A Synchronous Multi-Step Algorithm for Flexible and Efficient Virtual Network Reconfiguration

Nguyễn Tuấn Khải, Andreas Baumgartner, Thomas Bauschert
Technische Universität Chemnitz
Chair of Communication Networks
09126 Chemnitz, Germany
[tuan-khai.nguyen | andreas.baumgartner | thomas.bauschert]@etit.tu-chemnitz.de

***Abstract*—Reconfigurations of virtual networks in cloud environments are enabled by virtual machine (VM) migration technologies. A lot of research work has focused on the implementation of VM migration (such as pre/post-copy techniques) and the determination of the destination hosts for the VM placement. The latter is often related to virtual network embedding (VNE) because it entails the mapping of virtual network functions and traffic flows on the physical substrate. However, a lack of consideration of VNE in the research on VM migration (and vice versa) still prevails. We consider this joint problem and observe that when one regards this as a multi-step model, remarkable benefits in terms of migration time, migration traffic volume, and even migration feasibility are obtained. In this paper, we present a heuristic algorithm to accelerate the solution and show that high-quality solutions can be achieved in polynomial time with suitable parameter settings.**

Keywords—virtual network embedding, VNE, VM migration, multi-step, heuristic

I. Introduction and Related Work

Virtualisation technologies have enabled many new business models and opened up extensive room for research on both *virtual network embedding* (VNE) [1] and *virtual machine* (VM) *migration* [2]. The objective of VNE is to realise service chains by means of VM instantiation and concatenation, while VM migration can be used to reconfigure them. Applying dynamic VM migration during operation provides several benefits such as reducing the rejection probability of new service chains, reducing energy costs, and improving efficiency due to less resource fragmentation.

Live migration ensures that the service downtime is kept minimum by applying two well-known methods: *pre-copy* and *post-copy*. The pre-copy method has the advantage of being robust to failures and easy to model but incurs a long migration time. The post-copy technique takes less time, but is prone to failures and often leads to severe service degradation due to *paging* across the network.

When VNE and VM migration are considered in tandem, the issue of simultaneous relocation of virtual network functions and migration of traffic flows needs to be addressed. For certain destinations, the migration might be infeasible due to depleted bandwidth. In fact, this issue has not been addressed adequately so far. It is often assumed that migration traffic flows are kept minimum, or that there is enough bandwidth to realise the migration. Zangiabady et al [3] take a reinforcement-learning approach to trigger the reallocation of resources at a given network state. The decision is made by a learning agent to minimise unnecessary migrations. In [4], migration flags are set on VMs that need to migrate based on VNE solutions, but no rules on which path or how much bandwidth is taken are specified. Reference [5] gives quite a comprehensive algorithm for optimising both VNE and VM migrations across data centers (DCs) that are triggered by fluctuations in energy prices, DC workloads, and renewable energy productions. The demand for migration is checked every hour, and some simplistic assumptions about migration techniques as well as the actual migration time are made, e.g. the migration bandwidth is fixed and the VM migration rates are averaged. Esposito and Cerroni in [6] and [7] develop a mathematical model to balance downtime and total migration time. The model is however based on quite immoderate assumptions. For instance, the number of copy rounds must be fixed, copy rounds of all VMs must be in sync, and migration flows can be mapped differently in each copy round. No scalability can be achieved with such a model that includes many details on copy-round level. In our work, we also consider several details influencing the migration but develop an approximation formula for the migration time.

978-3-903176-31-7 © 2020 IFIP

We seek to contribute to the combined VNE and VM migration problem by introducing a novel *multi-step* approach. All in all, the main issue with other work concerns bandwidth bottlenecks in the substrate. With constricted bandwidth, two options are available: either prolong the migration or cancel it. In such a case, our multi-step approach allows for two additional options: postpone the migration or redirect the migration traffic flows to so-called *stopover hosts*. By postponing the migrations to a later migration step, competition for bandwidth is alleviated. With multiple steps and stopover hosts, migration hotspots (i.e. congestion) caused by concurrent migration traffic flows are dissipated.

In our previous work [8] we introduced a mathematical model that provides the proof-of-concept for this multi-step paradigm. We modelled the multi-step migration problem as a series of VNEs that relate to each other via some migration constraints. Each VNE represents the end of one migration batch and the start of the next batch; all migrations within a step start synchronously. Interestingly, allowing for multiple migration steps can greatly reduce both migration time and migration traffic volume. Even the migration feasibility itself also improves. For better scalability, in this paper, we solve the virtual network reconfiguration problem on a per-step basis using the Ant Colony metaheuristic approach.

Our notion of multi-step migration is closest to that of *live multi-hop migration*, a patent claimed by Glikson et al [9]. Reference [10] also formulates a similar mathematical multi-step model in order to serve as *model predictive control*, but it is based on very simple assumptions without considering the migration process. To the best of our knowledge, these are the only studies that address issues similar to ours.

The rest of this paper is structured as follows: an outline of the problem statement is given in Chapter II. In Chapter III, we specify the heuristic algorithm used for solving the multi-step migration problem. In Chapter IV, first our performance evaluation setup with two different test scenarios is described, followed by a discussion on the obtained results. A brief summary of our work is provided in Chapter V.

