Generating Virtual Network Embedding Problems with Guaranteed Solutions

Andreas Fischer and Hermann de Meer

***Abstract*—The efficiency of network virtualization depends on the appropriate assignment of resources. The underlying problem, called Virtual Network Embedding, has been much discussed in the literature, and many algorithms have been proposed, attempting to optimize the resource assignment in various respects. Evaluation of those algorithms requires a large number of randomly generated embedding scenarios. This paper presents a novel scenario generation approach and demonstrates how to produce scenarios with a guaranteed exact solution, thereby facilitating better evaluation of embedding algorithms.**

I. Introduction

THE rising complexity of todays communication networks demands new approaches to network management. The flexibility introduced with network virtualization [1] can help to achieve that goal. Network virtualization allows network operators to define virtual network topologies on demand. On a given Substrate Network (SN), arbitrary Virtual Networks (VNs) can be realized. The VNs are logically separated from each other, thereby ensuring parallel execution of diverse networked applications with minimal mutual influence and even providing some security through compartmentalization.

There is, however, the problem of **assigning** resources for such networks in an efficient way. **An** SN always has limited resources, and each further VN consumes some of those resources. This calls for resource assignment algorithms, trying to figure out for a given set of VNs how to best embed them into the SN. This problem is commonly known as Virtual Network Embedding (VNE) [2]. For the last few years, **many algorithms have** been proposed, providing solutions under **various** aspects of this problem.

The evaluation of these algorithms depends largely on randomly generated problem instances, as real-world scenarios are not available in sufficient numbers. Algorithms are evaluated with a simulator that generates random embedding scenarios and computes metrics to judge the performance of the algorithm. The random generation process is hard to control, however, and it is difficult to generate problem instances which are hard and which serve to exhibit particular properties of VNE algorithms. To improve this situation, in this paper three main contributions are presented by the authors:

- **An architectural pattern which allows** more flexible generation of evaluation scenarios;
- The concept of perfectly solvable scenarios—scenarios that have a known optimal solution;
- Two novel scenario generation approaches that enable generation of scenarios with an optimal solution that is known a priori to the experimenter.

A. Fischer and H. De Meer are with the University of Passau, Faculty of Computer Science, Innstr. 43, 94032 Passau, Germany. E-mail: {andreas.fischer,demeer}@uni-passau.de.

The contributions focus on VNE problems for which it is assumed that no co-hosting of nodes from the same VN is allowed, virtual links are unsplittable, and node and link resources are integral. The **new** scenario generation approach **is** implemented in a publicly available VNE simulator, providing the opportunity for researchers to generate problem instances that allow to **compare** the solution quality of any VNE algorithm **to** a known optimum.

The rest of the paper is structured as follows: Section II discusses background and related work. Section III introduces the problem of **solvable scenario generation**. Section IV discusses the generation process and **proposes a more flexible architectural pattern**. In Section V the concept of perfectly solvable scenarios is defined, and two elements for generating such scenarios are presented and analyzed. Section VI concludes the paper and gives directions for future research.

II. Background and Related Work

Work related to this paper can be roughly categorized in three areas: Experimental evaluation of algorithms, VNE in general, and simulation of VNE algorithms in particular.

A. *Experimental evaluation of algorithms*

Experimental evaluation of algorithms is not a new field. There has been a lot of work on experimental algorithmics (cf. [3], [4], [5], [6], [7]). Rardin and Uzsoy focus particularly on evaluation of heuristic algorithms [8]. Berberich et al. [9] discuss in particular the design of experiments for algorithmic evaluation. McGeoch [10] provides a good introduction on the subject. Most of the work, however, focuses on classical $\mathcal{NP}$ problems—the application to VNE algorithms has not been investigated, so far.

As for generation of problem instances, again various approaches exist for classical $\mathcal{NP}$ problems such as bin packing [11], the travelling salesman problem [12], graph partitioning [13], and other $\mathcal{NP}$-hard graph problems [14]. **These are not easily adaptable to the VNE problem, which, in the variant considered here, is actually the combination of two $\mathcal{NP}$ problems—node embedding and link embedding.** Current approaches for generation of VNE problem instances **up to now** rely on only loosely controlled random processes, without much discussion on the effects and properties of the generated problem instances.

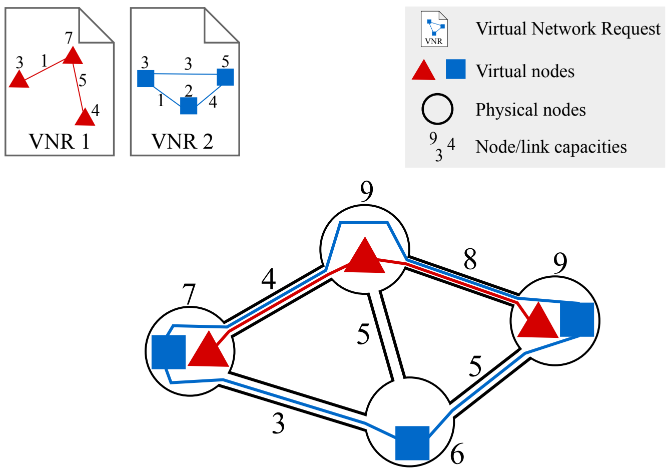


Fig. 1. Embedding two virtual network requests into one substrate network **(Example adapted from [2])**

B. Virtual Network Embedding

Virtual Network Embedding is a resource assignment problem at its core: Virtual nodes have to be assigned to nodes from **an** SN and the interconnecting virtual links have to be assigned to appropriate paths in the SN. Both substrate nodes and links have limited resources, whereas their virtual counterparts impose respective demands on those resources. **Here, it is assumed that resources and demands are represented by integers. It is also assumed that nodes of a single VN may not be assigned to the same substrate node. Moreover, virtual links are assumed to be unsplittable. These assumptions reflect settings found in a significant part of the literature (cf. [2]).**

Fig. 1 shows a problem instance with two Virtual Network Requests (VNRs) to be embedded in a SN, with a possible embedding depicted, already. While the given example is simple, the generic problem of trying to embed as many VNRs as possible into a given SN with limited resources is $\mathcal{NP}$-hard [15]. Algorithms attempting to solve this problem, therefore, mostly use heuristic or meta-heuristic approaches. This raises **an interesting question: Which algorithm can provide better solutions in a given situation?**

The first approaches of resource assignment for virtual networks have come up around 2006 (cf. [16], [17], [18]). Since then, a very large number of VNE algorithms has been proposed in the literature. Surveys on the topic are provided by Belbekkouche et al. [19] and Fischer et al. [2].

Comparative analysis of VNE algorithms is a building block of VNE evaluation (cf. [20]). Most authors reuse the same set of parameters for problem generation—often slightly modified. To increase comparability of different experiments, Zhu and Wolf [21] propose a common set of parameters to serve as a benchmark for VNE algorithms. However, the issue of random generation of VNE problems has not been discussed in detail in the literature, so far.

C. Simulation of VNE algorithms

Simulation of VNE algorithms can be performed in two different ways, depending on whether the algorithm is evaluated in an online or an offline environment. In an offline environment, the algorithm has full knowledge, and temporal aspects (apart from algorithm runtime) are not in the focus of evaluation. In online evaluation, the problem changes over time, as VNRs enter and leave the system (cf. Fischer et al. [2]). The focus here is on offline evaluation. This does not impede applicability of the discussed concepts, as every online algorithm can easily be evaluated in an offline environment. The extension to online evaluation is left as future work.

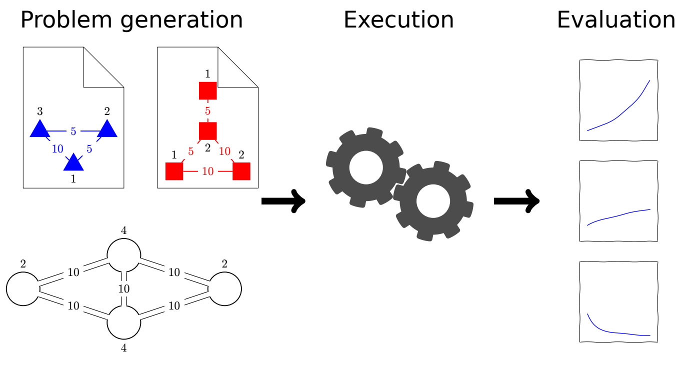


Fig. 2. The three steps of **offline** VNE simulation

Offline simulation of VNE algorithms consists of three distinct stages: Problem generation, algorithm execution, and evaluation of results (cf. Fig. 2).

First, problem instances have to be generated. In the context of VNE this means that substrate and virtual topologies have to be created and resources and demands have to be assigned to create the respective SNs and VNRs. Typically, the problem instances are randomly generated **(e. g., by creating random Waxman topologies and assigning random resources and demands)**. In that case, a number of parameters drives the generation of substrate and virtual networks (e. g., number of nodes in a network). The result of this step is a generated scenario for further evaluation. Such a scenario is defined as follows:

Definition 1: A *scenario* is represented by a tuple $(SN, (VNR_i)_{i=1,\ldots,n})$, consisting of a substrate network SN and a sequence of virtual network requests (VNR_i).

Once the scenario has been generated, it is fed to the VNE algorithm for experimentation. The algorithm tries to embed as many of the given VNRs as possible into the SN. Once the algorithm is done, results are evaluated and interpreted with the help of various metrics.

A number of simulation frameworks specializing on VNE simulation has been used in the VNE literature. Four of these have been made publicly available:

Yu et al. [22] presented the VNE Simulator[1]. It is written in ANSI C, uses the GT-ITM topology generator [23] and has been used in multiple further publications (cf. [24], [25], [26]).

