GRASP and Variable Neighborhood Search for the Virtual Network Mapping Problem[⋆]

Johannes Inführ and Günther Raidl

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

Abstract. Virtual network mapping considers the problem of fitting multiple virtual networks into one physical network in a cost-optimal way. This problem arises in Future Internet research. One of the core ideas is to utilize different virtual networks to cater to different application classes, each with customized protocols that deliver the required Quality-of-Service. In this work we introduce a Greedy Randomized Adaptive Search Procedure (GRASP) and Variable Neighborhood Search (VNS) algorithm for solving the Virtual Network Mapping Problem. Both algorithms make use of a Variable Neighborhood Descent with ruin-and-recreate neighborhoods. We show that the VNS approach significantly outperforms the previously best known algorithms for this problem.

Keywords: Virtual Network Mapping, Variable Neighborhood Search, GRASP.

1 Introduction

The Internet as it exists today suffers from ossification [20]. It is hard or even impossible to introduce new technologies, even though they would bring large improvements in Quality-of-Service. Examples for such technologies include Explicit Congestion Notification [21] or Differentiated Services (a Quality-of-Service framework) [5]. The most prominent example is probably IPV6 [9], which was first specified in 1998 and 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.

Network virtualization has been identified as a central technology for alleviating the ossification of the Internet in the Future Internet research community [3,4]. It is already being successfully employed in scientific network testbeds such as GENI [1], G-Lab [25] or PlanetLab [8]. In this context, network virtualization is used to share large scale research networks among different research groups. Each group uses its own virtual network to perform experiments, without fear of interference by other groups even though they are using the same

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

underlying physical network. With network virtualization, changes to the Internet technology can be employed in an incremental and non-disruptive manner. Old and new technologies can coexist in different virtual networks. However, virtualization does not have to be just a device to gradually move from one technology to the next. Having multiple virtual networks in place could be the preferred state, because it allows specialization of the virtual networks to better cater to the requirements of different application classes. For a survey on network virtualization, its application and available technologies, see [6].

The Virtual Network Mapping Problem (VNMP) arises in this context. The multitude of virtual networks (VNs), each with different characteristics and protocols, still has to be realized by utilizing the available physical network infrastructure (the substrate) and the available resources. Additionally, VNs have to be realized in such a way that they fulfill the required specification with respect to Quality-of-Service parameters such as available communication bandwidth and delay.

In this work, we introduce a Greedy Randomized Adaptive Search Procedure (GRASP) and a Variable Neighborhood Search (VNS) algorithm for solving the VNMP. Instead of simple Local Search, both algorithms make use of a Variable Neighborhood Descent with ruin-and-recreate neighborhoods [24]. We will show that the VNS approach significantly outperforms the previously best known algorithms from the literature.

The rest of this work is structured as follows: Section 2 defines the VNMP formally, followed by a discussion of the relevant background in Section 3. The GRASP and VNS approaches are presented in Sections 4 and 5. Section 6 contains the results of the experimental evaluation of our proposed algorithms and their comparison to other algorithms presented in the literature. We conclude in Section 7.

2 The Virtual Network Mapping Problem

Three types of information are required to fully specify a VNMP: The substrate network (i.e. the physical network) with its available resources, the virtual networks (VNs) that need to be realized with their resource requirements and the location constraints between the nodes of the VNs and the substrate nodes.

A directed graph $G = (V, A)$ with node set V and arc set A models the substrate network. Each substrate node $i \in V$ has a CPU power of $c_i \in \mathbb{N}^+$. This CPU power is used by the VN nodes mapped to i, but also by all implementations of VN arcs traversing it. We assume that routing one unit of bandwidth (BW) requires one unit of CPU power. It is inconsequential whether this BW is simply relayed or has originated from a virtual node mapped to the substrate node. Even if both, the sending and receiving virtual node are mapped to the same substrate node, CPU capacity is required to route traffic from one virtual node to the other. 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.

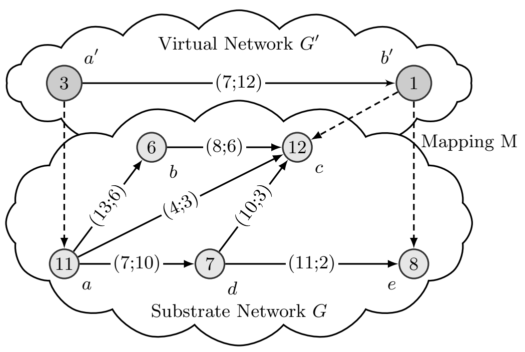

Fig. 1. An illustrative VNMP instance

The disconnected components of another directed graph $G' = (V', A')$ model the virtual networks. Each node $k \in V'$ requires a CPU power $c_k \in \mathbb{N}^+$. Each arc $f \in A'$ has a bandwidth requirement $b_f \in \mathbb{N}^+$ and a maximum allowed delay $d_f \in \mathbb{N}^+$.

The substrate nodes that a virtual node k is allowed to use are defined by the set $M \subseteq V' \times V$. By $s(a)$ and $t(a)$, $\forall a \in A \cup A'$, we denote arc a's source and target nodes, respectively.

Two components are required to specify a valid VNMP solution: A mapping $m : V' \to V$ such that $(k, m(k)) \in M$, $\forall k \in V'$ and a substrate path $P_f \subseteq A$ from $m(s(f))$ to $m(t(f))$ for every $f \in A'$ that does not exceed d_f. The total CPU load on each $i \in V$ (caused by virtual nodes hosted on i and traversing BW) is not allowed to exceed c_i and the BW capacities b_e have to be respected, too.