II. Problem Description

A. Network Models

As in most VNE models, we start the problem formulation with the modelling of the substrate network and the virtual networks. Let $G^{\mathbf{S}} = (N^{\mathbf{S}}, L^{\mathbf{S}})$ be the substrate network with $N^{\mathbf{S}}$ and $L^{\mathbf{S}}$ being the substrate nodes and substrate links respectively. This could be the model of an inter-data-center network where each node represents a physical data center (DC) site. We assign for each node certain resources, which can be of any type, such as vCPU, memory, or storage. Substrate links connect substrate nodes and are directional. In addition, they exist in pairs of equal bandwidth and opposite directions to reflect full-duplex channels. This modelling of substrate nodes and links gives the definition of a *capacitated substrate network*.

For modelling convenience, we regard all service chains that should be embedded as a single large virtual network denoted as $G^{\mathbf{V}} = (N^{\mathbf{V}}, L^{\mathbf{V}})$. Note that this large virtual network may differ in each step of the multi-step migration, if there are changes in the bandwidth demand over time. Each virtual node v requires a share of the substrate nodes' resources which amounts to $d(v)$; each virtual link $(v, \tilde{v})$ is directional and characterised by some bandwidth demand $d(v, \tilde{v})$.

B. VNE and VM Migration

A VNE framework involves mappings of $N^{\mathbf{V}}$ onto $N^{\mathbf{S}}$ and of $L^{\mathbf{V}}$ onto substrate paths, which are subject to capacity and flow conservation constraints. These are commonly referred to as *node mapping* and *flow mapping*, respectively.

Each virtual node is typically realised via an instantiation of a VM, and thus can migrate from one host to another, in effect changing the node and flow mapping states. To reflect possible changes of the mapping states, we multiply the substrate network so that there are several duplicates of it. This effectively corresponds to solving multiple VNE problems where the same substrate network and virtual network are reused as input, but the mapping outcomes may be different.

To incorporate VM migration costs and coordinate the mappings across steps, we consider the *pre-copy* technique [11], in which the migration time depends on various factors such as memory footprint $\mathcal{M}$, writable working set $\mathcal{W}$, page-dirtying rate $\mathcal{R}$, and memory state transmission rate $\mathcal{L}$. Two hyper-parameters, namely minimum progress $\mathcal{X}$ and switch-over goal time $\mathcal{T}$, also govern the migration time. However, when the memory transmission rate significantly exceeds the page-dirtying rate, one can safely ignore $\mathcal{X}$ and $\mathcal{T}$ and achieve a conservative estimation of the migration time as follows:

$$t_{\mathbf{MIG}} = \frac{\mathcal{LM} - \mathcal{MR} + \mathcal{RW}}{\mathcal{L}(\mathcal{L} - \mathcal{R})} \qquad (1)$$

With the adoption of the pre-copy technique, multiple mappings can now be coupled due to certain migration-related constraints. Due to the multi-step structure where each step involves a different VNE, VM migrations occur in sync on a per-step basis, meaning that a new migration step can only start if the previous one is completely finished.

A mixed-integer-programming (MIP) model based on this formulation was specified in our previous work [8] which provided a compelling proof-of-concept for the multi-step VM migration in terms of total migration time, migration traffic volume, and even feasibility of the migration.

C. Limitations of the Mathematical Model

The biggest drawback of problems formulated as MIP models is that they are generally known to be $\mathcal{NP}$-hard and thus exhibit poor scalability. While there exist some metaheuristics to speed up the solution, sometimes the task of finding a feasible solution is challenging enough, as an extensive use of auxiliary variables and constraints is involved. Moreover, the number of migration steps must be over-estimated since it is not known beforehand, which increases the model complexity. In effect, most of the solution time is spent on computing the inverse of the incident matrix, which is already expensive. In fact, it was proven that, unless $\mathcal{P} = \mathcal{NP}$, no approximation algorithm exists to solve this type of problem in polynomial time [12]. Our primary objective is therefore not to accomplish (near-)optimal solutions, but rather to provide a feasible, high-quality solution, considering the computational effort. As such, we take a per-step approach to develop a heuristic algorithm, in which both the VNE and VM migration problems are solved for each step, and more computation time is spent on constructing and evaluating feasible solutions, as opposed to the holistic approach (i.e., all the steps are considered collectively).

D. Research Issues Related to the Development of the Heuristic Algorithm

We consider a two-phase problem, where the initial VNE is given and the target VNE is obtained in the primary decision phase (this type of problem is already addressed in several VNE papers, such as [1] and the references therein). We focus on the secondary phase where we seek to realise an optimum step-wise migration from the initial towards the target VNE. The objective is to minimise the total migration time, so that the target VNE state is reached as soon as possible. The idea of the per-step approach is to determine the transition from the current VNE state to the next, and further next, which is repeated until the target VNE state is reached. This problem could for instance be tackled by reinforcement learning (RL) [13], where the action space would involve selecting migration stopovers, and the state space would refer to all possible VNE states. However, since the use-case entails a rather dynamic environment and the sizes of both spaces increase exponentially with respect to the number of VMs, training this RL model would become costly due to the exploratory nature of the technique on both the action and state levels. We instead develop an algorithm that is mostly exploratory on the action level and greedier on the state level.

