A Memetic Algorithm for the Virtual Network Mapping Problem[1]

Johannes Inführ, Günther Raidl

Vienna University of Technology
Favoritenstraße 9–11/1861, 1040 Vienna, Austria
{infuehr,raidl}@ads.tuwien.ac.at

Abstract

The Virtual Network Mapping Problem arises in the context of Future Internet research. The core idea is the introduction of virtual networks to the Internet to be able to improve its functionality in a non-disruptive way. This also enables the creation of specialized networks which directly provide functionality required by some application classes. The challenge of fitting all the virtual networks (and the resources they require) into a physical network is 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 an uniform crossover for the Grouping Genetic Algorithm that may also be interesting outside of the Virtual Network Mapping domain.

1 Introduction

Network virtualization is a central component of scientific network testbeds such as GENI [1], Planet-Lab [10] or G-Lab [23]. It is necessary to facilitate the sharing of a costly resource: large scale networks available for testing new technology. Each research group can create its own virtual networks incorporating the improvements they are working on and test them within those testbeds, without having to fear interference from other research groups and their tests. This network virtualization approach has been identified as a central paradigm of Future Internet research [4, 5]. The Internet as it exists today suffers from ossification [20]. It is hard or even impossible to introduce new technologies to the Internet, even though they would bring large improvements to the service quality. A good example is IPV6 [11], which is still not implemented completely despite the obvious demand. The main reason why upgrades are so problematic is that changes to the underlying technology, such as employed protocols, would be very disruptive for the users who depend on the Internet working exactly as it does now. With the help of virtualization, such changes can be implemented in an incremental and non-disruptive manner, because old and new technologies can coexist. But network virtualization has even more potential: instead of using it just as a device for testing and switching technologies, the coexistence of different technologies may be the intended state. Multiple virtual networks exist, each with different dedicated and efficient protocols and capabilities, geared exactly towards specific use cases. For a survey on network virtualization, its application and available technologies, see [8].

In this context arises the Virtual Network Mapping Problem (VNMP). 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 that significantly outperforms the best previously available algorithms for the VNMP [18]. 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?
- Is the time for local improvement well spent?
- Does the crossover operation have a beneficial influence on the final outcome?

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

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

The VNMP consists of three parts: The virtual networks (VNs) that need to be realized, the substrate network that hosts the virtual networks (i.e. the physical network), and the location constraints between the virtual networks and the substrate network.

The VNs are modeled by the disconnected 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 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 set $M \subseteq V' \times V$ defines for each virtual node $k \in V'$ the substrate nodes that can be used to host it. 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 substrate path $P_f \subseteq A$ from $m(s(f))$ to $m(t(f))$ for every $f \in A'$ that does not exceed the allowed delay d_f and in which the bandwidth capacities b_e on the substrate arcs are respected.

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.

Finding a valid solution to the VNMP is NP-hard [3]. Therefore we cannot expect heuristic 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 the possibility of 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. For valid solutions to the VNMP, $C_a = 0$. We will call an instance solved if a valid solution could be obtained. As in previous work [18], 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. With these costs, we create a meaningful result even without solving an instance, because they show how to change the substrate network in a cost effective way to be able to host the current VN load. To compare two solutions, we use lexicographic ordering. Solutions with lower C_a are better and if this cost is equal, lower C_u is preferred.

Figure 1 presents a simple VNMP instance. It shows the substrate network G (nodes a to e) and its available resources. The available CPU power is written inside the nodes, the arc labels define the available bandwidth and the incurred delay when data is transferred across this connection. The virtual network graph G' consists of one virtual network with two nodes (a' and b'). The number within the

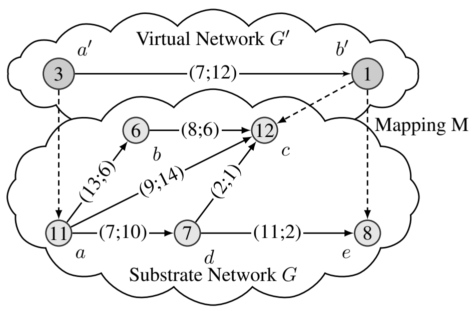

Figure 1: A simple VNMP Instance.

nodes define the required CPU power, the arc label specifies the required bandwidth and the maximum allowed delay. The dashed lines show the possible locations of the virtual network nodes. Usage costs for substrate nodes and arcs have been omitted for clarity.

