Solving the Virtual Network Mapping Problem with Construction Heuristics, Local Search and Variable Neighborhood Descent*

Johannes Inführ and Günther R. Raidl

Institute of Computer Graphics and Algorithms
Vienna University of Technology, Vienna, Austria
{infuehr|raidl}@ads.tuwien.ac.at
http://www.ads.tuwien.ac.at

Abstract. The Virtual Network Mapping Problem arises in the context of Future Internet research. Multiple virtual networks with different characteristics are defined to suit specific applications. These virtual networks, with all of the resources they require, need to be realized in one physical network in a most cost effective way. Two properties make this problem challenging: Already finding any valid mapping of all virtual networks into the physical network without exceeding the resource capacities is NP-hard, and the problem consists of two strongly dependent stages as the implementation of a virtual network's connections can only be decided once the locations of the virtual nodes in the physical network are fixed. In this work we introduce various construction heuristics, Local Search and Variable Neighborhood Descent approaches and perform an extensive computational study to evaluate the strengths and weaknesses of each proposed solution method.

Keywords: Virtual Network Mapping Problem, Construction Heuristics, Local Search, Variable Neighborhood Descent, Future Internet

1 Introduction

The Internet has ossified [18]. Core parts of the network protocols have not been updated for more than a decade and the introduction of new services and technology is difficult, time consuming and costly. Improvements to the network protocols, however desirable or necessary, do not see widespread adoption if they could break existing features. Examples include Explicit Congestion Notification [19] or Differentiated Services (a quality of service framework) [3].

The Future Internet research community is currently searching for ways to overcome the ossification of the Internet and network virtualization has been identified as a promising technology to do this [1,2]. With the help of virtualization, changes to the core protocols of the Internet can be deployed in an

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

incremental and non-disruptive fashion. This idea can be developed even further, if one does not view virtual networks as a necessary crutch to move from one technology to the next, but instead as an integral feature of the network. If multiple virtual networks are present, then each of them can have different properties, different protocols, tailored specifically to a user group. In scientific network testbeds such as GENI [7], PlanetLab [6] or G-Lab [22] network virtualization techniques are already in use to share the underlying network infrastructure (substrate) among different research groups. For a survey on network virtualization, its application and available technologies, see [4].

The Virtual Network Mapping Problem (VNMP) arises in this context. Given are multiple virtual networks (VNs) which need to be realized by using resources present in the substrate. We model the substrate by a directed graph $G = (V, A)$ with node set V and arc set A. The VNs are modeled by the disconnected components of the directed graph $G' = (V', A')$. The set $M \subseteq V' \times V$ defines the allowed mappings between virtual and substrate nodes. By $s(a)$ and $t(a)$, $\forall a \in A \cup A'$ we denote the arc's source and target node, respectively. Each VN node $k \in V'$ requires CPU power $c_k \in \mathbb{N}^+$ (to implement custom protocols, etc.), each arc $f \in A'$ requires bandwidth (BW) $b_f \in \mathbb{N}^+$ and has a maximum allowed delay d_f. Substrate nodes $i \in V$ have an associated CPU power $c_i \in \mathbb{N}^+$, which is used to power the VN nodes mapped to i, but also to route BW. One unit of CPU power is required to route one unit of BW. Substrate arcs $e \in A$ have a BW capacity $b_e \in \mathbb{N}^+$ and a delay $d_e \in \mathbb{N}^+$. The objective is to find a mapping $m : V' \to V$ 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 implementations of virtual arcs does not exceed c_i. Furthermore, we have to find for each $f \in A'$ a substrate path $P_f \subseteq A$ leading from $m(s(f))$ to $m(t(f))$ with a delay of at most d_f. The BW capacity of substrate arcs has to be respected by those paths. These are required properties of a valid solution.

We want to implement the virtual networks with as low costs as possible. Every substrate node $i \in V$ has an associated usage cost $p_i^V \in \mathbb{N}^+$ which is paid when at least one VN node is mapped to it. Additionally, every substrate arc $e \in A$ has a usage cost $p_e^A \in \mathbb{N}^+$ which is paid when at least one virtual connection uses it. The sum of substrate node and arc usage costs, the total usage cost C_u, is the objective to be minimized.