III. Heuristic Algorithm

The following per-step procedure is repeated until the target mapping state is reached: given a current node and flow mapping state, determine (i) a stopover host for each VM, (ii) on which paths to migrate the VMs, (iii) the required bandwidth for each migration path, and (iv) the next flow mapping.

The above procedure can be solved using any existing metaheuristic such as Simulated Annealing (SA) [14], Genetic Algorithm (GA) [15], or Ant Colony (AC) [16]. Except for SA which is single-agent-based, both GA and AC are population-based, where multiple solutions are evaluated in a row. Essentially, the performance of a certain metaheuristic depends very much on its hyper-parameter settings, with which one tries to balance exploration and exploitation in the heuristic search. We adopt the population-based AC method since it exhibits some characteristics from both SA and GA.

Specifically, initial temperature in SA and initial pheromone trails in AC both encourage exploration, whereas cooling rate in SA and evaporation rate in AC both promote exploitation. The population fitness improves over generations in GA thanks to the reproduction of the elite, and in AC, the population quality also improves over epochs thanks to the reinforcement of pheromone trails left behind by the best artificial ants.

A. *Searching for Node-mapping Candidates*

First, random sampling of the stopover host for each VM is conducted. For brevity, we define VMs that have not migrated yet to their target hosts as *unsettled VMs*. The set of VMs involved in this sampling is denoted by Q. In principle, Q can include any VMs, but one could consider unsettled VMs alone for computational cost-savings. The sampling is based on the probability distribution matrix P of size $|Q| \times |N^{\mathbf{S}}|$ where the row sums are equal to one. Each row of the matrix corresponds to a VM in Q, and each column to a substrate node. Associated with P is the pheromone trail matrix τ of the same shape. Each entry of P depends on the respective entry of τ. For each VM $v \in Q$, the higher the value of $\tau(v,s)$, the more likely the substrate node s is selected as the next stopover host. The sampling is more exploratory when many entries in a row are relatively high, and greedier when very few entries are relatively high. The matrix τ is updated after every epoch by evaporation and reinforcement, which promotes exploitation in subsequent epochs. Therefore, to encourage exploration during the first epochs, τ is generally initialised with some positive values.

Some heuristic information can also be leveraged. We introduce a heuristic information matrix η with the same shape as P. Now each entry of P is calculated as follows:

$$P(v,s) = \frac{\eta(v,s)\cdot\tau(v,s)}{\sum_{\tilde{s}\in N^{\mathbf{S}}}\eta(v,\tilde{s})\cdot\tau(v,\tilde{s})}, \qquad \forall v \in Q, s \in N^{\mathbf{S}}$$

Each sampling is done independently by an artificial ant, which is a search agent responsible for finding and evaluating a solution candidate. Let K denote the set of ants. The node-mapping candidates found by the ants' samplings are denoted by a matrix x of size $|K| \times |Q|$. Each node-mapping state is realised via the migrations of VMs. However, to ensure its feasibility, one needs to refine the node-mapping candidates.

B. *Refinement of Solution Candidates*

1) *Ensure Migration Feasibility*

For bandwidth-intensive processes like VM migration, to find the migration path, one possibility is to use the widest path (WP) [17]. However, the use of WPs could lead to quick depletion of bandwidth due to potentially large hop counts. An alternative is to use the shortest path (SP) [18], where we assign to each link a link cost of 10^8 times the inverse of its bandwidth [19]. The path cost then depends on both bandwidth and hop count.

Given the residual bandwidth as a result of the current traffic flow-mapping, the migration time can be estimated using Equation (1). The estimated migration time (EMT) can be prolonged by means of bandwidth throttling, but cannot be shortened without severely affecting the running services. For synchronous migrations, we define the VM with the largest EMT as the *lead VM*, whose EMT we name *designated migration time* (DMT). Once the lead VM migrates, the EMTs of the following VMs are re-estimated. If their re-estimated EMTs exceed the DMT, their next host candidates are reset to the current ones (denoted by $x_{\mathbf{cur}}$). To minimise the number of such migration-deferred VMs, the remaining VMs with EMTs lower than DMT can migrate with their migration bandwidth throttled since their migration time can be prolonged. Procedure 1 is executed by each artificial ant k, wherein x_k denotes the k'th row of the matrix x.

```
Procedure 1: Refine node-mapping candidate
INPUT: node-mapping candidate $x_k$ and lead VM $v_{\mathbf{lead}}$
OUTPUT: refined node-mapping candidate $x_{\mathbf{ref}}$
1  $x_{\mathbf{ref}} \coloneqq$ duplicate($x_k$)
2  for each $v \in Q$, starting with $v_{\mathbf{lead}}$:
3      Re-estimate EMT
4      if EMT > DMT:
5          $x_{\mathbf{ref}}(v) \coloneqq x_{\mathbf{cur}}(v)$
6      else:
7          Assume $v$ migrates to $x_{\mathbf{ref}}(v)$ with bandwidth throttled
8          Update residual substrate resource capacities
```

Note that bandwidth throttling should only be applied sparingly as the estimation of migration time becomes less accurate with too much throttling. In addition, some VMs may finish the migration before others. If this happens to two VMs that are both exchanging traffic and the time lag is critical, a temporary tunnel may be established to forward outstanding packets until the new traffic flow re-mapping is realised. More detailed discussion on this issue can be found in [20] and [21]. For simplicity, we do not consider this.

