Triptych: Multi-objective Optimisation of Service Deployment Costs, Application Delay and Bandwidth Usage

Miguel Rocha[1,2], Truong Khoa Phan[2], Joao Reis[2], David Griffin[2], Miguel Rio[2]
[1]University of Minho, [2]University College London
[1]mrocha@di.uminho.pt, [2]{t.phan,joao.reis.16,dgriffin,miguel.rio}@ucl.ac.uk

***Abstract*—Advanced Internet services increasingly rely on many components to implement their functionality. These composite services have three important features: they are expensive to deploy, components need to be placed intelligently close to the users to improve quality of experience and they will potentially consume significant amounts of bandwidth. This paper presents Triptych, a multi-objective optimisation framework that tries to optimise according these three dimensions to help the three main stakeholders in the Internet ecosystem: users, application providers and network providers. Triptych implements evolutionary computation approaches for this complex problem, which simultaneously optimise service deployment costs, latency-based user utility and network congestion. These algorithms provide possible operating points, bringing important tools for network managements and resource allocation. A large set of simulations under different scenarios are provided to validate the algorithms.**

I. Introduction

Emerging Internet applications such as virtual/augmented reality (VR/AR), online gaming, wearable cognitive assistance and edge analytics require tight constraints on quality of service (QoS) [4], [8], [23]. These applications require extreme low latency and high bandwidth consumption. The general approach is to move computational resources closer to the users to reduce network latency and bandwidth requirements [10], [17], [18], [20], [30]. Therefore, placement policies will play a key role in delivering good QoS for the end-users.

Finding optimal placement for services is an NP-hard problem [9], [20]. In this class of problems, a composite service comprises several components potentially deployed by servers in different locations. Those components need to be connected in a pre-defined topology or workflow, which can be defined as a combination of chain, parallel and/or loop structures [20]. A typical example of composite service is Network Functions Virtualisation (NFV) paradigm where network functionality is split up into multiple building blocks that need to be chained together to provide the required functionality [11]–[13].

In this scenario, the key challenges of composite services are: (a) how to optimally deploy services on different servers in distinct locations; and, (b) how to find optimal routing path connecting service instances as a pre-defined workflow. Several studies have previously attempted to solve one of these challenges without addressing both of them simultaneously. For instance, service placement optimisation has been addressed in [1], [17], [18], [20], [30], [31]. On the other hand, several optimal formulations and heuristic algorithms have been proposed for routing issues [5], [6], [15].

ISBN 978-3-903176-16-4

In this paper, we first show that optimising service placement only can cause serious network congestion (section IV-C). Therefore we propose Triptych framework to simultaneously optimise both network routing and service placement decision. This combination significantly increases the problem complexity, but it ensures user QoS while reducing operating costs for application and network providers.

In brief, the contributions of our work are as follows:

- We design a general-purpose definition of service topology, which allows to address different composition structures and combinations of these structures.
- We optimise both network routing and service placement at the same time. This combination helps to significantly improve user QoS as well as reducing operational costs.
- We introduce a weighting scheme in the objective function, as an approach to allow selecting operating points according to preferences of the application and the network providers.
- We conduct extensive numerical studies under synthetic datasets over real network topology. The simultion results show significant improvements on user latency, network congestion and operational cost.

The paper is organized as follows. In section II, we present a literature review of related work on service placement and routing optimisation. In section III, we design multi-objective genetic algorithms to trade-off between deployment costs, user latency and network congestion. We show evaluation results of those algorithms in section IV and conclude the work in section V.

II. Related Work

A. Service composition

Optimising service placement and service selection is an important research topic to improve QoS, as well as reducing operational costs. In this trend, work has been done for optimising atomic service performance [1], [17], [18], [24],

[30], [31]. Atomic service is a classical paradigm where all the required functionality is built into a single component. The authors in [17], [18] have introduced a novel utility-based service placement algorithm to significantly improve user utility compared with the classical closest-based placement approach. On the other hand, DONAR [30] has been proposed as a decentralised server selection method for cloud services aiming to optimise both client performance and server load. In this work, we consider to optimise several aspects of composite services, which translates into a much more complex problem than the atomic services one.

Service composition or composite services require multiple components to be executed sequentially or concurrently. For instance, recent work on NFV deployment has considered to optimise chain structure where each component is executed sequentially, thus the output of a component will be the input for the next one [2], [3], [20], [27].

The authors in [20] have proposed a general framework for maximising user utility in composite services. However, a major limitation of that work is that it does not consider the underlying network routing between service components. This limitation can potentially cause network congestion, as we show in our results (see Section IV-C).

Another work in this topic is DRENCH [27], a semi-distributed resource management framework for NFV based service function chaining. DRENCH incorporates a traffic load-balancing algorithm to estimate network function instance loads. Then, each NFV node independently directs flows to an appropriate least-loaded service instance. However, the DRENCH framework is designed only for NFV chaining which is the linear and the loop structures (see Figure 1 and Figure 3). In this work, we investigate a further step to optimise for both service placement and underlying routing between components, which is applicable for all kinds of composite service structures, including, but not limited to, linear, tree, loop structures or any combination of those (Section III-A). Furthermore, we consider more realistic models of cost and latency.