The objective of the VNMP is to minimize the total substrate usage cost. A price of $p_i^V \in \mathbb{N}^+$ has to be paid for every $i \in V$ that hosts at least one virtual node. Using a substrate arc $e \in A$ costs $p_e^A \in \mathbb{N}^+$. The sum of incurred node and arc usage costs is the total substrate usage cost C_u.

Already finding a valid solution to the VNMP is NP-hard [2]. Therefore we cannot expect an optimization approach to always be able to find valid solutions (which may not even exist) within practical time. To get around this problem, we allow the possibility of adding CPU power to the substrate, each additional unit costing C^CPU, and increasing the available BW on substrate arcs, costing C^BW per additional unit. The sum of the costs for additional resources is the additional resource cost C_a. For valid solutions to the VNMP, $C_\mathrm{a} = 0$. We call a VNMP instance solved if a valid solution could be obtained. We set $C^\mathrm{CPU} = 1$ and $C^\mathrm{BW} = 5$ in this work, which are values we also used in [19], to reflect the fact that it is cheaper to add additional CPU capacity to a router than to increase the BW of a network link. We use C_a as primary objective that has to be minimized. Only if two solutions have the same C_a the one with lower C_u is preferred. This has the advantage of guiding solutions towards validity during search.

Figure 1 shows a simple VNMP instance. The virtual network G' contains two virtual nodes, showing their CPU requirement, and a virtual arc connecting them, labeled with its BW requirement and allowed delay. The substrate network G contains the physical network nodes showing their CPU resources and the available links between the nodes, labeled by their BW capacity and the delay that is incurred when data is transmitted across them. The dashed lines show M, i.e. the allowed locations of the virtual nodes. Usage costs have been omitted for clarity. This example has only one valid solution, as b' cannot actually be mapped to c, even though c has enough resources available. The path implementing the virtual connection cannot use b, because it does not have enough resources to route the required BW. The direct connection from a to c lacks the required BW capacity, and the path (a, d, c) incurs too much delay. So the only valid solution is to map a' to a, b' to e and use the path (a, d, e) to implement the virtual arc between a' and b'.

3 Background and Related Work

The Greedy Randomized Adaptive Search Procedure (GRASP) [10] is a metaheuristic for combinatorial optimization problems. It works by continually repeating two steps. The first step is the randomized greedy construction of a solution to the problem to be solved. A second step is applying a local improvement technique to the constructed solution. These two steps are repeated until a termination criterion (like runtime or number of iterations) is reached. The best found solution is the final result of GRASP. How the randomized greedy solution construction works is a central aspect of GRASP. It iteratively builds a solution by adding components that seem good (but not necessarily the best) according to a greedy criterion. All possible components are collected in a candidate list (CL). The restricted candidate list (RCL) is created from the CL, usually by selecting all components from CL that are good enough (only a limited deviation from the perceived best alternative) or by selecting the best ones until the RCL has a specified length. The actual component that is added to the solution is selected uniformly at random from the RCL. This procedure usually leads to promising and at the same time diversified solutions for local optimization. A comprehensive overview of GRASP can be found in [11,22]. For hybridization techniques see [12].

The General Variable Neighborhood Search (VNS) [13] algorithm is built around Variable Neighborhood Descent. In Variable Neighborhood Descent, a local search is performed systematically switching between a series of neighborhood structures until a solution is reached that is local optimal w.r.t. all neighborhood structures. VNS adds diversification by applying random moves, called shaking, in successively larger neighborhood structures to escape the basins of attraction of local optima. VNS is a very successful metaheuristic for combinatorial optimization problems, for more details and a survey of applications see [14].

The VNMP appears in the literature as Network Testbed Mapping [23], Virtual Network Embedding [7], Virtual Network Assignment [29] and Virtual

Network Resource Allocation [26]. Embedding virtual networks into a shared substrate is always the central problem. Differences arise with the considered resources. For example, the authors of [29] do not consider any resources explicitly, [23,26] use bandwidth and [7,27] add CPU power. We extend on the latter by also considering the interaction between routing and hosting virtual networks and supporting delay constraints for the virtual connections. There are different methods for constraining the allowed mapping locations of virtual nodes present in the literature. The nodes of a VN might be required to be located at different substrate nodes [29], the mapping might be predetermined [26] or a a distance limit between a virtual node and the substrate node that hosts it might be in effect [7]. The VNMP model we utilize in this work can represent all of these variants and is thus most flexible. As for the paths used to implement a virtual connection, there are two approaches: using a single path or using multiple paths. Using multiple paths [7,28]) has the advantage that the problem of finding the implementations for the virtual connections becomes polynomially solvable when bandwidth may be arbitrarily split. However, using multiple paths to implement a virtual connection makes its observed behavior much more erratic, as it depends on multiple physical links. Therefore, we utilize here only a single path to implement virtual connections.