Chowdhury et al. [27], [28] introduced the Vineyard simulator[2]. Similar to the VNE Simulator it uses GT-ITM to generate topologies. It is written in C++ and has also been used for multiple publications, already (cf. [29], [30], [31]).

[1]https://github.com/USC-NSL/embed
[2]http://www.mosharaf.com/ViNE-Yard.tar.gz

Another simulator—the CVI-Sim[3]—has been proposed by Papagianni et al. [32]. It is written in Java and can generate Erdös-Rényi topologies. The SN can also be generated from a PlanetLab topology, instead. It has also been used multiple times in the literature, already (cf. [33], [34], [35]).

The authors have contributed to the ALEVIN[4] simulator [36], [37]. It is written in Java and has been developed to support comparative algorithm analysis [20]. It can generate multiple types of topologies (e. g., Waxman or Erdös-Rényi) or import pre-generated topologies from multiple formats (e. g., SNDLib [38]). Like the other simulators, it has been used multiple times in the literature (cf. [39], [40], [41], [42]).

The extensibility and the focus on comparative algorithm analysis of the ALEVIN simulator provide a solid foundation for integrating the scenario generation approaches presented in this paper. Therefore, the concepts proposed in this paper have been implemented in the ALEVIN simulator. They are available online as open source.

III. Problem Statement

When generating embedding scenarios for algorithm evaluation, one should strive to generate scenarios that are "good". The obvious question is: What constitutes a "good" scenario? Embedding scenarios are supposed to highlight particular qualities or properties of the VNE algorithm under investigation. Some types of scenarios are clearly not a good fit. E. g., a scenario that is trivial to solve by any algorithm will hardly help to shed insight onto the behavior of said algorithm.

This poses a challenge in particular to random generation processes. Generation parameters have to be chosen such that the solution to a given scenario is non-trivial for the algorithm. A straightforward approach is to set generation parameters such that the generated resources are sparse in comparison to the generated demands.

Considering that substrate resources are limited, any VNE algorithm will then have to deal at some point with a VNR that can not be embedded in the given SN. In this case, the VNR is typically rejected and embedding continues with the next VNR. There is, of course, still the option of deleting an already embedded virtual network instead, but in practice this will often be undesirable and, consequently, algorithms discussed in the literature rarely deal with that possibility.

There is a significant difference between a genuinely impossible scenario and a theoretically possible scenario that just could not be solved by a given VNE algorithm. Due to the $\mathcal{NP}$-hardness of the VNE problem, most VNE algorithms are heuristic and are therefore bound to make mistakes and reject a VNR although further search might have provided a solution. There are three separate cases to consider:

1) An algorithm fails to embed a VNR that is genuinely impossible to embed because the absolute number of substrate resources is too small.
2) An algorithm fails to embed a VNR that is genuinely impossible to embed because substrate resources are too fragmented.

[3]https://github.com/chrisap/CVI-SIM
[4]http://alevin.sf.net/

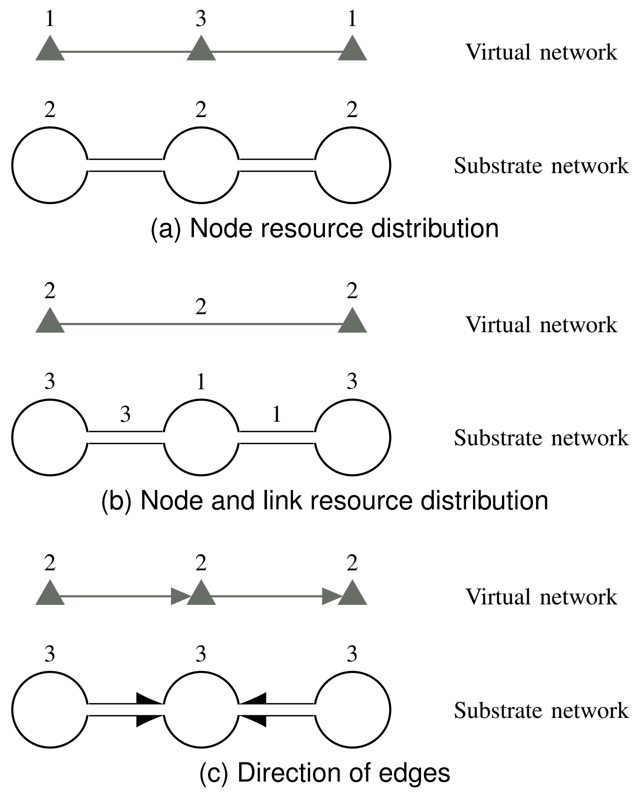


Fig. 3. Different types of impossible embedding scenarios and why they fail

3) An algorithm fails to embed a VNR although a solution would actually exist.

It is important to appreciate the difference between the three cases. The first case is easy to detect a priori. Even before starting the embedding process, one can sum up the virtual demands and compare them with the sum of all substrate resources. If the first number is higher than the second, the respective VNR should be rejected outright.

The second case is somewhat more subtle. Fig. 3 shows some examples. Here, three different configurations of a VNR and **an** SN are presented, where in all configurations it is impossible to embed the VNR, although the overall number of substrate resources appears to be sufficient. The first configuration depicts a scenario in which the VNR is impossible to embed because no substrate node can host the middle virtual node. Of course, a similar example could be set up with link resources, instead. In the second configuration it is the combination of node and link resources that causes a problem. The distribution of available node resources forces the virtual nodes to be embedded on the outer nodes of the substrate network. However, it is then impossible to realize the virtual link, as the right substrate link would be overloaded. The third configuration is specific to scenarios with directed graphs. **In this case, the direction of the virtual links clashes with the direction of the substrate links.** The available node resources dictate that each substrate node hosts at most one virtual node. **However, the second virtual link then cannot match the direction of the substrate link.** It is worthwhile to note that in all three **configurations** the sum of all virtual resources is less than the sum of all substrate resources, so it is not the absolute lack of substrate resources that causes these problems. Instead, the fragmentation (i. e., the unfortunate distribution) of the substrate resources causes the VNR to be rejected. In general it is difficult to detect such problems a priori—for the simulation environment as well as for the VNE algorithm. The algorithm will likely have to try to perform the (necessarily unsuccessful) embedding.

Finally, the third case is the most difficult one. Situations will occur where a solution is possible, but the algorithm fails to find it and rejects the VNR. This brings up an interesting observation: In the first two cases it was the specific configuration of SN and VNR that made embedding impossible. In contrast, in the third case it is a property of the algorithm whether embedding succeeds or not. Indeed, some algorithms may find a solution where others fail. This is highly relevant for evaluation, as the quality of VNE algorithms will be judged by their ability to find possible embeddings.

A major challenge, therefore, is to generate scenarios that fall into this third category: The scenarios should be possible to solve, but at the same time finding the solution should be non-trivial for any given VNE algorithm. In order to discuss how to generate such scenarios, first a more detailed look at the random scenario generation process is necessary.

IV. Construction of random embedding scenarios

Flexible scenario generation is a key requirement for a VNE simulator. Several different parameters such as network size, number of VNRs, or resource and demand ranges have to be supported. Moreover, various experiments will focus on different aspects of the VNE problem. For example, one experiment may investigate the effects of increasing the SN size, whereas another experiment may vary the distribution of resources and demands. This has to be reflected in the scenarios that have to be generated. In order to cover a sufficient range of possible configurations and to gain trust in evaluation results, scenarios are typically generated in large numbers with a random generation process. While it is certainly possible to implement individual scenario generators for each experiment, this leads to significant duplication of code and likely to additional bugs in software and, therefore, less trust in the simulation itself.

A better approach is to identify components that can be reused by different experiments. For example, generators for Waxman [43] and Erdös-Rényi [44], [45] topologies are likely to be used in many different experiments. Likewise, a random distribution of node and link resources and demands will have to be implemented in many experiments, as well. Identifying those components and exporting a common interface **allows the experimenter** to set up VNE experiments with minimal change to the code base. Ideally, the experimenter should be able to provide code that influences only a small, specific aspect of scenario generation without changing the rest of the generation process at all.

A. Conventional scenario generation

The main challenge for a scenario generation **approach** is to allow decomposition of functionality into individual items, while still supporting complex generation requirements. Conventionally, scenarios are generated by first using topology generators to create a set of topologies, one of which becomes the SN, and then randomly assigning weights (i. e., resources and demands) to the generated nodes and links.

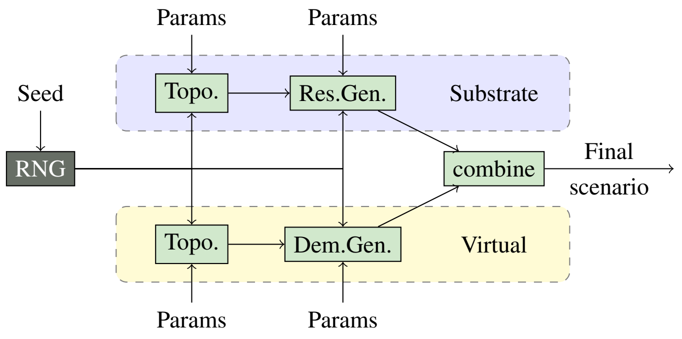


Fig. 4. Conventional random scenario generation process

Fig. 4 shows such a conventional scenario generation **approach**. Substrate and virtual networks are generated separately, each controlled by their own parameters. The substrate topology is handed to a resource generator, which assigns resources to nodes and links. Likewise, each virtual topology is handed on to a demand generator, assigning node and link demands. Typically, several VNR are generated by repeating the virtual network generation process multiple times. Also, resource and demand generators may actually be split into multiple generators, e. g., to have different random value ranges for node and link weights. At the end, **SN and VNRs are combined to form the final evaluation scenario.**

This approach is followed by all four VNE simulators presented before, although in some cases topology generation may be performed by an external tool like the BRITE or GT-ITM topology generators [46], [23]. While such an approach serves well for a number applications, there are still some shortcomings that call for a different approach:

- Avoiding the generation of scenarios that are impossible to solve completely is very hard **(i. e., it is hard to avoid “unfortunate resource fragmentation”; cf. Section III)**.
- Correlating the generation of virtual and substrate networks is not supported by this approach **(e. g., to stop generating demands when the sum of demands already exceeds the sum of resources)**.
- Changing the generation process itself is difficult and still requires changes to the simulation code base.