B. Network routing optimisation

Traffic engineering is an important approach to make efficient use of network resources with loose guarantees on delay, loss and throughput for end users [5], [6], [15]. For instance, Fortz et al. [5] proposed algorithms to optimise weight setting for Open Shortest Path First (OSPF) routing for better load balancing. In [6], the authors have proposed algorithm to optimise OSPF routing for more efficient use of network resources in traffic and network change scenarios. In [7], [15], the authors have proposed optimal and heuristic algorithms for finding routing solution in order to save network power consumption. A hybrid traffic engineering approach of application and network layers has been proposed to improve network efficiency especially for large-scale streaming applications [19]. The authors in [22] have designed optimisation and evolutionary meta-heuristic algorithms to find link weight setting for link-state protocols such as Open Shortest Path First (OSPF). Recently, machine learning has been applied to Internet routing domain [21], [28], [29]. The authors in [28], [29] show how routing be formulated as an machine learning problem. Then they solve the problem using both supervised learning [14] and reinforcement learning [26]. Their preliminary results show that supervised learning might be ineffective if the traffic conditions do not exhibit very high regularity. Then they suggest that applying deep reinforcement learning to this context yields high performance. However, all those aforementioned works assume the location of service instances is fixed and they try to optimise network routing only. In this work, by looking at both traffic engineering and service placement, Triptych tries to simultaneously optimise deployment cost, end-to-end latency and network congestion.

III. Triptych - Multi-objective Optimisation of Composite Services

In this section, we first introduce three basic structures of composite services. These structures can be combined to form any complex composite service structures. We then formally define the problem and design Triptych, an evolutionary algorithm to simultaneously optimise cost, latency and congestion for a general composite service structure.

A. Composite service structures

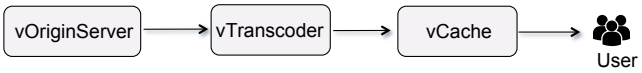


Fig. 1: Linear structure

One of the most common structures of composite service is the chain or linear structure. As an example, a user needs to watch a video on demand (VoD) content on his device (e.g. smart phone, tablet or smart TV). In Figure 1, the three virtual services $vOriginServer$, $vTranscoder$ and $vCache$ need to be connected as a chain to deliver a video stream to the user. The $vOriginServer$ acts as the root server sending the stream to the $vTranscoder$ and the $vCache$. The $vTranscoder$ is responsible for transcoding the video stream into varying levels of quality to support adaptive streaming. The $vCache$ is used to cache/store encoded video streams so that it can serve users quickly without connecting to the $vOriginServer$. For this linear structure, the end-to-end latency will be accumulated from each hop of the chain (including network latency and processing delay at each node).

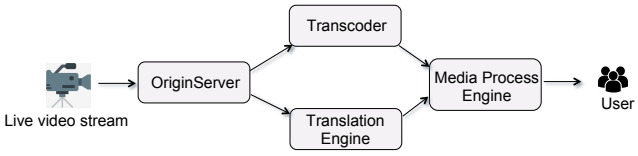


Fig. 2: Tree structure

Figure 2 shows an overview of the scenario covered by the remote production in broadcasting use case. The on-site

reporter needs to send high quality content back to the *media process engine* before being broadcasted. The encoded video and the translated audio produced by the *transcoder* and the *translation engine* will be merged at the *media process engine* before sending out to the viewers. The translating step can be done in parallel with the encoding step and the end-to-end latency will be the longest of the two branches. This structure forms a directed acyclic graph and is referred to as a tree or a parallel structure.

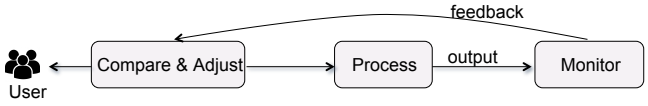


Fig. 3: Loop structure

The third structure we are presenting is a cycle or a loop structure. An example could be the closed loop system shown in Figure 3. It is similar to the chain structure except there is a loop to provide feedback for making decisions in subsequent rounds. The end-to-end latency is accumulated from each hop. Examples of this include services where components send any sort of application feedback to the source.

B. Triptych problem definition

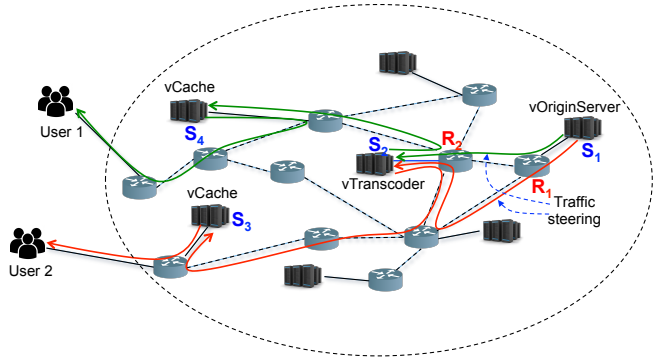


Fig. 4: Optimisation of routing and placement for composite services

We use the linear structure use case (Figure 1) as an example of the optimisation problem addressed by Triptych. In brief, high-quality communication paths between the $vOriginServer$ and the users are critical for VoD systems. Performance of the system is related to available bandwidth, latency of network links, computational capability and location of servers where service instances are deployed. To guarantee QoS, it may be necessary to modify the network path (e.g. via a SDN controller) or select service instances in different locations. In Figure 4, we show an example of two users watching the same VoD stream. In this example, there are several servers where we can deploy service instances ($vOriginServer$, $vTranscoder$ and $vCache$). We need to create a linear structure (or VNF chain) to connect between the $vOriginServer$, the $vTranscoder$, the $vCache$ and each of the users as a chain. To reduce operational costs,

TABLE I: Summary of key notations (alphabetical order)