2) *Ensure Traffic Flow-mapping Feasibility*

We compute the *constrained SP* to re-map each virtual flow, subject to its bandwidth demand. If its bandwidth demand cannot be fulfilled, we reject the candidate. Procedure 2 is executed after Procedure 1 by the same ant.

```
Procedure 2: Compute flow-mapping candidate
INPUT: refined node-mapping candidate $x_{\mathbf{ref}}$
OUTPUT: flow-mapping candidate $f_{\mathbf{can}}$
1  for each $(v,\tilde{v}) \in L^{\mathbf{V}}$:
2      $pth \coloneqq$ constrained SP based on $d(v,\tilde{v})$
3      if $pth$ is found:
4          $f_{\mathbf{can}}(v,\tilde{v}) \coloneqq pth$
5          Update residual substrate bandwidth
6      else:
7          Reset residual substrate bandwidth
8          return error
```

In case the node-mapping candidate from Procedure 1 is rejected in Procedure 2 (by returning an error), we start over and attempt to refine the node-mapping candidate again by taking the VM with the next largest EMT instead as the alternative lead VM. Procedure 3 represents the total work done by an ant in order to find the heuristic reward.

```
Procedure 3: Find heuristic reward
INPUT: node-mapping candidate $x_k$
OUPUT: heuristic reward $h_k$
1  $v_{\mathbf{lead}} \coloneqq$ VM with largest EMT
2  $x_{\mathbf{ref}} \coloneqq$ Procedure 1($x_k, v_{\mathbf{lead}}$)
3  $f_{\mathbf{can}} \coloneqq$ Procedure 2($x_{\mathbf{ref}}$)
4  if $f_{\mathbf{can}}$ is found:
5      Calculate heuristic reward $h_k$
6      go to line 10
7  else:
8      $v_{\mathbf{lead}} \coloneqq$ VM with next largest EMT
9      go to line 2
10 end
```

C. *Quality of Solution Candidates*

Solution quality can rely on the lead VM. Two ants could nominate almost identical solutions but with different lead VMs, so one DMT would be shorter than the other. However, too short DMT may allow very few VMs to migrate, so with many unsettled VMs, many migration steps would result.

To measure the solution quality, we first define the VNE state quality based on the VMs' closeness to their final destinations. We use EMTs from the current to the target hosts for the estimation of closeness and thus for the evaluation of a state. Our best guess of the EMTs is based on the *status quo* of the residual bandwidth due to the current traffic flow mapping. The quality of a current state is therefore:

$$g(state_{\mathbf{cur}}) \coloneqq \sum_{v \in Q} \mathrm{EMT}\left(v, \mathrm{SP}\big(x_{\mathbf{cur}}(v), x_{\mathbf{dst}}(v)\big)\right)$$

where $\mathrm{EMT}(v,p)$ denotes the EMT of a VM v that would migrate on a path p, and $\mathrm{SP}(s,t)$ is the SP from s to t.

We proceed now to the definition of the migration candidate quality. Each action (i.e., the selection of stopover hosts) leads to a new VNE state, and thus is incentivised by the state quality. However, identical actions in different states need not be identically incentivised. In other words, the same action might result to good outcomes in some states but bad outcomes in other states. To mitigate such adverse decisions, an incentive to take an action should take into account the migration time from the current to the next state candidate. Our definition of an action incentive is thus:

$$h = \frac{g(state_{\mathbf{cur}}) - g(state_{\mathbf{nxt}})}{\mathrm{DMT}}$$

The action with the highest incentive will be eventually taken. This incentive serves as heuristic reward for each ant and contributes to the pheromone trails.

D. Update Pheromone Trails

An epoch is a period during which the ants search for migration solutions, refine them, and evaluate them. The pheromone trails are initialised to some values and updated after every epoch by means of evaporation and reinforcement. The pheromone update involves first multiplying the pheromone trails by an evaporation rate $b < 1$, followed by the addition of fresh pheromones that depend on the heuristic reward h (i.e., incentive). For simplicity, we define the fresh pheromone value of an ant k as $\tau_{\mathbf{fresh}}(k) = \max(h_k, 0)$. The pheromone update is then expressed as follows:

$$\tau(v,s) \coloneqq b \cdot \tau(v,s) + \sum_{k \in K : x(k,v) = s} \tau_{\mathbf{fresh}}(k)\, , \forall v \in Q, s \in N^{\mathrm{S}}$$

Some research on AC has shown that a major performance boost can be accomplished with greedier reinforcement of the pheromone trails. To achieve this, the *MAX-MIN Ant System* [22] suggests that only the best ant may deposit its fresh pheromones. With this alternative configuration, the update of the pheromone trails after an epoch becomes:

$$\tau(v,s) \coloneqq b \cdot \tau(v,s) + \tau_{\mathbf{fresh}}(k_{\mathbf{best}}),$$
$$\forall v \in Q, s \in N^{\mathrm{S}} : x(k_{\mathbf{best}}, v) = s$$

wherein $k_{\mathbf{best}}$ is either the best ant so far (*global-best ant*), or the best ant in the current epoch (*epoch-best ant*).

