JOURNAL OF OBJECT TECHNOLOGY
Published by AITO — Association Internationale pour les Technologies Objets
http://www.jot.fm/

Dynamic Virtual Network Embedding: Using Incremental Model Transformation and Integer Linear Programming Techniques

Stefan Tomaszek[a] Lars Fritsche[a] Andy Schürr[a]

a. Real-Time Systems Lab, Technical University of Darmstadt, Germany
{stefan.tomaszek,lars.fritsche,andy.schuerr}@es.tu-darmstadt.de

Abstract

Network virtualization enables flexible placement, migration, and execution of virtual networks and machines on physical hardware. This results in an NP-hard optimization problem called virtual network embedding (VNE). Ensuring hardware and (non-)functional constraints while finding an optimal solution for a wide range of scenarios is a challenging task in these highly dynamic environments.

To develop and evaluate algorithms tailored for various environments, we present a model-driven approach to specify and solve dynamic VNE problems by using a high-level specification to declaratively specify the search space and constraints, incremental model transformation to prune the search space, and low-level ILP techniques to find an optimal solution in the pruned search space. This high-level specification is used to generate an executable program for solving dynamic VNE problems. Furthermore, we show in an evaluation that this generated program can solve a typical dynamic VNE problem in less time than a hand-crafted ILP-based program.

Keywords virtual network embedding; incremental model transformation; data center; integer linear programming;

1 Introduction

Today, online services such as social networking, e-commerce, and online gaming are ubiquitous and place high demands on service providers in terms of availability, scalability, and flexibility. The enormous amount of data and computing capacity required are pushing traditional network technologies, topologies, and management techniques to their limits. Cloud computing is one of the leading technologies in this area to meet these high demands. Data centers provide the necessary resources for data storage and processing via virtualized environments. This hardware virtualization decouples the underlying physical hardware from the

Stefan Tomaszek, Lars Fritsche, Andy Schürr. *Dynamic Virtual Network Embedding: Using Incremental Model Transformation and Integer Linear Programming Techniques*. In Journal of Object Technology, vol. 19, no. 2, 2020, pages 9:1–21. doi:10.5381/jot.2020.19.2.a9

running applications, enabling service providers to operate their data centers hardware- and vendor-independently. The administration of these highly complex virtual environments can be standardized and often centrally managed, enabling new services to be delivered and migrated to other physical servers or virtual environments automatically.

However, the advantages of virtualizing data centers are accompanied with the complex task of embedding virtual networks in an optimal way. This NP-hard optimization problem [ACKT16] is called *virtual network embedding (VNE) problem*, whereby an optimal solution can only be calculated with a high effort, which is usually associated with a long runtime for solving the problem. The common scenario for a VNE problem is that of embedding several virtual networks in a substrate network (e.g., a data center), whereby structural as well as (non-)functional constraints must be ensured in compliance with the optimization goal [FBB+13]. Structural constraints refer to the resources of the physical servers, switches, or links (e.g., computing capacity of the servers and bandwidth of the links) and (non-)functional constraints to service level agreements, security policies, or hardware-specific functionalities (e.g., firewalls). Common optimization goals for the VNE problem are the minimization of communication costs, monetary costs, or used hardware resources [YYRC08]. The abundance of possible combinations of structural and (non-)functional constraints with different network topologies, application scenarios, and optimization goals makes the development, adaptation, simulation, and comparison of VNE algorithms challenging. In addition, these environments are highly dynamic, so that other actions must be considered beside the common case of embedding new virtual networks. This can be the modification of existing virtual resources, the deletion of virtual networks, or the modification and failure of physical hardware, which can lead to migration of existing embeddings.

To solve the above stated problem, the main challenges are the characterization of the relevant search space, the (guaranteed) compliance to all constraints, and finding optimal embeddings and migrations in the specified search space. Two established categories of approaches for solving these problems [ACKT16, FBB+13] are heuristics-based (e.g., [BCKR11, LS17, YCLL17]) and based on integer linear programming (ILP) (e.g., [ZGH+15, YG16]). Heuristics-based approaches are tailored to specific infrastructures and application scenarios without ensuring that all constraints are respected and the found solutions are optimal in the specified search space. In contrast, general purpose ILP-based approaches can be used for a wide range of applications in compliance with constraints and requirements while achieving optimal results to a specific optimization goal. Due to the long runtime to solve the VNE problem, ILP-based approaches are only applicable for small data centers [YG16]. In order for ILP-based approaches to support dynamic changes, which means that the migration of virtual machines must also be considered, hand-crafted programs are used to encode and update the ILP problem for a concrete scenario during runtime. Therefore, every change (e.g., modification of a substrate server) has to be integrated into the ILP problem, which also means that the constraints and the optimization goal have to be adapted. However, the development of such a program for *dynamic VNE problems supporting migrations* is even more complex and error-prone when done by hand than expressing the common scenario where already embedded networks do not change. In addition, the generation of an efficient ILP formulation requires a lot of experience and knowledge in dealing with ILP techniques and dynamic VNE problems.

A combination of search space pruning strategies and ILP-based technologies is provided by the model-driven virtual network embedding (MdVNE) approach [TLWS18b]. This approach provides a way to define a high-level specification for a static VNE scenario, which is then automatically converted into executable program code. It incorporates model transformation (MT) techniques as a search space pruning strategy to describe the search space declaratively. Based on this declarative description and an optimization goal, we can automatically formulate

the ILP problem finding an optimal solution within the specified search space respecting all constraints. MT enables us to prune the search space beforehand filtering the set of all possible mappings, while ILP solving finds a valid and optimal solution in the now pruned search space. However, to this point MdVNE does not support *dynamic VNE scenarios* including changes to already embedded virtual or substrate networks and altering former embeddings.

In this paper, we introduce a model-driven approach supporting dynamic VNE scenarios based on a high-level specification respecting all constraints and ensuring optimality w.r.t. the objective function in the specified search space. We use graph patterns to characterize the search space declaratively and a high-level specification to define the constraints and objective function. From this, an executable program is derived which integrates the required incremental MT technologies, generates an ILP formulation, and updates this ILP formulation according to the incremental model changes. More precisely, the paper contains the following contributions:

- A high-level problem specification for dynamic and online VNE scenarios.
- Presentation of a model-driven approach supporting dynamic VNE scenarios using incremental MT and ILP techniques.
- In an experimental evaluation of a common dynamic VNE scenario, we compare the performance of the model-driven approach with a hand-crafted program for creating and updating an ILP program.

To the best of our knowledge, this is the first work that uses a combination of *incremental MT and ILP techniques* and supports optimal solutions of the dynamic VNE problem in data centers.

The remaining paper is structured as follows. In Sec. 2, we describe the VNE problem, the dynamic properties, and the existing MdVNE approach. In Sec. 3, we introduce the new model-based approach and high-level specification for dynamic VNE scenarios using incremental MT technologies followed by an evaluation in Sec. 4. Sec. 5 then presents related work followed by a summary of the paper and future work in Sec. 6.

2 Background

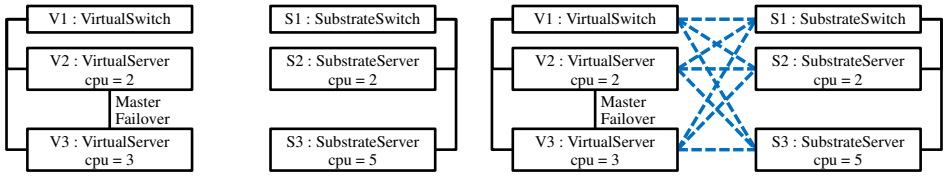
In this section, we define the VNE problem for data centers, explain the classification of dynamic VNE problems, and introduce the MdVNE approach from [TLWS18a].

