A Memetic Algorithm for the Virtual Network Mapping Problem

Inführ, Johannes and Raidl, Günther

Forschungsbericht / Technical Report

TR–186–1–14–01

January 15, 2014

Submitted to **Journal of Heuristics**

A Memetic Algorithm for the Virtual Network Mapping Problem

Johannes Inführ · Günther Raidl

Abstract The Internet has ossified. It has lost its capability to adapt as requirements change. A promising technique to solve this problem is the introduction of network virtualization. Instead of directly using a single physical network, working just well enough for a limited range of applications, multiple virtual networks are embedded on demand into the physical network, each of them perfectly adapted to a specific application class. The challenge lies in mapping the different virtual networks with all the resources they require into the available physical network, which is the core of the Virtual Network Mapping Problem. In this work, we introduce a Memetic Algorithm that significantly outperforms the previously best algorithms for this problem. We also offer an analysis of the influence of different problem representations and in particular the implementation of a uniform crossover for the Grouping Genetic Algorithm that may also be interesting outside of the Virtual Network Mapping domain. Furthermore, we study the influence of different hybridization techniques and the behaviour of the developed algorithm in an online setting.

Keywords Virtual Network Mapping · Memetic Algorithm · Hybrid Metaheuristic · Grouping Genetic Algorithm

This work has been funded by the Vienna Science and Technology Fund (WWTF) through project ICT10-027.

Johannes Inführ (✉)
Algorithms and Data Structures Group, Vienna University of Technology, Favoritenstraße 9–11/1861, 1040 Vienna, Austria
E-mail: infuehr@ads.tuwien.ac.at

Günther Raidl
Algorithms and Data Structures Group, Vienna University of Technology, Favoritenstraße 9–11/1861, 1040 Vienna, Austria
E-mail: raidl@ads.tuwien.ac.at

1 Introduction

As it exists today, the Internet suffers from ossification (National Research Council 2001). It is hard or even impossible to change existing protocols or introduce new technology to the Internet, even though changes would bring large improvements to the service quality. Examples of protocols that never saw wide-spread adoption include Explicit Congestion Notification (Ramakrishnan et al 2001) or Differentiated Services, which is a quality of service framework (Carlson et al 1998). The best current example for the ossification of the internet is IPV6 (Deering and Hinden 1998), which was introduced more than 15 years ago and is still not implemented completely despite the obvious demand.

The reasons for ossification are manifold. The history of the Internet shows that fundamental changes only occur if the network is about to collapse or if there is some immediate monetary gain. At the moment, the Internet is able to cope with the traffic demand and achieving monetary gain by improving the core Internet protocols is hard, because the Internet Service Providers (ISPs) need to agree on the changes. If all ISPs offer the same improvement, then there is no benefit for any of them (Anderson et al 2005). In National Research Council (2001), the general diagnosis of ossification is further refined. First, there is intellectual ossification. Any new technology has to be compatible with the current technology from the outset. This stifles innovation. Secondly, there is infrastructure ossification. Suggested improvements are not deployed in the infrastructure, not even for testing purposes. Thirdly, there is system ossification. Instead of fixing problems at their roots, workarounds and fixes are employed to keep the system running, making it more fragile and susceptible to even more problems (Anderson et al 2005).

Any solution to the ossification problem needs to have two properties if it wants to have any hope of actually being deployed. It needs to be backwards compatible and incrementally deployable. Network virtualization has been put forward as a suitable candidate (Touch et al 2003; Gold et al 2004; Anderson et al 2005; Tutschku et al 2008; Berl et al 2010).

The basic idea of network virtualization is straight forward. Instead of using one physical network that can do everything well, use multiple virtual networks embedded in the physical network. With network virtualization in place, changes to the underlying technology of the Internet can be implemented in an incremental and non-disruptive manner, because old and new technologies can coexist in different virtual networks. However, network virtualization has more to offer than being a crutch for switching protocols. At the moment, the Internet is a general purpose network, which supports a lot of applications rather well. With network virtualization, each application can have its own virtual network, perfectly adapted to the requirements of the particular application. Turner and Taylor (2005) describe a virtual network offering a learning environment with high quality audio and video multicast mechanisms. Format translators are available within the network to enhance compatibility. To allow for these kinds of applications, the nodes of the virtual networks receive compute capabilities within the routers of the physical network. Therefore, virtual networks can offer their own (and application specific) topology, routing, naming services, and resource management (Tutschku 2009). Network virtualization is already deployed in scientific network testbeds such as GENI (GENI.net 2012), PlanetLab (Chun et al 2003) and G-Lab (Schwerdel et al 2010), not as an enhancement to be studied, but as a central enabling technology for carrying out experiments. Virtual networks are used to partition the network testbeds so that different research groups can perform their experiments without interference from each other. For a survey on network virtualization, its application and available technologies, see Chowdhury and Boutaba (2010).

The Virtual Network Mapping Problem (VNMP) arises in this context. Even if there are multiple virtual networks with different characteristics and protocols, they still have to share the physically available resources in such a way that every network fulfills the required specifications, for instance with respect to quality of service parameters such as communication bandwidth and delay.

In this work, we will introduce a Memetic Algorithm (MA) that significantly outperforms the best previously available algorithms for the VNMP (Inführ and Raidl 2013b). In addition, we will answer the following questions in the context of the VNMP:

– What is the influence of different solution representations and crossover operators?
– Does the crossover operation have a beneficial impact on the final outcome?
– Is the time for local improvement well spent?
– How do different alternatives for local improvement perform?
– Is it beneficial to keep the population of solutions when the virtual networks change in an online setting?

The work presented here is a substantial extension of Inführ and Raidl (2013a). It contributes an analysis of refinement techniques and of the online behaviour of the presented MA.

Section 2 presents the formal definition of the VNMP, followed by a discussion of the relevant background and related work in Section 3. The proposed Memetic Algorithm is outlined in Section 4 and Section 5 contains the computational results. We conclude in Section 6.

2 The Virtual Network Mapping Problem

To specify a VNMP instance, three types of information are required: The substrate network to host the virtual networks (i.e. the physical network), the virtual networks (VNs) that need to be realized and the assignment constraints between virtual networks and substrate network.

The substrate network is modeled by a directed graph $G = (V,A)$. Each substrate node $i \in V$ has an associated CPU power $c_i \in \mathbb{N}^+$ which is used by the VN nodes mapped to i, but also to route BW. We assume that one unit of BW traversing a substrate node requires one unit of CPU power. This traversing BW could be internal to the substrate network, i.e. could be sent from and forwarded to another substrate node. It could also be sent or received by a virtual node mapped to the substrate node, and it is possible that both sending and receiving virtual nodes are mapped to the same substrate node. Substrate arcs $e \in A$ have a BW capacity $b_e \in \mathbb{N}^+$ and a delay $d_e \in \mathbb{N}^+$ that is incurred when data is sent across e.

The VNs are modeled by the individual connected components of a directed graph $G' = (V', A')$ with node set V' and arc set A'. Each VN node $k \in V'$ requires CPU power $c_k \in \mathbb{N}^+$ to implement custom protocols. Each VN arc $f \in A'$ has a bandwidth (BW) requirement $b_f \in \mathbb{N}^+$ and a maximum allowed delay $d_f \in \mathbb{N}^+$.

The set $M \subseteq V' \times V$ defines for each virtual node $k \in V'$ the substrate nodes that can be used to host it. The mapping of virtual nodes to substrate nodes has to be restricted, because a virtual node should be mapped close to its users. By $s(a)$ and $t(a)$, $\forall a \in A \cup A'$, we denote the arc's source and target nodes, respectively.

A valid VNMP solution specifies a mapping $m : V' \rightarrow V$ of virtual nodes to substrate nodes such that $(k, m(k)) \in M$, $\forall k \in V'$ and the total CPU load on each $i \in V$ (caused by mapped virtual nodes and traversing BW) does not exceed c_i. In addition, there has to be a simple substrate path $P_f \subseteq A$ from $m(s(f))$ to $m(t(f))$ implementing every $f \in A'$ that does not exceed the allowed delay d_f. The implementing paths have to respect the bandwidth capacities b_e on the substrate arcs and the CPU capacities on the substrate nodes.

The objective of the VNMP is to minimize the total substrate usage cost. Every substrate node $i \in V$ has an associated usage cost $p_i^V \in \mathbb{N}^+$ which has to be paid when at least one VN node uses it. Furthermore, every substrate arc $e \in A$ has a usage cost $p_e^A \in \mathbb{N}^+$ which has to be paid when it is used by at least one virtual arc. The total substrate usage cost C_u is the sum of the node and arc usage costs that have to be paid. The motivation for this cost function is that the infrastructure that gets used by the implementation of any virtual network has to be kept running. The unused parts can be shut down, which saves operational costs.

Already finding a valid solution to the VNMP is NP-hard (Inführ 2013, chap. 3). Therefore we cannot expect optimization techniques to always be able to find valid solutions (which may not even exist) within practical time. However, just reporting that no valid solution could be found is unsatisfactory for two reasons: for optimization purposes, there should be a way to distinguish between invalid solutions, to prefer those closer to validity and for practical purposes we would like to have a recourse strategy available so that the required virtual network load can be implemented. To be able to do that, we allow increasing the available CPU power at each substrate node, with the price of C^{CPU} per unit and the available BW on substrate arcs with the price of C^{BW} per unit. The sum of the incurred costs is the additional resource cost C_a. If a solution to a VNMP instance is valid, $C_a = 0$. We will call an instance solved if a valid solution could be obtained. As in previous work (Inführ and Raidl 2013b), we set $C^{\text{CPU}} = 1$ and $C^{\text{BW}} = 5$ to reflect the fact that it is easier to increase the CPU power of a router than to increase the BW of a network connection. We do not consider changing the delay of a substrate arc because of two reasons. First of all, it is difficult to change the delay of a connection in practice, because it is dependent on the employed technology. Changing the delay would typically mean changing the technology, which is very expensive. Secondly, given some implementing paths of virtual arcs that exceed their delay constraints, it is not straight forward to determine where in the substrate the delays have to be reduced to make all paths feasible while keeping C_a minimal.