An approach that can overcome these shortcomings has to support in particular a more flexible combination of functional generation elements. **An architectural pattern to support this** is proposed next.

B. A chain of generation elements for scenario generation

In order to overcome the shortcomings of a conventional scenario generation process, this paper presents a more flexible approach. The **pipe-and-filter pattern** (cf. Hohpe and Woolf [47]) is **adapted** to the generation of embedding scenarios. Specifically, a number of scenario generation elements

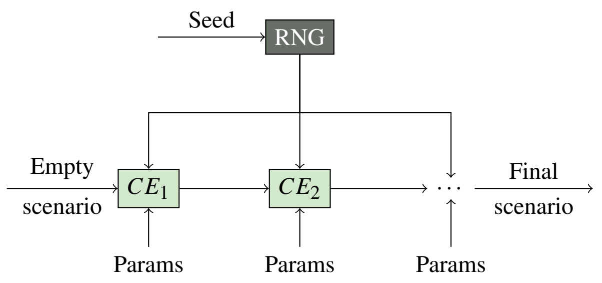


Fig. 5. A chain of scenario generation elements

(or "filters") are combined to form a chain of elements which generates a full scenario each time it is invoked.

In this scenario generation approach, a chain element is defined as follows:

Definition 2: Let $\mathcal{S}^{in}$ be a scenario, $\mathcal{R}$ be a Random Number Generator (RNG), and $\mathcal{P}$ be a sequence of parameters represented as key-value pairs. A *chain element* is then a function $CE : (\mathcal{S}^{in}, \mathcal{R}, \mathcal{P}) \mapsto \mathcal{S}^{out}$ that modifies a scenario according to the specified parameters with the help of the RNG, and returns the modified scenario.

Each individual element is responsible for a part of the scenario. The experimenter is responsible for specifying parameters that generate different scenarios for each parameter combination. A RNG is used as a source of randomness for the individual chain elements such that each invocation creates a different scenario. Assuming that the RNG is an implicit parameter to each chain element, a given element CE with parameters $(p1, p2, \dots)$ can be denoted as $[CE(p1, p2, \dots)]$ or simply $[CE]$, if parameters are clear from the context.

Chain elements can be simple or complex. The only requirement for them is to take a scenario and, based on their parameters, produce a scenario. This allows to concatenate an arbitrary number of these elements, not unlike the concept used in UNIX shell programming where individual console utilities are concatenated to form complex commands. Here, the concatenation of such elements produces a scenario generation chain:

Definition 3: Given a sequence of chain elements $(CE_i)_{i=1,\dots,n}$, a corresponding sequence of parameters $(\mathcal{P}_i)_{i=1,\dots,n}$ and a RNG $\mathcal{R}$. Let $\mathcal{S}_0$ be the empty scenario. A *scenario generation chain* is then the functional composition of the individual chain elements, generating intermediate results $(\mathcal{S}_i)_{i=1,\dots,n}$ such that $\forall i \in \{1, \dots, n\} : \mathcal{S}_i = CE_i(\mathcal{S}_{i-1}, \mathcal{R}, \mathcal{P}_i)$.

Each element takes the scenario that is generated by the preceding element, modifies it, and hands it on to the next element in the chain. The last result, $\mathcal{S}_n$, is the final generated scenario. Using the notation above, a given scenario generation chain with elements $(CE_i)_{i=1,\dots,n}$ can be written as:

$[CE_1]—[CE_2]—\dots—[CE_n]$

Figure 5 depicts this concept. Scenarios are generated by having the user specify a chain of scenario generation elements, a set of parameters to configure the elements, and a seed for the internal RNG. The chain is started with an initially empty scenario and produces the final scenario which is then handed on to the VNE algorithm by the simulator.

C. Replicating **the functionality of** a conventional scenario generation process

The conventional scenario generation process, as shown in Fig. 4, can be fully replicated by the concept of a scenario generation chain. By defining appropriate topology and constraint generation elements and chaining them in the right order, the same type of scenarios can be generated. The widely used approach of using Waxman topologies along with randomly assigned CPU and bandwidth resources and demands can be realized with the following elements:

Waxman (SN) This element generates a Waxman topology for the SN. It takes the number of nodes n to be generated, and the two Waxman parameters α and β as input. As a chain element, it is denoted as: $[W_S(n, \alpha, \beta)]$

Waxman (VNR) This element generates a number of Waxman topologies for the VNRs. Like its substrate counterpart, it takes the number of nodes n to be generated, and the two Waxman parameters α and β as input. Additionally, the number m of topologies to generate is taken as a parameter. As a chain element, it is denoted as: $[W_V(n, \alpha, \beta, m)]$

CPU and bandwidth resources These elements assign CPU resources to each substrate node and bandwidth resources to each substrate link. The respective values are taken from a random interval $[r_{low}, r_{high}]$. The interval may differ for CPU and bandwidth—therefore, two elements are actually defined: $[CPU_S(r^c_{low}, r^c_{high})]$ and $[BW_S(r^b_{low}, r^b_{high})]$

CPU and bandwidth demand Like the resource elements, these elements assign CPU demands to each virtual node and bandwidth demands to each virtual link. Virtual nodes and links are considered from all previously generated VNRs topologies. Again, the respective value is taken from a random interval $[r_{low}, r_{high}]$, which may differ for CPU and bandwidth. The chain element denotations are: $[CPU_V(r^c_{low}, r^c_{high})]$ and $[BW_V(r^b_{low}, r^b_{high})]$

In (shortened) chain notation a conventional scenario generation process that generates random Waxman topologies for substrate and virtual networks and assigns CPU and bandwidth resources and demands can then be denoted as:

$[W_S]—[CPU_S]—[BW_S]—[W_V]—[CPU_V]—[BW_V]$

Thus, with these elements in place, no functionality is lost compared to the conventional approach and the same type of scenarios can be generated.

D. Discussion

There are several advantages of the proposed new approach to scenario generation. Additional generation aspects are now easy to introduce via the concept of chain elements. For example, it is trivial to create and integrate generators for new types of resources and demands or additional topology generators. The chain structure for a particular experiment can be specified by the experimenter as needed, providing increased flexibility for experiment design. Moreover, **a generation chain implementing specific functionality can be defined at runtime** by the experimenter without having to change the simulation code base.

An additional advantage of this approach is repeatability of experiments. An experiment is fully specified by the order of chain elements and their respective input parameters. The RNG remains the only source of non-determinism. If a pseudo-random number generator based on an initialization seed is used, experiments become fully deterministic and the generated scenarios can be perfectly recreated on demand. This can be used not only to confirm previous results, but also to verify whether any change to a VNE algorithm has had effects on the results or not.

The scenarion generation chain concept, along with all of the chain elements defined above, has been implemented in the ALEVIN simulator. The results are publicly available online. Going beyond a conventional scenario generation approach, the scenario generation chain concept now also enables definition of more complex chain elements, taking previously generated parts of the scenario into account. This leads to the concept of a perfectly solvable scenario and the generation of such scenarios.

V. Perfectly solvable scenarios

An interesting question comes up when looking for scenarios that are neither trivial to solve, nor obviously impossible: Where is the exact turning point where solvable scenarios change to impossible scenarios? It seems that in this vicinity of the parameter space, the hardest scenarios are to be found. The exact turning point is then represented by a scenario that is just barely solvable—i. e., one where only optimal assignment of resources will lead to a successful embedding of all VNRs. For VNE evaluation it is very helpful to construct scenarios that fall close to this turning point. They allow to put VNE algorithms under stress, giving them the most difficult problems to solve.

A. Definition

There are various optimization criteria for VNE. Here it is assumed that optimization focuses on resource usage. An optimal solution is then defined as:

Definition 4: A solution for a scenario is called optimal if every other solution occupies **at least as many resources.**

An optimal solution is not necessarily unique: Two or more solutions can exist with the same resource consumption. **It should also be noted that whereas node demands are always satisfied by a single resource, regardless of the embedding, link demands may consume multiple resources if the respective virtual nodes are not assigned to adjacent substrate nodes. Thus, given two solutions for a problem, differences in resource consumption are always due to differences in path length of the virtual links.**

Bearing this in mind, a perfectly solvable scenario can be defined as follows:

Definition 5: A perfectly solvable scenario is one for which:

1) at least one solution exists.
2) every solution is also optimal.

I. e., every sub-optimal solution must necessarily lead to VNR rejection. An example of such a scenario is given in Fig. 6. Although maybe not immediately obvious at first, it is possible to embed both VNRs (a possible solution is shown below in Fig. 7). After embedding, all resources of the SN will be consumed.