As already mentioned, just finding a valid solution to this problem is NP-hard, so we cannot expect efficient heuristic methods to always be able to find valid solutions. For optimization purposes, we want to be able to determine how far solutions are away from validity so we can prefer solutions closer to validity. To do this, every node $i \in V$ can increase its available CPU power by a_i^{CPU} units for the cost of C^{CPU} per unit and every arc $e \in A$ can increase its available BW by a_e^{BW} units for the cost of C^{BW} per unit. The sum of the costs for additional resources in the substrate will be denoted as C_a; for valid solutions $C_a = 0$. In our experiments we set $C^{\mathrm{CPU}} = 1$ and $C^{\mathrm{BW}} = 5$ to reflect the fact that it is easier to add CPU power to a router than to increase the BW of a data link.

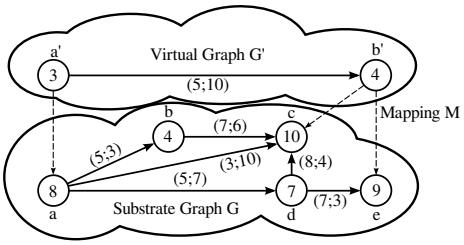

Fig. 1. Example of a VNMP instance.

For comparing two solutions, we used lexicographic ordering, i.e., smaller C_a is preferred and if it is equal smaller C_u is preferred.

Figure 1 shows a small VNMP instance. It contains the virtual network graph G' consisting of one VN with two nodes (a' and b'), the substrate graph G (nodes a to e) and the allowed mapping from the virtual network to the substrate nodes (dashed lines). Node labels define the CPU requirements for VN nodes and the available CPU power for substrate nodes. Costs have been omitted for clarity. Arc labels denote bandwidths and delays. Note that in this example, b' actually cannot be mapped to c, even though c offers enough CPU capacity. This is because there is no path from a to c that satisfies the constraints of the virtual connection between a' and b'. Node b cannot be used, because its CPU power of 4 is not enough to route the required BW of 5. The direct connection from a to c does not offer enough BW and the path using d exceeds the delay limit. The only feasible solution to this instance is to map a' to a and b' to e and implement the connection between a' and b' by the path (a, d, e).

In the following Sections, we will introduce construction heuristics, Local Search and Variable Neighborhood Descent algorithms for solving olso larger instances of the VNMP and show that depending on the available run-time, every heuristic can be the best choice. Local Search is the most versatile approach, which depending on configuration can either perform similar to the Construction Heuristics, to Variable Neighborhood Descent or somewhere in-between. We also compare the presented heuristics with the exact method proposed in [15].

2 Related Work

Virtual Network Mapping Problems have received considerable scientific interest in recent years due to their relevance to Future Internet research (e.g. [20,5,9,11,17,21,23,24,25]). The core problem solved in these works is the same: virtual networks have to be realized by means of a physical network. The details however, are always different. This can already be seen when comparing the names of the problems. Typical names include Virtual Network Embedding, Virtual Network Assignment or Network Testbed Mapping. A further area for differences are the resources the virtual networks require. The one demand that is nearly universally considered is bandwidth, but there is no consensus on how

this demand is taken into account. One method is to use traffic bounds to describe a whole range of BW requirements that there has to be a feasible routing for all of them (e.g. [9,17]). Another is to specify the node-to-node communication demand in the form of a traffic matrix (e.g. [23]). If another resource is taken into account, it is the required CPU processing power of each virtual node (e.g. [24,20]). The considered substrate sizes vary between 20 [17] and 100 [25] nodes and are either real or synthetic topologies.