C_e	capacity of link $e \in E$
D_i	demand of user $i \in U$
E	set of links in the network
F^f_{kj}	fixed cost of service j at server k
F^v_{kj}	variable cost of service j at server k
$G = (V, E)$	network topology
L_e	latency of link $e \in E$
L^{fh}_{min}	min first hop latency
L^{fh}_{max}	max first hop latency
L^{e2e}_{min}	min end-to-end latency
L^{e2e}_{max}	max end-to-end latency
$P_{latency}$	latency penalty
P_{cost}	cost penalty
$P_{congestion}$	congestion penalty
P^i_{fh}	first hop latency penalty of user i
P^i_{e2e}	E2E latency penalty of user i
r	root of the tree T
s_T	list of successors in the tree T
S_j	servers are able to deploy service j
$Serv_T$	list of distinct services in the tree T
SPO	service placement optimisation only
$T = (r, s_T)$	composite service tree
u_e	utilisation of link $e \in E$
$U = \{i\}$	set of users i
U_{min}	min link utilisation
U_{max}	max link utilisation
V	set of nodes in the network topology

the $vOriginServer$ and the $vTranscoder$ instances can be shared between the two chains as shown in Figure 4. However, we assume that the link $(R_1 - R_2)$ has limited bandwidth. Therefore, Triptych decides to route traffic flows of the two chains from S_1 to S_2 via different paths to avoid making the link $(R_1 - R_2)$ congested. On the other hand, to guarantee low latency, the $vCache$ needs to be located close to the end user, thus Triptych deploys the two instances of the $vCache$ in S_3 and S_4 for the two users. To sum up, the Triptych algorithm needs to take into account deployment cost, application delay and network congestion in making decisions on service instance placement and the corresponding underline network routing.

In this section, we will firstly define the data structures involved in the definition of the composite service structure. Then, we will introduce the concepts related to service placement and network routing optimisation and show how we connect them in Triptych.

1) General-purpose tree definition: A given instance of the composite service structure will be represented by a tree where each node represents a service instance that needs to be deployed in a suitable server and the tree will define the dependencies between service instances. The branches on the tree indicate services that can be executed in parallel, while a linear chain of nodes indicates services that need to be completed sequentially. We use nodes with replicated identifiers to indicate the same service instance in the loop structure (see Figure 5c). In this way, we have a general

representation scheme that allows to define different service structures.

We show in Figure 5 how the three composite structures be represented. In Figure 5b, the *Transcoder* and the *Translation Engine* need to connect to the root which is the *OriginServer*. Then they both connect to the *Media Process Engine* and the tree is terminated at the *user*. Note that in Figure 5c the "Com & Adj (1)" and "Com & Adj (2)" are the same "Compare & Adjust" instance (with replicated identifier) in the loop structure in Figure 3.

The tree is represented by a root node r and a function s_T that provides, for each node, the list of its successors in the tree: $T = (r, s_T)$. Note that for a given service s, the list $s_T(n)$ will provide the services that need to be completed in order for s to be able to start. For a given service tree T, we can also define $Serv_T$, which is the list of services in the tree, i.e. the full set of services that appear at least once in the tree's nodes. $|Serv_T|$ will therefore be used to denote the number of distinct services.

2) Service placement optimisation: The optimisation algorithm will define which servers to deploy service instances for each user. This decision is made by trading off cost with performance of the network, such as latency and congestion. Each server $s_{kj} \in S_j$ (S_j represents the set of all servers able to deploy service j), will be defined by a tuple (F^f_{kj}, F^v_{kj}) containing, respectively, the fixed deployment cost (F^f_{kj}) of the service j at server k and its variable cost (F^v_{kj}).

- Fixed cost: the cost of deploying the service instance for the first time at a server. For example, this can be thought as the cost of software installation in that server. The fixed cost is incurred only once and does not vary with the number/size of service instances at a certain location.
- Variable cost: this cost is proportional to the resources used by the service instances. The more service instances are deployed, the more resources are consumed and hence the cost increases with the number of instances.

The service placement optimisation algorithm can be executed at various timescales, including initial service deployment and ongoing reconfiguration to migrate existing services, instantiate new services, undertake service scaling as demand patterns change. On the other hand, composite service optimisation is used to determine which instances of services should be interconnected to meet performance and cost objectives for specific user session requests. This can be undertaken at initial session establishment, as well as for the optimisation of already running sessions/VNF forwarding graph (VNFFG) instances.

3) Routing optimisation: In this work, we add the network routing dimension to the problem. Thus, we will consider a network defined as a directed graph $G = (V, E)$, where V is the set of nodes, while E represents links connecting them. Each link $e \in E$ has a capacity C_e defining the amount of traffic it can accommodate. In addition, each link $e \in E$ is associated with a propagation delay L_e. When demands are allocated to the links in the network, the total amount of traffic allocated over a link will be given by its load l_e. From the link capacity and load, a link utilization u_e is calculated as: $u_e = l_e/C_e$.

We assume that each user $i \in U$ is connected to a node in the network topology (see Figure 4). The routing optimisation objective is to find an optimal end-to-end path connecting from the first to the subsequent service instances in the composite structure and finishing at the end-user. The optimal path needs to take into account network latency and link loads. In addition, the routing optimisation also provides feedback to the service placement optimisation. For instance when the placement optimisation receives feedback saying that the link (R_2, S_2) in Figure 4 is congested, it should instantiate new $vTranscoder$ in another location to reduce the load on the link (R_2, S_2). The Triptych optimisation algorithm needs to take into account three dimensions in the formulation of the objective function: latency, cost and network congestion. We define, in the next subsection the multi-objective cost function used in the Triptych algorithm.

C. Multi-objective cost function

1) Latency: We consider latency which includes both network latency and processing time at the server where service instances are deployed. Also, some services require an extreme low latency between the users and the first hop service instance. For example, users should connect to a low-latency rendering component in an online game service to reduce lag as the player moves viewpoint, while the game simulation engine itself could be located more remotely, if the positions of other players do not change rapidly and so a longer latency would not impact game play. Therefore, along with the end-to-end latency, we also consider the first hop latency as a component of the cost function when deploying a composite service.