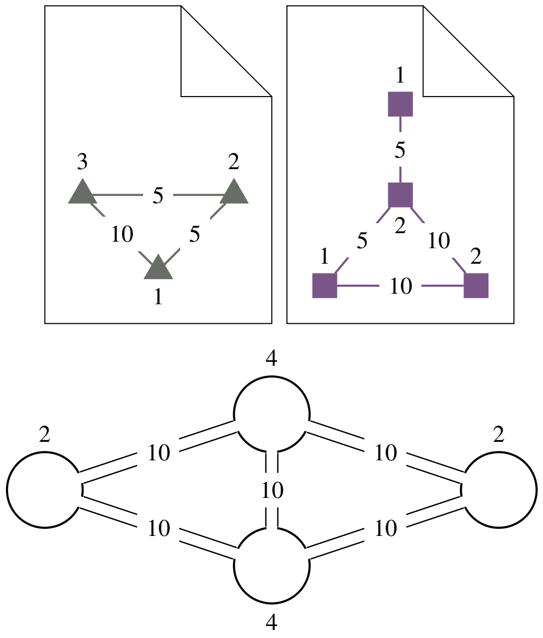


Fig. 6. Example of a perfectly solvable scenario with two VNRs

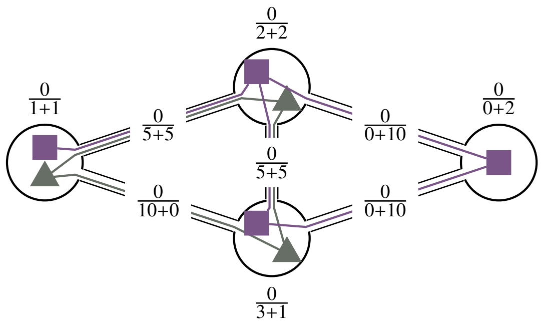


Fig. 7. A possible embedding solution. Resource occupation for nodes and links is denoted as: $\frac{\text{Remaining}}{\text{VNR1+VNR2}}$

While such scenarios can of course be constructed by hand, it is more relevant for experimentation to be able to construct them algorithmically with random variations, to be able to rapidly generate a large number of such scenarios such that a large part of the parameter space can be explored. The major advantage of such a scenario lies in the fact that if the optimal solution is known in advance, it can be compared to the solution produced by a VNE algorithm. The algorithm can then be judged in terms of how far its solution deviates from the known optimum.

Sanchis [48] shows that most $\mathcal{NP}$ problem generators that generate problem instances with known solutions are limited in that they **cannot** efficiently generate the full spectrum of possible problems. This means that any efficient problem generator is necessarily limited to generating only a particular type of problems. Properties of the instances generated by

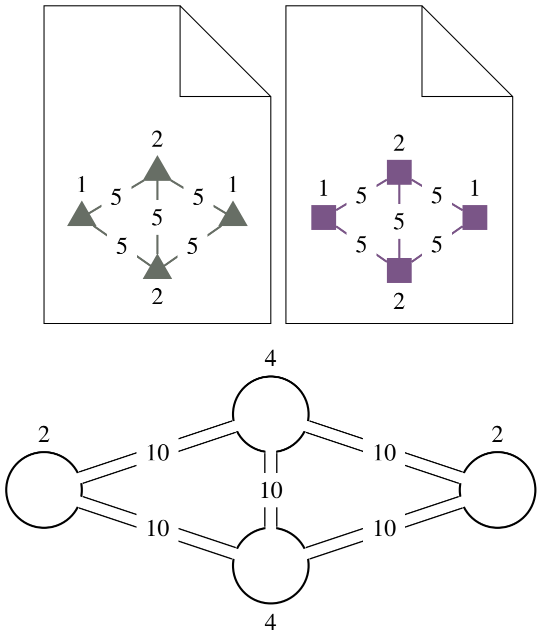


Fig. 8. Replication of the substrate network with scaled demands

such a generator thus have to be analyzed. Two examples of such scenario generators are discussed and analyzed here: A very simple generator that just duplicates the substrate topology, and a more complex generator that generates random subnetworks out of a given SN.

B. Substrate network replication

The first scenario generator presented here represents a very straightforward idea: Take any given SN, duplicate the topology, and replace all resources with demands. Use the resulting network as a VNR. Effectively, this creates a scenario, where the VNE algorithm tries to "embed **an** SN into itself". The solution to this VNE problem is, of course, trivial. Surprisingly, although the approach seems very obvious, no similar experiments have been mentioned in the VNE literature, so far. Still, it is worthwhile to check whether VNE algorithms are actually able to find the trivial solution. And, with a few small extensions, this scenario generator can already provide interesting insights into algorithm behavior.

1) Description of the scenario generator: The scenario generator presented here generates a given number of VNRs from any previously generated SN. Each generated VNR is topologically an exact copy of the SN. Demands in each VNR are scaled proportionally to the corresponding SN resources such that the total demand meets a predefined fraction of the available substrate resources. E.g., with ten VNRs, and each demand consuming 5% of its respective resource, exactly 50% of all substrate resources will be occupied in the optimal case.

An example is shown in Fig. 8. The SN is replicated twice, yielding two identical VNRs. Demands are scaled to 50% of their respective substrate resource for each VNR, such that the sum of all demands of both VNRs matches the sum of all resources in the SN. It is, of course, trivial to see that an embedding exists for this scenario, and that any successful embedding of both VNRs consumes all available SN resources. While the solution is not necessarily unique (in this case due to symmetry—the VNRs could be mirrored horizontally and/or vertically), it is clear that no better solution can exist.

```
Data: A pre-generated SN
Data: Number m of VNRs to generate
Data: Fraction f of SN resources to be demanded
Result: A set of m VNRs matching the SN in topology,
        such that at least a fraction f of each SN resource
        will be required for a successful embedding
while Not enough VNRs generated do
    Copy topology of SN into new VNR;
    for Each resource r in the SN do
        Generate a respective demand d in the VNR such
        that $d = (r \cdot f)/m$
    end
end
```
Algorithm 1: A scenario generator for SN replication

Pseudocode for this scenario generator is shown as Algorithm 1. Based on the generated SN, a number m of VNRs is created by copying the topology from the SN and assigning node and link demands in direct proportion to their respective substrate resources. Demands are scaled by a factor f. In the simplest case, a single VNR can be created as a perfect replication of the substrate network (with all resources converted to demands). As part of a scenario generation chain, this element will be denoted as: $[SNRepl(m, f)]$

This approach allows for precise control in two different dimensions: First, how many resources of the SN should be spent in the ideal case? This is easy to realize by just scaling virtual demands, respectively. For example, in order to generate a "load" of 20% for the SN, f is set to 0.2, and thus each demand in each VNR computes as the value of the respective substrate resource times 0.2, divided by the total number of VNRs. This allows to tune the resource scarcity of a particular scenario very precisely, making it possible to study the behavior of VNE algorithms in the face of varying resource scarcity levels.

Second, over how many VNRs should these resources be distributed in the optimal case? This can be controlled easily by **the factor** m. For example, to subdivide resources into ten different demands, m is set to 10, and thus each resource hosts ten demands in the optimal case. Thus, it becomes possible to study the behavior of algorithms when the granularity of the virtual demands increases.

Regarding VNRs that are topologically identical to the SN, there may be the problem that the internal data structures inadvertently give away the perfect solution to the problem. Indeed, a very naïve VNE algorithm that simply assigns virtual node i to substrate node i and performs shortest path mapping for the virtual links, would achieve a perfect score in these scenarios unless further measures are taken. For evaluation it is therefore advisable to randomize the order of virtual nodes in each VNR before embedding takes place.

2) Evaluation: To demonstrate the application of the new scenario generation element, it was implemented in the

ALEVIN simulator and experiments were performed with three different VNE algorithms:

- The vnmFlib algorithm (cf. Lischka and Karl [49]);
- The D-ViNE algorithm (cf. Chowdhury et al. [27]);
- The RW-MM-SP algorithm (cf. Cheng et al. [24]).

The experimentation machine was a AMD Opteron 4180 with 12 cores (only one core was used, however) and 8GB of RAM. The scenario generation chain was set up such that the SN was generated like in a conventional case, whereas the VNRs were generated with the new element:

$[W_S]$—$[CPU_S]$—$[BW_S]$—$[SNRepl]$

Two experiments were run: first, the number of VNRs m was varied and the influence on the algorithm runtime was measured. Second, the load parameter f was varied and the effect on algorithm runtime and the acceptance ratio (i. e., the number of embedded VNRs divided by the total number of VNRs) was measured. For each individual parameter setting, 30 runs were performed with different random seeds. Results are shown as notched box plots, using the standard interpretation of a box plot indicating the median, 95% confidence interval, upper and lower quartiles and outliers.

The results of the first experiment are shown in Fig. 9. Here, the load scaling factor f was set to 1, such that all generated scenarios were perfectly solvable scenarios. The number m of VNRs was varied from 5 to 25 in increments of 5, adding the special case of a single VNR. It can be seen that all three algorithms appear to have a linear runtime dependency on the fragmentation of resources. The more virtual networks there are, the longer the algorithms take. Acceptance ratio is not depicted for this experiment—however, it should be noted that with only a single VNR all algorithms found the correct embedding. For 25 VNRs, results are shown below.

Fig. 10 and Fig. 11 show the behavior of the algorithms when the load parameter f is varied. Here, the number m of VNRs was set to 25 and the load parameter f was varied between 0.1 and 1.1 in increments of 0.1. The last case of 1.1 was included to see the effect of scenarios which by definition **cannot** be solved completely. In this experiment, the three algorithms show distinct behavior. For vnmFlib, the runtime increases up to a load of about 0.8. This is correlated with its acceptance ratio, which decreases proportionally, indicating that this algorithm takes significantly longer when it is unsuccessful in embedding a VNR. While D-ViNE shows a similar decreasing acceptance ratio, there is no significant effect on algorithm runtime (though it should be noted that runtime for this algorithm was very high overall, compared to the other two algorithms). The RW-MM-SP algorithm is interesting here, as it manages to almost perfectly identify the artificial scenario as easy, and finds solutions very fast in almost all cases. This algorithm is also significant in that it quickly identifies the last VNR as impossible to embed when f is set to 1.1, therefore finishing slightly faster in that case.

3) Properties of the scenario generator: The scenario generator is straightforward to implement in any simulation environment. It has two control parameters: the number m of VNRs, and the fraction f of desired load on the SN. These two factors can be influenced independently from one another. **Thus, unlike in conventional scenario generators, it is now possible** to investigate the influence of resource scarcity without changing the demand granularity, and vice versa, thereby highlighting particular aspects of VNE algorithms.