2.1 Problem Description

The problem description based on [STR+15, ZGH+15, TLWS18a] can be divided into the network model, the mapping variables, the constraints, and the objective function.

2.1.1 Network Model

We model the *substrate network*, representing a data center network, and the *virtual networks* as undirected weighted graphs $G^S = (N^S, L^S)$ and $G^V = (N^V, L^V)$ with typed *nodes* N^S, N^V and *links* $l_{uv} \in L^S$, L^V from u to v. The superscript S and V refer to the substrate network and virtual network, respectively. In the substrate network, paths $p_{uv} \in P^S$ are defined additionally, which represent a sequence of acyclic connected links and nodes from the source node u to the target node v. The nodes in the networks represent either servers or switches, whereby in the substrate network each server can host virtual servers and switches. To model the resources provided by the substrate network or required by the virtual network, for every node u multiple



(a) Virtual and substrate networks (b) Networks with mapping candidates

Figure 1 – Running examples for the VNE problem

integer values exist to express computing capacities like CPU cores (C_u^S, C_u^V), memory (M_u^S, M_u^V), and storage (S_u^S, S_u^V). For links, only bandwidth resources $B_{l_{uv}}^S$, $B_{l_{uv}}^V$ between the nodes u and v are further considered.

> **Example: Virtual and substrate networks**
> Throughout the paper, we use the virtual and substrate networks from Fig. 1a as running example. For simplicity, the two networks (virtual and substrate) are presented as separate graphs with reduced resources and properties. On the left-hand side, the graph shows a virtual network with a central switch and two servers requiring 2 and 3 CPU cores, respectively. The virtual server *V3* is modeled as a failover server for *V2*. On the right-hand side, the substrate network also contains a central switch with two servers providing 2 and 5 CPU cores, respectively.

2.1.2 Mapping Variables

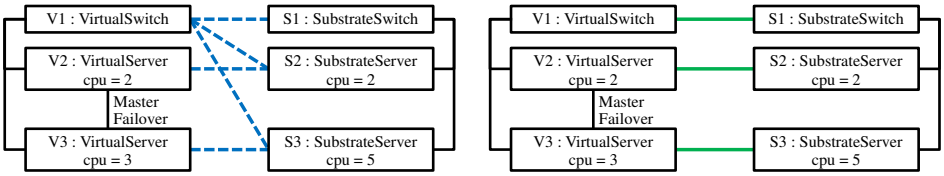
The *mapping variables* $\in \{0, 1\}$ define the embedding of a virtual element into a substrate element. Since every virtual node can be mapped to a substrate node and every virtual link to a substrate path, there exist mapping variables for each considered node and link mapping. The node-mapping variables x_u^i define whether the virtual node i is mapped to the substrate node u. Analogously, the link-mapping variables y_{uv}^{ij} indicate whether a virtual link l_{ij} is mapped to a substrate path p_{uv}.

> **Example: Mapping candidates**
> In Fig. 1b all node-mapping candidates, freely selectable ILP node-mapping variables, for the virtual elements (server, switches) are shown as blue dashed lines. These candidates are composed of the pairwise combination of all virtual and substrate elements, whereby no further constraints are taken into account yet.

2.1.3 Constraints

In the following, we define further *basic constraints* to ensure that (i) all virtual elements are embedded exactly once, (ii) every virtual server (switch) is mapped exactly to one substrate server (server or switch), and (iii) the resources of the substrate elements (computing capacity, memory, storage, bandwidth) are not overbooked.

In addition, we consider further *properties* for virtual servers or networks to demonstrate the different possibilities of constraints in this domain. A virtual *failover server* is a server that acts as a backup for a master server and takes over if the master server fails. The restriction here is that each master server may have a maximum of one failover server, whereby these two virtual servers must not be placed on the same substrate server. A virtual *high performance computing server* is characterized by a very fast clocked CPU, which is only available on a few



(a) Including the constraints (b) Valid embedding

Figure 2 – Running examples including the constraints and providing a valid embedding

substrate servers. This means that virtual high performance computing servers may only be operated on high performance computing substrate servers. The last constraint regards *low latency networks* that are virtual networks in which the connection between two servers may only have very low latencies. This property is enforced by allowing only a maximum of two hops between mapped substrate servers.

Example: Including the constraints
Fig. 2a shows the node-mapping candidates after incorporating the above constraints. This eliminates the candidates $V2 \rightarrow S1$ and $V3 \rightarrow S1$ as a virtual server cannot be embedded into a substrate switch. In addition, the candidate $V3 \rightarrow S2$ is rejected because the substrate server $S2$ cannot provide enough CPU cores for $V3$. Since $V3$ can only be mapped to $S3$ and the constraint that master and failover server must not run on the same substrate server, $V2$ must be mapped to $S2$. For the sake of clarity, additional dependencies between the candidates are not shown in the figures. This concerns, for example, the constraint that a virtual switch can only be mapped to one substrate element, whereby only one candidate is selected from the set of $V1 \rightarrow S1$, $V1 \rightarrow S2$, and $V1 \rightarrow S3$.

2.1.4 Objective Function

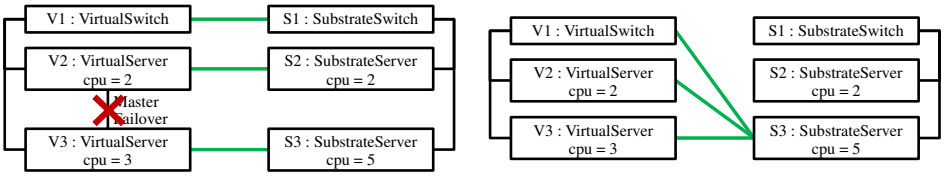
As a possible objective function, we minimize the sum of the aggregated communication costs for virtual servers in a substrate network [ZGH+15] ($cost^L$) and the costs for the migration of a virtual server to another substrate server ($cost^M$). To calculate the communication costs, we use the cost matrices for data center topologies from [MPZ10] and as migration costs the sum of all resources of a virtual server. This results in the following objective function:

$$\text{min:} \sum_{p_{uv} \in P^S} \sum_{l_{ij} \in L^V} y_{uv}^{ij} cost^L(p_{uv}, l_{ij}) + \sum_{i \in N^V} \sum_{u \in N^S} x_u^i cost^M(u, i) \tag{1}$$

Example: Valid embedding including the objective
Fig. 2b shows a possible valid embedding after solving the objective function. The embeddings are represented as green solid lines. Since the communication costs are equal for all mappings of the virtual switch and no migration costs are incurred, each candidate for embedding the virtual switch is potentially selectable.

2.2 Dynamic VNE

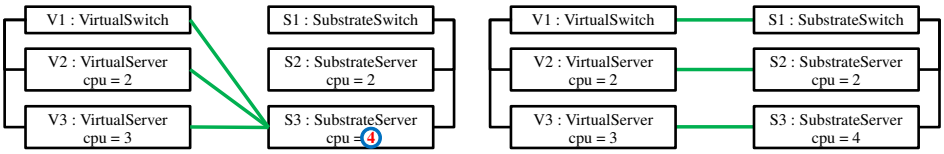
When classifying VNE (simulation) environments, a distinction can be made between (i) offline and online and (ii) static and dynamic scenarios [FBB+13]. In offline scenarios all virtual



(a) Removing a failover connection

(b) New embeddings after the removal

Figure 3 – Removing a failover connection after Fig. 2b



(a) Changing the CPU resource of $S2$

(b) New embeddings after the changes

Figure 4 – Changing the CPU resource of a substrate server after Fig. 3b

networks and the (dynamic) changes of the network elements are known in advance, whereas in online scenarios the virtual networks and dynamic changes must be handled as they arrive without further information about future virtual networks. Thus, it is possible to embed several or all virtual networks or changes simultaneously for an offline scenario. In addition, the scenario can be either static or dynamic. In a static scenario, only new virtual networks are embedded without changing already embedded elements. Hence, it is not possible to delete or change these networks or their servers and links. In an online scenario, however, changes to virtual and substrate networks or their elements are possible, whereby migration of existing embeddings must be integrated into the problem description. Realistic environments are typically characterized as online and dynamic scenarios.