Inspired by the utility function from the work [18], we define a minimum and the maximum first hop and end-to-end latency thresholds. Hence, each user request, $i \in U$, will be defined by a tuple $(D_i, L^{fh}_{min}, L^{fh}_{max}, L^{e2e}_{min}, L^{e2e}_{max})$, which define, respectively, the user demand, and the minimum and maximum required first hop and end-to-end latency. The minimum and maximum latency thresholds are used to capture the nature of the penalty imposed over the performance for services. For some services, if the latency is less than L_{min}, the improvement is not perceived by the users of that service, thus the penalty is always zero. For instance, in high-quality *voice over IP* service, if latency is less or equal to $L_{min} = 20$ ms, the users get a feeling of real voice communication [25]. Therefore, there is no need to consider any penalty ($P = 0$ - Figure 6) when the latency is less or equal to L_{min}. On the other hand, 150 ms is about the limit for keeping the user's attention focused. Therefore, a huge penalty needs to set if the latency is more than $L_{max} = 150$ ms. We show in Figure 6 a graph of penalty vs. metric, where the metric X in this case refers to latency.

vOriginServer: *vTranscoder*
vTranscoder: *vCache*
vCache: *User*

(a) Linear structure

OriginServer: *Transcoder, Translation Engine*
Transcoder: *Media Process Engine*
Translation Engine: *Media Process Engine*
Media Process Engine: *User*

(b) Tree structure

Com & Adj (1): *Process*
Process: *Monitor*
Monitor: *Com & Adj (2)*
Com & Adj (2): *User*

(c) Loop structure

Fig. 5: Example of composite service representation

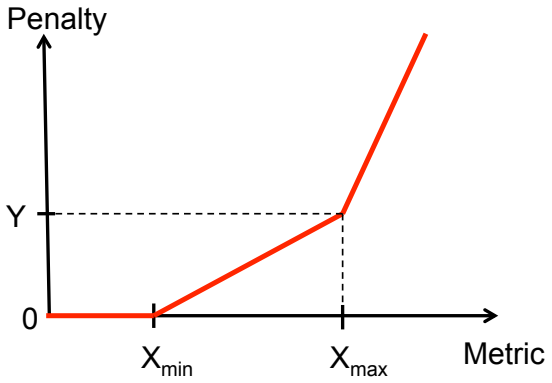


Fig. 6: Penalty function p used to normalize the latency and cost metrics

$$P_{latency} = \frac{1}{2 * |U|} \sum_{i \in U} (P^i_{e2e} + P^i_{fh}) \quad (1)$$

where P^i_{e2e} and P^i_{fh} represent the penalties for end-to-end and first-hop latencies for user request i, being obtained by applying the penalty function p depicted in Figure 6 to the computed values of the respective latencies (Y is set to 10 in our experiments), given an assignment of services to servers. Note that in the current model, we do not consider the impact of network congestion on the latency. For example, congestion on a link can increase queuing delay which in the end impacts on the overall latency. We plan a future work to incorporate queuing delay into the general latency model.

2) Deployment costs: We optimise both the fixed cost and the variable costs (see section III-B2) in the Triptych algorithm. So the total cost TC of a solution (an assignment of services to specific servers) is calculated by $TC = FC + VC$, summing the fixed costs (FC) incurred in the used servers and the variable costs (VC) for all user requests. The fixed cost is, thus, calculated as follows:

$$FC = \sum_{j \in Serv_T} \sum_{k \in S_j} b_{jk} \times F^f_{kj} \quad (2)$$

where b_{jk} is a binary variable, taking value 1 if the server s_k is used to deploy service j, and 0 otherwise.

On the other hand, the variable costs are calculated as:

$$VC = \sum_{i \in U} \sum_{j \in Serv_T} \sum_{k \in S_j} b_{jki} \times F^v_{kj} \times D_i \quad (3)$$

where b_{jki} is a binary variable, taking value 1 if the server s_k is used to deploy service j processing the user request i, which has a total demand of D_i.

To be able to create a unified cost function, we also normalise the cost penalty - $P_{cost} = p(TC)$ - using the convex piecewise linear function as shown in Figure 6. The penalty cost is zero if the sum of fixed cost and variable cost are less or equal than a defined minimum budget $Cost_{min}$, and we set a fast growing penalty for total costs above a threshold $Cost_{max}$. In our experiments, $Cost_{min}$ was calculated as the minimum possible cost to satisfy all demands, while $Cost_{max}$ was set to be equal to $2 \times Cost_{min}$.

3) Network congestion: In this work, we use the congestion measure proposed by Fortz and Thorup in their seminal paper [5]. It is defined by a cost (or penalty) Φ which is the sum of the cost function $\Phi_e(u_e)$ for each link $e \in E$. This is a convex piecewise linear penalty function, which has a low growth for small link utilisations, but increases more quickly with the link utilisation (see the piecewise convex penalty function in [5]). It has the advantage over metrics as maximum link utilisation of considering all links in the network, while also penalising heavily overloaded links.

As proposed in the original publication, this metric can be normalized over distinct network topologies, computing Φ^*, which divides the penalty Φ over the minimum load in the network, obtained by routing over shortest paths with link weights set to 1. In this case, the optimal theoretical value of the objective function is 1.

Note that the definition of this objective function follows the same principles of the two previous ones. Since the optimal value in this case is 1, we set $P_{congestion} = 1 - \Phi^*$. In this way, network congestion can be combined with latency and cost in the multi-objective formulation.

4) Overall cost function: Here, we consider two scenarios: in the first, which we will denote by service placement optimisation only (SPO), we address the optimisation of the assignment of the required services for all user requests seeking to minimize $P_{latency}$ and P_{cost}, while in the latter (our Triptych algorithm) we will consider also routing in the network, and therefore will add a third objective ($P_{congestion}$) to the optimisation framework.

In this work, we consider an aggregation of the different cost functions through a sum of their values, weighted by considering individual parameters for each component. Thus, our objective will be to minimize P, defined as:

$$P = \alpha P_{latency} + \beta P_{cost} + \gamma P_{congestion} \quad (4)$$

As a way to further normalize the results, we will consider only cases where $(\alpha + \beta + \gamma = 1)$. In the case of SPO, we will consider γ to be set to 0.