One obvious drawback of this scenario generator is that the topology of the generated VNRs is always identical to the given SN. Although it enables a new type of evaluation, the investigated scenarios remain artificial and will be deemed unrealistic in most contexts. As a remedy, a means for generating scenarios with random VNR topologies is discussed next.

C. Iterative VNR subtraction

While the previously discussed scenario generator provides an easy way to generate random scenarios with perfect solutions, it is obviously limited with regard to the topology of the VNRs. One can argue that scenarios where all VNRs will always have the same topology as the SN are hardly representative. But do alternatives exist? Indeed, there are more elaborate possibilities for scenario generators that produce perfectly solvable scenarios, one of which is presented in this section. The discussed scenario generator produces perfectly solvable scenarios with VNRs of variable size. Here, the algorithm concept is described first, followed by an evaluation with common VNE algorithms. Finally, the properties of the generated VNRs are analyzed.

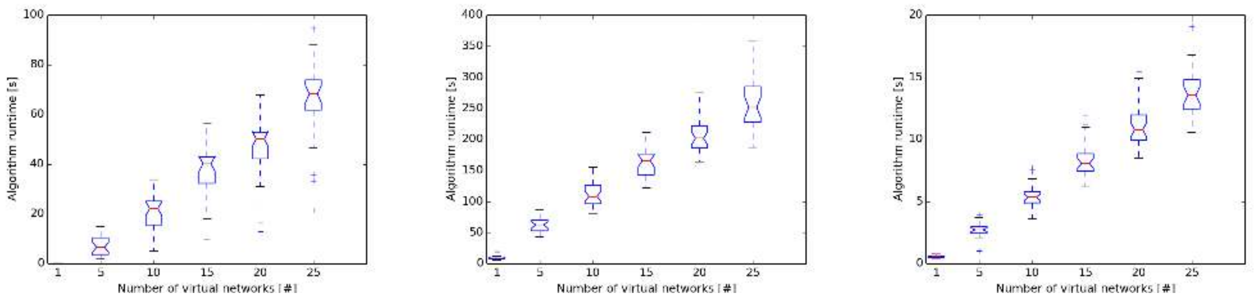
1) Description of the scenario generator: The generation of scenarios with random VNR topologies is somewhat more involved than simply replicating the SN topology. It is necessary to understand how the combination of node and link demands makes a seemingly straightforward problem more intricate. Therefore, after giving a short overview of the concept for the scenario generator, the problem of orphaned links is discussed, before details of the scenario generator along with a pseudocode implementation are presented.

a) Overall concept: The scenario generator tries to generate a set of VNRs from a given SN by repeatedly extracting a random subgraph from the SN. All extracted subgraphs are converted to VNRs. This is repeated until the SN is "empty". Care is taken, that the generated VNRs are actually connected networks. An example is shown in step 1 of Fig. 12: A VNR is extracted randomly from the SN. The resulting remainder becomes the SN of step 2.

Regarding nodes, **random problem generation can be performed similar to bin packing problems, setting virtual node demands to a random fraction of the respective substrate node (cf. [50] for a bin packing example).** However, the inclusion of links complicates the generation somewhat. In particular, a scenario generator has to take care to exhaust link resources in line with node resources.

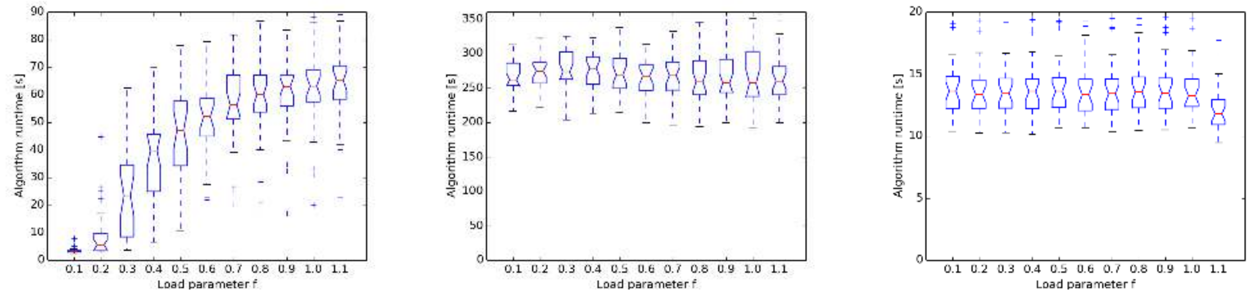
b) Problem of orphaned links and its solution: In terms of nodes, the scenario generation process **appears simple, at first**: For each new VNR one can select a random subset of connected, non-empty nodes and subtract a random fraction of the remaining node resources on each of them. If demands and resources are integral, this process is guaranteed to terminate at some point, occupying all node resources of the SN.

Links, however, pose a problem for this approach. If all resources of a node are consumed, but one of its adjacent



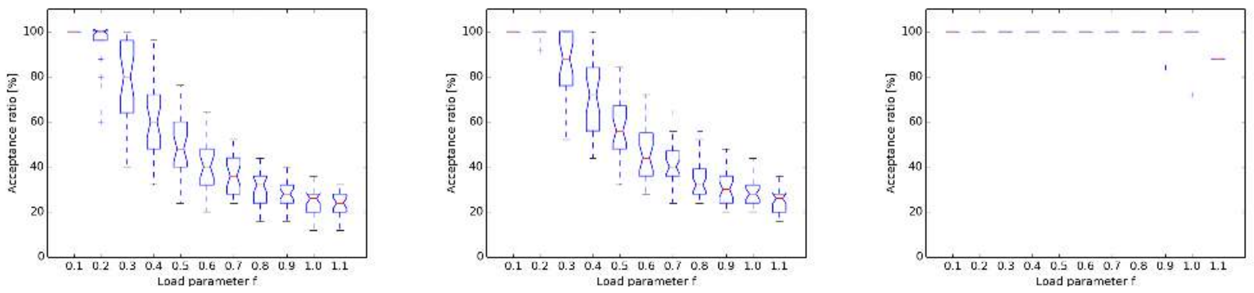
(a) vnmFlib (b) D-ViNE (c) RW-MM-SP

Fig. 9. Runtime dependency on number of VNRs m (f = 1.0)



(a) vnmFlib (b) D-ViNE (c) RW-MM-SP

Fig. 10. Runtime dependency on load parameter f (m = 25)



(a) vnmFlib (b) D-ViNE (c) RW-MM-SP

Fig. 11. Acceptance ratio dependency on load parameter f (m = 25)

links has resources left, these resources **cannot** be consumed without creating a multi-hop link. In general, this can lead to scenarios, where the a priori solution is not optimal, anymore.

Such a situation **occurs** when the example in Fig. 12 is continued. Three VNRs are generated iteratively for this scenario, until the SN has no more resources left. In the first step, one of the substrate nodes is already completely occupied. However, **some incident links still** have resources left. These links cause in the second and third step one hidden hop per generated VNR (depicted with a dot in the figure). While the scenario is valid and the VNRs can be embedded in the SN, there is actually a more optimal solution that leaves some of the SN resources unused. This is shown in Fig. 13. A tighter packing of the VNRs during embedding **causes** underutilization of the two links at the left side of the SN.

In order to solve this problem, the scenario generator has to make sure that, if a node becomes completely saturated, all its incident links will be saturated, as well. One strategy to achieve this goal is to set the demand for a virtual link in dependency of the demand of its incident virtual nodes. First, all nodes adjacent to a saturated node have to become part of the current VNR if they have resources left. For all substrate links which are incident to nodes that host virtual nodes in the VNR, a virtual link is created. Then, for every virtual link the percentage by which the two incident virtual nodes saturate their respective substrate node is computed. This gives two values: r_{source} and r_{dest}, for source node and destination node, respectively. The remaining resources on the substrate link are then saturated by the higher one of those two values $max(r_{\text{source}}, r_{\text{dest}})$. This ensures that no orphaned