This small example has a unique solution, since b' cannot be mapped to c, even though this mapping is allowed and c has enough CPU power to host b' and transfer the required seven units of bandwidth. The problem is that the virtual connection from a' to b' cannot be implemented from a to c. The path (a, b, c) is not possible, because b does not have the required CPU power to route the bandwidth. The direct connection from a to c incurs too much delay. The path (a, d, c) is not allowed, because the connection from d to c does not offer the required bandwidth. So the only remaining possibility is to map b' to e and implement the virtual connection with the path (a, d, e).

3 Background and Related Work

The Genetic Algorithm (GA) [15] is a nature-inspired population-based algorithm for combinatorial optimization, an overview can be found in [24]. The problem representation plays a crucial role for the performance of a Genetic Algorithm. In particular, there is a special representation designed for problems where entities have to be grouped together [12], like in the case of the VNMP virtual nodes mapped to the same substrate node. We will utilize this representation in this work. Successful applications of this representation include the access node location problem [2] and the multiple traveling salesman problem [6]. However, it is not clear that this representation is always advantageous. For instance, [13] reports a successful application of a GA to the generalized assignment problem, without using this representation. Also, its robustness is questioned in [7]. Therefore, we will analyze the performance implications of different representations for the VNMP.

The Memetic Algorithm (MA) is a combination of a population based optimization method (e.g. GA) and a local improvement technique [19, 21]. The main idea is to use the GA to find promising regions in the search space and then use the local improvement technique to find 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. Without enough time for the GA, it will fail to find promising regions, without enough time for the local improvement method, the found solutions will not be excellent. We will show how this tradeoff manifests itself for the VNMP.

The VNMP can also be found under the names Virtual Network Assignment [28], Virtual Network Embedding [9], Virtual Network Resource Allocation [25] and Network Testbed Mapping [22] in the literature. The basic problem is always the embedding of virtual networks in a substrate network. What varies are considered resources (e.g. the authors of [28] use no explicit resources, [22, 25] use bandwidth and [9, 26] add CPU power). We go considerably further by also considering link delays and the interaction between hosting virtual nodes and routing bandwidth on substrate nodes. Another possibility is using an additional resource type to model the capacity of a router to transfer bandwidth, which we did in previous work [17]. Different possibilities for constraining the mapping of virtual nodes considered in the literature are the requirement to map the nodes of a VN to different substrate nodes [28], prede-

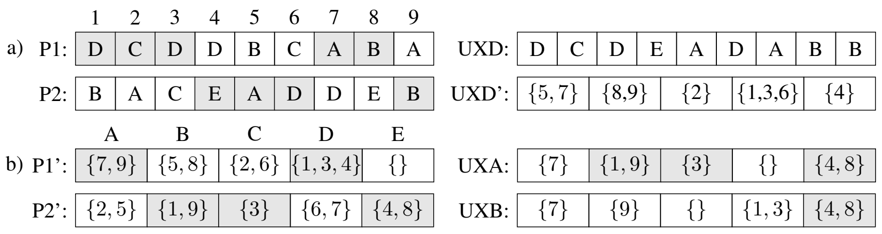
1 2 3 4 5 6 7 8 9
a) P1: D C D D B C A B A
P2: B A C E A D D E B
UXD: D C D E A D A B B
UXD': {5,7} {8,9} {2} {1,3,6} {4}
A B C D E
b) P1': {7,9} {5,8} {2,6} {1,3,4} {}
P2': {2,5} {1,9} {3} {6,7} {4,8}
UXA: {7} {1,9} {3} {} {4,8}
UXB: {7} {9} {} {1,3} {4,8}