Notice that as the latency, cost and congestion values are normalized, the optimal value of each of the components of the cost function is 0, and therefore this will also be the theoretical optimal value of P, although this might not be achievable given the specific instances. In addition, when we set $\alpha = \beta = \gamma = 1/3$, the three components are considered in the optimization process with the same priority.

D. Evolutionary Algorithm

1) Overall structure of the EA: We address the optimisation problem formulated in the previous section through the use of Evolutionary Algorithms (EAs). These are justified in this case by the complexity of the underlying optimisation problem and by the numerous cases where they have shown success in related tasks [20], [22]. We present in Figure 7 a general flowchart showing the main steps of the EA. We implemented the EA based on the *inspyred* package in Python[1], following the default structure of the provided EA.

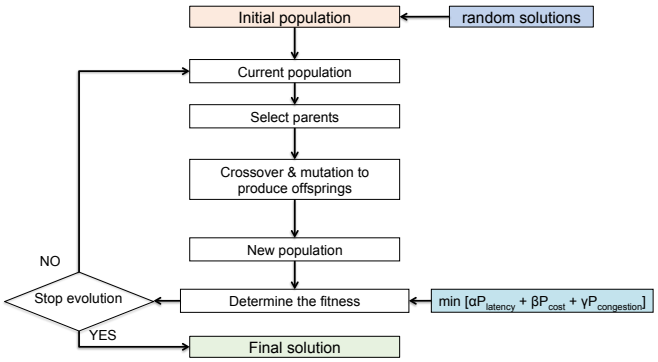


Fig. 7: Evolutionary algorithm flow chart

Next, we explain in detail the steps of the EA:

- Initialisation: The initial population is a set of potential solutions to the problem. Each solution contains a list of integers, each being randomly generated within the allowed range for each position.
- New population: after each generation, a new population is created by the following steps until reaching the stopping criterion:
  - Selection: select parents from a population according to their fitness (the lower the fitness, the higher chance to be selected).
  - Crossover: using the selected crossover operator, create new individuals (offspring) from the selected parents.
  - Mutation: over the generation, use the selected mutation operators to change part of their genes.
  - Reinsertion: the newly generated offspring will be combined with some individuals selected from the previous population to create the next population.
- Stop: if the termination criterion is reached (maximum number of solution evaluations), the algorithm stops, and returns the best solution in the current population.

2) Solution representation and evaluation: A solution for the considered optimisation problem is represented as an individual in the EA, and is encoded as a list of integer values. This list encompasses two parts, the first represents the assignment of the services to specific servers, for each user request, while the latter represents the way routing is done in the network, through the definition of weights for each network link, used to route the traffic using an intra-domain routing protocol as OSPF. Notice that in the SPO case, only the first part of the solutions exists.

Regarding the first part, the service-server assignment, the length is given by $|U| \times |Serv_T|$. Thus, for each user request, all services defined in the structure have an assigned server. The integer value at each position will specify the index of the server assigned to deploy that specific service for that user request. This part of the solution is decoded into a finite function indexed by user identifier. For each user, another finite function from service to server identifier keeps the assignment for that specific user request.

Regarding the second part, encoding routing weights, the length is given by the number of links in the network. Each value specifies the weight of a link, to be used in the shortest path calculation for each demand, using the defined intra-domain routing protocol (e.g. OSPF). The range of the values in this part of the individual is the set [1, $MaxW$]. In the experiments, $MaxW$ was considered to be 20.

Each individual is evaluated by first decoding its genome (list of integer values) into a service-server assignment for each user, and if routing is optimised into the OSPF weights, as described. The service-server assignment is used to compute the penalties for latency and costs, as described in Sections III-C1 and III-C2. The routing weights are used to compute the full paths for each user request, from those the respective link loads and utilisations, and finally compute the congestion penalty (see Section III-C3). The fitness of each individual is given by the overall penalty P, computed as given by the Equation 4.

3) Genetic operators: The EA implemented in this work, makes use of both crossover and mutation operators. Regarding the crossover, we use a single operator provided by the *inspyred* package, namely the *one-point crossover*. This takes two parents, cuts in a random position and combines the genes alternating from each parent, generating two new solutions (individuals).

Regarding mutation, we make use of three different operators: *random mutation*, an operator from the *inspyred* package which randomly changes the value of a given gene for another in the allowed range for that position (given a probability per position); *increment mutation*, which changes a randomly selected gene adding or subtracting one; and, an *intelligent mutation*, which works as random mutation but only accepts the change if it leads to a higher fitness value,

[1]https://pypi.org/project/inspyred/

being the process repeated N_{mut} times (in the experiments, we considered its value to be 20).

4) Other parameters: In the EA, the selection of the parents to undergo recombination is based on ranking selection, where individuals are ranked and selection is made based on values calculated from the ranking position. The recombination scheme is generational, being replaced in each generation 80% of the individuals in the population, with an elitism value of 2 (i.e. the two best individuals are always kept). The population size was set to 100, and the termination criterion is based on a maximum number of solutions evaluated (set to 20000 in the experiments).

In the objective function, the default values for the weights were the following: $\alpha = 0.4, \beta = 0.3, \gamma = 0.3$. For SPO, both α and β were set to 0.5.

IV. Evaluation

A. Simulation setup

We evaluated Triptych's performance testing with three composite structures: the linear (Figure 1), the tree (Figure 2), and the loop (Figure 3).

We use the Geant network as the core topology, which contains 22 nodes and 36 links, as shown in Figure 8 [16].