E. Heuristic Information

Some *a priori* information can be extracted to compute the η matrix. One option is to consider the constraints of the substrate network. For example, some substrate nodes cannot host certain VMs due to the lack of resources, they lie on highly congested paths, or their paths to the target host traverse many hops. These substrate nodes should be rarely explored. To quantify heuristic information for a stopover host candidate, the SPs from the current host to the host candidate, and in turn, from the host candidate to the target host can be used. Procedure 4 calculates the matrix η.

```
Procedure 4: Calculate heuristic information matrix $\eta$
INPUT: current mapping state
OUTPUT: heuristic information matrix $\eta$
1  for each $v \in Q, s \in N^{\mathrm{S}}$:
2      $p_1 \coloneqq \mathrm{SP}(\text{from} \coloneqq x_{\mathbf{cur}}(v), \text{to} \coloneqq s)$
3      $p_2 \coloneqq \mathrm{SP}(\text{from} \coloneqq s, \text{to} \coloneqq x_{\mathbf{dst}}(v))$
4      if $s$ has enough capacities for $v$:
5          $\eta(v,s) \coloneqq \frac{\mathrm{bottleneck}(p_1, p_2),}{c \cdot \mathrm{pathcost}(p_2) + 1}$
6      else:
7          $\eta(v,s) \coloneqq 0$
```

where $\mathrm{bottleneck}(p_1, p_2)$ is the bottleneck of the path composed of p_1 and p_2, and $\mathrm{pathcost}(p)$ is the sum of the link costs constituting p. The parameter c is a normalising multiplier such that on average, the product $c \cdot \mathrm{pathcost}(p_2)$ is approximately in the order of one.

F. All Negative Rewards Case

Our algorithm is greedy on the state level, meaning that the global-best ant's solution candidate is eventually selected to realise the next state, provided that the heuristic reward is positive. However, it is possible that none of the rewards are positive, in which case we take an exploratory action by choosing the solution candidate with minimum reward. We name the respective ant *negative-global-best ant.* The following pseudo-code summarises the overall procedure:

```
Main Procedure
1  while unsettled VMs persist:
2      Calculate $\eta$ with Procedure 4
3      Initialise $\tau$
4      repeat $\mathrm{EPOCHS}_{\mathbf{MAX}}$ times:
5          Calculate $P$ based on $\eta$ and $\tau$
6          Sample node-mapping matrix $x$ using $P$
7          for each $k \in K$:
8              Calculate heuristic reward $h_k$ with Procedure 3
9              $\tau_{\mathbf{fresh}}(k) \coloneqq \max(h_k, 0)$
10         Evaporate and reinforce $\tau$ with $\tau_{\mathbf{fresh}}$
11         if $h_{\mathbf{MAX}} > 0$:
12             Execute global-best ant's solution
13         else:
14             Execute negative-global-best ant's solution
```

IV. Performance Evaluation

A. Experiment Setup

We select a random substrate network topology with 50 nodes from SNDLib [23]. The minimum degree is two, the maximum degree is five, and the probability that a link exists between any two nodes is 7%. The node resource capacities and link bandwidths are selected randomly based on a realistic set of values given by a German network operator. As for the virtual network, we examine two test scenarios: one with 50 service chains (SCs) and another with 70 SCs. Each SC is generated randomly with 3–4 VMs and with random resource demands. The page dirtying rate $\mathcal{R}$ and writable working set $\mathcal{W}$ of each VM are also randomised. We assume that the memory footprint $\mathcal{M}$ is also the memory demand. Both the minimum progress amount $\mathcal{X}$ and switch-over goal time $\mathcal{T}$ are set to 0, so migration can only take place if the migration rate $\mathcal{L}$ exceeds the page-dirtying rate $\mathcal{R}$. The initial and target VNEs are determined by solving two optimisation problems with randomly chosen objectives.

As a reference migration solution for each scenario, we solve the MIP problem developed in [8] using the CPLEX MIP solver v12.9.0 on a 12-threaded Intel i7-3930K 3.2 GHz machine. The solution time is limited to 1.5 hours. Note that it is not possible to compare our work against [9] as [9] only provides general guidelines. The MPC model in [10] cannot be compared either, as the migration processes in [10] are neglected.