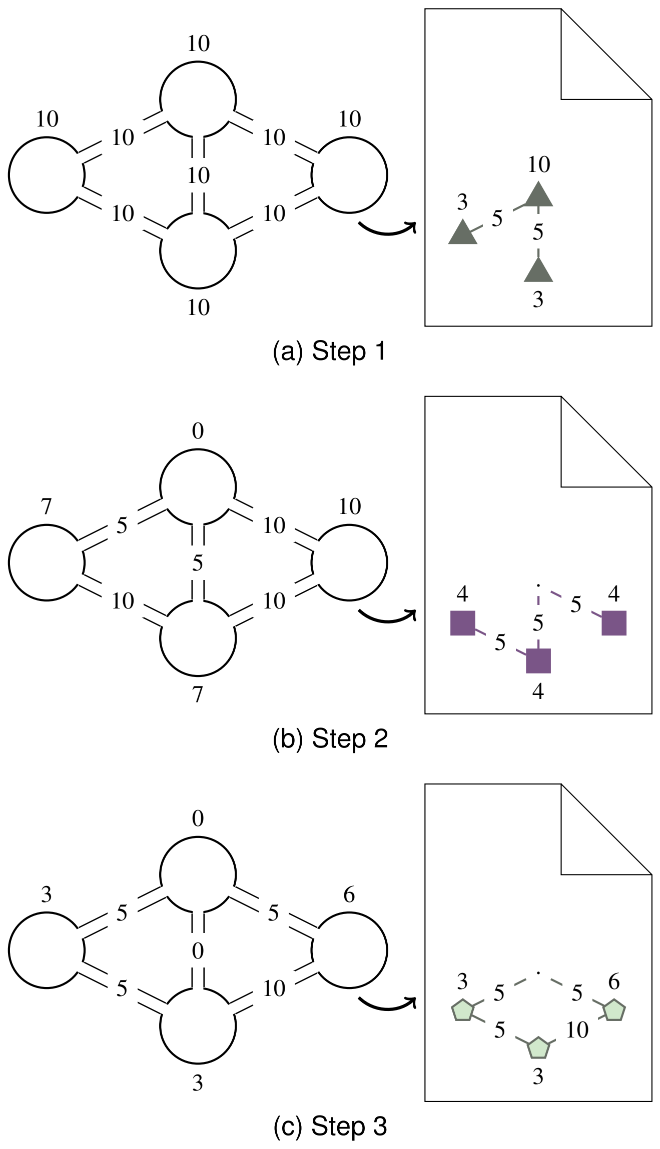


Fig. 12. An example of random VNR generation with a suboptimal solution

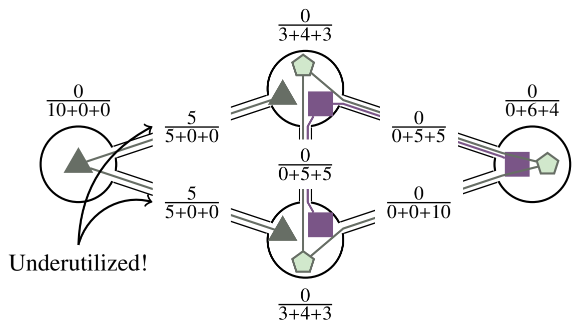


Fig. 13. An optimal solution for the previous example. Resource occupation for nodes and links is represented as: $\frac{\text{Remaining}}{\text{VNR1+VNR2+VNR3}}$

links remain, as any substrate link incident to a completely saturated substrate node will in turn be saturated, as well.

c) Scenario generation details: The scenario generator discussed here applies this concept to avoid orphaned links. It maintains three main data structures during the creation of each VNR. First, a candidate list, C, that contains all substrate nodes that are adjacent to the currently created partial VNR. Second, an urgent list, $\mathcal{U}$, that contains substrate nodes that are adjacent to a substrate node that has already been saturated by the currently created partial VNR. Finally, the scenario generator keeps track of previously assigned nodes by maintaining a mapping list, $\mathcal{M}$, which contains tuples of already assigned substrate nodes and virtual nodes.

```
Data: A pre-generated SN
Result: A set of n VNRs of varying size, such that the
        resulting scenario will be perfectly solvable
while $SN \neq \emptyset$ do
    Initialize new (empty) VNR;
    $\mathcal{M} = \emptyset, \mathcal{U} = \emptyset$;
    Initialize $C$ with random node from SN;
    while $\mathcal{U} \neq \emptyset$ or $C \neq \emptyset$ do
        if $\mathcal{U} \neq \emptyset$ then
            Pick SNode from $\mathcal{U}$ with resource $res$;
        else
            Pick random SNode from $C$ with resource
            $res$;
        end
        Create new VNode with random demand
        $dem \leq res$;
        Calculate SNode ratio $r_{SNode} = \frac{dem}{res}$;
        $\mathcal{M} \leftarrow \mathcal{M} \cup \{(SNode, VNode)\}$;
        Occupy dem on SNode;
        Add VNode to VNR;
        Initialize $\mathcal{N}$: Set of all nodes adjacent to SNode;
        while $\mathcal{N} \neq \emptyset$ do
            Pick neighbor SNeigh from $\mathcal{N}$;
            if $SNeigh \in \mathcal{M}$ then
                Set $r_{max} = max(r_{SNode}, r_{SNeigh})$;
                Create new VLink with bandwidth
                $bw_{dem} = bw_{res} \cdot r_{max}$;
                Add VLink to VNR;
                Occupy $bw_{dem}$ on respective SLink;
            else
                if $r_{SNode} = 1$ then
                    $\mathcal{U} \leftarrow \mathcal{U} \cup \{SNeigh\}$;
                else
                    $C \leftarrow C \cup \{SNeigh\}$;
                end
            end
        end
    end
end
Reset SN;
```

Algorithm 2: A scenario generator producing random VNRs

The scenario generator is sketched out as Algorithm 2. It starts by creating a new VNR. This VNR is initialized by

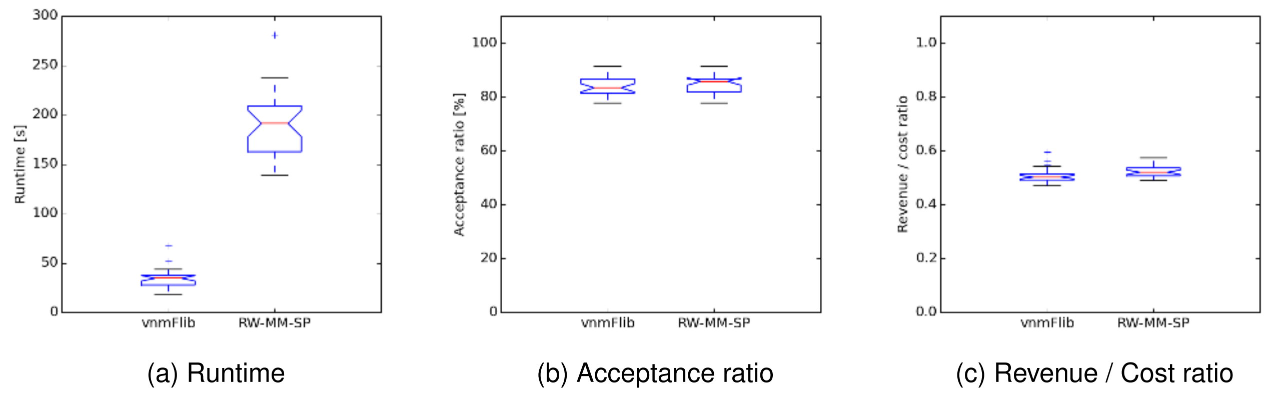


Fig. 14. Evaluation results for vnmFlib and RW-MM-SP

randomly selecting any non-empty node in the SN, creating a corresponding virtual node that consumes a random fraction of the remaining resources of the substrate node, and adding that node to the VNR. The demand of the generated virtual node is marked as occupied on the substrate node.

The scenario generator then adds nodes that are adjacent to the selected substrate node to a node candidate list. In general, the candidate list consists of all nodes that can still be added as virtual nodes to the VNR—i. e., all non-empty nodes that have not yet been used for the current VNR but are directly connected to nodes that have already been used.

Now, assuming the urgent list is empty and the candidate list is not, the scenario generator will randomly select a node from the candidate list and generate a corresponding virtual node in the same way as discussed above. For all substrate links that connect the current substrate node to previously used substrate nodes, a corresponding virtual link is generated. The demand for the link is set to the maximum of the demand ratio of the source node and the demand ratio of the destination node.This ensures that if a substrate node is occupied by 100%, adjacent substrate links will also be fully occupied.

If the substrate node is fully occupied after the virtual node was generated, all adjacent nodes that **have no corresponding virtual nodes in the VNR, yet,** are added to the urgent list, instead of the candidate list. These nodes are processed with higher priority, as they now *have* to be part of the currently generated VNR. Otherwise, those nodes are added to the candidate list (provided they are not already part of it).

The scenario generator continues with the next node. If the urgent list is empty, the next node is taken at random from the candidate list, and the same procedure ensues. Once the candidate list is empty, the VNR is added to the scenario, and generation continues with the next VNR.

This process is repeated until all nodes and links in the SN are fully occupied. At this point, the generated VNRs are guaranteed to saturate the SN completely, and a solution is guaranteed to exist, just by reversing the steps taken. To complete the scenario, the marked occupations are removed from the SN, and SN and the generated VNRs are combined to form the new scenario. As a chain element, this generator will be denoted as: $[ItSNSub]$

The VNRs generated by this approach are random subgraphs of the provided SN. With each iteration, the generated VNRs becomes smaller—while the first generated VNR covers the whole SN topology (provided the SN is fully connected), the last generated VNR only covers the remainder of the previous operations, likely only containing a single node. In other words: contrary to both the conventional approach and the previous approach, this scenario generator generates random networks of highly varying size. It is interesting to see, on the one hand, how well VNE algorithms cope with this kind of scenario and, on the other hand, to investigate and characterize the types of networks generated by this random process. These issues are discussed next.

2) Evaluation: Again, for demonstration of the new scenario generation element, it was implemented in ALEVIN and used for a series of experiments. The same environment as in section V-B2 was used. The scenario generation chain was set up again such that a conventional Waxman network was generated as SN. From this network, the VNRs were then formed with the new element:

$[W_S]$—$[CPU_S]$—$[BW_S]$—$[ItSNSub]$

For this series of experiments, the D-ViNE algorithm was excluded, as it suffered from excessive runtime (runs were stopped after taking more than 30 minutes per scenario).

Fig. 14 shows the results of these experiments. It can be seen that, while vnmFlib and RW-MM-SP achieve comparable values for the acceptance ratio, vnmFlib is significantly faster in this case. It should be noted that the generation element produces perfectly solvable scenarios—therefore, unlike with scenarios generated by a conventional scenario generator, it is known that an acceptance ratio of 100% is achievable.

As an additional metric, the commonly used “Revenue / Cost” ratio (i. e., the ratio between the sum of all realized demands and the sum of all consumed resources) was used to provide further insight. Again, both algorithms performed similar, achieving results in the vicinity of 0.5. This indicates that, on average, twice as many resources were consumed than demands realized. This hints at significant optimization potential, since similar to the acceptance ratio the optimal value for this metric is known to be 1.0 due to the scenario being perfectly solvable.