Example: Dynamic changes

After the complete virtual network is embedded in the substrate network (see Fig. 2b), the master/failover connection between $V2$ and $V3$ is removed (see Fig. 3a). This could result in a lower objective value if the virtual switch $V1$ or the virtual servers $V2$ or $V3$ are migrated because the communication costs are larger than the migration costs. In this example, all virtual elements are moved to the substrate server $S3$ (see Fig. 3b). Afterwards, however, the CPU resource of the substrate server $S3$ is reduced to 4 cores (see Fig. 4a), whereby the entire virtual network can no longer be operated on this single substrate server, since the sum of the CPU resources of $V2$ and $V3$ requires 5 cores. In this case, at least one virtual server must be migrated to comply with the constraints. The new post-migration embeddings are shown in Fig. 4b.

2.3 Model-driven Virtual Network Embedding

A schematic view of the *model-driven virtual network embedding* (MdVNE) approach from [TLWS18b] can be found in Fig. 5. At runtime, the user creates one or more *virtual network requests* (VNRs) to be embedded into the substrate network (Fig. 5 step (A)). This can be abstracted as an *VNR event* that adds new elements to the model (represented as a green +-symbol). Then, in step (1), the MT tool creates a set of possible mapping candidates still containing invalid embeddings (step (B)). The blue ⧖-symbol on the event indicates that

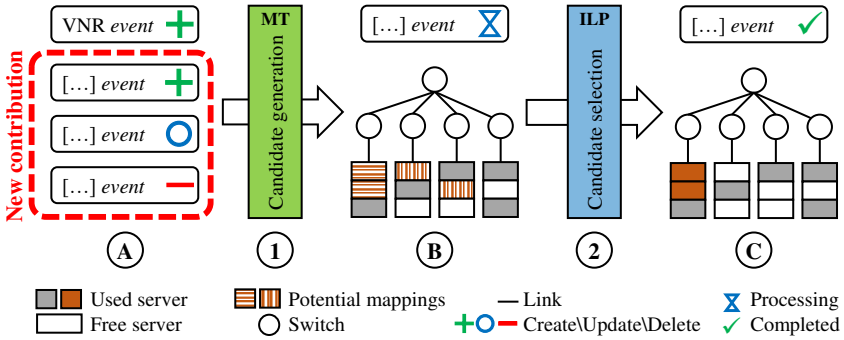


Figure 5 – Schematic view of the (i)MdVNE approach

solving the VNE problem for this event is currently in progress. Now, the mapping candidates together with additional ILP constraints, which cannot be ensured by MT, and the objective function are converted into an ILP problem and passed to the ILP solver (step ②). The optimal solution found by the ILP solver (if a solution exists at all) is then deployed (step Ⓒ) by embedding the virtual network and the VNR event is marked as processed (✓). Now, the next incoming VNR event (step Ⓐ) can be processed and embedded. Since these are online scenarios, it is not possible to predict the next VNR events with certainty.

In [TLWS18b], triple graph grammars (TGGs) [Sch94] are used as MT technique to create the mapping candidates (step ①). A TGG describes in a declarative and rule-based manner possible mappings between the substrate and the virtual network using a correspondence graph interlinking both. For every embedding of a new virtual network (VNR event) the following two steps are required: (i) the substrate network and (ii) the virtual network are created together with the possible mapping candidates. After performing these two steps, all possible mapping candidates are expressed as ILP mapping variables. While some constraints are implicitly integrated in the TGGs such as injective mappings between substrate and virtual servers, other constraints not ensured by the MT specification are added to the ILP problem, and the ILP objective function is created. After that, the embeddings are finalized and the generated intermediate data for the substrate network, the mapping candidates, and the complete ILP problem is deleted. Through this batch processing, the MdVNE approach supports static and online scenarios for the VNE problem.

3 Incremental MdVNE

In this section, we present the novel incremental model-driven virtual network embedding (iMdVNE) approach, based on [TLWS18b], to support and simulate dynamic online VNE scenarios while respecting all constraints and ensuring optimality w.r.t. the objective function in the specified search space. In order to obtain a highly customizable and extensible approach for VNE scenarios and environments, we divide the inputs for this algorithm into create, update, and delete events, based on the basic functionalities for a persistent memory (CRUD) [Ste09]. Since these model changes refer to a (usually) very small subset of network elements, only the incremental parts and their direct effects have to be considered. These incremental changes are called *deltas*.

Fig. 5 schematically shows iMdVNE, which consists of the two steps ① (candidate generation) and ② (candidate selection) and, therefore, three system states following the MdVNE approach. State Ⓐ shows the inputs of iMdVNE, which are abstracted as create (+), update

Model modification (delta)	New embed.	Del. embed.	Req. mig.	Opt. mig.
+ Create vir. element	X	-	X	X
+ Create sub. element	-	-	-	X
O Decrease vir. resource *C/M/S*	-	-	-	X
O Increase vir. resource *C/M/S*	-	-	X	X
O Decrease sub. resource *C/M/S*	-	-	X	X
O Increase sub. resource *C/M/S*	-	-	-	X
O Change property	-	-	X	X
– Delete vir. element	-	X	-	X
– Delete sub. element	-	X	X	-

Table 1 – Migration cases for the input events.

(O), and delete (–) events. These changes refer to deltas in the networks. All mapping candidates are then generated by the model transformations (1), whereby only the candidates that depend on the incremental changes have to be updated. After that, the intermediate state (B) consists of a set of possible mappings, which are then converted into an ILP problem. When creating the ILP problem (step (2)) only the deltas of the mapping variables and the constraints must be updated in the ILP problem formulation. Therefore, the ILP solver can reuse previously calculated solutions. In the end, in state (C), a solution is obtained that respects all constraints and is optimal with regard to the set of candidates generated.

In the transition from a static to a dynamic online VNE problem, the migration of currently existing embeddings, the re-embedding of previously rejected virtual networks and the rejection of a currently embedded virtual network must be considered. Therefore, three cases can be distinguished: (i) *new embeddings*, (ii) *deletion of embeddings*, and (iii) *migration of embeddings*. The first two cases (i) and (ii) are only relevant for the creation and deletion events and (can) trigger further migration steps to ensure optimal and valid solutions. The migration of existing embeddings (iii) can be triggered by the creation, update (e.g., de-/increase of resources), and deletion event. However, two cases (strategies) must be distinguished here: (a) *required migration* and (b) *optimizing migration strategy*. The required migration strategy (a) must always be performed if constraints are violated by the delta. This migration ensures that the embeddings respect all constraints. The optimizing migration strategy (b) includes the required migration strategy to ensure that all constraints are respected but additionally guarantees that all embeddings are optimal w.r.t. to the objective function. Table 1 summarizes the events for a adding (*new embed.*) or deleting an embedding (*del. embed.*) and their possible cases required migration strategy (*req. mig.*) or optimizing migration strategy (*opt. mig.*) for computing resources (*C*), memory (*M*), and storage (*S*).

Example: Modifying the virtual or substrate network

Starting from Fig. 2b, removing the master/failover connection (see Fig. 3) results in an migration of $V1$ and $V2$, which is only needed to ensure optimality while the constraints would still be satisfied without any migration. But, when we change the CPU of $S3$ according to Fig. 4a a required migration is necessary resulting in Fig. 4b.