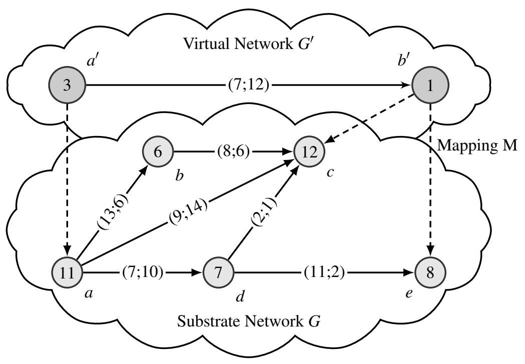

Fig. 1 A simple VNMP Instance

With the concept of additional resource costs in place, we are able to create meaningful results even without finding a valid solution to a VNMP instance, because we at least know a cost effective way to host the current VN load. When comparing two solutions, we need to take C_u and C_a into account. Since our first aim is to find a solution where no investment in infrastructure is required (i.e., $C_a = 0$), solutions with lower C_a are preferred, even if their C_u is higher. Only if the additional resource cost of two solutions is equal, lower substrate usage costs are preferred.

Figure 1 shows a simple VNMP instance. The virtual network graph G' consists of one virtual network with the two nodes a' and b'. The number printed in the nodes define the required CPU power, the arc label specifies the required bandwidth and the maximum allowed delay. The substrate network G (nodes a to e) shows in the nodes the available CPU power, arc labels define the available bandwidth and the delay that is incurred when data is transferred across this connection. The mapping constraints are visualized with dashed lines; a' may only be mapped to a, while b' may be mapped to c or e. The usage costs for the substrate nodes and arcs have been omitted for clarity.

This small example has a single valid solution, because b' cannot be mapped to c. The mapping itself would be possible since c has enough CPU capacity to host b' and receive the data of the incoming virtual connection (8 units of CPU are required, c offers 12). The problem is that the virtual connection from a' to b' cannot be implemented from a to c. The implementing path cannot cross b, because b does not have sufficient CPU power to forward seven units of bandwidth. Using the direct connection from a to c would violate the delay constraint and the path (a,d,c) is not allowed because the connection from d to c does not have enough bandwidth capacity. Therefore, the only solution to this VNMP instance is to map a' to a, b' to e and implement the virtual connection between a' and b' by using the substrate path (a,d,e).

3 Background and Related Work

Genetic Algorithms are nature-inspired population-based algorithms for optimization, an overview can be found in Sivanandam and Deepa (2007). When applying a Genetic Algorithm (GA) to solve a problem, one important design decision is the problem representation, which has a pronounced influence on its performance. For example, the influence of the chosen problem representation in the context of the travelling salesman problem is discussed in Larrañaga et al (1999). There exist many different problem representations for different problem classes. In particular, there is a special representation designed for problems where entities have to be grouped together (Falkenauer and Delchambre 1992), like in the case of the VNMP, where groups of virtual nodes are mapped to the different substrate nodes. In this work, we will utilize this representation. Successful applications of this representation include the access node location problem (Alonso-Garrido et al 2009) and the multiple traveling salesman problem (Evelyn et al 2007). However, it is not clear that this representation is always advantageous. For instance, Feltl and Raidl (2004) report a successful application of a GA to the generalized assignment problem, preferring a different representation. Also, its robustness is questioned in Brown and Sumichrast (2003). Therefore, we will analyze the performance implications of different representations for the VNMP.

Memetic Algorithms (MAs) are combinations of a population based optimization methods (e.g. GAs) and local improvement techniques (Moscato and Norman 1992; Radcliffe and Surry 1994; Moscato and Cotta 2010). The main idea is to use the GA to find promising regions in the search space and local improvement techniques for intensification, i.e., identifying excellent solutions in those promising regions. There is a trade-off between the time spent in the GA and the time spent executing the local improvement technique, corresponding to the usual trade-off between intensification and diversification. We will show how this tradeoff manifests itself for the VNMP.

The VNMP occurs under the names Virtual Network Assignment (Zhu and Ammar 2006), Virtual Network Embedding (Chowdhury et al 2009), Virtual Network Resource Allocation (Szeto et al 2003) and Network Testbed Mapping (Ricci et al 2003) in the literature. The basic problem of mapping virtual networks in a substrate network is always the same, but there are substantial differences in the details.

In related work, the substrate and virtual networks are predominantly given as undirected graphs (Gupta et al 2001; Ricci et al 2003; Zhu and Ammar 2006; Houidi et al 2008; Chowdhury et al 2009; Razzaq and Rathore 2010; Yeow et al 2010; Fajjari et al 2012; Qing et al 2012; Wang et al 2012; Zhang et al 2012a,b). We are only aware of the works by Szeto et al (2003), who use directed graphs, and Lu and Turner (2006), who use directed graphs only for the virtual networks. We chose the directed approach, since directed graphs allow for example the specification of asymmetric resource requests, which are important for applications that require a lot of bandwidth in one direction but have a control path in the other direction which is not bandwidth-heavy but delay sensitive. The substrate and virtual networks used for testing are usually random graphs, with substrate network sizes going up to 100 nodes and virtual network sizes of up to 10 nodes. The VNMP instances we use in this work employ substrate graphs extracted from real network topologies with up to 1000 nodes. The virtual networks are designed to mimic different real application scenarios in structure and resource requirements and have sizes up to 30 nodes. For more details, see Inführ (2013, chap. 5).

Resources are used to limit the number of virtual networks that fit into a substrate network. Different approaches exist in the literature. Zhu and Ammar (2006) limit the number of virtual nodes mapped to substrate nodes and virtual arcs crossing substrate arcs directly. Besides this exception, bandwidth is a resource that is universally considered, exclusively so for example in Gupta et al (2001). The CPU capacity of substrate nodes is the second most popular resource for VNMP and is used among others by Razzaq and Rathore (2010); Qing et al (2012); Wang et al (2012).

In addition to resource restrictions, limiting the mapping possibilities of virtual nodes is popular. One method to implement this is to assign a location to substrate nodes and virtual nodes. Virtual nodes may only be mapped to substrate nodes not too far away. This approach is employed by Lu and Turner (2006); Chowdhury et al (2009); Zhang et al (2012b). Another way of limiting the mapping possibilities is to forbid that a substrate node hosts multiple virtual nodes of the same virtual network (Zhu and Ammar 2006; Fajjari et al 2012; Yeow et al 2010). An in-depth discussion about the possibilities of restricting virtual node placement can be found in (Yeow et al 2010).

Some less usual resources and restrictions can also be found in the literature. One example is the work by Zhang et al (2012a), where virtual nodes may be split up and mapped to multiple substrate nodes, which causes additional overhead. Fajjari et al (2012) consider available memory as additional resource. Furthermore, Fajjari et al allow the "overbooking" of bandwidth resources since not all virtual networks will have their peak communication demand at the same time. A limit is placed on the probability that the bandwidth capacity of a substrate connection is exceeded.

The last constraint we want to mention is the way virtual arcs are implemented. In this work, we require a single simple path in the substrate for every virtual arc, an approach that is also taken for example by Zhu and Ammar (2006). A possible alternative is to implement a virtual arc by multiple paths in the substrate. This has the advantage that virtual connections can request more bandwidth than is available on any arc in the substrate. In addition, algorithms based on multicommodity flow can be used to solve the problem, see for example the work of Szeto et al (2003). The disadvantage is that the behaviour of the virtual connection, especially with respect to the observed delay, becomes more erratic the more substrate paths are used to implement it.

4 A Memetic Algorithm for the VNMP

In this section, we present a Memetic Algorithm for solving the VNMP. First, Section 4.1 will deal with the basic algorithm, while possible refinements are presented in Section 4.2.

4.1 The Basic Algorithm

Algorithm 1 shows the Memetic Algorithm for the VNMP in pseudocode, the components of which we will describe in this section. From our previous work on the

Algorithm 1: Memetic Algorithm for the VNMP

```
Input  : VNMP instance I
Output: Solution S for I
1  Population P;
2  InitializePopulation(P,I);
3  while !terminate() do
4      Solution p1=select(P);
5      Solution p2=select(P);
6      Solution offspring=crossover(p1,p2);
7      mutate(offspring);
8      copyArcs(offspring,p1,p2);
9      localImprovement(offspring);
10     insert(P,offspring);
11 end
12 return best(P);
```

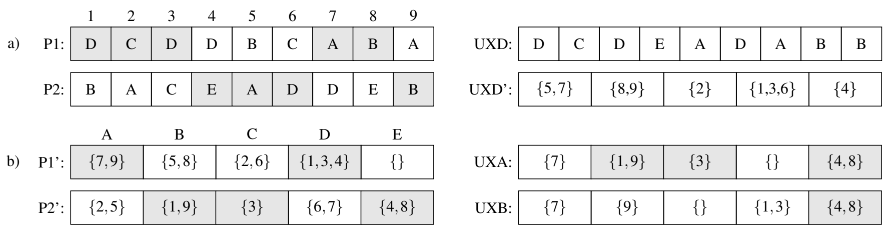
1 2 3 4 5 6 7 8 9
a) P1: D C D D B C A B A
P2: B A C E A D D E B
UXD: D C D E A D A B B
UXD': {5,7} {8,9} {2} {1,3,6} {4}
A B C D E
b) P1': {7,9} {5,8} {2,6} {1,3,4} {}
P2': {2,5} {1,9} {3} {6,7} {4,8}
UXA: {7} {1,9} {3} {} {4,8}
UXB: {7} {9} {} {1,3} {4,8}

Fig. 2 Comparison of different implementations of uniform crossover for the direct (a) and grouping (b) representations

VNMP (Inführ and Raidl 2013b), we know that the choice of location for a virtual node is the most important step when constructing a solution. Because of that, we decided to utilize a Genetic Algorithm to work primarily on this node packing aspect of the problem and use existing algorithms to derive complete solutions, i.e. to implement the VN connections. The main task of the GA is thus to assign virtual nodes to substrate nodes, or, when put in another way, finding for each substrate node a group of virtual nodes mapped to it so that a good complete solution can be created.