VNMPs have been solved by simulated annealing [21], (quadratic) mixed integer programming [5,9,17,15], approximation algorithms [9], distributed algorithms [11], multicommodity flow algorithms [23,24] or algorithms especially tailored to the considered problem variant [25,20]. To the best of our knowledge, this is the first application and comparison of the trade-off of construction heuristics, Local Search and Variable Neighborhood Descent in the context of virtual network mapping. The VNMP variant solved in this work is very general, since it considers CPU and BW resources, path delays, mapping constraints and the influence of routing overhead on CPU resources. Therefore most algorithms presented in previous work do not apply, with the exception of the exact approach of [15], which we will use for comparisons. Furthermore, we use test instances that are freely available for comparison and designed with a focus on realism, in both size (up to 1000 nodes) and structure.

3 Construction Heuristics

A Construction Heuristic (CH) is used to create solutions to problems by following heuristic rules that guide the construction process towards feasible solutions of high quality. For the VNMP, we can already see that these are conflicting objectives; guiding towards feasibility means spreading resource use across the whole substrate, which causes C_u to be unnecessarily high. Trying to pack VNs densely will most likely lead to high C_a, so some kind of balancing is required. Fortunately, constructing a solution to the VNMP can be split into four different phases that are iterated and in each phase we can focus on different aspects of the solution. The four phases are: selecting a virtual node to map (SVN), selecting a target for the node (TVN), selecting a virtual arc to implement (SVA) and implementing the arc (IVA). Additionally, there can be an emphasis on mapping the nodes (NE) or an emphasis on implementing virtual arcs (AE). With NE, all nodes will be mapped before virtual arcs are implemented. With AE, all implementable virtual arcs will be implemented before a next virtual node is mapped. Since a virtual arc f can only be implemented if $m(s(f))$ and $m(t(f))$ have already been fixed, AE variants will also start by mapping at least two nodes, but then mapping of virtual nodes and implementing virtual arcs will be interleaved instead of strictly sequentially as it is done with NE.

Table 1 lists the considered strategies. Note that for the TVN strategies, only substrate nodes allowed by M are regarded. If a strategy does not find a feasible node, the one with the most free resources is chosen. The SVA strategies only consider implementable virtual arcs. All IVA strategies implement a virtual arc

Table 1. Implemented SVN, TVN, SVA and IVA strategies.

Name	Description
SVN1	Selects the next unmapped node of V'.
SVN2	Selects the node with the highest sum of CPU requirement and connected BW.
SVN3	Selects with SVN2 from the VN with highest total CPU and BW requirement that has still unmapped nodes left. Concentrating on one VN when selecting nodes supports AE variants, because virtual arcs become implementable much faster.
SVN4	Selects with SVN2 from the VN with the lowest total sum of allowed delays and unmapped nodes.
TVN1	Maps a virtual node to the first substrate node with enough free CPU capacity.
TVN2	Maps to the first substrate node with enough free CPU capacity to support the CPU requirements and the total connected BW of the virtual node (total CPU load).
TVN3	Maps to the substrate node with the most free CPU capacity. If there are multiple choices, the one with the most free incoming and outgoing BW is used as map target.
TVN4	Maps to the substrate node with enough free resources (w.r.t. total CPU load) and least increase of C_u.
SVA1	Selects the next arc.
SVA2	Selects the arc with the highest BW requirement.
SVA3	Selects arc with the smallest delay.
SVA4	Selects the arc f with the smallest fraction of allowed delay to shortest possible delay between $m(s(f))$ and $m(t(f))$.
IVA1	Arcs have a cost of 0 if they are already used, or their usage cost otherwise. Arcs without enough free BW cost 10^6.
IVA2-n	The cost of an arc is the sum the fraction of the arcs free BW the virtual arc would use and the fraction of free CPU power the virtual arc would use on the node the substrate arc connects to. This cost is then taken to the power of $n \in \{0.5, 1, 2\}$.

f by finding a Delay Constrained Shortest Path in the substrate from $m(s(f))$ to $m(t(f))$ via the Dynamic Program from [8]. The only difference between the strategies is the calculation of the substrate arc costs, which define the length of a path. If the following strategies define no specific order of nodes or arcs, it is arbitrary.