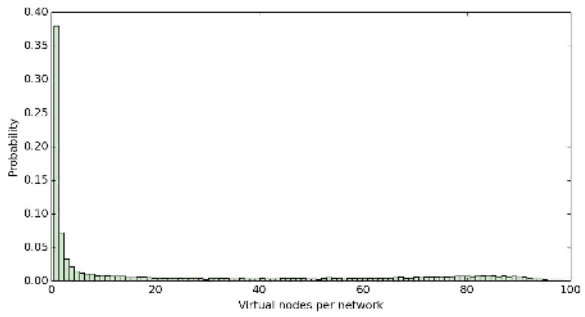


Fig. 15. Virtual nodes per VNR as generated by iterative VNR subtraction

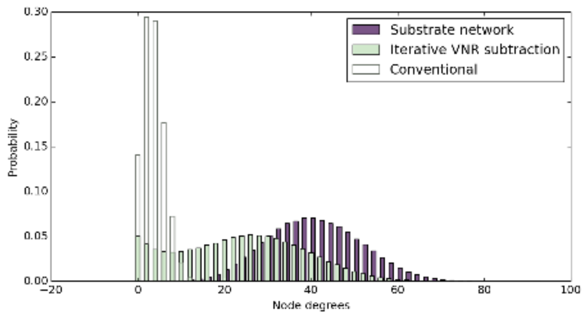


Fig. 16. Comparison of node degrees in the generated VNRs for iterative VNR subtraction and a conventional scenario generator

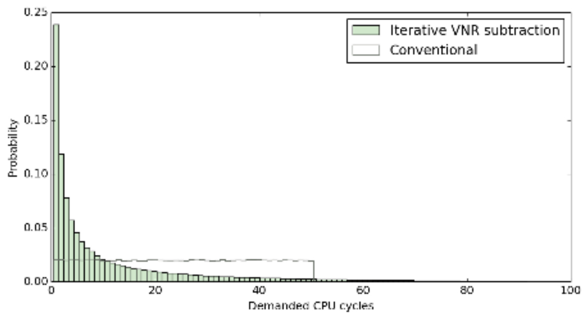


Fig. 17. Comparison of demanded CPU cycles (node demands) for iterative VNR subtraction and a conventional scenario generator

3) Algorithm properties: Obviously, the scenario generator described here produces scenarios that are very different to **scenarios described in VNE literature**. In particular, the generated VNRs now depend on the SN, whereas in conventional approaches the VNRs were generated independently of the SN. To show the differences, here two scenario generation chains will be compared side by side. The scenario generation chains used here are on the one hand **one that generates scenarios similar to those found in current VNE literature:**

$[W_S]$—$[CPU_S]$—$[BW_S]$—$[W_V]$—$[CPU_V]$—$[BW_V]$

This chain is compared with the same chain used in the evaluation above:

$[W_S]$—$[CPU_S]$—$[BW_S]$—$[ItSNSub]$

With each chain, 1000 scenarios were generated. For each scenario **an** SN with 100 nodes was generated with a Waxman topology generator. **For the VNRs, the conventional scenario generator used Waxman topologies, generating 20 VNRs with 10 nodes each.** These values are similar or equal to those found in common VNE literature. For the iterative VNR subtraction approach, no further parameters are needed.

A first, obvious difference lies in the **generated number of virtual nodes per VNR.** In the conventional approach, the number is a parameter to be set by the user. Here, this number can vary wildly. In particular, due to the problem of orphaned links, it is not easily possible to have the scenario generator generate VNRs with a fixed number of virtual nodes. **Fig. 15 shows the distribution of nodes per VNR**[5]. What is striking is the large number of small to very small networks—i.e., networks with only one or two nodes. Almost 40% of all networks consist of isolated nodes that form a network of their own. This is an effect that is caused at the end of the generation process, when there are only few resources left in the SN and the scenario generator fails to find potential neighbors for newly generated virtual nodes.

One can argue that the large number of trivial networks is a drawback for the scenario generator. However, it is informative to have a look at the distribution of node degrees and to compare them to the results of a conventional scenario generator and to the initial distribution of node degrees in the SN. This comparison is depicted in Fig. 16. One can identify the inclination of the iterative VNR subtraction process to generate small networks by the higher probability to generate nodes of degree zero. What is surprising, though, is the fact that the **Waxman-based approach produces** more nodes that are also unconnected. Almost 15% of all nodes (n.b. "nodes", not "networks" as before) have degree zero. In other words: A significant number of VNRs generated by a conventional scenario generator will be unconnected, **unless the topology is fixed explicitly. In contrast, the novel scenario generator proposed here produces scenarios that are guaranteed to be connected**[6]. Comparing to the SN, it is clear that the node degree distribution of the iterative VNR subtraction process matches the original SN characteristics better than the conventional approach.

Fig. 17 shows another interesting difference between a conventional scenario generator and the iterative VNR subtraction process presented here. Whereas in the conventional approach node demands are distributed equally between a specified minimum and maximum, the scenario generator discussed here shows the characteristics of an exponential falloff with a large number of very small demands (almost one quarter of all demands have a value of 1) and a long tail of large demands[7].

Finally, an interesting aspect is the number of VNRs that is generated by the scenario generator. Like with the number of virtual nodes per VNR, the total number of VNRs generated will vary for different scenarios. In contrast, the conventional

[5]The distribution of links per VNR, though not shown here, has a similar characteristic

[6]**It should be noted that the scenario generation chain approach makes it easy to define chain elements that solve either of the two problems by eliminating unconnected networks or removing isolated nodes.**

[7]Again, link resources, though not shown here, have similar characteristics

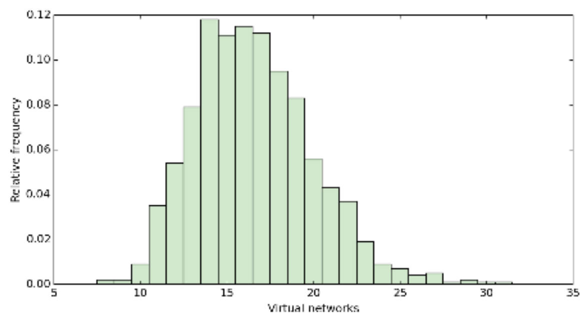


Fig. 18. Number of VNRs generated

approach lets the user decide how many VNRs to generate. For the investigated parameter set, the distribution of the number of VNRs is shown in Fig. 18. The distribution approximately reflects a normal distribution with an arithmetic mean of about 16.5 VNRs per scenario. This is in comparison to 20 VNRs set for the conventional approach. The iterative VNR subtraction process generates networks with a significantly higher number of nodes, and therefore will generate less networks overall, even though isolated nodes are considered to be a VNR of their own for this scenario generator.

Overall, one can conclude that, while the scenario generator discussed here certainly produces scenarios that are quite different to those generated by a conventional approach, they provide quite interesting additional insights and may be used to exhibit particular properties of VNE algorithms. This can be used either to perform more detailed evaluations or to fine-tune the behavior of a particular VNE algorithm. The generation of perfectly solvable scenarios thus does not negate the need for conventional scenarios, but rather complements the evaluation with more focused experiments.

VI. Conclusion and Future Work

Virtual Network Embedding is an important problem for **network virtualization.** The evaluation of algorithms attempting to provide optimized resource allocations for virtual networks is, therefore, of high interest. This paper introduced concepts to better control randomness in evaluation scenarios, producing scenarios with a solution that is known a priori to the experimenter. This enables experimenters for the first time to precisely measure the quality of an embedding produced by a VNE algorithm. To facilitate this, a new **architectural pattern for** scenario generation was presented and implemented as open source in the publicly available ALEVIN VNE simulation environment. Moreover, two generation elements producing perfectly solvable scenarios were discussed, implemented, and analyzed. The type of generated scenarios can complement a more conventional generation approach, allowing for more diverse evaluation of VNE algorithms.

The proposed scenario generation approach focuses on a conceptually simple VNE model. Extending this model and the generators such that a more varied set of VNE problems can be generated represents important future work. Also, the extension to online evaluation should provide interesting additional insight.

Acknowledgment

The research leading to these results was supported by the "Bavarian State Ministry of Education, Science and the Arts" as part of the FORSEC research association and by the EU project DC4Cities (FP7 STREP, grant no. 609304).