4 GRASP

A key component for a well working GRASP approach is the randomized greedy heuristic. We use the best identified construction heuristic configuration from our previous work in [19] as basis for randomization. In a nutshell, as long as virtual arcs are implementable (its source and target node have been mapped), the virtual arc f with the smallest fraction of d_f to shortest possible delay between $m(s(f))$ to $m(t(f))$ is implemented by the path with the least increase in C_u without increasing C_a. If no such virtual arc exists, the unmapped node with the highest total CPU requirement (CPU requirement of the virtual node and BW of connected virtual arcs) is selected from the VN that has the highest sum of total CPU requirements. It is mapped to the substrate node with the highest amount of free CPU capacity, ties are broken by using the amount of free incoming and outgoing bandwidth. Based on our previous work, we know that the substrate node selection strategy is the most influential for the overall performance of the construction heuristic. Therefore we concentrate on randomizing this strategy and keep all other parts of the randomized construction heuristic 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 BW, the candidate list. Let $f^{\mathrm{CPU}}_{\mathrm{Best}}$ denote the free CPU capacity and $f^{\mathrm{BW}}_{\mathrm{Best}}$ the free BW 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 $f^{\mathrm{CPU}}_i \geq \alpha f^{\mathrm{CPU}}_{\mathrm{Best}} \wedge f^{\mathrm{BW}}_i \geq \alpha f^{\mathrm{BW}}_{\mathrm{Best}}$. If $f^{\mathrm{CPU}}_{\mathrm{Best}}$ or $f^{\mathrm{BW}}_{\mathrm{Best}}$ is negative (i.e. more resources are used than actually are available), α is replaced by $2-\alpha$ in the relevant acceptance criterion. The actual mapping target is chosen uniformly at random from the restricted candidate list.

After the randomized greedy solution is generated, a local improvement strategy is applied. For comparison purposes, we choose the same method the Genetic Algorithm (GA) presented in [18] uses. It is a Variable Neighborhood Descent approach based on three ruin-and-recreate [24] neighborhoods, which are searched in a first-improvement fashion. They remove a part of a solution and reconstruct it using a construction heuristic designed for this rebuilding task (CH3 from [19]). The following short description of the used neighborhoods skips this rebuilding step. The first neighborhood is the set of all solutions reachable by removing the mapping of a single virtual node. The second neighborhood is the set of all solutions reachable by clearing a substrate arc, which means that all virtual arc implementations using this arc are removed. The third neighborhood is the set of all solutions reachable by clearing a substrate node. This means that all virtual nodes mapped to the substrate node, and all virtual arc implementations using this node, are removed from the solution. This neighborhood configuration was selected because it offers a good balance between required runtime and solution quality. Also, preliminary experiments showed that using simple Local Search is not competitive. In this work, we will call this configuration VND. We use VND without timelimit to improve solutions generated by the randomized greedy heuristic. If the found solution is better than the best solution found so far, we keep it. Then we repeat the randomized construction and improvement steps until the timelimit is reached, the best found solution is the result of GRASP.

5 VNS

Our proposed VNS algorithm uses a single type of shaking neighborhood in multiple configurations. Let this neighborhood be called $N^s(v)$, with $v \in [0,1]$ as parameter controlling the shaking vigor. N^s is based on the idea of clearing substrate nodes. When $N^s(v)$ is applied to a VNMP solution, N^s randomly selects $\lceil v \cdot |V| \rceil$ substrate nodes. All virtual arc implementations that traverse the selected nodes are removed from the solution. All virtual nodes mapped to the selected substrate nodes are mapped to a substrate node that is allowed by M but not selected. If no such node exists, the mapping remains unchanged. The resulting solution is completed and improved by VND to create the final solution of one VNS iteration. During the execution of VNS we apply N^s with different values for v. The used values are determined by two parameters, the base neighborhood size v_b and the count of iterations that have not resulted in an improvement of the best found solution n_ni. At the beginning of a new iteration, $N^s(v_\mathrm{b} n_\mathrm{ni})$ is applied to the currently best found solution and the result is improved by VND. If the solution created in this manner is better than the currently best known solution, n_ni is reset to one, otherwise n_ni is increased by one. The upper limit for n_ni is n_max. If this value is exceeded, n_ni is reset to one. The largest shaking neighborhood applied during VNS is $N^s(v_\mathrm{b} n_\mathrm{max})$. Values for v_b and n_max have to be chosen such that $v_\mathrm{b} n_\mathrm{max} \leq 1$. The shaking and improvement steps are applied until the timelimit is reached. The initial solution

```
VNMPSolution best=initialize();
$n_{\mathrm{ni}}$=1;
while(!terminate()){
    VNMPSolution candidate=shake($N^s(v_{\mathrm{b}} n_{\mathrm{ni}})$,best);
    applyVND(candidate);

    if(candidate<best){ //New best solution found
        best=candidate;
        $n_{\mathrm{ni}}$=1;
    }else{
        ++$n_{\mathrm{ni}}$;
        if($n_{\mathrm{ni}}$>$n_{\mathrm{max}}$)$n_{\mathrm{ni}}$=1;
    }
}
return best;
```

Listing 1.1. VNS for the VNMP

for VNS is built by the same method as for GRASP, but without randomization. Listing 1.1 shows the general outline of the proposed VNS.