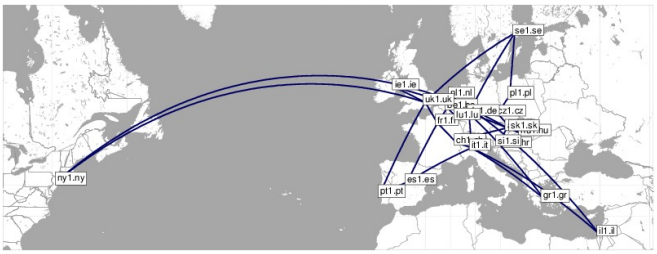


Fig. 8: Geant network topology

Over this network core, we simulated a set of problem instances, by generating realistic values of user demands, server costs and latency requirements (end-to-end and first-hop). We considered 2 distinct levels of latency requirements and of overall demands (named A and B) to create distinct levels of difficulty (where A is the easier and B the more difficult scenario). For instance, latency thresholds L_{min} and L_{max} in A are bigger than those in B. As a result, there is more chance for finding good solutions in A than in B.

Users and servers are randomly placed in network nodes, although the proportion of nodes that can deploy servers was considered to be 30% of the whole set. All generated instances have an average of 5 user requests per network node.

B. Algorithm convergence

We show in Figure 9 how the EAs converge over 250 generations, corresponding to 20000 solution evaluations in a selected set of cases which are representative of the whole set of simulations. We show the convergence of SPO and Triptych algorithms for the linear, the tree and the loop structures. As SPO is simpler than Triptych in terms of algorithm complexity, SPO has better fitness values in the first generations. However, the two algorithms quickly converge to stable fitness values in less than 100 generations. The other observation is that computational complexity increases from the linear to the tree and the loop structures. As a result, the linear structure converges in less than 50 generations while it takes around 100 generations for the loop structure to reach its point of convergence.

To benchmark Triptych, one would argue that we need to design an optimal formulation (e.g. using integer linear program) and compare Triptych's result with the optimal one. Recall that Triptych tries to minimize the fitness function P (Equation 4) where $P \geq 0$. Therefore, we can derive the optimal fitness value which is $P_{optimal} \geq 0$. As shown in Figure 9, the fitness values in all the three structures converge to a value which is close to zero (see Figure10, Table II and Table III for the exact value of P). In other word, we can see that Triptych algorithm performs well and is able to find near-optimal solutions.

C. Triptych vs. service placement optimisation only

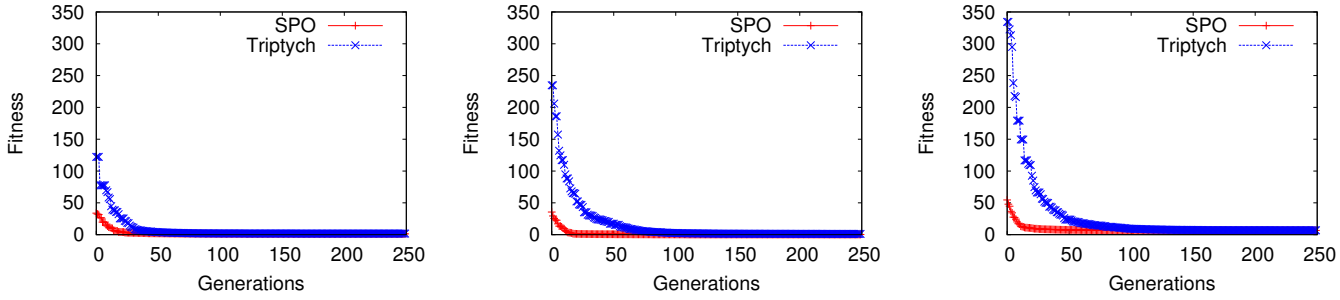
In this section, we show a comparison between Triptych and the service placement optimisation only (SPO). The full results are shown in Table II. Column **T** stands for the topology - L (linear), T (tree) and P (loop). Columns **D** and **L** represent the two levels of demands and latency requirements, respectively (see section IV-A). The other columns represent the results of the EA for the three components of the objective function. Each EA was run 5 times and the mean of the results for the best solution are shown in the table. Notice that for SPO the penalty regarding the congestion is not optimised and is shown to illustrate the values of congestion obtained when this is not taken into account.

TABLE II: Results for the Triptych algorithm vs SPO.

T	**D**	**L**	**SPO**			**Triptych**		
			P_{lat}	P_{cost}	P_{cong}	P_{lat}	P_{cost}	P_{cong}
L	A	A	2.09	0.30	2708.8	2.23	1.92	0.61
L	A	B	4.13	0.53	2178.8	4.42	1.91	0.57
L	B	A	2.08	0.30	2899.0	2.34	2.22	0.84
L	B	B	4.11	0.49	2475.4	4.52	2.21	0.88
T	A	A	0.86	0.26	1963.8	0.75	1.24	1.22
T	A	B	2.27	0.32	1779.3	2.91	1.26	1.06
T	B	A	0.44	0.30	2196.2	1.10	1.43	2.10
T	B	B	1.92	0.30	2117.9	3.16	1.41	1.85
P	A	A	4.88	0.12	2920.5	4.95	2.25	0.90
P	A	B	7.89	0.17	2944.7	8.01	2.26	1.00
P	B	A	4.88	0.10	3240.3	5.28	2.49	1.73
P	B	B	7.87	0.15	3168.0	8.30	2.41	1.59

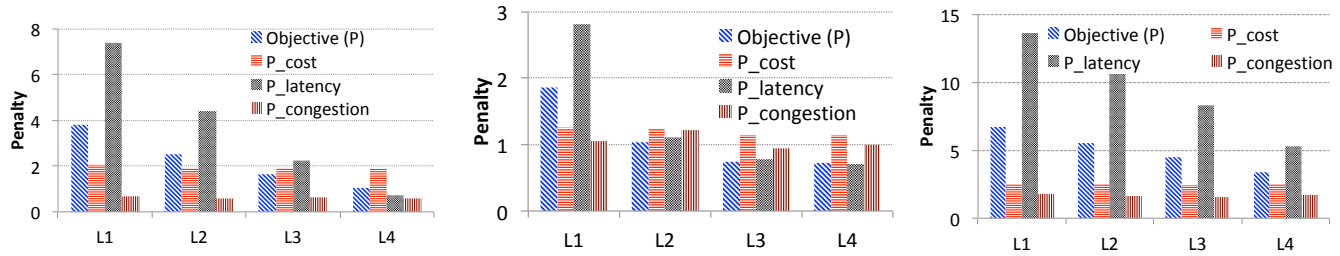
The first conclusion we may take from the analysis of these results is that the SPO provides good results for latency and costs in the tested instances, but by not taking into account the routing in the network it causes unacceptable levels of congestion in the network, with values for P_{cong} around 2000, which means that there are several highly overloaded links in the network.