These two different ways of viewing the problem suggest two different representations. The first representation is the direct representation; a simple vector specifies the mapping target for each virtual node. The second representation focuses on the grouping aspect and represents a solution as a vector of sets determining which virtual nodes are mapped to a particular substrate node. We will call the latter grouping representation. Figure 2 shows examples both representations for the same VNMP solution. P1 and P1' (and P2 and P2') represent the same mapping of virtual nodes to substrate nodes, P1 with the direct representation and P1' with the grouping representation. For instance, P1 specifies that virtual node 1 is mapped to substrate node D, while P1' specifies that virtual nodes 7 and 9 are mapped to substrate node A.

The way the crossover operation works is heavily influenced by the chosen representation. The basis of the utilized crossover operators will be the classical uniform crossover, as we have seen in preliminary experiment that one-point and two-point crossover variants do not perform as well in this context. For the direct representation, applying the uniform crossover principle is straight forward. The mapping target of every virtual node is adopted randomly from one of the parents. We will denote the uniform crossover for the direct representation by UXD. Figure 2 shows a possible result (labeled as UXD) of applying UXD with P1 and P2 as parents. The marked mapping decisions of P1 and P2 are selected to be carried over to the offspring. UXD' shows the translation of the offspring to the grouping representation.

The uniform crossover for the grouping representation utilizes the same principle. For every substrate node, the virtual nodes mapped to it are chosen randomly from one of the parents. From here on we will call the set of virtual nodes mapped to a substrate node a virtual node group. Because we copy sets from different parents, two effects may occur that are not possible with the direct representation. In each parent solution, a virtual node is member of exactly one virtual node group. When none of those groups are selected to be present in the offspring, a virtual node remains unmapped after the crossover operation. If the groups of both parents are selected, the virtual node would be mapped twice. Both results are not allowed.

To solve the first problem, we adopt the mapping decision of one of the parents for all virtual nodes that remain unmapped after the crossover procedure. To avoid the second problem, we override an old mapping with a newer mapping. The consequence of this is that the sequence in which the groups are copied matters. We will compare two different copying strategies: copying all selected groups of one parent, then all selected groups of the other (which we will denote as UXA), and copying the groups in the order of the substrate node labels (UXB).

The result of applying the UXA and UXB crossover operators with parents P1' and P2' is shown in figure 2. For UXA, first the virtual node groups for substrate nodes A and D get copied from P1', then the groups for substrate nodes B, C and E from P2'. The mapping decisions from P2' override those of P1', for example, after the virtual node groups of P1' are added to the offspring, virtual node 9 is mapped to A, but this is overridden by P2', where node 9 is mapped to B. For UXB, we can observe that later mapping decisions influence earlier ones. After the groups for A to D are copied from their respective parents, virtual node 4 is mapped to substrate node D. The final step of UXB is copying the group for E from P2'. This group contains node 4, so it is removed from D. Note that in the result of UXA and UXB some nodes remain unmapped.

The main idea of crossover is to combine important solution properties from the parents to generate superior offspring. In our case, we want to keep the virtual node groups intact. The marked regions in the results of UXD, UXA and UXB show the groups that have survived without node removal. We can see that for UXA three groups have survived, for UXB one group has survived and no group survived UXD. The bad performance of UXD with respect to groupings was the reason why the grouping representation was introduced in the first place (Falkenauer and Delchambre 1992). However, there is also a big difference between UXA and UXB. With UXA, at least all virtual node groups selected for crossover of the second parent will survive (which are half of the groups in the expected case). With UXB, only the last group that is copied is guaranteed to survive. Therefore we use UXA when comparing the different representation possibilities for the VNMP.

After the crossover operation has finished, we apply the mutation operator with a probability of p_m. It clears a fraction of substrate nodes by mapping all virtual nodes they host to substrate nodes that are not marked to be cleared, if it is allowed by the mapping constraints. This fraction of cleared nodes is chosen uniformly at random from $[0,r]$, but at least one node is cleared. In this work we used $p_m = 0.2$ and $r = 0.2$ based on preliminary results, which also showed that mutation is required for good performance.

Until now, we have only considered the mapping of the virtual nodes. To specify a complete solution, the solution representation also contains the implementation of the virtual arcs. This implementation is derived by applying one of the improvement methods described later on. Note that these improvement methods utilize a Construction Heuristic to generate a complete solution before starting the improvement. Since the arc implementations may represent a significant amount of work, and the basic idea of crossover is to transfer as much information as possible from the parents to the offspring, we copy the arc implementation of the parents once the mapping for the virtual nodes is fixed. For every virtual arc f, we check the locations of $s(f)$ and $t(f)$ in the substrate graph for both parents and the offspring. If one parent utilizes the same mapping locations as the offspring, we copy its arc implementation. If both parents are compatible, the arc implementation is chosen randomly from one of the parents. If the mapping is different from both parents, the arc remains unimplemented. Unimplemented arcs will be assigned an implementation during the local improvement phase.

One of our main aims in this work is to check whether the time spent for local improvement actually improves the performance of the algorithm. Therefore, we either use a Variable Neighborhood Descent (Hansen and Mladenović 2001) to perform local improvement, or we skip local improvement and apply a Construction Heuristic instead. We need to apply the Construction Heuristic, because some virtual arcs might be unimplemented and we need to guarantee that only complete solutions are generated. We selected the best Construction Heuristic presented in (Inführ and Raidl 2013b), which means that from all virtual arcs to be implemented, we select in each step the arc for which the fraction of the delay of the shortest path in the substrate and the allowed delay is smallest. The path implementing this virtual arc is the path with the least increase in substrate usage cost C_u among those paths with the minimal increase in additional resource cost C_a. We will call this method CH.

The Variable Neighborhood Descent algorithm used in this work is based on the results presented by Inführ (2013, chap. 6). It employs three neighborhood structures, RemapVnode, ClearSarc and ClearSnode, which will be explained shortly. These are ruin-and-recreate neighborhoods which are searched in a first-improvement fashion. That means they remove a part of a solution and then reconstruct it to create a neighboring solution. For the reconstruction task we use a construction heuristic (denoted by CH-R) that is an extension of CH, as CH only implements virtual arcs. CH-R implements virtual nodes by choosing the mapping to a substrate node that increases C_u the least. We will skip the reconstruction step in the following description of the three neighborhood structures.

The RemapVnode neighborhood structure removes the mapping of a virtual node (and all implementations of virtual arcs connected to the virtual node). The ClearSarc neighborhood structure removes all virtual arc implementations crossing a particular substrate arc. The ClearSnode neighborhood structure follows the same principle, but for substrate nodes. It removes all virtual nodes mapped to the substrate node (including the implementations of the incoming and outgoing virtual arcs connected to the virtual nodes) in addition to all virtual arc implementations crossing the substrate node. These neighborhood structures are searched in the first-improvement way. We will denote this method simply as VND and execute it without time-limit. It is not the best identified configuration from Inführ (2013), but one that offers a good balance between solution quality and required runtime. We will however compare the Memetic Algorithm with the best Variable Neighborhood Descent approach from Inführ (2013), which we will call B-VND.

After the newly created solution has been improved by VND (or at least completed by CH), it is inserted back into the population immediately and the worst solution is removed in a steady-state fashion. The newly created solution is not inserted if it is already present in the population. This concludes one MA iteration, the next one begins by utilizing a binary tournament to select the parents for the next crossover operation.

The one component missing for a complete description of the MA is the population initialization. The main aim when initializing a population is the creation of a diverse set of good solutions. Several possibilities exist for the VNMP. One could simply map virtual nodes to one of the allowed substrate nodes randomly. Mapping virtual nodes in a way that tries to minimize the increase in C_u is another. Preliminary results showed that these approaches, while creating a very diverse set of initial solutions, do not work well, because VND requires a lot of time to improve the offspring during the initial iterations. Therefore, we chose a different approach: we create one good solution by using VND, and then apply the mutation operator with $r = 0.2$ to generate all other initial solutions. This has the additional benefit that the MA will have a promising solution from the beginning. In this work we used a population size of 10, we will show an analysis of the influence of the population size on the performance of the proposed MA in the following section.

4.2 Refinements

In this section, we discuss some modifications to the presented MA which might help to improve its performance. One area that could be improved is the initialization. Currently, the population is created by applying VND once and then mutate to create the required number of individuals. Our reason for this choice was that we need to create a diverse set of initial solutions, but these solutions also have to be reasonably good, as abysmal initial solutions substantially increase the time required for the local improvement phase in the initial iterations of the MA. An alternative way to achieve this goal would be to use a randomized construction heuristic to generate each individual and possibly further improve those individuals via an improvement heuristic. As a basis for the randomized construction heuristic, we chose CH. We have already defined how CH selects virtual arcs and implements them, now we cover virtual node selection and implementation. CH selects the (unmapped) virtual node with the highest total CPU load (the sum of the direct CPU demand and the bandwidth of incoming and outgoing virtual arcs) from the virtual network with the lowest sum of allowed delays. It is mapped to the allowed substrate node with the most free CPU capacity. In case of ties, the substrate node with the most free connected bandwidth is chosen as map target. A virtual node is selected for mapping if no virtual arc can be implemented, i.e., if there is no arc with mapped virtual source and target nodes.