6 Results

The proposed GRASP and VNS algorithms have been tested on the instances available from [16]. This instance set contains VNMP instances from 20 to 1000 substrate nodes, with 30 instances of each size. Every instance includes 40 VNs that have to be implemented. The VNs have different properties to cover different use-cases, like high BW requirements for P2P applications or low delays for VoIP applications. Table 1 shows the main properties of the used instances, for more information see [19]. To analyze the behaviour of the proposed algorithms in different load cases, we also tested with instances from the instance set that had some of their VNs removed. A load of 0.5 means that only 50% of the available VNs were used. We considered load levels of 0.1, 0.5, 0.8 and 1. This results in

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

a total of 840 VNMP instances, 120 for every size and 210 for every load level, so the results presented later in this section are averages of 120 or 210 runs respectively. 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. All reported results of statistical tests are based on a paired Wilcoxon signed rank test with a 5% level of significance.

Our design goal for the proposed algorithms was finding good solutions to the VNMP with respect to the objective function. This is significantly different from finding just valid solutions. The following results will show cases where algorithms find better results on average while solving fewer instances. To recognize the algorithms that create good solutions, we cannot look for low substrate usage costs C_{u}, because higher values might be better if the additional resource costs C_{a} are lower. Therefore we use the following ranking procedure as introduced in [19]. The achieved results of each algorithm under comparison for a specific instance are sorted in ascending order. The algorithm that achieved the best results gets rank 0, the second best rank 1 and so on. Algorithms with the same result get the same rank. To have a value that is comparable across different instances, the rank is divided by the maximum rank to create the relative rank R_{rel}. If all results are the same (i.e. the highest rank is zero), R_{rel} is zero as well for all algorithms. A R_{rel} of 0.1 means that the results of the algorithm in question are within the top 10% of compared algorithms.

Section 6.1 compares the performance of the GRASP approach for different values of α, Section 6.2 analyzes the performance of the VNS approach for different shaking neighborhood configurations and Section 6.3 shows a comparison of the best GRASP and VNS approach with other approaches from the literature.

6.1 GRASP

To evaluate the influence of α on the GRASP approach, we tested values for α from 0 (completely random initial solution) to 0.9 in 0.1 increments and 0.99 (very similar initial solutions). The average performance depending on the instance size can be seen in Table 2. The symbol next to the relative rank shows the relation to the best R_{rel} based on a statistical test, $>$ means that the reported R_{rel} is significantly larger than the best, $=$ means that no significant difference could be observed. Immediately visible is the tendency of the best α value to rise with the instance size. For size 20, $\alpha \in [0, 0.4]$ yields the best results w.r.t. R_{rel}, while for size 1000 $\alpha \in [0.5, 0.8]$.

The reason for this behaviour is that for small instances, the randomized construction heuristic does not have to make as many random choices as for the larger instance sizes. Therefore, to get the same search space coverage w.r.t initial solutions, α has to be small for small instances. The results for the larger instances show that if α is too small, then the performance degrades, because the initial solution is far too random. Another contributing factor is that VND takes longer to optimize a very random initial solution, as can be seen by the iteration counts. Therefore, fewer iterations can be performed in the same amount of time.

Table 2. Average relative rank R_{rel} and its relation to the best result, average number of iterations (Its.) and fraction of solved instances (Solv.) in percent for different values of α per instance size

	Size	GR-0.00	GR-0.10	GR-0.20	GR-0.30	GR-0.40	GR-0.50	GR-0.60	GR-0.70	GR-0.80	GR-0.90	GR-0.99
R_{rel}	20	0.278 =	0.216 =	0.219 =	0.235 =	**0.215** =	0.331 >	0.384 >	0.445 >	0.516 >	0.618 >	0.842 >
	30	0.431 >	0.337 >	0.318 =	0.288 =	**0.266** =	0.341 >	0.337 >	0.461 >	0.551 >	0.626 >	0.826 >
	50	0.549 >	0.512 >	0.463 >	0.362 >	**0.311** =	0.329 =	0.402 >	0.484 >	0.536 >	0.640 >	0.868 >
	100	0.870 >	0.665 >	0.468 >	0.362 =	**0.319** =	0.359 =	0.410 >	0.384 >	0.460 >	0.554 >	0.692 >
	200	0.889 >	0.737 >	0.488 >	0.361 >	**0.301** =	0.301 =	0.313 =	0.339 =	0.436 >	0.531 >	0.740 >
	500	0.856 >	0.718 >	0.511 >	0.449 >	0.381 >	**0.306** =	0.325 =	0.358 >	0.390 >	0.488 >	0.617 >
	1000	0.902 >	0.665 >	0.623 >	0.529 >	0.425 >	0.354 =	**0.338** =	0.341 =	0.375 =	0.426 >	0.470 >
Its.	20	1998	2323	2641	2916	3142	3291	3453	3677	3751	3836	3897
	30	780	896	1034	1153	1248	1399	1534	1588	1676	1667	1664
	50	282	329	390	442	481	497	531	553	565	567	566
	100	39	47	56	62	66	71	77	82	83	87	87
	200	45	54	66	78	90	98	103	112	116	127	129
	500	15	18	22	26	30	33	36	39	41	41	42
	1000	4	5	7	7	9	10	11	12	12	12	12
Solv.	20	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	99.2
[%]	30	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	50	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	100	100.0	100.0	100.0	99.2	98.3	97.5	97.5	99.2	99.2	97.5	95.8
	200	95.8	99.2	99.2	99.2	98.3	97.5	97.5	96.7	95.8	95.0	91.7
	500	71.7	80.8	81.7	81.7	76.7	80.8	78.3	75.8	77.5	75.0	70.0
	1000	34.2	58.3	57.5	55.8	60.0	55.0	54.2	55.8	55.0	55.0	55.8

Note that for finding valid solutions, low α values seem to be beneficial, even for large instances.