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

```
Population P;
InitializePopulation(P);
while(!terminate()){
    VNMPSolution p1=select(P);
    VNMPSolution p2=select(P);
    VNMPSolution offspring=crossover(p1,p2);
    mutate(offspring);
    copyArcs(offspring,p1,p2);
    localImprovement(offspring);
    insert(P,offspring);
}
return best(P);
```

Listing 1: Memetic Algorithm for the VNMP

fine a mapping [25] or only allow a certain distance between the location of the virtual node and the substrate node that has to host it [9]. The VNMP model utilized in this work can accommodate all of those constraint types. In our model, we implement virtual arcs by using a single path, as is also done for instance in [28]. However, using multiple paths to implement a virtual connection is also prominent in the literature (e.g. [9]) since it makes the problem of implementing the virtual connections polynomially solvable and the authors of [27] argue for substrate support for this. However, using multiple paths to implement a single virtual connection makes the observed behaviour of the virtual connection more erratic, as multiple physical connections influence it. Therefore, we concentrate on finding simple paths to implement virtual connections.

4 A Memetic Algorithm for the VNMP

Listing 1 shows the Memetic Algorithm for the VNMP in pseudocode, which we will describe in this section. Based on previous experience with the VNMP [18], the most important step while constructing a solution is the choice of the location of the virtual nodes in the substrate. Therefore we chose to utilize a Genetic Algorithm to work primarily on this node packing aspect of the problem and use other algorithms to create a complete solution, 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 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 lead to different problem representations. The first representation is a simple vector that specifies the mapping target for each virtual node. We will call this representation the direct representation. The second representation focuses on the grouping aspect and represents a solution as a vector of sets which specify which virtual nodes are mapped to a particular substrate node. We will call this representation the grouping representation. Figure 2 shows examples of the direct and grouping representations of the same VNMP solution. For instance, P1 shows that virtual node 1 is mapped to substrate node D, while P1' shows that virtual nodes 7 and 9 are mapped to node A.

The employed representation influences how the crossover operator works. Based on preliminary results, we decided to utilize only uniform crossover (instead of the simpler one-point and two-point crossover variants). For the direct representation, the uniform crossover is straight forward; for every virtual node the mapping target is randomly selected from one of the parents. Figure 2 shows a possible result of the uniform crossover of P1 and P2 in direct representation (UXD). The marked components are the ones selected to be carried over to the offspring. UXD' shows the translation of UXD 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. We will call the set of virtual nodes mapped to a substrate node a virtual node group from here on out. Due to this solution structure, two effects can occur that are not possible with the direct representation. In each solution, a virtual node is part 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 both groups are selected, then the virtual node would be mapped twice, which is not allowed. The first problem can be remedied by just utilizing the mapping decision of one of the parents after the crossover procedure has finished for all unmapped virtual nodes. For the second problem, we override the old mapping with a newer mapping. This means that the sequence in which the groups are copied matters. We will compare two different copying strategies: copying all groups of one parent, then all groups of the other (UXA), and copying the groups in order of the substrate node labels (UXB). Figure 2 shows the result of applying these crossover operators using parents P1' and P2'. Note that in both cases some virtual nodes remain unmapped. The main idea of the 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 crossover 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 [12]. 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 we apply the mutation operator, which we will call ClearSnode mutation, with a probability of p_m. The ClearSnode mutation clears a fraction of substrate nodes by mapping virtual nodes to substrate nodes that are not selected 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. Due to space limitations we cannot show the detailed analysis.

Until now, we have neglected the implementation of virtual arcs. It is also a part of the solution representation, even though the crossover and mutation operators do not work on them directly. Since the arc implementation may represent a significant amount of work done by the local improvement, 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 node 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.