To derive a randomized Construction Heuristic from CH, we randomize the selection of the mapping target for a virtual node, all other parts stay deterministic. We introduce a parameter $\alpha \in [0,1]$ that controls the level of randomization. When selecting a suitable substrate node for a virtual node, we collect a list of possible targets sorted by the available CPU and bandwidth, the candidate list. Let $f^{\text{CPU}}_{\text{Best}}$ denote the free CPU capacity and $f^{\text{BW}}_{\text{Best}}$ the free bandwidth capacity of the node that would have been selected by the deterministic strategy. We build the restricted candidate list by selecting all nodes i with $f^{\text{CPU}}_i \geq \alpha f^{\text{CPU}}_{\text{Best}} \wedge f^{\text{BW}}_i \geq \alpha f^{\text{BW}}_{\text{Best}}$. If $f^{\text{CPU}}_{\text{Best}}$ or $f^{\text{BW}}_{\text{Best}}$ is negative (i.e., more resources are used than are actually available), α is replaced by $2-\alpha$ in the relevant acceptance criterion. The mapping target is chosen uniformly at random from the restricted candidate list. We will denote this method as RCH.

The solutions created by RCH might be further improved by applying a metaheuristic. Using VND is out of the question, because generating the initial population would take far too long. Therefore, we select two Local Search approaches based on the results published in (Inführ and Raidl 2013b), one with more emphasis on speed, the other with more emphasis on solution quality. The first local search algorithm, which we will call LS1, uses the RemapSlice neighborhood structure. Like the already described neighborhoods, RemapSlice is a ruin-and-recreate neighborhood, recreation is done by applying CH-R. RemapSlice removes the implementation of a complete slice from a solution. LS1 searches RemapSlice in a first-improvement fashion. LS2 focuses more on solution quality and applies the already discussed ClearSnode neighborhood structure with first-improvement.

Another area for refinement of the presented MA is the application of the local improvement method. At the moment, we simply apply VND to every newly generated individual. However, this newly created individual might be so bad in terms of solution quality that trying to improve it would most likely be a waste of time. A tiered approach might improve this situation. Every individual benefits from basic improvement (for instance by applying LS1), and more powerful (and time-consuming) methods like LS2 or VND are applied only to promising individuals. We will evaluate three variants of this idea. The first one, which we call NewBest, simply states that if an individual (after the basic improvement) is the new best known solution, a more powerful improvement heuristic is applied. The second method, NewTopX, extends the definition of which individual is worthy of more powerful improvement. If a newly created individual (again after basic improvement) falls within the top $\alpha \in [0,1]$ fraction of the population, it gets improved further. With $\alpha = 0.2$, a newly generated individual would have to be better than 80% of the population to benefit from further improvement. The third method takes a slightly different approach in that it does not tie the application of the stronger method to the creation of a new individual. Instead, the top α fraction of the population is improved periodically. The length of the period is set with parameter β. A value of $\beta = 2$ means that the

Table 1 Properties of the used VNMP instances: average number of substrate nodes (V) and arcs (A), virtual nodes (V') and arcs (A') and the average number of allowed map targets for each virtual node ($M_{V'}$)

Size	$\|\mathbf{V}\|$	$\|\mathbf{A}\|$	$\|\mathbf{V'}\|$	$\|\mathbf{A'}\|$	$\|\mathbf{M_{V'}}\|$
20	20	40.8	220.7	431.5	3.8
30	30	65.8	276.9	629.0	4.9
50	50	116.4	398.9	946.9	6.8
100	100	233.4	704.6	1753.1	11.1
200	200	490.2	691.5	1694.7	17.3
500	500	1247.3	707.7	1732.5	30.2
1000	1000	2528.6	700.2	1722.8	47.2

best fraction of the population is improved every two generations. Since we utilize a steady-state GA, the improvement is executed after two times the population size new individuals have been created. We call this method ImproveAfter.

5 Results

To evaluate the performance of the proposed MA, we used the test set available from Inführ and Raidl (2011b). This set contains VNMP instances with 20 to 1000 substrate nodes, with 30 instances of each size. Each instance contains 40 VNs. Table 1 shows the main properties of the instance set, for more information on the instances see (Inführ 2013, chap. 5). We tested the MA on all instances of the instance set, and in addition also with a reduced load, i.e., fewer VNs. A load of 0.5 means that only 50% of the available VNs are used. We solved the instances with load levels 0.1, 0.5, 0.8 and 1, yielding a total of 840 test instances. All algorithms compared in this section have been run on one core of an Intel Xeon E5540 multi-core system with 2.53 GHz and 3 GB RAM per core. A CPU-time limit of 200 seconds was applied for sizes up to 100 nodes, 500 seconds for larger instances. The reported results of statistical tests are based on a paired Wilcoxon signed rank test with a 5% level of significance.

In the previous section, we defined three different types of crossover and two methods for local improvement. To be able to fully answer what the influence of different solution representations and crossover operators is and whether the time for local improvement is well spent, we evaluated every combination of crossover and local improvement. These are:

D-CH:	Direct representation with CH as local improvement and UXD
D-VND:	Direct representation with VND as local improvement and UXD
G-CH:	Grouping representation with CH as local improvement and UXA
G-VND:	Grouping representation with VND as local improvement and UXA
G-CH-B:	Grouping representation with CH as local improvement and UXB
G-VND-B:	Grouping representation with VND as local improvement and UXB

Another configuration of interest is G-VND-N, a variant of G-VND with disabled crossover operator, i.e., one individual is selected from the population, mutated, improved and then reinserted. This configuration allows us to determine if the crossover operation is actually necessary for good performance. We also compare the performance of the MA approaches to VND on its own, to see if and by how much the GA changes the performance, i.e., whether the VNMP problem solving capability comes primarily from VND or GA. We also compare our results to the best Variable Neighborhood Descent configuration (B-VND) from Inführ and Raidl (2013b) and the multicommodity-flow based integer linear programming formulation (FLOW) presented by Inführ and Raidl (2011a) with small modifications to match the VNMP model used in this work. The results of FLOW were achieved using a time-limit of 10 000 seconds.

In this work, we do not try to find valid VNMP solution, we try to find valid VNMP solutions with minimal C_u. However, we cannot simply compare C_u values for different algorithms, because higher values might be better if C_a is lower. We could have used a fitness function like $M \cdot C_a + C_u$, with M being a constant larger than the usage cost of the complete substrate. However, this introduces a strong bias towards algorithms that are able to find valid solutions for VNMP instances. Consider two algorithms applied to ten VNMP instances, one finds very cheap solutions but fails to find a valid solution for one instance while the second algorithm finds expensive but valid solutions to all instances. It might happen that the second algorithm has a better average fitness value, even though the first algorithm is more useful. Therefore, we employ a ranking procedure to compare different algorithms for the VNMP: For a single VNMP instance, order the compared algorithms based on the results they achieved, best algorithm first. The best algorithm gets assigned rank 0, the second best rank 1, and so on. Algorithms with the same result share the same rank. The rank of an algorithm still cannot be compared across multiple instances, because the number of ranks may be different for each instance. Therefore we calculate the relative rank R_{rel} of an algorithm when solving a particular instance as its rank divided by the highest rank for this instance. If all algorithms achieve the same result (i.e., the highest rank is zero), then R_{rel} is zero for all algorithms. Averages of R_{rel} can be compared in a meaningful way. An algorithm achieving an average relative rank of 0.1 generates results that are among the top 10% on average of all compared algorithms.

The average performance of the tested algorithms for different instance sizes is shown in Table 2. The symbol next to the reported relative ranks shows the relation to the best R_{rel} (disregarding FLOW) based on the mentioned Wilcoxon signed rank test, $=$ means that there is no significant difference between the reported R_{rel} and the best observed value, $>$ means that the reported rank is significantly higher than the best.

We can see that D-VND and G-VND achieve the best results for all instances up to and including size 200. For the largest two size classes, no MA configuration can beat B-VND. However, B-VND also requires more time (a maximum of 1000 seconds was allowed in (Inführ and Raidl 2013b)) than the 500 seconds allowed for all GA variants for these sizes. The GA variants based on CH achieve the best results at sizes 100 and 200. With smaller instances, local improvement with VND is better than a higher number of iterations made possible by not spending time on local improvement. The number of iterations starts getting relevant for sizes 100 and above as CH matches or outperforms VND. Even though the used Variable Neighborhood Descent configuration was selected for low runtime-requirements, the number of it-

Table 2 Average relative rank R_{rel} and its relation to the best result, average number of iterations (Its.) for GA based algorithms or average runtime for the other algorithms, fraction of solved instances (Solv.) in percent and average C_a for all compared algorithms per instance size