These strategies result in a total of 512 different construction heuristics, the results of their evaluation can be found in Sect. 7. The strategies were kept simple to keep running times short as the following heuristics build on the best CH variants.

4 Local Search

The basic idea of Local Search (LS) is that a found solution to a problem may be improved by iteratively making small changes. The solutions immediately reachable from a starting solution S are defined by a neighborhood $N(S)$. LS starts with a solution S and replaces it with a better solution from $N(S)$ until no more improvements can be found. For selecting the neighbor, we use the two standard strategies first-improvement (select the first improving solution) and best-improvement (select the best solution from a neighborhood).

Table 2. Implemented neighborhoods for LS.

Name	Description
N_1	Removes the implementation of an arc.
N_2	Removes a virtual node and the implementations of its adjacent arcs.
N_3	Removes all virtual nodes and implementations of all virtual arcs of a VN.
N_4	Removes the implementation of all virtual arcs using a specific substrate arc.
N_5	Removes all virtual nodes and the implementation of all arcs using a specific substrate node.
N_6	Like N_2, but tries mapping the virtual node to all allowed substrate nodes instead of delegating this task to the CH during rebuilding.

The six implemented neighborhoods are listed in Table 2. They all share the common idea that they remove a part of a complete solution (like the implementation of a virtual arc) and then complete the solution again by applying a CH. The neighborhood descriptions will skip this rebuilding step.

For each neighborhood, there is a natural order in which to evaluate the neighbors (e.g., clearing the first substrate node, clearing the second one and so on). However, we might be able to speed up the search process by trying other, more promising, neighbors first. For finding valid solutions, the most promising neighbors are those that might change C_a, e.g., changing the mapping of a virtual node that is mapped to an overloaded substrate node. We will call this strategy OverloadingFirst. A more extreme variant of this is OnlyOverloading, which only considers the neighbors that OverloadingFirst prefers.

5 Variable Neighborhood Descent

The neighborhoods discussed in the previous section can be applied in combination with a variable neighborhood descent (VND) algorithm [10]. A VND utilizes a series of neighborhoods $N_1 \dots N_k$. An initial solution is improved by N_1 until no more improvements can be found, then N_2 is applied to the solution and so on. If N_k fails, VND terminates. If an improved solution is found in some neighborhood, VND restarts with N_1. We use the neighborhoods of the previous section in two variants: as described and in the OnlyOverloading variant, which we will denote with a prime. For example, N_5' is the neighborhood of all solutions reachable by clearing an overloaded substrate node. Table 3 lists the tested neighborhood configurations.

6 Test Instances

This Section describes the used VNMP instances. The substrates are subgraphs of the nren network [16] which contains European research networks and their interconnects. It's one of the largest freely available networks based on physical network structure (1100 nodes) and has geo-location information embedded which is used for defining meaningful mapping constraints. The instance set

Table 3. Implemented neighborhood configurations for VND.

Name	Description
C_1	$N'_1\ N'_2\ N'_3\ N'_4\ N'_5\ N'_6\ N_1\ N_2\ N_3\ N_4\ N_5\ N_6$. All neighborhoods, in order of their size.
C_2	$N'_1\ N'_2\ N'_3\ N'_4\ N'_5\ N'_6$. All OnlyOverloading neighborhoods.
C_3	$N_1\ N_2\ N_3\ N_4\ N_5\ N_6$. All complete neighborhoods.
C_4	$N_6\ N_5\ N_4\ N_3\ N_2\ N_1$. Neighborhoods that produce the largest changes first.
C_5	$N'_1\ N'_2\ N'_3$. Only neighborhoods of C_2 yielding improvements based on preliminary results.
C_6	$N'_3\ N'_2\ N'_1$. C_5 in reverse order.

Table 4. Properties of the VNMP instances: average number of substrate nodes (V) and arcs (A), virtual nodes (V') and arcs (A'), total usage costs (C) and the average number of allowed map targets for each virtual node ($M_{V'}$).