Since we want to check whether the time spent for local improvement actually improves the performance of the algorithm, we either use a Variable Neighborhood Descent [14] to perform local improvement, or we skip local improvement and apply a Construction Heuristic instead. The only reason for applying the Construction Heuristic is to implement all virtual arcs that have not been implemented yet to guarantee that after this step a complete solution has been generated. We selected the best Construc-

Size	$\|V\|$	$\|A\|$	$\|V'\|$	$\|A'\|$	$\|M_{V'}\|$
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

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 ($M_{V'}$).

tion Heuristic presented in [18], which means that virtual arc implementations are paths that cause the least increase in the substrate usage cost C_u without increasing the additional resource cost C_a. We will call this method CH. As for the Variable Neighborhood Descent, we chose neighborhoods N_2, N_4 and N_5 from [18]. These are destroy-and-recreate neighborhoods which are searched in a first-improvement fashion. That means that they remove a part of a solution and then reconstruct it to create a neighboring solution. For this reconstruction step, we use a construction heuristic designed for this reconstruction task (CH3 from [18]). We will skip the reconstruction step in the following description of the used neighborhoods. N_2 removes the mapping of a single virtual node (and all implementations of virtual arcs connected to the virtual node). N_4 removes all virtual arc implementations using a substrate arc. N_5 clears substrate nodes, i.e. all virtual nodes mapped to the substrate node are removed (which again includes the implementations of all connected virtual arcs), in addition to all virtual arc implementations using the substrate node. We will call this local improvement method VND and execute it without time-limit. We selected VND and not the best identified Variable Neighborhood Descent configuration in [18] because this configuration offers a good balance between solution quality and required runtime. However, we will compare our Memetic Algorithm with the best Variable Neighborhood Descent approach from [18], which we will call B-VND.

The newly created and improved offspring is immediately inserted back into the population and replaces the worst solution present (steady-state GA), unless the offspring is already present in the population. At this point, one GA iteration is complete, and the next one begins by utilizing a binary tournament to select the parents for the next crossover operation. Until now, we have neglected the problem of population initialization for reasons that will become obvious shortly. The main aim when initializing a population is the creation of a diverse set of good solutions. For the VNMP, there are different possibilities. One could simply randomly map virtual nodes to one of the allowed substrate nodes. 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 good solution from the beginning. In this work we used a population size of 10.

5 Results

To evaluate the performance of the proposed MA, we used the test set available at [16]. The 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 [18]. The MA was tested on all instances, and additionally with different loads, i.e. reduced numbers of VNs. A load of 0.1 means that only 10% of the available VNs are used. We tested load levels of 0.1, 0.5, 0.8 and 1. This results in 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.

	Size	D-CH	G-CH	D-VND	G-VND	G-CH-B	G-VND-B	G-VND-N	VND	B-VND	FLOW
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
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

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 run-time for the other algorithms, fraction of solved instances (Solv.) in percent and average C_a for all compared algorithms per instance size.

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.

We compare four different MA configurations: direct representation with CH as local improvement (D-CH), with VND as local improvement (D-VND), grouping representation using UXA and CH as local improvement (G-CH) and with VND as local improvement (G-VND). To fully compare the influence of the employed crossover operator, we also test the grouping representation with UXB and CH as local improvement (G-CH-B) and VND as local improvement (G-VND-B). Furthermore, we investigate the G-VND variant with disabled crossover (G-VND-N), i.e. one individual is chosen from the population, mutated, improved and then reinserted. Finally, we also consider VND on its own, the best VND configuration from [18] (B-VND) and compare the results to the multicommodity-flow based integer linear programming formulation presented in [17] (FLOW) 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.

Our aim in this work was finding good solutions to the VNMP (instead of only finding valid solutions for example). However, we cannot simply compare C_u values for different algorithms, because higher values might be better, if C_a is lower. Therefore we use the same ranking procedure as employed in [18]: 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 is 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.