For defining the VNE problem as a high-level specification, we developed the metamodel presented in Fig. 6. This UML class diagramm [Fow04] can be divided into three parts: (i) the virtual networks on the left-hand side (prefix *Virtual*), (ii) embeddings between the virtual and the substrate network in the middle, and (iii) the substrate network on the right-hand side (prefix *Substrate*). In the metamodel, constraints such as that a virtual server must only be mapped to a substrate server or a virtual server must only have one master/failover server are

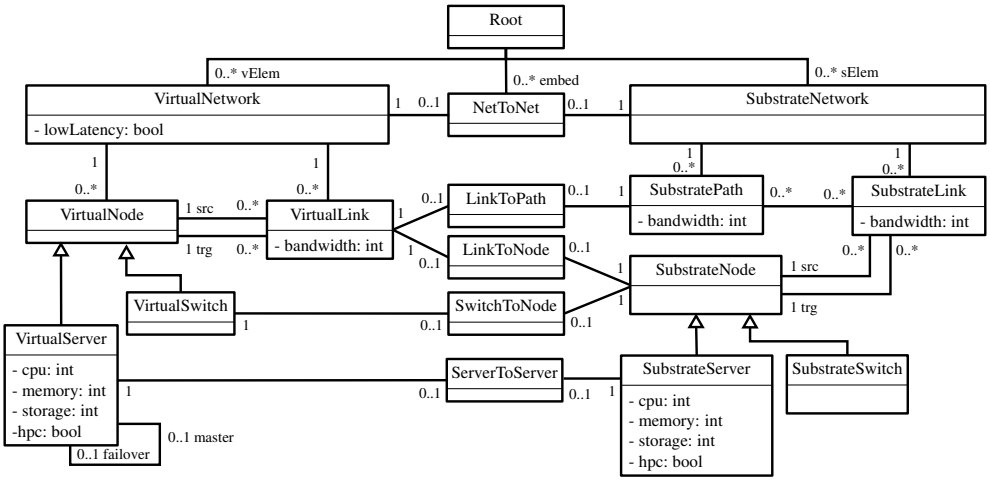


Figure 6 – Metamodel

already included and realized by the multiplicities at the associations. The Object Constraint Language (OCL) [WK04] is used to ensure constraints that cannot be expressed directly in the metamodel, e.g., the number of CPU resources of a substrate server must be less than or equal to all embedded virtual servers (see Eq. (2)).

$$\textbf{context}\ \text{SubstrateServer}\ \textbf{inv}\ (\text{self.virtualServers.cpu} \rightarrow \text{sum}()) \leq \text{self.cpu} \tag{2}$$

Thus, all necessary constraints from the problem description (see Sec. 2.1) can be guaranteed by the metamodel and additional OCL constraints. According to this metamodel and the OCL constraints, we create MT rules manually for the candidate generation step (Fig. 5 ①).

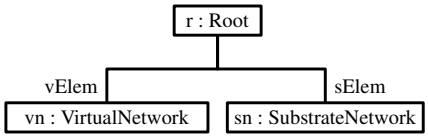
3.1 Candidate Generation (MT)

To create, update, and delete the mapping candidates (step ①) from Fig. 5, we use incremental MT techniques [MG06]. Therefore, we employ an incremental pattern matcher which finds occurrences of patterns in a model, referred to as matches. These matches can be understood as injective mappings of model elements to the pattern elements, hence, every model element occurs at most once in a match. The advantage of using incremental pattern matching is that it keeps track of changes (deltas) to a model and notifies the user of dis-/appeared matches rather than collecting all matches from scratch. This allows us to focus on the deltas, created by the incoming events, of the virtual and substrate networks, which are usually very small compared to the entire model. Since in incremental pattern matching all deltas are monitored, we distinguish between three cases: (i) a new match is added, (ii) an existing match is changed, and (iii) a match disappears. Table 2 summarizes these cases in correlation to the input events for iMdVNE.

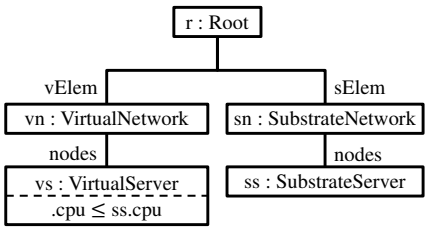
In Fig. 7, we present a selection of MT patterns for creating, modifying, and deleting mapping candidates. Fig. 7a shows the pattern for an embedding of a virtual into a substrate network. The elements (*r*, *vn*, and *sn*) refer to model instances and a match is found if these instances with the defined links exist. For every new match of this pattern, a new mapping candidate is generated resulting in a set of all combinations of virtual and substrate networks. In this case, a new *NetToNet* element (see Fig. 6), representing a new potential mapping candidate, is created for each new match. Since a network can be both created and deleted, this pattern

Event	New match	Update match	Delete match
+ Create element	X	-	-
O Increase virtual resource *C/M/S*	-	X	X
O Decrease virtual resource *C/M/S*	X	X	-
O Decrease substrate resource *C/M/S*	-	X	X
O Increase substrate resource *C/M/S*	X	X	-
O Change property	X	-	X
– Delete element	-	-	X

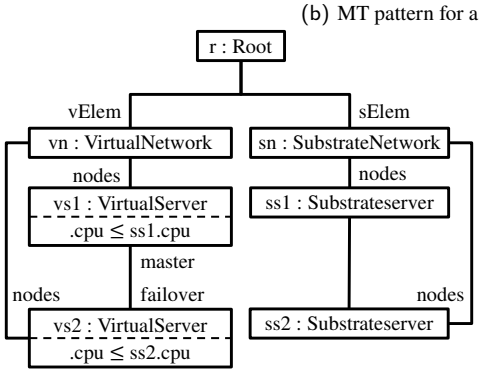
Table 2 – Effect of changes (events) in networks on matches



(a) MT pattern for a network embedding



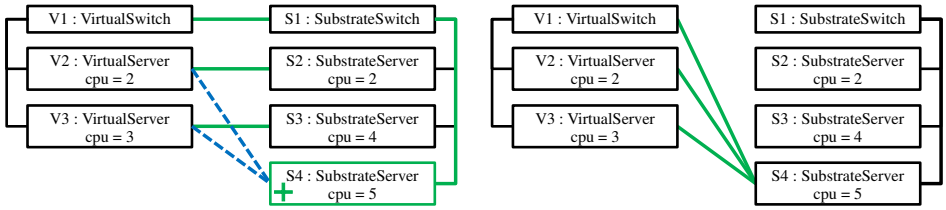
(b) MT pattern for a server embedding



(c) MT pattern for a server embedding with an master/failover connection

Figure 7 – MT patterns for server embeddings

can also be used to find and delete affected mapping candidates. Therefore, we delete the corresponding mapping candidate (*NetToNet* element) if a match of this pattern disappears. Fig. 7b shows the pattern for embedding a virtual into a substrate server. Similar to Fig. 7a, all elements (*r*, *vn*, *sn*, *vs*, and *ss*) with the defined links are located in the model with an additional check if the number of CPU resources of the virtual server does not exceed the CPU resources of the substrate server (.cpu $\leq$ ss.cpu). Analogously to Fig. 7a, a new *ServerToServer* element (see Fig. 6), representing a new potential mapping candidate, is created when a new match is found and, therefore, a new server is added. A deletion of a server or changing the CPU resource so that .cpu $\leq$ ss.cpu does not hold, results in the disappearance of a match and, therefore, a deletion of the *ServerToServer* mapping candidate. In the last pattern Fig. 7c, two virtual servers, where *vs2* acts as failover server for *vs1*, are found together with two substrate servers with sufficient CPU resources. The procedure is analogously to Figs. 7a and 7b, whereby it is additionally ensured that master and failover servers are not placed on the same substrate server.