Size	$\|V\|$	$\|A\|$	$\|V'\|$	$\|A'\|$	C	$\|M_{V'}\|$
20	20	40.8	220.7	431.5	1536.0	3.8
30	30	65.8	276.9	629.0	2426.6	4.9
50	50	116.4	398.9	946.9	4298.1	6.8
100	100	233.4	704.6	1753.1	8539.1	11.1
200	200	490.2	691.5	1694.7	17584.2	17.3
500	500	1247.3	707.7	1732.5	44531.8	30.2
1000	1000	2528.6	700.2	1722.8	89958.4	47.2

contains substrates of 20 to 1000 nodes, 30 instances of each size. The VN sizes are chosen uniformly at random from $[5, \min(30, 0.3 * |V|)]$. In order to reflect realistic use cases, the VNMP instances contain 10 VNs of each of four different types: Stream, Web, Peer-to-Peer (P2P) and Voice-over-IP (VoIP).

Stream VNs have a tree structure and model video streaming services. They have high BW and CPU requirements but are not delay constrained. Web VNs have a star structure and very low BW and CPU requirements but hard delay constraints. P2P and VoIP VNs have small world structure. P2P VNs have high BW and medium CPU requirements, but no delay constraints. VoIP VNs have medium CPU and BW requirements and moderate delay constraints.

Bandwidth and CPU capacities of the substrates are based on the requirements of a random implementation of all VNs. Table 4 shows the main properties of the instance set, which is available at [14].

7 Results

Each CH, LS and VND variant was tested on the full instance set as described in Section 6. Additionally, each instance was tested with different loads, i.e., reduced numbers of VNs. A load of 0.5 means that only 50% of the VNs of each type were used. Load levels of 0.1, 0.5, 0.8 and 1 were tested, which results in a total of 840 test instances (120 per size). The proposed algorithms 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 1000 seconds was applied.

We evaluated each algorithm from two points of view: Capability of finding valid solutions and capability of finding a best solution among all considered algorithms. To evaluate the second aspect, we cannot search for the lowest average C_u values, because higher values might be better if C_a is lower. Therefore we used the following ranking procedure to compare different algorithms: Considering a specific instance, the algorithm that achieves the best solution gets assigned rank 0, the second best rank 1 and so on. Algorithms with the same results share the same rank (no rank is skipped). The relative rank R_{rel} of an algorithm when solving a particular instance is its rank divided by the highest rank for this instance. Average R_{rel} values can be compared in a meaningful way. For example, an average R_{rel} of 0.1 means that the algorithm in question belongs to the top 10% of all compared algorithms over the compared instances.

7.1 Construction Heuristics

Before we could compare all implemented heuristics, we needed to identify promising CH variants which can be used to generate the initial solution for LS and VND and perform the rebuilding step of the proposed neighborhoods.

Considering the average R_{rel} of each construction heuristic variant over all tested instances, the best construction heuristic (CH1) reached a R_{rel} of 0.093. It used the strategies SVN3, TVN3, SVA4 and IVA1 with AE and was able to find valid solutions to 60.8% of all instances. This strategy is geared towards reducing C_u. For initialization purposes it might be interesting to use a strategy that focuses on finding valid solutions and leave cost reductions to the used neighborhoods, so we changed the IVA to IVA2-1. This variant is denoted with CH2 and is able to find valid solutions to 70.8% of all instances. The results showed that more than the 100 best CH variants (according to R_{rel}) used TVN3, which introduces a strong bias towards validity that might hamper LS and VND during the search for minimal C_u with neighborhoods that remove the mapping of a node. So for the third CH variant (CH3), we changed the TVN of CH1 to TVN4. Both CH1 and CH2 were tested for initialization, all three were tested for rebuilding. This led to 216 LS and 72 VND variants. Now follows their evaluation and comparison.

7.2 Comparing CH, LS and VND

Figure 2 shows the trade-off between low R_{rel} and low run-time for all tested heuristics over all instances. Label (A) marks the best non-dominated construction heuristics. They all use SVN4 and TVN3. They emphasise implementing arcs, so the selected SVA strategy has not a lot of influence. This can be seen here since the two visible configurations are actually multiple configurations using different SVA strategies. The faster but slightly worse clusters use IVA2-1, while the better CH methods use IVA2-2.