B. Parameter Settings

A number of factors can influence the heuristic results. For instance, increasing either the population size (i.e., the number of ants) or the number of epochs increases the chance of reaching optimality in each migration step. This is because either of these settings encourages both exploitation and exploration. Table I lists the main factors that influence the exploitation and exploration behaviour of the AC metaheuristic. Note that lowering the evaporation rate b means more evaporation since pheromones evaporate by multiplying themselves by b. For the deposit of fresh pheromone trails, we select the global-best ant with 20% probability and the epoch-best ant with 80% probability.

TABLE I. INFLUENCE FACTORS OF THE METAHEURISTIC

Influence Factor	Exploit	Explore
Population size	✓	✓
Number of epochs	✓	✓
Initial pheromone trails		✓
Evaporation	✓	

While it is not yet certain how our per-step objective correlates to the overall migration time, we aim to evaluate the influence of different parameter settings on the performance of our algorithm holistically. We also constrain the parameters based on the assumption that the computational effort is limited. Heuristically, we regard the computational effort as the product of the population size and the number of epochs. For example, deploying 60 artificial ants in 20 epochs requires $60 \cdot 20 = 1200$ computational units. Deploying either 1 ant in 1200 epochs or 1200 ants in 1 epoch corresponds to the same 1200 computational units. These two settings represent two extreme cases with a small population existing in many epochs vs. a large population existing in very few epochs. We fix 1200 as the limit of the heuristic computational effort.

The order of magnitude of fresh pheromone trails deposited by the artificial ants depends largely on both the test scenario and the current state. Therefore, we refrain from initialising the pheromone trails until the end of the first epoch (epoch 0) when every ant's reward is calculated. The initial pheromone trails are then obtained by multiplying the maximum absolute value of the reward by a so-called *initial pheromone scalar* (denoted m). This is because heuristically, we consider the maximum reward—in terms of its magnitude—acquired from epoch 0 to be representative of the overall order of magnitude of the heuristic reward, as far as the current state is concerned. Five settings of m are evaluated, namely 0.1, 1, 10, 100, and 1000. We also consider four different values of b, namely 0.5, 0.8, 0.9, and 0.95.

C. Results

Our heuristic algorithm is implemented in Python [24]. The results are all feasible and obtained within 1–20 minutes each. This is a significant improvement in the solution time, considering the fact that the C-core of CPLEX often takes at least 30 minutes to find the first feasible solution.

TABLE II. BEST AND WORST PARAMETER SETTINGS FOR TEST SCENARIO 1 (WITH 50 SERVICE CHAINS)

		Ants	Epochs	m	b	Norm. mig. time
Best	Setting 1	150	8	0.1	0.5	1.081
	Setting 2	240	5	0.1	0.5	1.083
	Setting 3	80	15	0.1	0.9	1.084
Worst	Setting 4	2	600	0.1	0.5	1.488
	Setting 5	1	1200	1	0.5	1.498
	Setting 6	1	1200	0.1	0.5	1.724

TABLE III. BEST AND WORST PARAMETER SETTINGS FOR TEST SCENARIO 2 (WITH 70 SERVICE CHAINS)

		Ants	Epochs	m	b	Norm. mig. time
Best	Setting 7	80	15	0.1	0.8	1.363
	Setting 8	150	8	0.1	0.8	1.367
	Setting 9	30	40	0.1	0.95	1.368
Worst	Setting 10	1	1200	0.1	0.8	1.896
	Setting 11	1	1200	1	0.5	1.995
	Setting 12	1	1200	0.1	0.5	2.182

We normalise the total migration time obtained from the heuristic algorithm by dividing it by the one obtained from CPLEX. Quite a few good parameter settings yield high-quality results, but for brevity, we exemplify three best parameter settings and three worst parameter settings for each of the test scenarios in Tables II and III. The quality of each parameter setting is evaluated based on the mean normalised total migration time (i.e. the right-most column). Note that these are best and worst settings under the constraint on the computational effort, so the products of the "Ants" and "Epochs" columns are constant (i.e. 1200).

It can be seen in Table II that with lower complexity (with 50 SCs), a larger population size and a more aggressive exploitation in terms of pheromone trails are preferable. In Table III, for higher complexity (with 70 SCs), to achieve good-quality results, more epochs (and hence fewer ants) with less aggressive pheromone evaporation are favourable. In other words, in case of higher complexity, when the computational effort is limited, more time should be spent on exploration.

Both Tables II and III also indicate that a heuristic search with very few agents often yields outcomes of inferior quality. Although not shown here, even a more exploratory single-agent heuristic search (e.g. by means of higher initial pheromone scalar and less evaporation of pheromones) does not give good-quality solutions either.

Figures 1 (a) and 2 (a) show the cumulative probabilities of the normalised total migration time for the two test scenarios. In some cases, our algorithm outperforms CPLEX (with solution time constraint 1.5 hours), yielding normalised total migration times lower than one. In these Figures, the blue curves (no setting) represent 1200 ants and only one epoch. This is when the parameters initial pheromone scalar and evaporation rate play no role; the outcomes then merely result from the heuristic information matrix η.