Table 3 shows the influence of α for different load cases. Again we can observe that higher values of α allow for more iterations, but they do not lead to improved performance for high load. Instead, a value for $\alpha \in [0.4, 0.5]$ seems to be best suited when performance at a specific load level across different sizes is most important. Low α values are again beneficial for finding valid solutions.

Based on these results, we select the GRASP approach with $\alpha = 0.4$ for further comparisons.

6.2 VNS

To analyze the influence of different shaking neighborhood configurations, we tested $n_{\max} \in 2, 5, 10$ and $v_{\text{b}} \in 0.01, 0.05, 0.1$ to cover the range from very small changes with few shaking neighborhoods (i.e. few different configurations for N^s) to large changes with a lot of neighborhoods. Table 4 shows the performance of different neighborhood configurations based on instance size. The different configurations are labeled as "VNS-$n_{\max}.v_{\text{b}}$", e.g. VNS-2.05 uses $n_{\max} = 2$ and $v_{\text{b}} = 0.05$. We can see a similar behaviour to GRASP. For smaller sizes, large shaking neighborhoods are beneficial, while large instance sizes require small neighborhoods for the best levels of performance. Smaller shaking neighborhoods lead to an increased number of iterations in the same amount of time. Also note the similarity in number of iterations between VNS-5.05 and VNS-2.10, caused by the very similar maximum shaking neighborhood sizes. Indeed, between sizes

Table 3. Average relative rank R_{rel} and its relation to the best result, average number of iterations (Its.) and fraction of solved instances (Solv.) in percent for different values of α per load

	Load	GR-0.00	GR-0.10	GR-0.20	GR-0.30	GR-0.40	GR-0.50	GR-0.60	GR-0.70	GR-0.80	GR-0.90	GR-0.99
R_{rel}	0.10	0.520 >	0.413 >	0.330 >	**0.288** =	0.299 =	0.335 =	0.419 >	0.495 >	0.573 >	0.683 >	0.766 >
	0.50	0.744 >	0.586 >	0.445 >	0.384 >	**0.307** =	0.313 =	0.324 =	0.354 =	0.473 >	0.574 >	0.773 >
	0.80	0.782 >	0.634 >	0.521 >	0.430 >	**0.305** =	0.318 =	0.312 =	0.350 >	0.397 >	0.482 >	0.695 >
	1.00	0.682 >	0.568 >	0.470 >	0.377 =	**0.356** =	0.360 =	0.380 =	0.407 >	0.423 >	0.482 >	0.654 >
Its.	0.10	1529	1737	1990	2193	2348	2490	2638	2781	2871	2917	2947
	0.50	108	142	173	208	239	267	298	322	330	333	336
	0.80	80	105	120	137	155	169	180	190	192	195	201
	1.00	90	114	126	139	151	160	167	173	175	176	173
Solv.	0.10	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
[%]	0.50	90.0	97.1	97.6	97.1	97.1	96.7	96.7	96.7	96.2	95.7	95.7
	0.80	81.0	90.5	89.0	89.5	90.0	87.6	87.6	89.0	88.6	86.7	85.2
	1.00	72.9	77.1	78.1	76.7	74.8	76.2	74.3	72.9	73.8	73.3	69.0

Table 4. Average relative rank R_{rel} and its relation to the best result, average number of iterations (Its.) and fraction of solved instances (Solv.) in percent for different shaking neighborhood configurations per instance size

	Size	VNS-2.01	VNS-5.01	VNS-10.01	VNS-2.05	VNS-5.05	VNS-10.05	VNS-2.10	VNS-5.10	VNS-10.10
R_{rel}	20	0.389 >	0.436 >	0.396 >	0.416 >	0.240 >	0.244 >	0.340 >	0.198 =	**0.163** =
	30	0.402 >	0.404 >	0.394 >	0.344 >	0.305 >	0.293 >	**0.229** =	0.304 >	0.283 =
	50	0.457 >	0.390 =	0.356 =	0.396 =	**0.333** =	0.368 =	0.338 =	0.390 =	0.472 >
	100	0.432 >	0.371 =	0.372 =	**0.349** =	0.486 >	0.506 >	0.425 >	0.578 >	0.630 >
	200	0.460 >	0.360 =	**0.316** =	0.376 >	0.490 >	0.554 >	0.479 >	0.591 >	0.685 >
	500	0.467 >	0.420 >	**0.344** =	0.395 =	0.500 >	0.520 >	0.547 >	0.658 >	0.624 >
	1000	0.468 >	**0.339** =	0.366 =	0.462 >	0.459 >	0.518 >	0.579 >	0.596 >	0.665 >
Its.	20	7327	7327	7345	7325	6796	5812	6822	5701	4742
	30	3721	3699	3670	3590	3132	2577	3156	2511	2010
	50	1766	1758	1664	1583	1321	1042	1327	1001	786
	100	415	389	346	311	237	181	233	169	127
	200	504	450	399	349	270	208	260	192	147
	500	157	143	124	110	85	67	83	62	48
	1000	51	44	39	33	25	20	25	18	14
Solv.	20	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
[%]	30	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	50	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	100	98.3	99.2	100.0	100.0	99.2	100.0	100.0	100.0	99.2
	200	93.3	95.8	97.5	95.8	95.0	99.2	99.2	98.3	100.0
	500	74.2	74.2	76.7	79.2	76.7	77.5	80.0	76.7	76.7
	1000	55.8	55.0	59.2	53.3	55.0	54.2	55.0	55.0	53.3

50 and 500, there is no significant difference between the two configurations. Larger shaking neighborhoods seem to increase the chance of finding valid solutions. The results indicate that increasing the shaking neighborhood size in multiple small steps works better than few large steps. This can be seen with configurations that have the same maximum shaking neighborhood size. VNS-10.01 and VNS-2.05 show no significant difference in R_{rel}, except for sizes 200 and 1000 where using smaller steps is significantly better. The difference is more pronounced for VNS-10.05 and VNS-5.10. Until size 50 there is no difference in performance, for larger instances using smaller steps is significantly better.