Label (B) marks the first LS strategies. They use N_2 in the OnlyOverloading variant with first-improvement. Because they use the reduced neighborhoods, they are very fast, even faster than some of the tested CH variants. A marginally

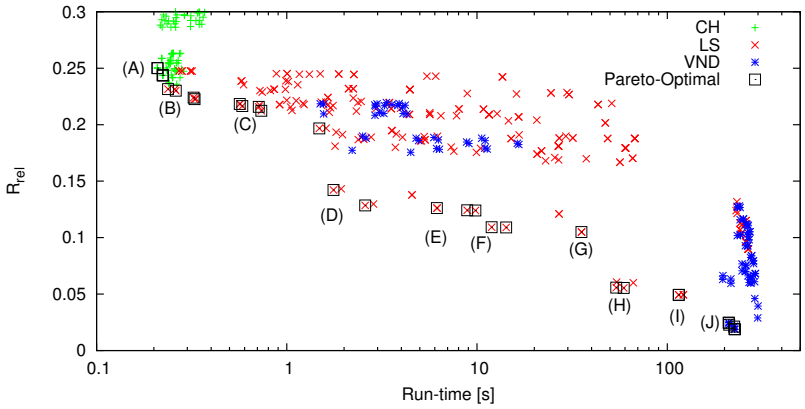

Fig. 2. Pareto-front of the tested heuristics regarding average R_{rel} and run-time over all instances.

better ranking can be achieved by initializing with CH1 instead of CH2. The small visible differences are caused by different rebuilding strategies.

The LS variants at (C) use neighborhood N_4 (faster) and N_5 (slower). Otherwise they are equivalent to the better ranked variants at (B).

The run-time jump from (C) to (D) is caused by not using the neighborhoods in the OnlyOverloading variant. Also, starting with the LS configurations at (D), only CH3 is used to rebuild solutions. The effect of using CH3 can be seen comparing the unlabeled LS configuration between (C) and (D) and the faster variant at (D). They are equivalent except for the rebuilding strategy. The variants at (D) use N_3 with first-improvement and OverloadingFirst. Again using CH1 instead of CH2 for initialization causes better ranking but longer run-times. The variants close to the marked ones do not use OverloadingFirst. Variant (E) offers a slight improvement in rank at a high run-time cost by switching to best-improvement. The pattern visible at (D) and (E) is repeated twice with (F) and (G), and (H) and (I). The difference is the used neighborhood, at (F) N_2 is used, at (H) N_5.

The heuristics at (J) mark the emergence of VND as best solution heuristic. Using VND instead of the best LS variant halves R_{rel} at the cost of doubling the average run-time. The two visible different clusters are caused by the difference between first-improvement (faster) and best-improvement (lower R_{rel}). Both clusters contain VND variants using C_1 and C_3.

Figure 3 shows the trade-off between solving instances, i.e., just finding a feasible solution and low run-time. Label (A) marks the best non-dominated CHs. They use SVN4, TVN3 and IVA2-1 and AE. Seven percent more instances can be solved when switching to SVN3, marked with (B). The small differences in performance visible at (A) and (B) are caused by different SVA strategies, SVA2 performs better than SVA3.

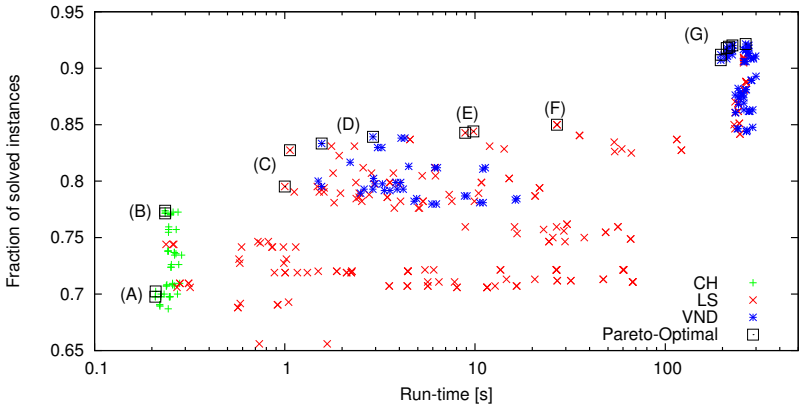

Fig. 3. Pareto-front of the tested heuristics regarding the fraction of solved instances and run-time over all instances.