(a) Adding a substrate server (b) Embeddings after the optimizing migration

Figure 8 – Adding a new substrate server after Fig. 4b

Example: Incremental pattern matching
Let us assume that the model is in the state shown in Fig. 2b and that the master/failover connection between *V2* and *V3* is removed now (see Fig. 3a). Removing the master/failover connection also removes the match of the pattern presented in Fig. 7c resulting in the deletion of the two *ServerToServer* mapping candidates between *V2*-*S2* and *V3*-*S3*. Now, the constraint that all virtual elements are embedded exactly once (see Sec. 2.1.3) is violated. Therefore, *V2* and *V3* must be re-embedded shown in Fig. 3b. Then, the CPU resource of *S3* is reduced from 5 to 4 (see Fig. 4a) violating the constraint that the resources of the substrate elements must not be overbooked. This constraint violation can only be detected during the candidate selection step, since condition .cpu $\leq$ ss.cpu in the pattern from Fig. 7b is not violated. In the next step in Fig. 8a, a new substrate server *S4* with a CPU resource of 5 is added. By this event, new matches of the pattern Fig. 7b are found, whereby also new mapping candidates between *V2*-*S4* and *V3*-*S4* are created. Now, we must distinguish between the two strategies required or optimizing migration. In the first case (required migration), the embedding from Fig. 4b is retained, since no constraint has been violated. In the second case (optimizing migration), the embedding would change to Fig. 8b, since a solution with a lower value for the optimization function exist.

3.2 Candidate Selection (ILP)

To generate and update the currently existing ILP problem, the deltas initialized by the incoming events as well as the deltas and changes of the candidate generation step are integrated into the ILP problem. Four cases have to be distinguished when updating the ILP problem: (i) Adding new mapping variables and constraints, (ii) adjusting the optimization function, (iii) updating existing constraints, and (iv) deleting existing mapping variables and constraints. Table 3 summarizes the possible cases for adding a variable or constraint (*add v./c.*), changing the objective (*change o.*), changing a constraint (*change c.*) or deleting a variable or constraint (*delete v./c.*) in correlation to the input event. In Fig. 5 step (B), the deltas are integrated into the existing ILP problem and solved. This allows the ILP Solver to (partially) reuse previously calculated solutions or approximations. After that, the solutions from the ILP solver are propagated back into the model.

4 Evaluation

In this section, the developed iMdVNE approach using a high-level VNE problem specification is evaluated in comparison to a manually coded Java program generating and updating an

Modification	Add v./c.	Change o.	Change c.	Delete v./c.
+ New embedding	X	X	X	-
O Update virtual embedding	-	X	X	-
O Update substrate embedding	-	-	X	-
– Delete embedding	-	X	X	X

Table 3 – Effect of embedding modifications for the ILP formulation

ILP formulation without MT techniques (*No MT*). The Java program uses an established ILP formulation and generates ILP programs that are state-of-the-art. The following research question is investigated in detail:

Is the performance of the iMdVNE approach comparable to a manually developed and optimized (Java) program generating and updating an ILP program?

To answer this research question, two migration strategies are considered that represent extreme cases: *required migration* (**RQ1**) and *optimizing migration* (**RQ2**).

4.1 Setup

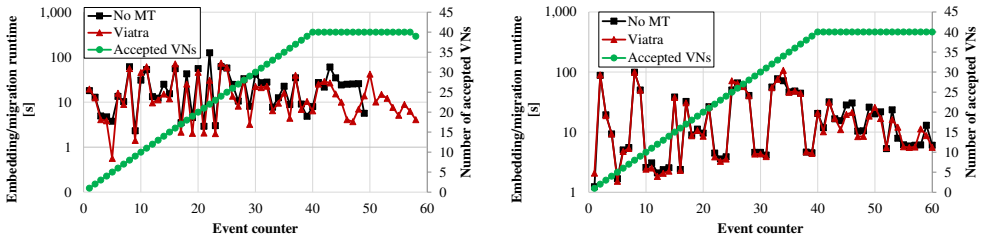
The evaluation setup consists of a 2-tier substrate network with 2 core switches connected to each rack switch via a bandwidth of 10 Gbit/s. Each rack in turn consists of a rack switch and 10 servers, each equipped with 32 CPU cores, 512 GB memory, and 1 TB storage. The servers are connected to the rack switch with a bandwidth of 1 Gbit/s. Thus, the substrate network has, in total, 80 servers distributed among 8 racks.

Virtual networks use a star topology with 2 to 10 virtual servers. Realistic values from [SvBI15] are used as resources of the servers and links. The values for the CPU for each server are thus between 1 to 32 and for the RAM between 1 GB to 511 GB. Since [SvBI15] does not contain any information about storage, values between 50 GB to 300 GB per server are assumed. The required bandwidths between the virtual servers and the central virtual switch are between 0.1 Gbit/s to 1 Gbit/s. For the exact probability distributions of CPU, memory, and bandwidth, we refer to [SvBI15].

We define two constraint setups: (i) a *basic* and (ii) an *all constraint setup*. The basic constraint setup (BC setup) refers to the basic constraints defined in Sec. 2.1.3. In the all constraint setup (AC setup), we additionally include the properties for master/failover and high performance server as well as the low latency networks. Therefore, in every virtual network between 0 to 2 virtual failover servers with the corresponding master servers are integrated. The probability that a virtual server is a high performance server and a virtual network is a low latency network is 20 %.

The following functions are defined to calculate the objective from Eq. (1). The communication costs $cost^L(p_{uv}, l_{ij})$ for the 2-tier substrate network used are based on the cost matrix for the VL2 network topology by [MPZ10] and is 0 if p_{uv} has length 0, $B^V_{l_{ij}}$ if p_{uv} has length 1, and $5B^V_{l_{ij}}$ if p_{uv} has length 2 or more. The migration costs $cost^M$ are defined as static costs, which consist of the sum of all required resources of a virtual server (CPU, memory, and storage).

As a dynamic VNE scenario, we first add 40 virtual networks consecutively and then remove 20 substrate servers after each other. If possible, we delete a substrate server with existing embeddings. Therefore, a migration is required to resolve the violated constraints. Thus, an event counter with 60 steps is used in the diagrams, whereby the first 40 steps refer to



(a) AC setup (b) BC setup

Figure 9 – Embedding/migration runtime with the required migration strategy

		Add virtual network [s]		Remove substrate server [s]	
		AC setup	**BC setup**	**AC setup**	**BC setup**
Fig. 9	***No MT***	24.4/20.9	26.0/22.6	27.8/24.3	15.2/12.4
	Viatra	20.3 (83 %)/18.2	27.0 (104 %)/24.0	13.6 (49 %)/11.7	13.2 (87 %)/10.3
Fig. 10	***No MT***	3.5/0.9	3.0/0.6	497/495	3489/3487
	Viatra	2.4 (64 %)/0.8	2.6 (86 %)/0.6	454 (91 %)/453	3497 (100 %)/3495

Table 4 – (Neu Diss) Mean values for Fig. 9 and Fig. 10

the embedding of the new virtual networks and the last 20 steps refer to the deletion of the substrate server including the necessary migration.

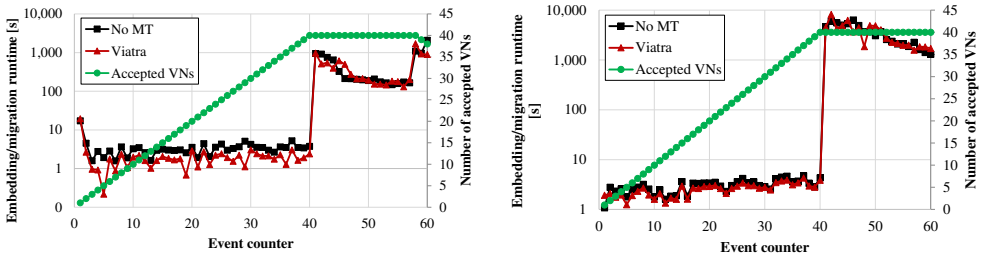
As metrics, we use the *embedding/migration runtime*, *ILP solving time*, and *number of accepted virtual networks*. The embedding/migration runtime is the complete runtime for processing the add virtual network or remove substrate server event. The ILP solving time is the runtime of the ILP solver to solve the ILP problem. The number of accepted virtual networks indicates the number of successfully embedded virtual networks.