	Size	D-CH	G-CH	D-VND	G-VND	G-CH-B	G-VND-B	G-VND-N	VND	B-VND	FLOW
$\mathbf{R_{rel}}$	20	0.352 >	0.367 >	0.206 =	0.196 =	0.366 >	0.231 >	0.205 =	0.914 >	0.761 >	0.000
	30	0.442 >	0.452 >	0.232 =	0.231 =	0.445 >	0.230 =	0.247 =	0.922 >	0.727 >	0.000
	50	0.475 >	0.475 >	0.249 =	0.259 =	0.471 >	0.288 >	0.378 >	0.942 >	0.746 >	0.030
	100	0.409 =	0.408 =	0.388 =	0.411 =	0.419 =	0.364 =	0.546 >	0.969 >	0.614 >	0.359
	200	0.393 =	0.373 =	0.425 =	0.410 =	0.389 =	0.461 >	0.633 >	0.941 >	0.379 =	0.656
	500	0.438 >	0.444 >	0.609 >	0.645 >	0.402 >	0.628 >	0.757 >	0.992 >	0.143 =	0.750
	1000	0.525 >	0.547 >	0.715 >	0.729 >	0.536 >	0.715 >	0.788 >	0.664 >	0.240 =	0.818
GA: Its.	20	393268	357568	8185	8169	359589	8141	8265	0.2	0.4	131.2
Other: t[s]	30	259702	241245	3899	3854	238717	3869	3912	0.7	1.3	1338.8
	50	163663	151068	1663	1671	151207	1657	1691	2.1	4.2	2832.1
	100	63276	59591	314	325	59571	315	328	16.0	29.7	6117.2
	200	109125	104063	333	352	103817	340	355	40.2	119.7	7140.3
	500	43412	42076	94	95	42057	93	99	126.6	605.1	3211.1
	1000	13631	13348	23	25	13407	24	27	397.1	828.1	9144.5
Solv. [%]	20	97.5	97.5	100.0	100.0	97.5	100.0	100.0	96.7	97.5	100.0
	30	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	50	99.2	99.2	100.0	100.0	99.2	100.0	100.0	99.2	98.3	97.5
	100	95.0	95.0	100.0	100.0	95.0	99.2	99.2	95.0	97.5	64.1
	200	94.2	93.3	95.8	96.7	94.2	96.7	97.5	90.0	98.3	35.0
	500	77.5	78.3	76.7	79.2	78.3	77.5	76.7	73.3	90.8	25.0
	1000	60.0	59.2	58.3	57.5	59.2	61.7	57.5	57.5	61.7	18.3
$\mathbf{C_a}$	20	9.9	8.4	0.0	0.0	7.4	0.0	0.0	13.1	4.5	0.0
	30	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	50	4.9	4.9	0.0	0.0	4.9	0.0	0.0	4.9	2.1	0.0
	100	5.3	6.3	0.0	0.0	5.3	2.1	0.6	6.3	3.3	19142.5
	200	5.5	4.1	3.0	3.4	5.6	7.6	4.4	19.0	1.0	71648.7
	500	62.4	73.6	77.3	76.6	70.9	64.4	65.9	97.6	13.9	3413.8
	1000	215.4	215.5	215.9	214.7	216.2	214.1	215.9	184.1	198.9	3952.2

erations for larger instances is very low. For the largest instances and highest loads, the final result is basically the one created during population initialization. As to the difference between UXA and UXB, results show that surprisingly UXB does not cause any noteworthy performance degradation. G-VND has a slight advantage compared to G-VND-B, but no clear pattern is visible. Disabling crossover (G-VND-N) on the other hand has a pronounced negative effect on the outcome for medium sized instances.

With respect to the differences between the direct and the grouping representation, no significant differences could be observed. The choice and type of local improvement has a much greater impact on the performance. The results for VND show that the combination with the GA has a significant positive effect. When comparing to FLOW, we can observe that FLOW is able to obtain the best results for sizes 20 and 30 and is also able to find valid solutions to all instances of this size. But with respect to the required runtime, FLOW is only competitive for size 20. Increasing the instance size reduces the fraction of valid solutions drastically, for the largest instance size, FLOW only finds a valid solution to 18% of the instances. The GA based algorithms, on the other hand, achieve 60%. Also note that the average C_a is far worse for FLOW.

In Table 3 we show the average performance of the tested algorithms depending on the load of an instance. For the lowest load, every tested GA configuration achieves basically the same results, except for G-VND-N which performs far worse

Table 3 Average relative rank R_{rel} and its relation to the best result, average number of iterations (Its.) for GA based algorithms or average runtime for the other algorithms, fraction of solved instances (Solv.) in percent and average C_a for all compared algorithms per load

	Load	D-CH	G-CH	D-VND	G-VND	G-CH-B	G-VND-B	G-VND-N	VND	B-VND	FLOW
$\mathbf{R_{rel}}$	0.10	0.302 =	0.297 =	0.317 =	0.319 =	0.289 =	0.312 =	0.411 >	0.893 >	0.527 >	0.045
	0.50	0.355 =	0.382 >	0.449 >	0.454 >	0.358 =	0.432 >	0.561 >	0.981 >	0.479 >	0.394
	0.80	0.492 >	0.492 >	0.425 =	0.473 >	0.500 >	0.475 >	0.559 >	0.912 >	0.495 >	0.517
	1.00	0.584 >	0.582 >	0.423 =	0.401 =	0.582 >	0.448 >	0.499 >	0.839 >	0.562 >	0.538
GA: Its. **Other: t[s]**	0.10	430129	401525	6354	6321	401711	6323	6373	5.6	41.7	1946.6
	0.50	82299	75242	1016	1020	74971	1010	1027	50.2	218.3	3216.1
	0.80	48183	43667	537	547	43522	533	566	111.2	316.2	4441.3
	1.00	37147	33257	385	393	33147	385	420	166.0	331.6	5668.0
Solv. [%]	0.10	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	95.7
	0.50	99.0	98.6	97.1	97.6	98.6	99.0	97.1	95.7	99.0	61.0
	0.80	88.6	88.6	88.6	88.1	88.6	90.0	88.1	85.7	91.9	48.6
	1.00	68.6	68.6	74.8	76.2	69.0	73.8	75.2	68.1	77.1	46.2
$\mathbf{C_a}$	0.10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	567.3
	0.50	0.1	0.1	0.6	0.4	0.1	0.2	0.5	0.7	0.1	4306.8
	0.80	19.0	18.3	21.4	26.5	21.1	23.1	19.4	30.0	11.4	20967.5
	1.00	154.3	160.3	147.3	141.6	156.1	141.5	144.0	155.0	116.3	43651.8

Table 4 Comparison of the average R_{rel} of D-VND with different population sizes per VNMP instance size

Size	2	5	10	25	50	100	200	500
20	0.380 >	0.329 >	0.308 >	0.296 >	0.209 =	0.270 =	0.310 >	0.531 >
30	0.346 >	0.330 >	0.243 =	0.232 =	0.235 =	0.297 =	0.473 >	0.589 >
50	0.413 >	0.283 =	0.281 =	0.281 =	0.345 >	0.505 >	0.616 >	0.737 >
100	0.441 >	0.377 =	0.347 =	0.390 =	0.447 >	0.551 >	0.566 >	0.643 >
200	0.566 >	0.331 =	0.335 =	0.402 >	0.439 >	0.522 >	0.588 >	0.692 >
500	0.597 >	0.416 =	0.388 =	0.451 =	0.388 =	0.483 >	0.524 >	0.714 >
1000	0.450 =	0.463 >	0.379 =	0.394 =	0.420 =	0.479 >	0.605 >	0.804 >

due to the disabled crossover operator. To solve instances of load 0.5, using the direct representation with CH as local improvement is essential. Interestingly, the grouping representation only achieves the same performance when combined with UXB. It seems as if the additional disruption caused by the crossover operation is the key for good performance for this load case. For higher loads, a MA configuration is required for the best performance. For load 0.8 the direct representation (D-VND) is significantly better, for load 1 the grouping representation (G-VND) has an advantage, but the difference is not statistically significant. As we have already seen previously, disabling the crossover operation reduces the performance significantly. Also, B-VND is outperformed by the MAs for every load case. Note that for the higher loads, B-VND also requires a similar amount of time as the GA variants, which have an average runtime of 328.5 seconds due to the set runtime limits. The advantage of B-VND is that it is able to solve more instances of the highest load than all other algorithms. FLOW is best used for load 0.1. Increasing the load causes significantly longer runtimes and reduces the number of solved instances. We will continue our analysis with D-VND as best MA configuration, since it is slightly better than G-VND.

Before we discuss the effect of possible refinements of D-VND, we look briefly into the influence of different population sizes on the performance of D-VND, which

Table 5 Comparison of the average R_{rel} and initialization time t_{init} of the standard D-VND configuration and alternative initialization methods

	Size	D-VND	RCH	RCH-LS1	RCH-LS2
$\mathbf{R}_{\text{rel}}$	20	0.285 =	0.291 =	0.323 =	0.335 =
	30	0.322 =	0.274 =	0.283 =	0.319 =
	50	0.409 =	0.400 =	0.421 =	0.392 =
	100	0.435 =	0.435 =	0.494 =	0.515 >
	200	0.497 =	0.419 =	0.490 =	0.615 >
	500	0.462 =	0.475 =	0.583 >	0.511 =
	1000	0.388 =	0.473 >	0.621 >	0.533 >
$\mathbf{t}_{\text{init}}$	20	0.2 >	0.1 =	0.2 >	0.9 >
	30	0.6 >	0.1 =	0.4 >	2.6 >
	50	1.8 >	0.2 =	0.9 >	9.0 >
	100	13.3 >	0.7 =	3.3 >	77.1 >
	200	37.7 >	1.5 =	8.7 >	258.7 >
	500	127.6 >	4.7 =	40.0 >	346.1 >
	1000	312.6 >	11.8 =	157.2 >	407.9 >

is presented in Table 4. It can be seen that the population size of 10 chosen for D-VND works well across nearly the whole range of instance sizes. Only for the smallest instances can the performance be substantially improved by increasing the size of the population. Generally, a trend towards lower population sizes as the instance size rises is visible. For the largest instance sizes, there is not enough runtime available to allow the influence of the population size to manifest itself.