The heuristics at (C) are the first Pareto-optimal LS heuristics. They use N_3 with first-improvement in the OnlyOverloading configuration. Both use CH2 to construct the initial solution. Using CH3 instead of CH1 for rebuilding causes the increase in solved instances. All of the following algorithms use this configuration for initialization and rebuilding.

The fist VND variants (using C_6) can be seen at (D). Since they start with the same neighborhood as the LS at (C) but also search other neighborhoods, they perform slightly better than LS. The better but slower variant at (D) uses best-improvement instead of first-improvement.

Once again we can observe the run-time increase caused by considering the complete neighborhoods, this time between (D) and (E). The LS variants at (E) use N_2 with first-improvement and prioritisation (faster) or without (marginally better). The variant at (F) is the same as the faster one at (E), but with best-improvement. The VND configurations C_1 and C_3 can be found at (G). The faster configurations use first-improvement, the others best-improvement.

We further compare the heuristics with the Integer Linear Programming (ILP) approach presented in [15] (with slight modifications to account for differences in the problem definition). For solving the ILP with CPLEX 12.4 [12], we used a time-limit of 10000 seconds and a memory limit of 4 GB. It has to be noted that this method solved instances none of the tested heuristics could and even solved four instances of the largest size to optimality. All instances of size 20 could be solved to optimality with an average run-time of 131 seconds. The best algorithm variants of all classes are compared in Table 5. Among other things, it shows the average relative decrease in C_u required (C_u-Gap) for each algorithm to match the performance of the ILP. This value is only based on instances that could be solved by the considered algorithm. Due to space limitations we cannot show a more detailed analysis here and refer instead to [13].

Table 5. The best algorithms of each class (according to the number of solved instances S or their average R_{rel}), their average runtime t and C_u-Gap over all instances. Bracketed values are the number of instances considered for calculating C_u-Gap.

Algorithm	S	R_{rel}	t[s]	C_u-Gap[%]
CH: SVN4, TVN3, SVA4, IVA1, AE	511	0.236	0.3	24.0 (434)
CH: SVN3, TVN3, SVA1, IVA2-1, AE	650	0.248	0.2	30.8 (512)
LS: N_2, best-improvement, OverloadingFirst, CH2, CH3	703	0.049	115.0	8.7 (511)
LS: N_6, best-improvement, OverloadingFirst, CH2, CH1	764	0.096	258.5	18.3 (518)
VND: C_1, best-improvement, CH2, CH3	773	0.019	226.9	7.1 (519)
VND: C_1, best-improvement, CH1, CH2	774	0.093	264.7	18.1 (520)
ILP	527	-	2466.0	0.0 (527)

8 Conclusion

In this work, we compared 512 CH, 216 LS and 72 VND algorithms. We could show that for the VNMP, each algorithm class has its application area: CHs for finding solutions fast, VND for finding the best solutions and LS covering the range in-between, depending on the used neighborhoods. For CHs, the most important strategy is the target choice for virtual nodes, so this is a clear area of interest for future improvements. For LS we could see that best-improvement works slightly better than first-improvement, but at a significant run-time cost. Reducing the neighborhood size also reduced the performance, but brought the execution speed into CH territory. VND benefited from the reduced neighborhoods too when searching for valid solutions. For LS, the initialization strategy has a pronounced influence on the result. For the best ranking, CH1 was used while CH2 was the better strategy when comparing the number of found valid solutions. The discussed VND variants produced the best results, but at a high run-time cost. Some fine-tuning with respect to the neighborhood configurations is still necessary.