The software tools used in this evaluation are Viatra [CHM+02], a state-of-the-art incremental graph pattern matcher, the Eclipse Modeling Framework (EMF) [SBMP08], and Gurobi [GO16], a state-of-the-art ILP solver. All experiments were run on a Ubuntu 19.04 machine performed with an AMD Ryzen Threadripper 2990WX (32 cores) CPU, 128 GB RAM, and OpenJDK 12. In the following, each data point is the median of three repeated experiments.

4.2 RQ 1: Required Migration

Fig. 9 shows the embedding/migration runtime for adding a virtual network or removing a substrate server for the 2 configurations (*Viatra* and *No MT*) on the left y-axis and the number of accepted virtual networks on the right y-axis over the event counter on the x-axis. The required migration strategy is used with the AC setup (Fig. 9a) or the BC setup (Fig. 9b). Table 4 summarizes the mean values for the add virtual network or the remove substrate server event for the embedding/migration runtime (first value) and the ILP solving time (second value). The percentage of the embedding/migration runtime compared to the *No MT* configuration is also given in brackets.

> **Example: Measurements**
> *Viatra* needs 20.3 s to embed a virtual network (embedding runtime) and *No MT* 24.4 s (see Fig. 9). This means that *Viatra* requires only 83 % of the embedding runtime compared to *No MT*. This embedding runtime includes the ILP solving time, which is 20.9 s for *No MT* and 18.2 s for *Viatra*.



(a) AC setup

(b) BC setup

Figure 10 – Embedding/migration runtime with the optimizing migration strategy

In both scenarios (Fig. 9), all virtual networks can be embedded one after the other (steps 1 to 40). In the case of migration, in the AC setup (Fig. 9a), after deleting 18 substrate servers in *Viatra* (9 for *No MT*) the VNE problem cannot be solved. This can be related to decisions made during the embedding of new virtual networks, which of course affect all subsequent embeddings. The average runtime for embedding a new virtual network differs in *Viatra* by 17 % in comparison to the *No MT* runtimes. When deleting a substrate server, *Viatra* can solve the VNE problem 51 % faster than *No MT* in the AC setup (*Viatra* is on average only 13 % faster in the BC setup). Since there are many more structural properties in the AC setup than in the BC setup, which can be integrated into the MT rules, many possible candidates can be discarded during candidate generation (e.g., the number of hops for a low latency network must be less than 2). This also leads to a reduced ILP solving time, which is shown by the mean time for deleting a substrate server in the AC setup for *Viatra*.

4.3 RQ 2: Optimizing Migration

Fig. 10 shows, similar to Fig. 9, the embedding/migration runtime, as well as the accepted networks over the event counter for the optimizing migration strategy. In Fig. 10a the AC setup is presented and in Fig. 10b the BC setup. Their mean values are presented in Table 4 and refer to the embedding/migration runtime, the ILP solving time and the percentage of *Viatra* in comparison to *No MT*.

When adding virtual networks, *Viatra* can solve the VNE problem up to 36 % faster than *No MT* in the AC and up to 14 % in the BC setup. Due to the migration strategy, already embedded elements can be migrated, which increases the number of possible solutions drastically. Therefore, the migration runtime is dominated by the ILP solving time and the increased number of (structural) constraints (AC setup) leads to a reduced migration runtime compared to the BC setup. This is shown for *Viatra* as well as for *No MT*.

When deleting substrate servers, we can also observe an average reduction of 9 % in the runtime for the AC setup and the ILP solving time. In the case of the BC setup, *Viatra* needed on average the same time to solve the VNE problem than *No MT*. The number of accepted networks were identically in both cases.

4.4 Threats to validity

In order to maximize the probability that *No MT*, representing the manually implemented incremental dynamic VNE tool, and *Viatra*, representing the iMdVNE tool generated from the high-level specification, are equivalent, the following techniques are used: (i) consistency

checks during the embedding or migration of virtual elements, (ii) unit tests, and (iii) less algorithm specific code for *Viatra* or *No MT*.

For technique (i), all embeddings and migrations are checked for constraints before being deployed to the model, so that only valid embeddings and migrations are performed regardless of the found solution. In technique (ii), 295 unit tests are used to check the correctness and optimality of the embeddings using sample networks and scenarios. In technique (iii), *Viatra* and *No MT* differ only in the code for the pattern matching, which was created manually for *No MT*. All other components like the processing of the found deltas, the updating and solving of the ILP problem, as well as the deployment of the embeddings are identically. Since the virtual and substrate networks in this evaluation were created using the probability distributions from [SvBI15], all experiments were repeated three times to minimize the influence of random events. Since the ILP solver used has a large influence on the runtime, we decided to integrate Gurobi, a state-of-the-art solver that is also used in industry.

4.5 Summary of the Evaluation

By using a program derived from a high-level specification with iMdVNE (*Viatra*), performance could be increased on average by up to 51 % for required migration strategy and up to 36 % for optimizing migration strategy compared to a manually tailored Java program for generating and updating an ILP program. Disadvantages in the runtime only occurred in certain BC setups, whereby the runtime increase was in the range of up to 4 %. This shows that the ILP programs created by iMdVNE are (in most cases) more efficient for solving dynamic VNE problems. The number of accepted virtual networks was comparable between *Viatra* and *No MT*. Thus, we presented in the evaluation that the program derived from a high-level specification with iMdVNE can (in most cases) solve the VNE problem faster than a manually tailored Java program for generating and updating an ILP program.

5 Related Work

In this section, we present an overview about related work for dynamic and online VNE approaches and incremental MT technologies.

5.1 Dynamic and online VNE approaches

The research in algorithms for solving the online VNE problems for static data centers has been extensively investigated. An overview of solutions can be found in [BBE+13] and a comparison of algorithms in [YDZ+17]. However, these two survey papers do not focus on dynamic approaches or the supported resource constraints.

Therefore, we present an overview of dynamic and online VNE algorithms for data centers. We searched for the terms *Virtual Network Embedding Data Center* in the IEEE Xplore Digital Library [IEE] and selected all suitable papers that are comparable to the problem description in Sec. 2.1. At the end 12 papers could be found for solving the dynamic and online VNE problem, which are using heuristics-based algorithms. Table 5 gives an overview of these papers with their supported resources slots, computing capacity (*C*), memory (*M*), storage (*S*), bandwidth (*B*) and their dynamic events for deleting virtual networks (*delete VN*) or changing the resources (*change res.*). Both events can be triggered during runtime while changing the resources, e.g., when the CPU value of a virtual server is changed, may require a migration of the virtual server. The *slot* column indicates whether the paper uses slots or slices as an abstraction for the resources.

Reference	Slot	C	M	S	B	Del. VN	Change Res.
[ZLW+12, LYL+14, ZZSB13]	x	-	-	-	x	x	-
[XDHK12]	x	-	-	-	x	-	x
[YC16, DY16]	x	-	-	-	x	x	x
[FSSC16, GCS14, YWPS19]	-	x	-	-	x	-	x
[NHDN16]	-	x	x	-	x	x	-
[YCLL17]	-	x	x	x	x	x	-
[LS17]	-	x	x	x	x	x	x

Table 5 – Heuristics-based dynamic and online VNE algorithms for data centers

Only [LS17] provides a heuristics-based algorithm which supports all resources (CPU, memory, storage, and bandwidth) as well as both dynamic features (delete virtual networks and change resources). However, no dynamic algorithm could be found that guarantees optimal and valid solutions, ensuring all constraints, for the VNE problem in data centers. This can be explained by the fact that finding optimal solutions is only suitable for small data centers due to the very large search space and the associated long runtime [YG16]. However, optimal solutions for dynamic and online VNE scenarios can be used as a benchmark for quality measurements of search space pruning strategies like heuristics-based approaches.