5.1 Refinement of D-VND

As the first step of refinement of D-VND, we evaluate different initialization methods. We initialize the individuals by applying RCH and RCH with subsequent improvement by LS1 and LS2. Based on experiments with α-values from 0.05 to 0.99 we determined that $\alpha = 0.5$ is best for initialization with RCH, when LS1 or LS2 are used to improve the solution generated by RCH, $\alpha = 0.7$ achieves the best results.

Table 5 shows the comparison of D-VND and different alternative initialization methods. It can be seen that the chosen initialization method generally has little influence on the final result. The best variant is still to use VND and mutate, but RCH is very close and has a deciding advantage: the initialization is significantly faster. That means that more time is available to do more GA iterations and with improved hybridization the standard D-VND configuration could be beaten. Therefore, we will continue with RCH as initialization.

The three methods for local improvement we discussed as alternatives to simply applying VND to every new offspring solution require two improvement methods. We have CH, LS1, LS2 and VND available, therefore every combination of a weaker method with a stronger method was evaluated, which in this case corresponds to six combinations. For NewBest, the combination LS2-VND performed best, based on a comparison of the average R_{rel}. As a side note, if the main aim is simply to find valid solutions, LS1-LS2 performs much better, being able to solve every instance up to and including size 200 and finding valid solutions to 90% of the instances of size 500 and 66% of size 1000.

For NewTopX, we evaluated the six improvement method combinations, in addition to values of α from 0.2 to 0.8. We could observe that the value of α has only a weak influence on the quality of the final solution, with a tendency of small values being better for larger instances. Again, the combination LS2-VND performed best, for α we chose a value of 0.3.

To select the best configuration for ImproveAfter, we evaluated the six improvement methods, α values from 0.2 to 0.8 and $\beta \in \{0.5, 1, 5, 20\}$. The general behaviour we could observe was that for weaker improvement method combinations, the best value for α was high (i.e., a large fraction of the population gets improved), as was the value for β (i.e., the improvement happens only seldomly). For instance, the best configuration when choosing CH and LS1 as improvement methods was $\alpha = 0.8$ and $\beta = 5$. With increasing power of the improvement methods (up to LS1-VND), the best β value decreases to 0.5. Among all configurations, the best results are again achieved by using the combination LS2-VND, with $\alpha = 0.8$ and $\beta = 0.5$, i.e., we improve a large fraction of the population by VND rather often.

Before we compare the refined hybridization approaches with the D-VND configuration, we want to point out two properties that we could observe while working on the refinement techniques. The first is the success rate when applying the stronger optimization method. We count an application as success if the offspring could be improved and is a new best known solution. The success rate is heavily dependent on instance size and load. For low load and small instances, the success rate is below 10%, but exceeds 80% for high sizes and high loads. That means that the additional improvement is most beneficial for challenging instances. Another behaviour that we noticed during our experiments was that the number of new best known solutions during search (the number of improvements) is not correlated with the final solution quality. Indeed, we could often observe that methods finding a high number of improvements had significantly worse results than the alternatives.

Table 6 shows a comparison of the quality of the final result between the standard D-VND configuration and the discussed alternative improvement methods. As can be seen, for a wide range of instance sizes and loads, the initial configuration performs best or at least not significantly worse than the alternatives, so as a general purpose solution method, the D-VND MA configuration is still best. Only if high quality solutions for large instance sizes with low loads are the main objective, ImproveAfter would be a better choice.

5.2 Online Behaviour of D-VND

Now, we will analyze the behaviour of D-VND in an online setting, i.e., when slices continuously arrive and depart. This is interesting for two reasons. First of all, this application is more likely in practice, because usually we do not know all virtual networks beforehand. They arrive dynamically and we need to fit them into the current substrate together with the already present virtual networks. Secondly, we can expect that a population based method performs well in this scenario, since it keeps a col-

Table 6 Comparison of the average R_{rel} of the standard D-VND configuration and improvement methods

Size	Load	D-VND	NewBest	NewTopX	ImproveAfter
20	0.10	0.067 =	0.167 =	0.067 =	0.167 =
	0.50	0.156 =	0.311 =	0.339 >	0.411 >
	0.80	0.167 =	0.583 >	0.361 =	0.439 >
	1.00	0.256 =	0.478 >	0.467 >	0.400 =
30	0.10	0.000 =	0.233 >	0.250 >	0.167 >
	0.50	0.211 =	0.233 =	0.306 =	0.317 =
	0.80	0.378 =	0.561 =	0.517 =	0.411 =
	1.00	0.450 =	0.533 =	0.489 =	0.528 =
50	0.10	0.089 =	0.233 =	0.128 =	0.250 >
	0.50	0.472 =	0.578 >	0.367 =	0.467 =
	0.80	0.567 =	0.483 =	0.450 =	0.517 =
	1.00	0.556 =	0.517 =	0.456 =	0.472 =
100	0.10	0.389 =	0.311 =	0.372 =	0.561 >
	0.50	0.394 =	0.433 =	0.567 =	0.606 >
	0.80	0.322 =	0.522 >	0.622 >	0.533 >
	1.00	0.344 =	0.533 =	0.478 =	0.644 >
200	0.10	0.578 =	0.533 =	0.439 =	0.417 =
	0.50	0.378 =	0.578 =	0.611 >	0.433 =
	0.80	0.361 =	0.578 >	0.589 >	0.489 =
	1.00	0.189 =	0.511 >	0.544 >	0.756 >
500	0.10	0.800 >	0.417 =	0.472 =	0.294 =
	0.50	0.444 =	0.489 =	0.522 =	0.544 =
	0.80	0.356 =	0.467 =	0.511 =	0.667 >
	1.00	0.222 =	0.489 >	0.467 >	0.822 >
1000	0.10	0.667 >	0.461 =	0.433 =	0.422 =
	0.50	0.344 =	0.467 =	0.544 =	0.644 >
	0.80	0.300 =	0.500 >	0.478 =	0.722 >
	1.00	0.044 =	0.600 >	0.711 >	0.644 >

lection of good results and when a new virtual network arrives one of those results might be easily extended to accommodate the new virtual network with only a small increase in cost.

We test two different scenarios for dynamic behaviour, which we call RampUp-Down and RampDownUp. For RampUpDown, we start with a VNMP instance at load 0.1 and increase the load in steps of 0.1 until we reach full load. Then the load is decreased again with the same step size until we reach load 0.1 again. The set of virtual networks to implement at a particular load level is the same, regardless whether virtual networks are currently being added or being removed; e.g., the VNMP instance at load 0.4 when virtual networks are being added is the same as the VNMP instance at load 0.4 when virtual networks are being removed. RampDownUp goes the other way, i.e., we start at full load, decrease the load until we reach 0.1, and then increase the load again. The VNMP instances at a particular load level are again the same (and also the same as the ones for RampUpDown). At each load level, the MA is run with the time limits used previously, i.e., 200 seconds for instances of size 100 and below, 500 seconds for instances of size 200 and beyond. When the load is increased, the solutions in the current population are not complete. We implement the newly added virtual networks by applying CH. We will call this MA-Keep. As we have stated in the introduction, our main goal when analyzing the online behaviour of the MA is determining whether keeping the population during changes is actually beneficial. Therefore, we also test the alternative of rebuilding the population every time virtual networks are added or removed, which we will denote by MA-Reset.

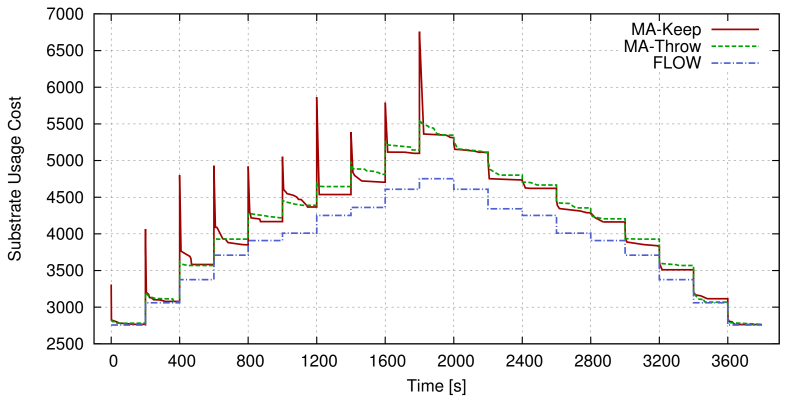

Fig. 3 Development of the substrate usage cost with the RampUpDown dynamic behaviour for MA-Keep and MA-Reset for an instance of size 100, the optimal values derived by FLOW are shown for reference

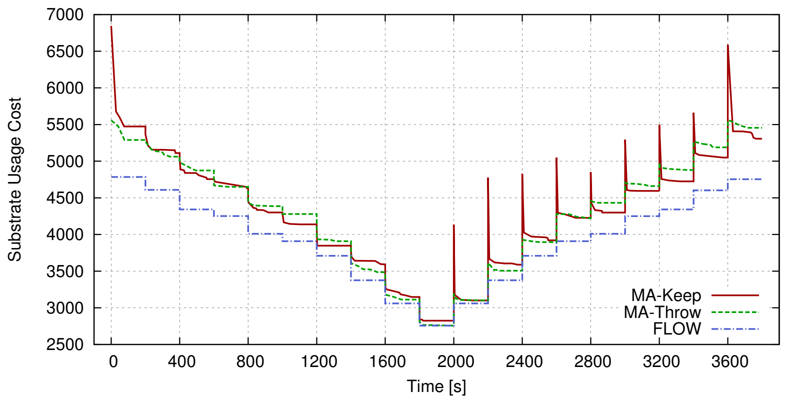

Fig. 4 Development of the substrate usage cost with the RampDownUp dynamic behaviour for MA-Keep and MA-Reset for an instance of size 100, the optimal values derived by FLOW are shown for reference

We evaluated MA-Keep and MA-Reset with RampUpDown and RampDownUp for every (full load) instance in the VNMP instance set.