Every migration step could incur a certain overhead for negotiating, establishing, and realising the migration traffic flows. Therefore, the number of migration steps might be important if administrative costs are considered. Furthermore, the runtime of our heuristic algorithm linearly depends on the number of migration steps. In this regard, fewer migration

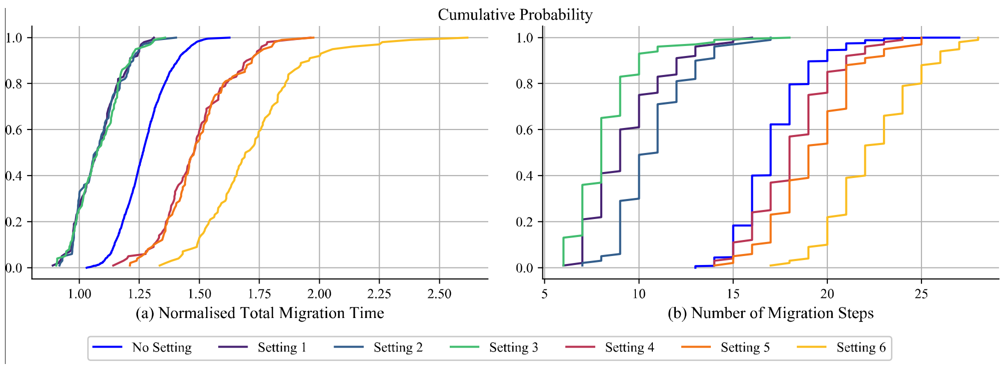

Fig. 1. Cumulative Probability of Normalised Total Migration Time and Number of Migration Steps (with 50 Service Chains)

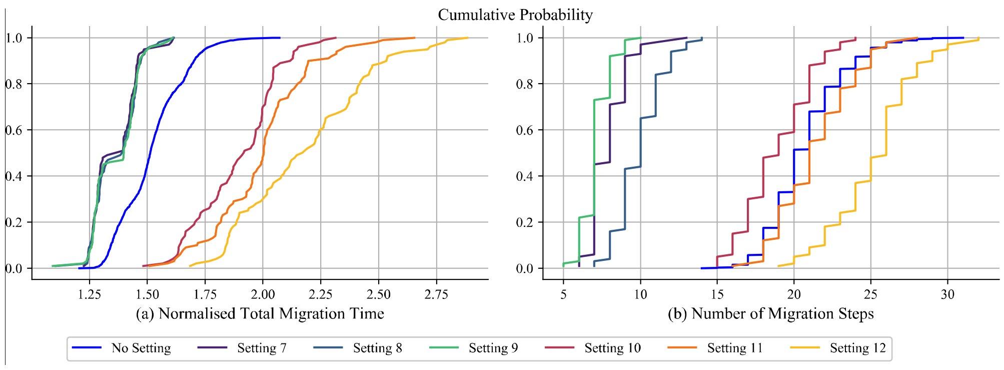

Fig. 2. Cumulative Probability of Normalised Total Migration Time and Number of Migration Steps (with 70 Service Chains)

steps incur less overhead and thus are preferable. We also show the cumulative probability graphs for the number of migration steps in Figures 1 (b) and 2 (b). If the number of migration steps is taken as the main objective, the best and worst parameter settings would certainly be somewhat different from that shown in Tables II and III.

With only the heuristic matrix η (i.e., with one epoch), one can achieve a fairly low total migration time (although not as low as the outcomes of the best settings), but many migration steps are incurred. This is because stopover hosts that are in the vicinity of the final destinations are preferred and a lot of unsettled VMs get deferred due to too much competition for bandwidth (see Procedure 1). As a result, only a small number of VMs get settled in each step, which in turn leads to many steps. Retuning the parameter c in the calculation of η might somewhat mitigate this issue.

It should be noted that the observations described above are based on the pre-setting of the heuristic computational effort to 1200. For larger problems, more computational effort would be required to yield comparable outcomes. However, we do not expect the increase in computational effort to be linear due to the complexity proof in [12].

Although not shown here, another minor observation from our study is that initialising pheromone trails to large values in the order of 10–1000 does not help much, even in the presence of many epochs. However, we expect that these large initial pheromone scalars could play a more significant role, in case the limit in computational effort increases.

V. Conclusion

In this paper, we adapt the concept of multi-step virtual network reconfiguration by means of VM migration, which was first formulated in our previous work [8] to allow for synchronous multi-step migration. The introduction of this novel approach of virtual network reconfiguration is imperative for efficiency and even feasibility. When an expensive reconfiguration process involves migration of many VMs, links might get crammed and cause severe congestions. Indeed, it has been proven that these congestions lead to a prolonged migration time and an increase in migration traffic volume, which can be very efficiently mitigated by enabling multi-step migration.

Due to the $\mathcal{NP}$-hardness of the problem formulated in [8], we develop a heuristic algorithm to speed up the solution for scalability. As described in Section III, the heuristic algorithm is based on the population-based Ant Colony metaheuristic whose runtime depends linearly on the number of VMs and the number of migration steps; the latter can be predicted to some extent based on the parameter setting.