Table 5. Average relative rank R_{rel} and its relation to the best result, average number of iterations (Its.) and fraction of solved instances (Solv.) in percent for different shaking neighborhood configurations per load

	Load	VNS-2.01	VNS-5.01	VNS-10.01	VNS-2.05	VNS-5.05	VNS-10.05	VNS-2.10	VNS-5.10	VNS-10.10
R_{rel}	0.10	0.393 >	0.289 =	0.253 =	0.242 =	**0.239** =	0.250 =	0.250 =	0.304 >	0.390 >
	0.50	0.464 >	0.408 =	**0.360** =	0.407 >	0.415 >	0.458 >	0.436 >	0.486 >	0.516 >
	0.80	0.446 =	0.451 =	**0.408** =	0.467 >	0.471 >	0.479 >	0.486 >	0.559 >	0.535 >
	1.00	0.454 =	**0.407** =	0.432 =	0.449 =	0.482 >	0.528 >	0.506 >	0.546 >	0.572 >
Its.	0.10	6220	6176	6100	5992	5451	4640	5472	4542	3744
	0.50	924	905	877	853	712	550	708	526	410
	0.80	483	477	458	444	360	274	367	262	204
	1.00	339	335	329	312	257	196	256	186	141
Solv.	0.10	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
[%]	0.50	95.2	96.7	98.1	96.7	97.1	97.1	97.6	97.1	96.2
	0.80	87.1	87.1	88.6	85.7	86.7	87.1	88.6	87.1	88.6
	1.00	72.9	72.9	75.2	76.7	73.8	76.2	76.2	75.7	74.8

The influence of the shaking neighborhood configuration across different load cases can be seen in Table 5. Small shaking neighborhoods lead to the best performance. Load 0.10 is an exception, as larger shaking neighborhoods achieve the best results. As for the configurations with the same maximum shaking neighborhood size, smaller steps are an significant advantage for half of the load cases.

Based on these results, we chose VNS-10.01 for further comparison.

6.3 Comparison

In this section, we compare our proposed algorithms GR-0.4 and VNS-10.01 with approaches from the literature. These are GA-D-VND, the Genetic Algorithm for the VNMP introduced in [18], B-VND, the Variable Neighborhood Descent algorithm with the best performance with respect to R_{rel} from [19] and FLOW, a multicommodity-flow based integer linear programming formulation presented in [17] with small modifications to match the VNMP model used in this work. FLOW was solved by CPLEX 12.4 [15]. We also compare to VND, the Variable Neighborhood Descent algorithm used within GR-0.4, VNS-10.01 and GA-D-VND. The timelimits used in [18] were the same as the ones used in this work. The results of VND and B-VND had a timelimit of 1000 seconds and FLOW had 10000 seconds. Note that we do not directly compare FLOW to the other algorithms, because it can fail to generate any solution to a VNMP instance due to runtime or memory limits. This is true starting with instances of size 50, and for size 1000 FLOW only generates a solution for 30 out of 120 instances. However, for instances with a result generated by FLOW, this result was used for the calculation of R_{rel}. For comparison purposes, missing results were treated as $R_{\mathrm{rel}} = 1$ and as instance that could not be solved, so these values can be directly compared with the other reported results. The reported values for the average runtime and C_{a} are only based on instances where FLOW generated a solution and are therefore not directly comparable to the other results.

Table 6. Average relative rank R_{rel} and its relation to the best result, average number of iterations (Its.) or runtime, fraction of solved instances (Solv.) in percent and average C_{a} for different solution methods per instance size

