in *INFOCOM'18: IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1943–1951.

[28] A. Marotta, E. Zola, F. D'Andreagiovanni, and A. Kassler, "A fast robust optimization-based heuristic for the deployment of green virtual network functions," *J. Network and Computer Applications*, vol. 95, pp. 42–53, 2017.

[29] T. Kuo, B. Liou, K. C. Lin, and M. Tsai, "Deploying chains of virtual network functions: On the relation between link and server usage," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1562–1576, 2018.

[30] H. Guo, Y. Wang, Z. Li, X. Qiu, H. An, P. Yu, and N. Yuan, "Cost-aware placement and chaining of service function chain with VNF instance sharing," in *NOMS'20: IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2020, pp. 1–8.

[31] D. Qi, S. Shen, and G. Wang, "Towards an efficient VNF placement in network function virtualization," *Comput. Commun.*, vol. 138, pp. 81–89, 2019.

[32] W. Rankothge, F. Le, A. Russo, and J. Lobo, "Optimizing resource allocation for virtualized network functions in a cloud center using genetic algorithms," *IEEE Trans. Network and Service Management*, vol. 14, no. 2, pp. 343–356, 2017.

[33] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: a scalable and fault-tolerant network structure for data centers," in *SIGCOMM'08: Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2008, pp. 75–86.

[34] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *SIGCOMM:'08 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2008, pp. 63–74.

[35] Cisco, "Spine and leaf architecture: Design overview white paper," 2019. [Online]. Available: http://www.youtube.com/watch?v=8RrbUyw9uSg

[36] H. A. Alameddine, L. Qu, and C. Assi, "Scheduling service function chains for ultra-low latency network services," in *CNSM'17: Conference on Network and Service Management*. IEEE Computer Society, 2017, pp. 1–9.

[37] P. Vizarreta, M. Condoluci, C. M. Machuca, T. Mahmoodi, and W. Kellerer, "Qos-driven function placement reducing expenditures in NFV deployments," in *ICC'17: IEEE International Conference on Communications*. IEEE, 2017, pp. 1–7.

[38] L. Qu, C. Assi, K. B. Shaban, and M. J. Khabbaz, "A reliability-aware network service chain provisioning with delay guarantees in nfv-enabled enterprise datacenter networks," *IEEE Trans. Network and Service Management*, vol. 14, no. 3, pp. 554–568, 2017.

[39] H. Hawilo, M. Jammal, and A. Shami, "Network function virtualization-aware orchestrator for service function chaining placement in the cloud," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 643–655, 2019.

[40] D. M. Manias, M. Jammal, H. Hawilo, A. Shami, P. Heidari, A. Larabi, and R. Brunner, "Machine learning for performance-aware virtual network function placement," in *GLOBECOM'19: IEEE Global Communications Conference*. IEEE, 2019, pp. 1–6.

[41] D. M. Manias, H. Hawilo, M. Jammal, and A. Shami, "Depth-optimized delay-aware tree (DO-DAT) for virtual network function placement," *CoRR*, vol. abs/2006.01790, 2020. [Online]. Available: https://arxiv.org/abs/2006.01790

[42] P. Roy, A. Tahsin, S. Sarker, T. Adhikary, M. A. Razzaque, and M. M. Hassan, "User mobility and quality-of-experience aware placement of virtual network functions in 5g," *Comput. Commun.*, vol. 150, pp. 367–377, 2020.

[43] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," in *CNSM'15: International Conference on Network and Service Management*, M. Tortonesi, J. Schönwälder, E. R. M. Madeira, C. Schmitt, and J. Serrat, Eds., 2015, pp. 50–56.

[44] A. Alleg, R. Kouah, S. Moussaoui, and T. Ahmed, "Virtual network functions placement and chaining for real-time applications," in *CAMAD'17: IEEE International Workshop on Computer Aided Modeling*

and Design of Communication Links and Networks. IEEE, 2017, pp. 1–6.

[45] J. Pei, P. Hong, K. Xue, D. Li, D. S. L. Wei, and F. Wu, "Two-phase virtual network function selection and chaining algorithm based on deep learning in sdn/nfv-enabled networks," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1102–1117, 2020.

[46] F. C. T. Chua, J. Ward, Y. Zhang, P. Sharma, and B. A. Huberman, "Stringer: Balancing latency and resource usage in service function chain provisioning," *IEEE Internet Comput.*, vol. 20, no. 6, pp. 22–31, 2016.