The algorithm is evaluated with various parameter settings using two test scenarios that correspond to different utilisations of the substrate network as well as problem complexities. We define the heuristic computational effort as the product of the population size and the number of epochs and use the normalised total migration time and the number of migration steps as performance metrics. By carrying out experiments with different parameter settings and taking into account the heuristic computational effort limit, the best parameter settings can be determined so that our algorithm will either target towards minimising the migration time or towards minimising the number of migration steps.

The evaluation is by no means comprehensive, but the obtained results indicate that our algorithm can be used as a tool for generating high-quality multi-step solutions with appropriate parameter settings. In addition, with a solution time that is considerably faster than with CPLEX, we also envisage its use for near real-time cloud resource management.

References

[1] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, “Virtual network embedding: A survey,” IEEE Communications Surveys & Tutorials, vol. 15, no. 4, pp. 1888–1906, 2013.

[2] V. Medina and J. M. García, “A survey of migration mechanisms of virtual machines,” ACM Comput. Surv., vol. 46, no. 3, 2014, Art. no. 30.

[3] M. Zangiabady, A. Garcia-Robledo, J.-L. Gorricho, J. Serrat-Fernandez, and J. Rubio-Loyola, “Self-adaptive online virtual network migration in network virtualization environments,” Transactions on Emerging Telecommunications Technologies, vol. 30, no. 9, p. e3692, 2019.

[4] G. Schaffrath, S. Schmid, and A. Feldmann, “Optimizing long-lived cloudnets with migrations,” in Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on. IEEE, 2012, pp. 99–106.

[5] D. Laganà, C. Mastroianni, M. Meo, and D. Renga, “Reducing the operational cost of cloud data centers through renewable energy,” Algorithms, vol. 11, no. 10, p. 145, 2018.

[6] F. Esposito and W. Cerroni, “Geomig: Online multiple vm live migration,” in Cloud Engineering Workshop (IC2EW), 2016 IEEE International Conference on. IEEE, 2016, pp. 48–53.

[7] W. Cerroni and F. Esposito, “Optimizing live migration of multiple virtual machines,” IEEE Transactions on Cloud Computing, vol. 6, no. 4, pp. 1096–1109, 2016.

[8] N. T. Khải, A. Baumgartner, and T. Bauschert, “Optimising virtual network functions migrations: A flexible multi-step approach,” in 2019 IEEE Conference on Network Softwarization (NetSoft). IEEE, 2019, pp. 188–192.

[9] A. Glikson, A. Israel, and D. Har’el Lorenz, “Live multi-hop vm remote-migration over long distance,” Feb. 5 2013, uS Patent 8,370,473.

[10] K. Kawashima, T. Otoshi, Y. Ohsita, and M. Murata, “Dynamic placement of virtual network functions based on model predictive control,” in NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium. IEEE, 2016, pp. 1037–1042.

[11] V. Mann, A. Gupta, P. Dutta, A. Vishnoi, P. Bhattacharya, R. Poddar, and A. Iyer, “Remedy: Network-aware steady state vm management for data centers,” NETWORKING 2012, pp. 190–204, 2012.

[12] M. Rost and S. Schmid, “On the hardness and inapproximability of virtual network embeddings,” IEEE/ACM Transactions on Networking, vol. 28, no. 2, pp. 791–803, 2020.

[13] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press, 2018.

[14] P. J. Van Laarhoven and E. H. Aarts, “Simulated annealing,” in Simulated annealing: Theory and applications. Springer, 1987, pp. 7–15.

[15] D. E. Goldberg, Genetic algorithms. Pearson Education India, 2006.

[16] M. Dorigo, M. Birattari, and T. Stutzle, “Ant colony optimization,” IEEE computational intelligence magazine, vol. 1, no. 4, pp. 28–39, 2006.

[17] V. S. Donavalli, “Algorithms for the widest path problem,” 2013.

[18] G. Gallo and S. Pallottino, “Shortest path algorithms,” Annals of operations research, vol. 13, no. 1, pp. 1–79, 1988.

[19] S. Halabi, “Ospf design guide,” Cisco Systems Network Supported Accounts, 1996.

[20] S. Ghorbani, C. Schlesinger, M. Monaco, E. Keller, M. Caesar, J. Rexford, and D. Walker, “Transparent, live migration of a software-defined network,” in Proceedings of the ACM Symposium on Cloud Computing, 2014, pp. 1–14.

[21] O. Michel, E. Keller, and F. M. Ramos, “Network defragmentation in virtualized data centers,” in 2019 Sixth International Conference on Software Defined Systems (SDS). IEEE, 2019, pp. 17–24.

[22] T. Stützle and H. H. Hoos, “Max–min ant system,” Future generation computer systems, vol. 16, no. 8, pp. 889–914, 2000.

[23] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski, “Sndlib 1.0—survivable network design library,” Networks, vol. 55, no. 3, pp. 276–286, 2010.

[24] G. Van Rossum et al., “Python programming language.” in USENIX annual technical conference, vol. 41, 2007, p. 3.