Table 2 shows the average performance of the tested algorithms for different instance sizes. The symbol next to the relative rank shows the relation to the best R_{rel} (disregarding FLOW) based on a

	Load	D-CH	G-CH	D-VND	G-VND	G-CH-B	G-VND-B	G-VND-N	VND	B-VND	FLOW
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.	0.10	430129	401525	6354	6321	401711	6323	6373	5.6	41.7	1946.6
Other: t[s]	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
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 3: Average relative rank R_{rel} and its relation to the best result, average number of iterations (Its.) for GA based algorithms or average run-time for the other algorithms, fraction of solved instances (Solv.) in percent and average C_a for all compared algorithms per load.

statistical test, $>$ means that the reported R_{rel} is significantly larger than the best, $=$ means that no significant difference could be observed. It can be seen that D-VND and G-VND achieve the best results for all instances up to and including size 200. For sizes 500 and 1000 B-VND performs best. However, B-VND also takes more time (a maximum of 1000 seconds was allowed in [18]) 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. However, starting with size 100, performing more iterations gets more important and CH outperforms VND. Even though the VND was selected for low runtime-requirements, the number of iterations for larger instances is very low. For the largest instances, the final result is basically the one created during population initialization. Surprisingly, UXB achieves the same results as the algorithms using UXA. G-VND has a slight advantage compared to G-VND-B, but no clear pattern is visible. Disabling crossover (G-VND-N) however has a pronounced negative effect on the results for medium sized instances. Generally, no significant differences could be observed between direct and grouping representations. The influence of the type of local improvement is far more pronounced. The results for VND show that the combination with the GA has a significant positive effect on the achieved results. FLOW is able to obtain the best results for sizes 20 and 30 and is also able to solve all instances of this size. However, only the required runtime at size 20 is competitive. For the largest instance size, FLOW only finds a valid solution to 18% of instances, while the GA based algorithms achieve 60%. Also note that the average C_a is far worse for FLOW.

Table 3 shows the average performance of the tested algorithms for different loads. For low loads, every tested GA achieves basically the same results, except G-VND-N which performs far worse due to the disabled crossover operator. For medium load, a direct representation and CH as local improvement are essential. Interestingly, the grouping representation is only able to achieve the same level of performance by using UXB. It seems as if the additional disruption caused by the crossover operation is the key for good performance for this load case. Higher loads require a MA for the best performance, for load 0.8 the direct representation is significantly better, for load 1 the grouping representation has an advantage, but is not significantly better. Note that disabling the crossover operation results in bad solutions for every tested load case. Also, B-VND is outperformed by the MAs for every load case. Keep in mind that for the highest load, B-VND requires the same amount of time as all the tested GA variants, which have an average runtime of 328.5 seconds due to the set runtime limits. However, B-VND is able to solve more instances of the highest load than all other algorithms. FLOW is best used for load 0.1. Higher loads increase the runtime significantly and reduce the number of solved instances.

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.

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. Also the dynamic behaviour of the proposed algorithm in the presence of arriving and departing virtual networks needs to be studied. We believe that a GA is a good basis for dynamic VNMP problems because it creates a collection of excellent solutions, some of which might lead to good solutions when VNs are added.

References

[1] GENI.net Global Environment for Network Innovations, http://www.geni.net.

[2] O. Alonso-Garrido, S. Salcedo-Sanz, L. Agustín-Blas, E.G. Ortiz-García, A.M. Pérez-Bellido, and J.A. Portilla-Figueras. A Hybrid Grouping Genetic Algorithm for the Multiple-Type Access Node Location Problem. *Intelligent Data Engineering and Automated Learning*, pages 376–383, 2009.

[3] D. Andersen. Theoretical Approaches To Node Assignment. `http://www.cs.cmu.edu/~dga/papers/andersen-assign.ps`, December 2002. Unpublished Manuscript.

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

[5] A. Berl, A. Fischer, and H. de Meer. Virtualisierung im Future Internet. *Informatik-Spektrum*, 33:186–194, 2010.

[6] E.C Brown, C.T Ragsdale, and A.E. Carter. A grouping genetic algorithm for the multiple traveling salesperson problem. *International Journal of Information Technology & Decision Making*, 6(02):333–347, 2007.