Figures 3 and 4 show the behaviour of MA-Keep and MA-Reset for RampUpDown and RampDownUp, respectively, for a single instance of size 100. Also shown are the optimal solution values as derived by FLOW (with a time-limit of 10000 seconds) for reference. The first notable observation is that the initial solution of MA-Keep is much worse than that of MA-Reset when the load increases. We would have expected the exact opposite, as new solutions are built from already optimized solutions for lower load, but it seems that CH is not good enough to be able to add additional virtual networks without substantial increase in substrate usage cost. For the downward direction, it seems that the initial solutions of MA-Keep are better than those of MA-Reset. The final solution before the load changes is of course more interesting, and MA-Keep has advantages here, even though for some load levels

Table 7 Comparison of the average R_{rel} of the initial ($R_{\mathrm{rel}}^{\mathrm{Init}}$) and final ($R_{\mathrm{rel}}^{\mathrm{Final}}$) solutions for MA-Keep and MA-Reset for each load level of RampUpDown and RampDownUp in the upward direction

		RampUpDown		**RampDownUp**	
	Load	**MA-Keep**	**MA-Reset**	**MA-Keep**	**MA-Reset**
$\mathbf{R}_{\mathrm{rel}}^{\mathrm{Init}}$	0.10	0.965 >	0.155 =	0.453 >	0.155 =
	0.20	0.892 >	0.169 =	0.908 >	0.178 =
	0.30	0.913 >	0.210 =	0.889 >	0.203 =
	0.40	0.907 >	0.239 =	0.891 >	0.227 =
	0.50	0.891 >	0.227 =	0.876 >	0.224 =
	0.60	0.895 >	0.260 =	0.876 >	0.273 =
	0.70	0.883 >	0.291 =	0.882 >	0.280 =
	0.80	0.880 >	0.331 =	0.889 >	0.319 =
	0.90	0.894 >	0.316 =	0.880 >	0.319 =
	1.00	0.593 >	0.093 =	0.591 >	0.072 =
$\mathbf{R}_{\mathrm{rel}}^{\mathrm{Final}}$	0.10	0.282 =	0.284 =	0.300 =	0.289 =
	0.20	0.324 =	0.393 >	0.302 =	0.334 =
	0.30	0.410 =	0.387 =	0.349 =	0.400 =
	0.40	0.427 >	0.376 =	0.441 =	0.412 =
	0.50	0.437 =	0.442 =	0.434 =	0.448 =
	0.60	0.449 =	0.449 =	0.416 =	0.473 =
	0.70	0.493 =	0.511 =	0.449 =	0.507 =
	0.80	0.476 =	0.534 >	0.473 =	0.492 =
	0.90	0.442 =	0.539 >	0.432 =	0.545 >
	1.00	0.385 =	0.564 >	0.374 =	0.516 >

MA-Reset achieves the best results. One last point of interest with respect to the final solution is, whether the final solution of MA-Keep for RampUpDown is better on the downward direction than when going up. That would indicate that keeping the population intact really is useful. Unfortunately, such a tendency is not readily visible in the presented figures.

Of course, these observations are only based on a single run with a single instance, so we need to check whether they are representative for all instances and the relevant comparisons will follow shortly. Before that, just one technical detail: for RampUpDown and RampDownUp, every load level occurs twice (once while increasing the load, once while decreasing), except load 1 for RampUpDown and load 0.1 for RampDownUp. We will consider the results at those load levels to belong to both the upward (increasing load) and downward (decreasing load) direction.

Table 7 shows a comparison of initial and final results achieved by MA-Keep and MA-Reset when increasing the load. The comparison is done separately for RampUpDown and RampDownUp. It can be observed that indeed, when increasing the load the initial results of MA-Reset are significantly better than those of MA-Keep. For the final solution, however, the situation is reversed. Final solutions generated by MA-Keep are significantly better than those of MA-Reset. Only for low loads is MA-Reset competitive.

Table 8 presents the same comparison, but now for the downward direction. It can be seen that for RampDownUp, the initial solutions of MA-Reset are much better than those of MA-Keep. For RampUpDown, the situation is more complicated. For high loads, it seems that having a population optimized for even higher loads and then simply removing the virtual networks yields better results than creating a new population. This also explains why the initial solution of MA-Reset for load 1 is bet-

Table 8 Comparison of the average R_{rel} of the initial ($R_{\mathrm{rel}}^{\mathrm{Init}}$) and final ($R_{\mathrm{rel}}^{\mathrm{Final}}$) solutions for MA-Keep and MA-Reset for each load level of RampUpDown and RampDownUp in the downward direction

		RampUpDown		**RampDownUp**	
	Load	**MA-Keep**	**MA-Reset**	**MA-Keep**	**MA-Reset**
$\mathbf{R}_{\mathrm{rel}}^{\mathrm{Init}}$	0.10	0.449 >	0.146 =	0.453 >	0.155 =
	0.20	0.357 >	0.170 =	0.338 >	0.176 =
	0.30	0.336 >	0.213 =	0.309 >	0.206 =
	0.40	0.271 =	0.231 =	0.327 >	0.231 =
	0.50	0.242 =	0.238 =	0.336 >	0.224 =
	0.60	0.215 =	0.277 >	0.326 >	0.266 =
	0.70	0.193 =	0.279 >	0.315 =	0.274 =
	0.80	0.164 =	0.333 >	0.326 >	0.293 =
	0.90	0.125 =	0.322 >	0.367 >	0.291 =
	1.00	0.593 >	0.093 =	0.891 >	0.049 =
$\mathbf{R}_{\mathrm{rel}}^{\mathrm{Final}}$	0.10	0.309 =	0.313 =	0.300 =	0.289 =
	0.20	0.385 =	0.358 =	0.362 =	0.348 =
	0.30	0.354 =	0.426 >	0.344 =	0.380 =
	0.40	0.444 =	0.441 =	0.411 =	0.442 =
	0.50	0.379 =	0.407 =	0.499 >	0.428 =
	0.60	0.340 =	0.458 >	0.497 =	0.438 =
	0.70	0.354 =	0.500 >	0.518 >	0.466 =
	0.80	0.292 =	0.549 >	0.519 =	0.481 =
	0.90	0.266 =	0.568 >	0.511 =	0.489 =
	1.00	0.385 =	0.564 >	0.611 >	0.488 =

Table 9 Comparison of the average R_{rel} of the solutions generated by MA-Keep for RampUpDown in the upward and the downward direction

Load	**Up**	**Down**
0.10	0.282 =	0.309 =
0.20	0.324 =	0.385 >
0.30	0.410 >	0.354 =
0.40	0.427 =	0.444 =
0.50	0.437 >	0.379 =
0.60	0.449 >	0.340 =
0.70	0.493 >	0.354 =
0.80	0.476 >	0.292 =
0.90	0.442 >	0.266 =
1.00	0.385 =	0.385 =

ter than that of MA-Keep, as no population for even higher load exists. With respect to the final results, we can see that for RampUpDown and high loads, MA-Keep is significantly better than MA-Reset. For RampDownUp, however, MA-Reset is advantageous. This implies that MA-Keep is better for RampUpDown in the downward direction not because keeping the population is an advantage when removing virtual networks, but because we revisit load levels that have already been optimized.

This brings us to our final question: are the solutions of MA-Keep for RampUpDown in the downward direction better than in the upward direction? This is answered in Table 9. We can see that especially for high loads, the second visit of a particular load level yields much better results.

What do these results mean for a virtual network operator who continually wants to optimize the implementation of the virtual networks? Generally, the population should be kept when virtual networks arrive or depart, especially if the load is high.

If the operator wants to take advantage of found improved solutions immediately (and does not wait until the MA has converged), then some care has to be taken, since the initial solutions when keeping the population and adding virtual networks are quite bad. Using stronger methods than CH to rebuild the solutions in the population might be advisable, but this of course increases the time until the MA can actually run and improve the solutions.

6 Conclusion and Future Work

In this work we have introduced the Memetic Algorithm D-VND that significantly outperforms the best previously available algorithm for the VNMP over a wide range of instance sizes and load cases. With reference to the main questions we set out to answer with this paper, we have shown that there is no significant difference between different representations if performance for specific instance sizes is relevant. For high loads, the grouping representation might offer an advantage. As for the difference between UXA and UXB, we have shown that UXB can cause performance degradations, but also seen one case where it is beneficial. We believe that analyzing the difference between those crossover variants warrants further research. Whether or not the time for local improvement is well spent depends on the instance size and load. For small sizes, local improvement increases performance, while for large instances executing more GA iterations is more important. For high loads, using local improvement is essential. We have shown that disabling crossover decreases performance in all cases.

We studied various improvements to D-VND, both with respect to the population initialization and how the local improvement of newly created offspring solutions works. We could find an initialization method that holds the performance levels of D-VND while substantially reducing the time required for initialization, but none of the studied local improvement methods could improve on D-VND in general. Just for the special case of large instances with low loads an improvement over D-VND could be observed.

We could show that keeping the population when the VNMP instance changes in a dynamic setting is preferable to rebuilding it. More work is still required to improve the completion of solutions when virtual networks are added, as currently the initial solutions are rather bad.

The current VNMP model assumes constant delay on substrate arcs without regard for the current load and no overhead for mapping a lot of virtual nodes to a substrate node. More work is needed to address these shortcomings. We have shown that keeping the solutions in the presence of changes is beneficial. It would be interesting to see whether this still holds for single solution based optimization methods, or if the solution to a slightly different problem is misleading.