5.2 Incremental MT

Graph pattern matching techniques can roughly be divided into batch and incremental pattern matching. Batch solutions rely mostly on local-search [Zü96] or solving a constraint satisfaction problem [LV02] which has the disadvantage that changing the model makes it necessary to collect all matches from scratch. In contrast, incremental pattern matching keeps track of model changes such that a set of new or invalidated matches can be found more efficiently. There have been some proposals for incremental pattern matching such as Varro et al. [VVS06] that introduce basic data structures for this purpose, Kanezashi et al. [KSGG+18] which employ reinforcement learning, and Fan et al. [FLL+11] that focus on social networks and other big data scenarios. However, the most popular approach w.r.t. model transformations is probably the RETE-algorithm [For82]. The key concept of RETE is to keep track of all partial matches in order to efficiently evaluate new matches or re-evaluate old ones in case that the model changes. This leads to an approach that scales with the size of the model change rather than the size of the model itself. RETE has been implemented/adapted by several tools such as Viatra [VBH+16] and Democles [VD13]. Besides pattern matching, Viatra offers a textual syntax to specify model transformations on EMF models together with advanced features such as finding the transitive closure. In contrast, Democles is a stand-alone incremental graph pattern matching engine which interprets the patterns and comes without the need to generate code.

6 Conclusion and Future Work

In this paper, we presented a novel model-driven approach that uses incremental MT and ILP technologies to solve dynamic VNE scenarios in data centers. Since the hand-crafted implementation of a tool to generate and update an ILP program that only considers model changes and passes them on to the ILP solver is very time-consuming and error-prone, the presented model-driven approach automatically derives a tool for solving dynamic VNE scenarios from a high-level specification. By using a high-level specification, it is possible to

develop and evaluate the quality of the embeddings and migrations as well as the performance to solve the VNE problem for new search space pruning strategies while ensuring that all constraints are respected. Thus, programming errors can be avoided by using MT techniques of these (partly) very complex programs to solve dynamic VNE problems. In many cases, the generated tool is even more efficient than a hand-crafted tool, since best practice approaches for efficient ILP programs can be automatically adhered to. Thus, when deleting a substrate server, the runtime for solving the VNE problem for the required migration strategy is reduced on average by up to 51 % (36 % for the optimizing migration strategy) while the number of accepted networks remains the same.

In future research, we plan to investigate the influence of MT tools and the impact of changing the ILP problem more intensively. Extensive parallelization of processes inside the pattern matcher could also improve the performance in solving the VNE problem. Furthermore, we want to apply this approach to other domains such as resource management in clouds [JS15] or service chain allocation problems [BB17].

References

[ACKT16] Edoardo Amaldi, Stefano Coniglio, Arie M. C. A. Koster, and Martin Tieves. On the computational complexity of the virtual network embedding problem. *Electronic Notes in Discrete Mathematics*, 52:213–220, 2016. doi:10.1016/j.endm.2016.03.028.

[BB17] Michael Till Beck and Juan Felipe Botero. Scalable and coordinated allocation of service function chains. *Computer Communications*, 102:78–88, 2017. doi:10.1016/j.comcom.2016.09.010.

[BBE+13] Md. Faizul Bari, Raouf Boutaba, Rafael Pereira Esteves, Lisandro Zambenedetti Granville, Maxim Podlesny, Md. Golam Rabbani, Qi Zhang, and Mohamed Faten Zhani. Data center network virtualization: a survey. *Communications Surveys and Tutorials*, 15(2):909–928, 2013. doi:10.1109/SURV.2012.090512.00043.

[BCKR11] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Antony I. T. Rowstron. Towards predictable datacenter networks. In *Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, pages 242–253, 2011. doi:10.1145/2018436.2018465.

[CHM+02] György Csertán, Gábor Huszerl, István Majzik, Zsigmond Pap, András Pataricza, and Dániel Varró. VIATRA - visual automated transformations for formal verification and validation of UML models. In *Intl. Conf. Automated Software Engineering (ASE)*, pages 267–270, 2002. doi:10.1109/ASE.2002.1115027.

[DY16] Jun Duan and Yuanyuan Yang. Efficient virtual network embedding for variable size virtual machines in fat-tree data centers. In *Intl. Conf. on Parallel Processing (ICPP)*, pages 1–10, 2016. doi:10.1109/ICPP.2016.8.

[FBB+13] Andreas Fischer, Juan Felipe Botero, Michael Till Beck, Hermann de Meer, and Xavier Hesselbach. Virtual network embedding: a survey. *Communications Surveys and Tutorials*, 15(4):1888–1906, 2013. doi:10.1109/SURV.2013.013013.00155.

[FLL+11] Wenfei Fan, Jianzhong Li, Jizhou Luo, Zijing Tan, Xin Wang, and Yinghui Wu. Incremental graph pattern matching. In *Proceedings of the 2011 ACM*

SIGMOD International Conference on Management of Data, SIGMOD '11, pages 925–936, 2011. doi:10.1145/1989323.1989420.

[For82] Charles Forgy. Rete: A fast algorithm for the many patterns/many objects match problem. *Artificial Intelligence*, 19(1):17–37, 1982. doi:10.1016/0004-3702(82)90020-0.

[Fow04] Martin Fowler. *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004.

[FSSC16] Carlo Fuerst, Stefan Schmid, P. Lalith Suresh, and Paolo Costa. Kraken: online and elastic resource reservations for multi-tenant datacenters. In *Intl. Conf. on Computer Communications (INFOCOM)*, pages 1–9, 2016. doi:10.1109/INFOCOM.2016.7524466.

[GCS14] Xinjie Guan, Baek-Young Choi, and Sejun Song. Topology and migration-aware energy efficient virtual network embedding for green data centers. In *Intl. Conf. on Computer Communication and Networks (ICCCN)*, pages 1–8, 2014. doi:10.1109/ICCCN.2014.6911768.

[GO16] I Gurobi Optimization. Gurobi optimizer reference manual. 2016.

[IEE] IEEE. IEEE xplore digital library. URL: https://ieeexplore.ieee.org/Xplore/home.jsp.

[JS15] Brendan Jennings and Rolf Stadler. Resource management in clouds: survey and research challenges. *J. Network Syst. Manage.*, 23(3):567–619, 2015. doi:10.1007/s10922-014-9307-7.

[KSGG+18] Hiroki Kanezashi, Toyotaro Suzumura, Dario Garcia-Gasulla, Min hwan Oh, and Satoshi Matsuoka. Adaptive pattern matching with reinforcement learning for dynamic graphs. *Conf. on High Performance Computing (HiPC)*, pages 92–101, 2018. doi:10.1109/HiPC.2018.00019.

[LS17] Federico Larumbe and Brunilde Sansò. Elastic, on-line and network aware virtual machine placement within a data center. In *Intl. Symposium on Integrated Network Management (IM)*, pages 28–36, 2017. doi:10.23919/INM.2017.7987261.

[LV02] Javier Larrosa and Gabriel Valiente. Constraint satisfaction algorithms for graph pattern matching. *Mathematical. Structures in Comp. Sci.*, 12(4):403–422, August 2002. doi:10.1017/S0960129501003577.

[LYL+14] Shouxi Luo, Hongfang Yu, Lemin Li, Dan Liao, and Gang Sun. Traffic-aware vdc embedding in data center: A case study of fattree. *China Communications*, 11(7):142–152, 2014. doi:10.1109/CC.2014.6895393.