References

[1] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862–876, 2010.
[2] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys and Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.
[3] C. McGeoch, "Analyzing algorithms by simulation: Variance reduction techniques and simulation speedups," *ACM Comput. Surv.*, vol. 24, no. 2, pp. 195–212, Jun. 1992.
[4] C. C. McGeoch, "Feature article—toward an experimental method for algorithm simulation," *INFORMS Journal on Computing*, vol. 8, no. 1, p. 1–15, Feb 1996.
[5] ——, "Experimental analysis of algorithms," *Notices of the AMS*, vol. 48, no. 3, pp. 304–311, 2001.
[6] B. M. Moret, "Towards a discipline of experimental algorithmics," in *Data structures, near neighbor searches, and methodology*, ser. DIMACS Monographs, M. H. Goldwasser, D. S. Johnson, and C. C. McGeoch, Eds. AMS Press, 2002, no. 59, pp. 197–213.
[7] D. S. Johnson, "A theoretician's guide to the experimental analysis of algorithms," in *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, M. H. Goldwasser, D. S. Johnson, and C. C. McGeoch, Eds. AMS and DIMACS, 2002, vol. 59, pp. 215–250.
[8] R. L. Rardin and R. Uzsoy, "Experimental evaluation of heuristic optimization algorithms: A tutorial," *Journal of Heuristics*, vol. 7, no. 3, pp. 261–304, 2001.
[9] E. Berberich, M. Hagen, B. Hiller, and H. Moser, "Chapter 8. experiments," in *Algorithm Engineering*, ser. Lecture Notes in Computer Science, M. Müller-Hannemann and S. Schirra, Eds. Springer Berlin Heidelberg, 2010, vol. 5971, pp. 325–388.
[10] C. C. McGeoch, *A Guide to Experimental Algorithmics*. Cambridge University Press, Jan. 2012.
[11] P. Schwerin and G. Wäscher, "The bin-packing problem: A problem generator and some numerical experiments with ffd packing and mtp," *International Transactions in Operational Research*, vol. 4, no. 5-6, pp. 377–389, 1997.
[12] M. G. Pilcher and R. L. Rardin, "A random cut generator for symmetric traveling salesman problems with known optimal solutions," Institute for Interdisciplinary Engineering Studies, Purdue University Lafayette, IN, Tech. Rep. CC-87-4, 1987.
[13] B. Krishnamurthy, "Constructing test cases for partitioning heuristics," *Computers, IEEE Transactions on*, vol. C-36, no. 9, pp. 1112–1114, Sept 1987.
[14] L. A. Sanchis, "Generating hard and diverse test sets for np-hard graph problems," *Discrete Applied Mathematics*, vol. 58, no. 1, p. 35–66, Mar 1995.
[15] D. G. Andersen, "Theoretical approaches to node assignment," Dec. 2002, unpublished Manuscript.
[16] J. Fan and M. Ammar, "Dynamic topology configuration in service overlay networks: A study of reconfiguration policies," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, April 2006, pp. 1–12.
[17] Y. Zhu and M. Ammar, "Algorithms for assigning substrate network resources to virtual network components," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, April 2006, pp. 1–12.
[18] J. Lu and J. Turner, "Efficient mapping of virtual networks onto a shared substrate," Washington University in St. Louis, Tech. Rep. WUCSE-2006-35, 2006.
[19] A. Belbekkouche, M. Hasan, and A. Karmouch, "Resource discovery and allocation in network virtualization," *Communications Surveys Tutorials, IEEE*, vol. PP, no. 99, pp. 1–15, 2012.
[20] J. F. Botero, X. Hesselbach, M. Duelli, D. Schlosser, A. Fischer, and H. De Meer, "Flexible VNE algorithms analysis using ALEVIN," in *Proc. of the 11th Würzburg Workshop on IP: Joint ITG, ITC, and Euro-NF Workshop "Visions of Future Generation Networks" (EuroView 2011)*. ACM, 2011.

[21] J. Zhu and T. Wolf, "Vnmbench: A benchmark for virtual network mapping algorithms," in *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*, July 2012, pp. 1–8.
[22] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: Substrate support for path splitting and migration," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 17–29, Mar. 2008.
[23] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an internetwork," *Proceedings of IEEE INFOCOM '96. Conference on Computer Communications*, vol. 2, pp. 594–602, 1996.
[24] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang, "Virtual network embedding through topology-aware node ranking," *SIGCOMM Comput. Commun. Rev.*, vol. 41, pp. 38–47, April 2011.
[25] B. Lü, T. Huang, Z.-k. Wang, J.-y. Chen, Y.-j. Liu, and J. Liu, "Adaptive scheme based on status feedback for virtual network mapping," *The Journal of China Universities of Posts and Telecommunications*, vol. 18, no. 5, pp. 87 – 94, 2011.
[26] X. Chen, Y. Luo, and J. Wang, "Virtual network embedding with border matching," in *Communication Systems and Networks (COMSNETS), 2012 Fourth International Conference on*, Jan 2012, pp. 1–8.
[27] N. Chowdhury, M. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *INFOCOM 2009, IEEE*, April 2009, pp. 783–791.
[28] M. Chowdhury, M. Rahman, and R. Boutaba, "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping," *Networking, IEEE/ACM Transactions on*, vol. 20, no. 1, pp. 206–219, Feb 2012.
[29] M. R. Rahman, I. Aib, and R. Boutaba, "Survivable virtual network embedding," in *NETWORKING 2010*, ser. Lecture Notes in Computer Science, M. Crovella, L. Feeney, D. Rubenstein, and S. Raghavan, Eds., vol. 6091. Springer Berlin Heidelberg, 2010, pp. 40–52.
[30] N. F. Butt, N. M. Chowdhury, and R. Boutaba, "Topology-awareness and reoptimization mechanism for virtual network embedding," in *Networking 2010*, vol. 6091. Springer Berlin Heidelberg, 2010, pp. 27–39.
[31] S. Masti and S. Raghavan, "Vna: An enhanced algorithm for virtual network embedding," in *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*, July 2012, pp. 1–9.
[32] C. Papagianni, A. Leivadeas, S. Papavassiliou, V. Maglaris, C. Cervello-Pastor, and A. Monje, "On the optimal allocation of virtual resources in cloud computing networks," *Computers, IEEE Transactions on*, vol. 62, no. 6, pp. 1060–1071, June 2013.
[33] A. Leivadeas, C. Papagianni, and S. Papavassiliou, "Socio-aware virtual network embedding," *Network, IEEE*, vol. 26, no. 5, pp. 35–43, September 2012.
[34] ——, "Efficient resource mapping framework over networked clouds via iterated local search-based request partitioning," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 24, no. 6, pp. 1077–1086, June 2013.
[35] C. Pittaras, C. Papagianni, A. Leivadeas, P. Grosso, J. van der Ham, and S. Papavassiliou, "Resource discovery and allocation for federated virtualized infrastructures," *Future Generation Computer Systems*, vol. 42, pp. 55 – 63, 2015.
[36] A. Fischer, J. F. Botero, M. Duelli, D. Schlosser, X. Hesselbach, and H. De Meer, "ALEVIN - a framework to develop, compare, and analyze virtual network embedding algorithms," *Electronic Communications of the EASST*, vol. 37, pp. 1–12, 2011.
[37] M. T. Beck, A. Fischer, F. Kokot, C. Linnhoff-Popien, and H. De Meer, "A simulation framework for virtual network embedding algorithms," in *6th International Telecommunications Network Strategy and Planning Symposium (Networks 2014)*. IEEE, Sep. 2014, pp. 1–6.
[38] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0–Survivable Network Design Library," in *Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium*, April 2007, http://sndlib.zib.de, extended version accepted in Networks, 2009.
[39] J. F. Botero, X. Hesselbach, M. Duelli, D. Schlosser, A. Fischer, and H. De Meer, "Energy efficient virtual network embedding," *IEEE Communications Letters*, vol. 16, no. 5, pp. 756–759, May 2012.
[40] A. Fischer, M. T. Beck, and H. De Meer, "An approach to energy-efficient virtual network embeddings," in *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. IEEE, 2013, pp. 1142 – 1147.
[41] M. T. Beck, A. Fischer, H. de Meer, J. F. Botero, and X. Hesselbach, "A distributed, parallel, and generic virtual network embedding framework," in *IEEE International Conf. on Communications (ICC 2013)*, 2013, pp. 3471–3475.
[42] M. T. Beck, A. Fischer, J. F. Botero, C. Linnhoff-Popien, and H. De Meer, "Distributed and scalable embedding of virtual networks," *Journal of Network and Computer Applications*, vol. 56, pp. 124–136, Oct. 2015.
[43] B. M. Waxman, "Routing of multipoint connections," *IEEE Journal on selected areas in communications*, vol. 6, no. 9, pp. 1617–1622, 1988.
[44] P. Erdös and A. Rényi, "On random graphs," *Publicationes Mathematicae*, vol. 6, pp. 290—-297, 1959.
[45] E. N. Gilbert, "Random graphs," *The Annals of Mathematical Statistics*, vol. 30, no. 4, p. 1141–1144, Dec 1959.
[46] A. Medina, A. Lakhina, I. Matta, and J. Byers, "Brite: an approach to universal topology generation," in *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001. Proceedings. Ninth International Symposium on*, 2001, pp. 346–353.
[47] G. Hohpe and B. Woolf, *Enterprise Integration Patterns*. Addison-Wesley, 2003.
[48] L. A. Sanchis, "On the complexity of test case generation for np-hard problems," *Information Processing Letters*, vol. 36, no. 3, pp. 135–140, 1990.
[49] J. Lischka and H. Karl, "A virtual network mapping algorithm based on subgraph isomorphism detection," in *VISA '09: Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*. New York, NY, USA: ACM, 2009, pp. 81–88.
[50] E. Falkenauer and A. Delchambre, "A genetic algorithm for bin packing and line balancing," in *IEEE International Conference on Robotics and Automation*, vol. 2, May 1992, pp. 1186–1192.

Andreas Fischer (http://www.net.fim.uni-passau.de/fischer) received his Diploma in Computer Science from University of Passau in 2008. Since then he has been a scientific assistant at the Computer Networks and Computer Communications group of Prof. De Meer. His research interests are network virtualization and network resilience in virtualized environments.

Hermann de Meer (http://www.net.fim.uni-passau.de/demeer) has been appointed as Full Professor of Computer Science at the University of Passau, Germany, since 2003. He is heading the Computer Networking Lab and co-heading the Institute of IT-Security and Security Law (ISL). His interests of research include Network Virtualization, Digitization of Energy Systems, IT-Security of Critical Infrastructures, and Distributed Control and Optimization. In addition to numerous publications in Journals and at Conferences, H. de Meer is also co-authoring a textbook on Queuing Networks and Markov Chains, published by John Wiley. Before joining University of Passau, he was affiliated with University College London (UCL), UK, and with Columbia University, New York City, USA. He had received his PhD from University Erlangen-Nuremberg, Germany, in 1992, and his Habilitation from Hamburg University, Germany. He is member of the ACM and of the GI (Gesellschaft fuer Informatik) and fellow of the DFG (Deutsche Forschungsgemeinschaft). In addition to hosting and chairing editions of IWQoS and IWSOS, he was host and initiator of 1st e-Energy Conference, later to be sponsored by ACM SIGCOMM, at the University of Passau in 2010.