[47] Q. Zhang, Y. Xiao, F. Liu, J. C. S. Lui, J. Guo, and T. Wang, "Joint optimization of chain placement and request scheduling for network function virtualization," in *ICDCS'17: IEEE International Conference on Distributed Computing Systems*, K. Lee and L. Liu, Eds. IEEE Computer Society, 2017, pp. 731–741.

[48] R. Gouareb, V. Friderikos, and A. Aghvami, "Virtual network functions routing and placement for edge cloud latency minimization," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2346–2357, 2018.

[49] A. Leivadeas, M. Falkner, I. Lambadaris, M. Ibnkahla, and G. Kesidis, "Balancing delay and cost in virtual network function placement and chaining," in *NetSoft'18: IEEE Conference on Network Softwarization and Workshops*. IEEE, 2018, pp. 433–440.

[50] K. Qu, W. Zhuang, Q. Ye, X. Shen, X. Li, and J. Rao, "Dynamic flow migration for embedded services in sdn/nfv-enabled 5g core networks," *IEEE Trans. Communications*, vol. 68, no. 4, pp. 2394–2408, 2020.

[51] G. Pongor, "Omnet: Objective modular network testbed," in *MASCOTS'93: Proceedings of the International Workshop on Modeling, Analysis, and Simulation On Computer and Telecommunication Systems*, 1993, pp. 323–326.

[52] J. Shi, Q. Zhang, and J. Sun, "PPLS/D: parallel pareto local search based on decomposition," *IEEE Trans. Cybern.*, vol. 50, no. 3, pp. 1060–1071, 2020.

[53] Q. Zhang and H. Li, "MOEA/D: A multiobjective evolutionary algorithm based on decomposition," *IEEE Trans. Evol. Comput.*, vol. 11, no. 6, pp. 712–731, 2007.

[54] I. Vlasic, M. Durasevic, and D. Jakobovic, "Improving genetic algorithm performance by population initialisation with dispatching rules," *Comput. Ind. Eng.*, vol. 137, 2019.

[55] C. L. Ramsey and J. J. Grefenstette, "Case-based initialization of genetic algorithms," in *ICGA'93: International Conference on Genetic Algorithms*, 1993, pp. 84–91.

[56] E. Osaba, F. Díaz, R. Carballedo, E. Onieva, and P. López-García, "A study on the impact of heuristic initialization functions in a genetic algorithm solving the n-queens problem," in *GECCO'14: Genetic and Evolutionary Computation Conference*, 2014, pp. 1473–1474.

[57] L. Kleinrock, *Theory, Volume 1, Queueing Systems*. Wiley-Interscience, 1975.

[58] G. Miotto, M. C. Luizelli, W. L. da Costa Cordeiro, and L. P. Gaspary, "Adaptive placement & chaining of virtual network functions with NFV-PEAR," *J. Internet Services and Applications*, vol. 10, no. 1, pp. 3:1–3:19, 2019.

[59] X. F. Liu, Z. Zhan, J. D. Deng, Y. Li, T. Gu, and J. Zhang, "An energy efficient ant colony system for virtual machine placement in cloud computing," *IEEE Trans. Evolutionary Computation*, vol. 22, no. 1, pp. 113–128, 2018.

[60] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[61] E. Zitzler and S. Künzli, "Indicator-based selection in multiobjective search," in *PPSN'04: Parallel Problem Solving from Nature*, ser. Lecture Notes in Computer Science, vol. 3242. Springer, 2004, pp. 832–842.

[62] E. Zio and R. Bazzo, "A clustering procedure for reducing the number of representative solutions in the pareto front of multiobjective optimization problems," *Eur. J. Oper. Res.*, vol. 210, no. 3, pp. 624–634, 2011.

[63] J. Hartigan and M. Wong, "Algorithm AS 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society*, vol. 28, no. 1, pp. 100–108, 1979.

[64] K. Kennedy and K. S. McKinley, "Optimizing for parallelism and data locality," in *ICS'92: International conference on Supercomputing*, 1992, pp. 323–334.

[65] M. Otokura, K. Leibnitz, Y. Koizumi, D. Kominami, T. Shimokawa, and M. Murata, "Application of evolutionary mechanism to dynamic virtual network function placement," in *ICNP'16: International Conference on Network Protocols*, 2016, pp. 1–6.

[66] E. Alba, A. Nakib, and P. Siarry, *Metaheuristics for dynamic optimization*. Springer, 2013, vol. 433.