References

1. Anderson, T., Peterson, L., Shenker, S., Turner, J.: Overcoming the Internet impasse through virtualization. Computer 38(4), 34 – 41 (2005)
2. Berl, A., Fischer, A., de Meer, H.: Virtualisierung im Future Internet. Informatik-Spektrum 33, 186–194 (2010)
3. Carlson, M., Weiss, W., Blake, S., Wang, Z., Black, D., Davies, E.: An architecture for differentiated services. IETF, RFC 2475 (1998)
4. Chowdhury, N.M.K., Boutaba, R.: A survey of network virtualization. Computer Networks 54(5), 862 – 876 (2010)
5. Chowdhury, N., Rahman, M., Boutaba, R.: Virtual network embedding with coordinated node and link mapping. In: INFOCOM 2009. pp. 783 –791 (2009)
6. 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)

7. GENI.net: Global Environment for Network Innovations. http://www.geni.net (2012)
8. Gouveia, L., Paias, A., Sharma, D.: Modeling and solving the rooted distance-constrained minimum spanning tree problem. Computers & Operations Research 35(2), 600–613 (2008)
9. Gupta, A., Kleinberg, J., Kumar, A., Rastogi, R., Yener, B.: Provisioning a virtual private network: a network design problem for multicommodity flow. In: STOC '01. pp. 389–398 (2001)
10. Hansen, P., Mladenović, N.: Variable neighborhood search: Principles and applications. European Journal of Operational Research 130(3), 449 – 467 (2001)
11. Houidi, I., Louati, W., Zeghlache, D.: A distributed virtual network mapping algorithm. In: Communications, 2008. ICC '08. IEEE International Conference on. pp. 5634 –5640 (2008)
12. IBM ILOG: CPLEX 12.4. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer
13. Inführ, J., Raidl, G.R.: Data supplement, `https://www.ads.tuwien.ac.at/projects/optFI-wiki/images/a/a7/DataSupplement.pdf`
14. Inführ, J., Raidl, G.R.: The Virtual Network Mapping Problem benchmark set, `https://www.ads.tuwien.ac.at/projects/optFI/`
15. 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.) Network Optimization: 5th International Conference, INOC 2011. LNCS, vol. 6701, pp. 105–117. Springer, Hamburg, Germany (2011)
16. Knight, S., Nguyen, H., Falkner, N., Roughan, M.: Realistic network topology construction and emulation from multiple data sources. Tech. rep., The University of Adelaide (2012)
17. Lu, J., Turner, J.: Efficient mapping of virtual networks onto a shared substrate. Tech. rep., Washington University in St. Louis (2006)
18. National Research Council: Looking Over the Fence at Networks. National Academy Press (2001)
19. Ramakrishnan, K.K., Floyd, S., Black, D.: The addition of explicit congestion notification (ECN) to IP. IETF, RFC 3168 (2001)
20. Razzaq, A., Rathore, M.S.: An approach towards resource efficient virtual network embedding. In: Proceedings of the 2010 2nd International Conference on Evolving Internet. pp. 68–73. INTERNET '10, IEEE Computer Society (2010)
21. Ricci, R., Alfeld, C., Lepreau, J.: A solver for the network testbed mapping problem. Special Interest Group on Data Communication Comput. Commun. Rev. 33(2), 65–81 (2003)
22. Schwerdel, D., Günther, D., Henjes, R., Reuther, B., Müller, P.: 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, pp. 1–10. Springer (2010)
23. Szeto, W., Iraqi, Y., Boutaba, R.: A multi-commodity flow based approach to virtual network resource allocation. In: Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE. vol. 6, pp. 3004 – 3008 vol.6 (2003)
24. Yeow, W.L., Westphal, C., Kozat, U.: 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. pp. 33–40. VISA '10, ACM, New York, NY, USA (2010)

25. Zhu, Y., Ammar, M.: Algorithms for assigning substrate network resources to virtual network components. In: INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings. pp. 1 –12 (2006)