[7] E.C. Brown and R.T. Sumichrast. Impact of the replacement heuristic in a grouping genetic algorithm. *Computers & Operations Research*, 30(11):1575–1593, 2003.

[8] N.M.M.K. Chowdhury and R. Boutaba. A Survey of Network Virtualization. *Computer Networks*, 54(5):862–876, 2010.

[9] N.M.M.K. Chowdhury, M.R. Rahman, and R. Boutaba. Virtual Network Embedding with Coordinated Node and Link Mapping. In *INFOCOM 2009, IEEE*, pages 783–791, April 2009.

[10] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33:3–12, 2003.

[11] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification, December 1998.

[12] E. Falkenauer and A. Delchambre. A Genetic Algorithm for Bin Packing and Line Balancing. In *IEEE International Conference on Robotics and Automation*, pages 1186–1192. IEEE, 1992.

[13] H. Feltl and G.R. Raidl. An Improved Hybrid Genetic Algorithm for the Generalized Assignment Problem. In *Proceedings of the 2004 ACM symposium on Applied Computing*, pages 990–995. ACM, 2004.

[14] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467, 2001.

[15] J.H. Holland. Genetic Algorithms. *Scientific American*, 267(1):66–72, 1992.

[16] J. Inführ and G.R. Raidl. The Virtual Network Mapping Problem benchmark set. `https://www.ads.tuwien.ac.at/projects/optFI/`.

[17] J. Inführ and G.R. Raidl. Introducing the Virtual Network Mapping Problem with Delay, Routing and Location Constraints. In J. Pahl, T. Reiners, and S. Voß, editors, *Network Optimization: 5th International Conference, INOC 2011*, volume 6701 of *LNCS*, pages 105–117. Springer, 2011.

[18] J. Inführ and G.R. Raidl. Solving the Virtual Network Mapping Problem with Construction Heuristics, Local Search and Variable Neighborhood Descent. In M. Middendorf and C. Blum, editors, *The 13th European Conference on Evolutionary Computation in Combinatorial Optimisation, EvoCOP 2013*, volume 7832 of *LNCS*, pages 250–261. Springer, to appear, 2013.

[19] P. Moscato and M.G. Norman. 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*, volume 1, pages 177–186. IOS Press, 1992.

[20] National Research Council. *Looking Over the Fence at Networks*. National Academy Press, 2001.

[21] N. Radcliffe and P. Surry. Formal Memetic Algorithms. *Evolutionary Computing*, pages 1–16, 1994.

[22] R. Ricci, C. Alfeld, and J. Lepreau. A Solver for the Network Testbed Mapping Problem. *SIGCOMM Comput. Commun. Rev.*, 33(2):65–81, 2003.

[23] D. Schwerdel, Da. Günther, R. Henjes, B. Reuther, and P. Müller. German-Lab Experimental Facility. In A. Berre, A. Gómez-Pérez, K. Tutschku, and D. Fensel, editors, *Future Internet - FIS 2010*, volume 6369 of *LNCS*, pages 1–10. Springer, 2010.

[24] S.N. Sivanandam and S.N. Deepa. *Introduction to Genetic Algorithms*. Springer, 1st edition, 2007.

[25] W. Szeto, Y. Iraqi, and R. Boutaba. A Multi-Commodity Flow Based Approach to Virtual Network Resource Allocation. In *Global Telecommunications Conference*, volume 6, pages 3004–3008, 2003.

[26] W.L. Yeow, C. Westphal, and U. Kozat. Designing and Embedding Reliable Virtual Infrastructures. In *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*, VISA '10, pages 33–40, New York, NY, USA, 2010. ACM.

[27] M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking Virtual Network Embedding: Substrate Support for Path Splitting and Migration. *ACM SIGCOMM Computer Communication Review*, 38(2):17–29, 2008.

[28] Y. Zhu and M. Ammar. Algorithms for Assigning Substrate Network Resources to Virtual Network Components. In *Proceedings of the 25th IEEE International Conference on Computer Communications*, pages 1–12, 2006.