The MA presented in this work has various parameters, which were tuned by hand, and only with one objective in mind. Automatic parameter tuning techniques should also be considered, especially when the objective changes, for instance just finding valid solutions.

References

Alonso-Garrido O, Salcedo-Sanz S, Agustín-Blas LE, Ortiz-García EG, Pérez-Bellido AM, Portilla-Figueras JA (2009) A Hybrid Grouping Genetic Algorithm for the Multiple-Type Access Node Location Problem. In: Corchado E, Yin H (eds) Intelligent Data Engineering and Automated Learning - IDEAL 2009, Lecture Notes in Computer Science, vol 5788, Springer Berlin Heidelberg, pp 376–383

Anderson T, Peterson L, Shenker S, Turner J (2005) Overcoming the Internet Impasse Through Virtualization. Computer 38(4):34–41

Berl A, Fischer A, de Meer H (2010) Virtualisierung im Future Internet. Informatik-Spektrum 33:186–194

Brown EC, Sumichrast RT (2003) Impact of the Replacement Heuristic in a Grouping Genetic Algorithm. Computers & Operations Research 30(11):1575–1593

Carlson M, Weiss W, Blake S, Wang Z, Black D, Davies E (1998) An Architecture for Differentiated Services. IETF, RFC 2475

Chowdhury NMMK, Boutaba R (2010) A Survey of Network Virtualization. Computer Networks 54(5):862–876

Chowdhury NMMK, Rahman MR, Boutaba R (2009) Virtual Network Embedding with Coordinated Node and Link Mapping. In: 28th IEEE International Conference on Computer Communications (INFOCOM 2009), IEEE, pp 783–791

Chun B, Culler D, Roscoe T, Bavier A, Peterson L, Wawrzoniak M, Bowman M (2003) PlanetLab: An Overlay Testbed for Broad-Coverage Services. ACM SIGCOMM Computer Communication Review 33:3–12

Deering S, Hinden R (1998) Internet Protocol, Version 6 (IPv6) Specification, RFC 2460. `http://tools.ietf.org/html/rfc2460`

Evelyn CB, Cliff TR, Arthur EC (2007) A Grouping Genetic Algorithm for the Multiple Traveling Salesperson Problem. International Journal of Information Technology & Decision Making 6(02):333–347

Fajjari I, Aitsaadi N, Pujolle G, Zimmermann H (2012) Adaptive-VNE: A Flexible Resource Allocation for Virtual Network Embedding Algorithm. In: IEEE Global Communications Conference (GLOBECOM 2012), IEEE, pp 2640–2646

Falkenauer E, Delchambre A (1992) A Genetic Algorithm for Bin Packing and Line Balancing. In: IEEE International Conference on Robotics and Automation, IEEE, pp 1186–1192

Feltl H, Raidl GR (2004) An Improved Hybrid Genetic Algorithm for the Generalized Assignment Problem. In: Proceedings of the 2004 ACM Symposium on Applied Computing, ACM, pp 990–995

GENInet (2012) Global Environment for Network Innovations. `http://www.geni.net`

Gold R, Gunningberg P, Tschudin C (2004) A Virtualized Link Layer with Support for Indirection. In: Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture, ACM, pp 28–34

Gupta A, Kleinberg J, Kumar A, Rastogi R, Yener B (2001) Provisioning a Virtual Private Network: A Network Design Problem for Multi-Commodity Flow. In: STOC '01, ACM New York, pp 389–398

Hansen P, Mladenović N (2001) Variable Neighborhood Search: Principles and Applications. European Journal of Operational Research 130(3):449–467

Houidi I, Louati W, Zeghlache D (2008) A Distributed Virtual Network Mapping Algorithm. In: IEEE International Conference on Communications (ICC '08), IEEE, pp 5634–5640

Inführ J (2013) Optimization Challenges of the Future Federated Internet. PhD thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, Vienna, Austria

Inführ J, Raidl GR (2011a) Introducing the Virtual Network Mapping Problem with Delay, Routing and Location Constraints. In: Pahl J, Reiners T, Voß S (eds) Network Optimization: 5th International Conference (INOC 2011), Springer, Hamburg, Germany, Lecture Notes in Computer Science, vol 6701, pp 105–117

Inführ J, Raidl GR (2011b) The Virtual Network Mapping Problem Benchmark Set and Achieved Solutions by Heuristic and Exact Methods. URL `https://www.ads.tuwien.ac.at/projects/optFI/`, `https://www.ads.tuwien.ac.at/projects/optFI/`

Inführ J, Raidl GR (2013a) A memetic algorithm for the virtual network mapping problem. In: Lau H, Van Hentenryck P, Raidl G (eds) Proceedings of the 10th Metaheuristics International Conference, Singapore, pp 28/1–28/10

Inführ J, Raidl GR (2013b) Solving the Virtual Network Mapping Problem with Construction Heuristics, Local Search, and Variable Neighborhood Descent. In: Middendorf M, Blum C (eds) Evolutionary Computation in Combinatorial Optimisation – 13th European Conference, EvoCOP 2013, Springer, Lecture Notes in Computer Science, vol 7832, pp 250–261

Larrañaga P, Kuijpers CMH, Murga RH, Inza I, Dizdarevic S (1999) Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators. Artificial Intelligence Review 13:129–170

Lu J, Turner J (2006) Efficient Mapping of Virtual Networks Onto a Shared Substrate. Tech. rep., Washington University in St. Louis, WUCSE-2006-35

Moscato P, Cotta C (2010) A Modern Introduction to Memetic Algorithms. In: Gendreau M, Potvin J (eds) Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol 146, Springer, pp 141–183

Moscato P, Norman MG (1992) A Memetic Approach for the Traveling Salesman Problem Implementation of a Computational Ecology for Combinatorial Optimization on Message-Passing Systems. In: Proceedings of International Conference on Parallel Computing and Transputer Applications, IOS Press, vol 1, pp 177–186

National Research Council (2001) Looking Over the Fence at Networks. National Academy Press, Washington D.C.

Qing S, Qi Q, Wang J, Xu T, Liao J (2012) Topology-Aware Virtual Network Embedding through Bayesian Network Analysis. In: IEEE Global Communications Conference (GLOBECOM 2012), IEEE, pp 2621–2627

Radcliffe N, Surry P (1994) Formal Memetic Algorithms. In: Fogarty T (ed) Evolutionary Computing, Lecture Notes in Computer Science, vol 865, Springer Berlin Heidelberg, pp 1–16

Ramakrishnan KK, Floyd S, Black D (2001) The Addition of Explicit Congestion Notification (ECN) to IP. IETF, RFC 3168

Razzaq A, Rathore MS (2010) An Approach Towards Resource Efficient Virtual Network Embedding. In: Proceedings of the 2010 2nd International Conference on Evolving Internet, IEEE Computer Society, INTERNET '10, pp 68–73

Ricci R, Alfeld C, Lepreau J (2003) A Solver for the Network Testbed Mapping Problem. Special Interest Group on Data Communication Computer Communication Review 33(2):65–81

Schwerdel D, Günther D, Henjes R, Reuther B, Müller P (2010) German-Lab Experimental Facility. In: Berre A, Gómez-Pérez A, Tutschku K, Fensel D (eds) Future Internet - FIS 2010, Lecture Notes in Computer Science, vol 6369, Springer, pp 1–10

Sivanandam SN, Deepa SN (2007) Introduction to Genetic Algorithms, 1st edn. Springer Publishing Company, Incorporated

Szeto W, Iraqi Y, Boutaba R (2003) A Multi-Commodity Flow based Approach to Virtual Network Resource Allocation. In: Global Telecommunications Conference (GLOBECOM 2003), IEEE, vol 6, pp 3004–3008

Touch J, Wang Y, Eggert L, Finn G (2003) A Virtual Internet Architecture. ISI Technical Report ISI-TR-2003-570

Turner JS, Taylor DE (2005) Diversifying the Internet. In: IEEE Global Telecommunications Conference (GLOBECOM 2005), IEEE, vol 2, pp 755–760

Tutschku K (2009) Towards the Future Internet: Virtual Networks for Convergent Services. e & i Elektrotechnik und Informationstechnik 126(7-8):250–259

Tutschku K, Tran-Gia P, Andersen F (2008) Trends in Network and Service Operation for the Emerging Future Internet. AEU-International Journal of Electronics and Communications 62(9):705–714

Wang Z, Han Y, Lin T, Tang H, Ci S (2012) Virtual Network Embedding by Exploiting Topological Information. In: IEEE Global Communications Conference (GLOBECOM 2012), IEEE, pp 2603–2608

Yeow W, Westphal C, Kozat U (2010) Designing and Embedding Reliable Virtual Infrastructures. In: Proceedings of the Second ACM Special Interest Group on Data Communication Workshop on Virtualized Infrastructure Systems and Architectures, ACM, New York, NY, USA, VISA '10, pp 33–40

Zhang S, Wu J, Lu S (2012a) Virtual Network Embedding with Substrate Support for Parallelization. In: IEEE Global Communications Conference (GLOBECOM 2012), IEEE, pp 2615–2620

Zhang Z, Su S, Niu X, Ma J, Cheng X, Shuang K (2012b) Minimizing Electricity Cost in Geographical Virtual Network Embedding. In: IEEE Global Communications Conference (GLOBECOM 2012), IEEE, pp 2609–2614

Zhu Y, Ammar M (2006) Algorithms for Assigning Substrate Network Resources to Virtual Network Components. In: 25th IEEE International Conference on Computer Communications (INFOCOM 2006), IEEE, pp 1–12