	Size	GR-0.4	VNS-10.01	GA-D-VND	VND	B-VND	FLOW
R_{rel}	20	0.476 >	0.222 =	**0.192** =	0.912 >	0.753 >	0.000
	30	0.518 >	**0.222** =	0.241 =	0.920 >	0.705 >	0.000
	50	0.598 >	**0.214** =	0.275 >	0.930 >	0.696 >	0.029
	100	0.606 >	**0.187** =	0.368 >	0.916 >	0.564 >	0.362
	200	0.577 >	**0.197** =	0.413 >	0.859 >	0.372 >	0.655
	500	0.628 >	0.489 >	0.538 >	0.846 >	**0.171** =	0.750
	1000	0.623 >	0.592 >	0.589 >	0.569 >	**0.228** =	0.817
Its. / t[s]	20	3142	7345	8185	0.2	0.4	131.2
	30	1248	3670	3899	0.7	1.3	1338.8
	50	481	1664	1663	2.1	4.2	2832.1
	100	66	346	314	16.0	29.7	6117.2
	200	90	399	333	40.2	119.7	7140.3
	500	30	124	94	126.6	605.1	3211.1
	1000	9	39	23	397.1	828.1	9114.5
Solv. [%]	20	100.0	100.0	100.0	96.7	97.5	100.0
	30	100.0	100.0	100.0	100.0	100.0	100.0
	50	100.0	100.0	100.0	99.2	98.3	97.5
	100	98.3	100.0	100.0	95.0	97.5	64.1
	200	98.3	97.5	95.8	90.0	98.3	35.0
	500	76.7	76.7	76.7	73.3	90.8	25.0
	1000	60.0	59.2	58.3	57.5	61.7	18.3
C_{a}	20	0.0	0.0	0.0	13.1	4.5	0.0
	30	0.0	0.0	0.0	0.0	0.0	0.0
	50	0.0	0.0	0.0	4.9	2.1	0.0
	100	2.5	0.0	0.0	6.3	3.3	19142.5
	200	0.4	6.1	3.0	19.0	1.0	71648.7
	500	47.1	68.5	77.3	97.6	13.9	3413.8
	1000	245.5	311.2	215.9	184.1	198.9	3952.2

Note that we show the average runtime only for VND, B-VND and FLOW, since the other algorithms were run until the timelimit was reached, so we show the performed iterations for them instead. For reference, the average runtime when considering different load cases is 328.5 seconds.

Table 6 shows the performance of the compared algorithms in relation to each other. It can be seen that the results achieved by the GR-0.4 are disappointing. The GRASP approach is significantly outperformed by the VNS and GA algorithms. However, using GRASP around VND is significantly better than using VND alone, except for size 1000, where both perform equally well. B-VND can only be beaten or matched up to size 100, then B-VND achieves significantly better results. The VNS approach works far better, achieving the best solutions

Table 7. Average relative rank R_{rel} and its relation to the best result, average number of iterations (Its.) or runtime, fraction of solved instances (Solv.) in percent and average C_{a} for different solution methods load

	Load	GR-0.4	VNS-10.01	GA-D-VND	VND	B-VND	FLOW
R_{rel}	0.10	0.497 >	**0.215** =	0.315 >	0.876 >	0.528 >	0.045
	0.50	0.616 >	**0.294** =	0.384 >	0.913 >	0.450 >	0.393
	0.80	0.602 >	**0.333** =	0.397 >	0.841 >	0.484 >	0.517
	1.00	0.586 >	**0.371** =	0.400 =	0.771 >	0.532 >	0.538
Its. /	0.10	2348	6100	6354	5.6	41.7	1946.6
t[s]	0.50	239	877	1016	50.2	218.3	3216.1
	0.80	155	458	537	111.2	316.2	4441.3
	1.00	151	329	385	166.0	331.6	5668.0
Solv.	0.10	100.0	100.0	100.0	100.0	100.0	95.7
[%]	0.50	97.1	98.1	97.1	95.7	99.0	61.0
	0.80	90.0	88.6	88.6	85.7	91.9	48.6
	1.00	74.8	75.2	74.8	68.1	77.1	46.2
C_{a}	0.10	0.0	0.0	0.0	0.0	0.0	567.3
	0.50	0.5	6.9	0.6	0.7	0.1	4306.8
	0.80	18.3	29.6	21.4	30.0	11.4	20967.5
	1.00	150.0	183.9	147.3	155.0	116.3	43651.8

for sizes 30 to 200. For size 20, the GA approach works marginally better. Keep in mind however, that we selected a shaking configuration for the VNS that was significantly worse for the smallest instance sizes than the alternatives, so it should be possible to at least match the GA with a different configuration. For the two largest sizes, VNS is beaten by B-VND, partly because the B-VND had more runtime available (and also made use of it) as evidenced by the average runtimes. Also, it is not a coincidence that there is no significant difference between the GRASP, VNS, GA and VND approaches for size 1000. They all use VND as local improvement strategy, and as can be seen by the iteration count, not enough iterations could be performed to reap the benefits of the more involved heuristics within the available runtime. FLOW generates the best results for sizes 20 to 50, but based on the runtimes it is only competitive for size 20. Also note the quick degradation of the number of solved instances and the average C_{a} compared to the heuristic approaches.

For solving instances at a specific load level, Table 7 shows that the VNS approach is the best choice across all load levels, achieving significantly better results than all of the other compared algorithms. There is no reason to use GR-0.4, it is matched or outmatched by B-VND within the same or lower runtime. For the least challenging problem class (load of 0.1), FLOW again achieves better results than the heuristics. Note however that it does not even find valid solutions for all instances in this class while requiring a lot more runtime.

7 Conclusions

In this work, we have presented a GRASP and VNS algorithm for solving the Virtual Network Mapping Problem. We have shown that the VNS algorithm produces significantly better results than the GA and VND approaches previously introduced. It also compares favourably against an integer linear programming approach. Based on the presented results, we can conclude that the main idea of VNS (successively larger random moves away from local optima) works better than learning from a set of good solutions (GA) or improving good random solutions (GRASP) for the Virtual Network Mapping Problem. The comparison is fair since the same improvement strategy (VND) was used, the parameters of all algorithms have been optimized and the same timelimits were employed.

The main direction for future work will be testing the presented algorithms in an online setting that allows for the arrival and departure of virtual networks. In particular, the fact that GA produces a set of good solutions instead of a single one might prove useful.