On the other hand, the Triptych algorithm provides acceptable results for the three objectives in all instances. Of course,



(a) Linear structure (b) Tree structure (c) Loop structure

Fig. 9: Convergence of SPO and Triptych



(a) Linear structure (b) Tree structure (c) Loop structure

Fig. 10: Triptych with different delay configuration L1 - L4

with more constraints on the solution space, Triptych has, in most cases, slightly higher penalties on cost and latency, as compared to SPO. Still, these are normally not very significant, being still on a range of quite acceptable values, are largely compensated by major improvements on network congestion.

Indeed, Triptych significantly improves on the congestion path. Note that as SPO does not consider routing, we assume that shortest path is used between two service instances. For example, in Figure 4, if using SPO the routing between S_1 and S_2 for the two flows will always be $[S_1 - R_1 - R_2 - S_2]$ which causes serious congestion on the link $R_1 - R_2$. Therefore, Triptych is far better in term of routing flexibility to overcome congested links.

D. Triptych results for instances with different latency requirements

One of the observations from the results in the previous section is that the levels of latency requirements seem to affect heavily the results. We show in this section how Triptych reacts with different latency levels. We consider one of the instances for each topology and vary the latency requirements in four levels ($L1 - L4$), from the hardest to the easiest. As shown in Figure 10, the cost, latency and congestion penalties reduce from the $L1$ to $L4$ scenarios. This observation is true in all the composite service structures. This shows some consistency in the results of the algorithm, and demonstrates it can be used over a large range of possible scenarios.

E. Triptych results with different weighted parameters

One of the main advantages of Triptych is that the weights defined for each component of the cost function provides flexibility to network operators to choose different operating points with distinct trade-offs of the objectives.

To illustrate this, we show in Table III how Triptych reacts with different weights in the objective function described in section III-C4, for a selected instance (in this case, using the linear structure). We vary α, β and γ which respectively control the importance of latency (including first-hop and end-to-end latency), cost (including fixed and variable costs) and congestion in the objective function. Recall that $\alpha+\beta+\gamma = 1$, and thus we take the default configuration and try variants where one (or two) of these weights is increased or reduced, being this compensated by the other one(s).

If we set $\alpha > \beta$, the Triptych algorithm tends to reduce latency, but potentially increases the cost. For instance, when we increase α in the configurations $w1 - w12$ and as a result, the latency penalties tend to be reduced. A similar observation can be found for β and γ.

However, as we are dealing with the three weights (α, β and γ) at the same time, changing one of them also has impact on the others. For example, the weight settings $w10, w11$ and $w12$ have the same α value, but the latency penalties are different as we change β and γ. Finding the correlation between the three weights is important as it helps to decide suitable operating points. For example, based on these results, if one needs to minimise the cost while having congestion penalty to be less

TABLE III: Results for the Triptych algorithm with different weights configurations

Weight config.	Weights α	β	γ	Results P	P_{lat}	P_{cost}	P_{cong}
default	0.4	0.3	0.3	1.85	2.34	2.22	0.84
w1	0.2	0.5	0.3	1.82	2.70	1.91	1.10
w2	0.2	0.3	0.5	1.56	2.57	2.24	0.74
w3	0.2	0.4	0.4	1.66	2.44	2.17	0.72
w4	0.3	0.2	0.5	1.50	2.31	2.42	0.65
w5	0.3	0.5	0.2	1.85	2.34	1.84	1.15
w6	0.4	0.45	0.15	2.01	2.38	1.89	1.34
w7	0.4	0.15	0.45	1.54	2.20	2.50	0.63
w8	0.5	0.1	0.4	1.65	2.34	2.44	0.59
w9	0.5	0.4	0.1	1.95	2.17	1.79	1.53
w10	0.6	0.1	0.3	1.78	2.21	2.47	0.70
w11	0.6	0.3	0.1	1.92	2.07	1.80	1.42
w12	0.6	0.2	0.2	1.93	2.19	2.11	0.97

than 1, the weight setting $w12$ should be used. In summary, by changing the weights, Triptych can provide an approximation of the Pareto front of the trade-off between the three objectives (latency, cost and congestion).

V. Conclusions

In this paper, we propose Triptych, a multi-objective optimisation framework for deployment of cost, application delay and bandwidth usage for composite services. Since this is an NP-hard problem, we design an evolutionary algorithm for solving the problem. Based on simulation results on several realistic instances, we show that comparing to service placement optimisation only, Triptych significantly improves network congestion, while keeping relatively fair penalties on cost and latency. In addition, Triptych reacts well with different levels of network configuration.

In future work, we plan to further develop online algorithms to dynamically change placement/routing solutions on-demand. Also, the exploration of alternatives for multiobjective optimisation based on EAs specifically designed for this purpose will be pursued.

Acknowledgment

This work has been supported by the US Army Research Laboratory and the UK Ministry of Defence (agreement number W911NF-16-3-0001) and has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 761699 (5G-MEDIA).