[MG06] Tom Mens and Pieter Van Gorp. A taxonomy of model transformation. *Electr. Notes Theor. Comput. Sci.*, 152:125–142, 2006. doi:10.1016/j.entcs.2005.10.021.

[MPZ10] Xiaoqiao Meng, Vasileios Pappas, and Li Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *Intl. Conf. on Computer Communications (INFOCOM)*, pages 1154–1162, 2010. doi:10.1109/INFCOM.2010.5461930.

[NHDN16] Tran Manh Nam, Nguyen Van Huynh, Le Quang Dai, and Huu-Thanh Nguyen. An energy-aware embedding algorithm for virtual data centers. In *Intl. Teletraffic Congress (ITC)*, pages 18–25, 2016. doi:10.1109/ITC-28.2016.112.

[SBMP08] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. *EMF - Eclipse Modeling Framework*. Pearson Education, 2008.

[Sch94] Andy Schürr. Specification of graph translators with triple graph grammars. In *Graph-Theoretic Concepts in Computer Science*, pages 151–163, 1994. doi:10.1007/3-540-59071-4_45.

[Ste09] Rod Stephens. *Beginning database design solutions*. John Wiley & Sons, 2009.

[STR+15] Sahel Sahhaf, Wouter Tavernier, Matthias Rost, Stefan Schmid, Didier Colle, Mario Pickavet, and Piet Demeester. Network service chaining with optimized network function embedding supporting service decompositions. *Computer Networks*, 93:492–505, 2015. doi:10.1016/j.comnet.2015.09.035.

[SvBI15] Siqi Shen, Vincent van Beek, and Alexandru Iosup. Statistical characterization of business-critical workloads hosted in cloud datacenters. In *Intl. Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 465–474, 2015. doi:10.1109/CCGrid.2015.60.

[TLWS18a] Stefan Tomaszek, Erhan Leblebici, Lin Wang, and Andy Schürr. Virtual network embedding: reducing the search space by model transformation techniques. In *Intl. Conf. on Model Transformation (ICMT)*, pages 59–75, 2018. doi:10.1007/978-3-319-93317-7_2.

[TLWS18b] Stefan Tomaszek, Erhan Leblebici, Lin Wang, and Andy Schürr. Model-driven development of virtual network embedding algorithms with model transformation and linear optimization techniques. In *Modellierung 2018*, pages 39–54, 2018.

[VBH+16] Dániel Varró, Gábor Bergmann, Ábel Hegedüs, Ákos Horváth, István Ráth, and Zoltán Ujhelyi. Road to a reactive and incremental model transformation platform: three generations of the VIATRA framework. *Software and System Modeling*, 15(3):609–629, 2016. doi:10.1007/s10270-016-0530-4.

[VD13] Gergely Varró and Frederik Deckwerth. A rete network construction algorithm for incremental pattern matching. In *Intl. Conf. on Model Transformation (ICMT)*, pages 125–140, 2013. doi:10.1007/978-3-642-38883-5_13.

[VVS06] Gergely Varró, Dániel Varró, and Andy Schürr. Incremental graph pattern matching: Data structures and initial experiments. *Electronic Communications of the EASST*, 4, 2006. doi:10.14279/tuj.eceasst.4.12.

[WK04] Jos B. Warmer and Anneke G. Kleppe. *Object constraint language 2.0*. Software-Entwicklung. mitp-Verl., 2004.

[XDHK12] Di Xie, Ning Ding, Y. Charlie Hu, and Ramana Rao Kompella. The only constant is change: incorporating time-varying network reservations in data centers. In *Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, pages 199–210, 2012. doi:10.1145/2342356.2342397.

[YC16] Lei Yu and Zhipeng Cai. Dynamic scaling of virtual clusters with bandwidth guarantee in cloud datacenters. In *Intl. Conf. on Computer Communications (INFOCOM)*, pages 1–9, 2016. doi:10.1109/INFOCOM.2016.7524355.

[YCLL17] Yang Yang, Xiaolin Chang, Jiqiang Liu, and Lin Li. Towards robust green virtual cloud data center provisioning. *Trans. on Cloud Computing*, 5(2):168–181, 2017. doi:10.1109/TCC.2015.2459704.

[YDZ+17] Hanene Ben Yedder, Qingye Ding, Umme Zakia, Zhida Li, Soroush Haeri, and Ljiljana Trajkovic. Comparison of virtualization algorithms and topologies for data center networks. In *Intl. Conf. on Computer Communication and Networks (ICCCN)*, pages 1–6, 2017. doi:10.1109/ICCCN.2017.8038524.

[YG16] Z. Yang and Y. Guo. An exact virtual network embedding algorithm based on integer linear programming for virtual network request with location constraint. *China Communications*, 13(8):177–183, 2016. doi:10.1109/CC.2016.7563720.

[YWPS19] Ying Yuan, Cong Wang, Sancheng Peng, and Keshav Sood. Topology-oriented virtual network embedding approach for data centers. *IEEE Access*, 7:2429–2438, 2019. doi:10.1109/ACCESS.2018.2886270.

[YYRC08] Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration. *Computer Communication Review*, 38(2):17–29, 2008. doi:10.1145/1355734.1355737.

[ZGH+15] Deze Zeng, Song Guo, Huawei Huang, Shui Yu, and Victor C.M. Leung. Optimal vm placement in data centers with architectural and resource constraints. *Intl. Journal of Autonomous and Adaptive Communications Systems (IJAACS)*, 8(4):392–406, 2015. doi:10.1504/IJAACS.2015.073187.

[ZLW+12] Jing Zhu, Dan Li, Jianping Wu, Hongnan Liu, Ying Zhang, and Jingcheng Zhang. Towards bandwidth guarantee in multi-tenancy cloud computing networks. In *Intl. Conf. on Network Protocols (ICNP)*, pages 1–10, 2012. doi:10.1109/ICNP.2012.6459986.

[ZZSB13] Mohamed Faten Zhani, Qi Zhang, Gwendal Simon, and Raouf Boutaba. VDC planner: dynamic migration-aware virtual data center embedding for clouds. In *Intl. Symposium on Integrated Network Management (IM)*, pages 18–25, 2013.

[Zü96] Albert Zündorf. Graph pattern matching in progres. In *Workshop on Graph Gramars and Their Application to Computer Science*, pages 454–468, 1996. doi:10.1007/3-540-61228-9_105.

About the authors

Stefan Tomaszek is a Ph.D. student at the Real-Time Systems Laboratory of Prof. Andy Schürr at the Technical University of Darmstadt. His research interests are model-based software engineering and virtual network embedding problems. His research is part of the DFG project SFB 1053 MAKI. Contact him at stefan.tomaszek@es.tu-darmstadt.de.

Lars Fritsche is a Ph.D. student at the Real-Time Systems Laboratory of Prof. Andy Schürr at the Technical University of Darmstadt. His research interests are model-based consistency restoration and graph transformations. His research is part of the DFG project Triple Graph Grammars (TGG) 2.0. Contact him at lars.fritsche@es.tu-darmstadt.de.

Andy Schürr is Professor and head of the Real-Time Systems Laboratory of the Technical University of Darmstadt University. His main research interests are related to model-based development (MBD) of embedded systems with a special emphasis on three application domains: automotive software development, automation engineering, and (self-adaptive)

communication systems. MBD-related research interests include (1) bidirectional model transformation languages, (2) integration of commercial-of-the-shelf engineering tools, (3) model-based testing of software product lines, and (4) self-adaptive distributed communication systems. Contact him at andy.schuerr@es.tu-darmstadt.de, or visit https://www.es.tu-darmstadt.de/.

Acknowledgments This work was funded by the German Research Foundation (DFG) as part of project A1 within the Collaborative Research Center (CRC) 1053 – MAKI. This work was partially funded by the German Research Foundation (DFG), project *Triple Graph Grammars (TGG) 2.0*.