References

1. GENI.net Global Environment for Network Innovations, `http://www.geni.net`
2. Andersen, D.: Theoretical Approaches to Node Assignment. Unpublished Manuscript (December 2002), `http://www.cs.cmu.edu/~dga/papers/andersen-assign.ps`
3. Anderson, T., Peterson, L., Shenker, S., Turner, J.: Overcoming the Internet impasse through virtualization. Computer 38(4), 34–41 (2005)
4. Berl, A., Fischer, A., de Meer, H.: Virtualisierung im Future Internet. Informatik-Spektrum 33, 186–194 (2010)
5. Carlson, M., Weiss, W., Blake, S., Wang, Z., Black, D., Davies, E.: An architecture for differentiated services. IETF, RFC 2475 (1998)
6. Chowdhury, N., Boutaba, R.: A survey of network virtualization. Computer Networks 54(5), 862–876 (2010)
7. Chowdhury, N., Rahman, M., Boutaba, R.: Virtual network embedding with coordinated node and link mapping. In: INFOCOM 2009, pp. 783–791. IEEE (2009)
8. Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., Bowman, M.: Planetlab: an overlay testbed for broad-coverage services. SIGCOMM Comput. Commun. Rev. 33, 3–12 (2003)
9. Deering, S., Hinden, R.: Internet protocol, version 6 (ipv6) specification (December 1998), `http://tools.ietf.org/html/rfc2460`
10. Feo, T., Resende, M.: Greedy randomized adaptive search procedures. Journal of Global Optimization 6, 109–133 (1995)
11. Festa, P., Resende, M.: An annotated bibliography of grasp–part i: Algorithms. International Transactions in Operational Research 16(1), 1–24 (2009)
12. Festa, P., Resende, M.G.C.: Hybrid GRASP heuristics. In: Abraham, A., Hassanien, A.-E., Siarry, P., Engelbrecht, A. (eds.) Foundations of Computational Intelligence Volume 3. SCI, vol. 203, pp. 75–100. Springer, Heidelberg (2009)
13. Hansen, P., Mladenović, N.: Variable neighborhood search: Principles and applications. European Journal of Operational Research 130(3), 449–467 (2001)

14. Hansen, P., Mladenović, N., Moreno Pérez, J.: Variable neighbourhood search: methods and applications. 4OR 6, 319–360 (2008)
15. IBM ILOG: CPLEX 12.4, `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer`
16. Inführ, J., Raidl, G.R.: The Virtual Network Mapping Problem benchmark set, `https://www.ads.tuwien.ac.at/projects/optFI/`
17. Inführ, J., Raidl, G.R.: Introducing the virtual network mapping problem with delay, routing and location constraints. In: Pahl, J., Reiners, T., Voß, S. (eds.) INOC 2011. LNCS, vol. 6701, pp. 105–117. Springer, Heidelberg (2011)
18. Inführ, J., Raidl, G.R.: A memetic algorithm for the virtual network mapping problem. In: Lau, H., Van Hentenryck, P., Raidl, G. (eds.) The 10th Metaheuristics International Conference, MIC13, Singapore (2013), submitted for review
19. Inführ, J., Raidl, G.R.: Solving the Virtual Network Mapping Problem with Construction Heuristics, Local Search and Variable Neighborhood Descent. In: Middendorf, M., Blum, C. (eds.) EvoCOP 2013. LNCS, vol. 7832, pp. 250–261. Springer, Heidelberg (2013)
20. National Research Council: Looking Over the Fence at Networks. National Academy Press (2001)
21. Ramakrishnan, K.K., Floyd, S., Black, D.: The addition of explicit congestion notification (ECN) to IP. IETF, RFC 3168 (2001)
22. Resende, M., Ribeiro, C.: Greedy randomized adaptive search procedures. In: Handbook of Metaheuristics, pp. 219–249 (2003)
23. Ricci, R., Alfeld, C., Lepreau, J.: A solver for the network testbed mapping problem. SIGCOMM Comput. Commun. Rev. 33(2), 65–81 (2003)
24. Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., Dueck, G.: Record breaking optimization results using the ruin and recreate principle. Journal of Computational Physics 159(2), 139–171 (2000)
25. Schwerdel, D., Günther, D., Henjes, R., Reuther, B., Müller, P.: German-lab experimental facility. In: Berre, A.J., Gómez-Pérez, A., Tutschku, K., Fensel, D. (eds.) FIS 2010. LNCS, vol. 6369, pp. 1–10. Springer, Heidelberg (2010)
26. Szeto, W., Iraqi, Y., Boutaba, R.: A multi-commodity flow based approach to virtual network resource allocation. In: Global Telecommunications Conference, GLOBECOM 2003, vol. 6, pp. 3004–3008. IEEE (2003)
27. Yeow, W.L., Westphal, C., Kozat, U.: Designing and embedding reliable virtual infrastructures. In: Proceedings of the Second ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures, VISA 2010, pp. 33–40. ACM, New York (2010)
28. Yu, M., Yi, Y., Rexford, J., Chiang, M.: Rethinking virtual network embedding: substrate support for path splitting and migration. ACM SIGCOMM Computer Communication Review 38(2), 17–29 (2008)
29. Zhu, Y., Ammar, M.: Algorithms for assigning substrate network resources to virtual network components. In: Proceedings of the 25th IEEE International Conference on Computer Communications, pp. 1–12 (2006)