References

[1] A. Ascigil, T. K. Phan, V. Sourlas, I. Psaras, and G. Pavlou. On Uncoordinated Service Placement in Edge Clouds. In *IEEE Int. Conf. on Cloud Computing Technology and Science (CloudCom)*, 2017.

[2] M. Bouet, J. Leguay, and V. Conan. Cost-based Placement of Virtualized Deep Packet Inspection Functions in SDN. In *IEEE Military Communications Conference (MILCOM)*, 2013.

[3] M. Bouet, J. Leguay, and V. Conan. Cost-based Placement of vDPI Functions in NFV Infrastructures. In *IEEE NetSoft*, 2015.

[4] J. Cho, K. Sundaresan, R. Mahindra, J. Merwe, and S. Rangarajan. ACACIA: Context-aware Edge Computing for Continuous Interactive Applications over Mobile Networks. In *ACM CoNEXT*, 2016.

[5] B. Fortz and M. Thorup. Internet Traffic Engineering by Optimizing OSPF Weights. In *IEEE INFOCOM*, 2000.

[6] B. Fortz and M. Thorup. Optimizing OSPF/IS-IS Weights in a Changing World. *IEEE Journal on Selected Areas in Communications*, 2002.

[7] F. Giroire, J. Moulierac, and T. K. Phan. Optimizing Rule Placement in Software-defined Networks for Energy-aware Routing. In *IEEE Global Communications Conference (GLOBECOM)*, 2014.

[8] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan. Towards Wearable Cognitive Assistance. In *ACM MobiSys*, 2014.

[9] N. Laoutaris, G. Smaragdakis, K. Oikonomou, I. Stavrakakis, and A. Bestavros. Distributed Placement of Service Facilities in Large-scale Networks. In *IEEE INFOCOM*, 2007.

[10] J. Li, T. K. Phan, W. K. Chai, D. Tuncer, G. Pavlou, D. Griffin, and M. Rio. DR-Cache: Distributed Resilient Caching with Latency Guarantees. In *IEEE INFOCOM*, 2018.

[11] M. Mechtri, C. Ghribi, and D. Zeghlache. A Scalable Algorithm for the Placement of Service Function Chains. *IEEE Transactions on Network and Service Management*, 2016.

[12] A. M. Medhat, T. Taleb, A. Elmangoush, G. A. Carella, S. Covaci, and T. Magedanz. Service Function Chaining in Next Generation Networks: State of the Art and Research Challenges. *IEEE Communications Magazine*, 2016.

[13] H. Moens and F. D. Turck. VNF-P: A Model for Efficient Placement of Virtualized Network Functions. In *CNSM*, 2014.

[14] M. Mohri, A. Rostamizadeh, and A. Talwalkar. Foundations of Machine Learning. *The MIT Press*, 2012.

[15] J. Moulierac and T. K. Phan. Optimizing IGP Link Weights for Energy-efficiency in Multi-period Traffic Matrices. *Computer Communications*, 2015.

[16] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly. SNDlib 1.0–Survivable Network Design Library. *Networks*, 2010.

[17] T. K. Phan, D. Griffin, E. Maini, and M. Rio. Utility-maximizing Server Selection. In *IFIP Networking*, 2016.

[18] T. K. Phan, D. Griffin, E. Maini, and M. Rio. Utility-centric Networking: Balancing Transit Costs with Quality of Experience. *IEEE/ACM Transactions on Networking*, 2018.

[19] T. K. Phan, J. Moulierac, C. N. Tran, and N. Thoai. Xcast6 Treemap Islands: Revisiting Multicast Model. In *ACM CoNEXT Student Workshop*, 2012.

[20] T. K. Phan, M. Rocha, D. Griffin, and M. Rio. Utilitarian Placement of Composite Services. *IEEE Transactions on Network and Service Management*, 2018.

[21] J. Reis, M. Rocha, T. K. Phan, D. Griffin, F. Le, and M. Rio. Deep Neural Networks for Network Routing. In *International Joint Conference on Neural Networks (IJCNN)*, 2019.

[22] M. Rocha, P. Sousa, P. Cortez, and M. Rio. Evolutionary Computation for Quality of Service Internet Routing Optimization. In *Workshops on Applications of Evolutionary Computation (LNCS)*, 2007.

[23] M. Satyanarayanan, P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, W. Hu, and B. Amos. Edge Analytics in the Internet of Things. *IEEE Pervasive Computing*, 2015.

[24] P. Simoens, D. Griffin, E. Maini, T. K. Phan, M. Rio, L. Vermoesen, F. Vandeputte, F. Schamel, and D. Burstzynowski. Service-centric Networking for Distributed Heterogeneous Clouds. *IEEE Communications Magazine*, 2017.

[25] M. Stone and B. Moore. Tolerable Hearing Aid Delays. Est. of Limits Imposed by the Auditory Path Alone using Simulated Hearing Losses. *Ear and Hearing*, 20(3), 1999.

[26] R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction. *MIT press Cambridge*, 1998.

[27] A. G. Tasiopoulos, S. G. Kulkarni, M. Arumaithurai, I. Psaras, K. Ramakrishnan, X. Fu, and G. Pavlou. DRENCH: A Semi-Distributed Resource Management Framework for NFV based Service Function Chaining. In *IFIP Networking*, 2017.

[28] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar. Learning to Route. In *ACM Workshop on Hot Topics in Networks (HotNets)*, 2017.

[29] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar. Learning To Route with Deep RL. In *Conference on Neural Information Processing Systems (NIPS)*, 2017.

[30] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford. DONAR: Decentralized Server Selection for Cloud Services. In *ACM SIGCOMM*, 2010.

[31] Z. Zhang, Y. Hu, M.Zhang, R.Mahajan, A. Greeberg, and B. Christian. Optimizing Cost and Performance Online Service Provider Networks. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2010.