Graph-Based Specification and Automated Construction of ILP Problems

GCM'22

Real-Time Systems Lab

Prof. Dr. rer. nat. Andy Schürr

Dept. of Electrical Engineering and Information Technology

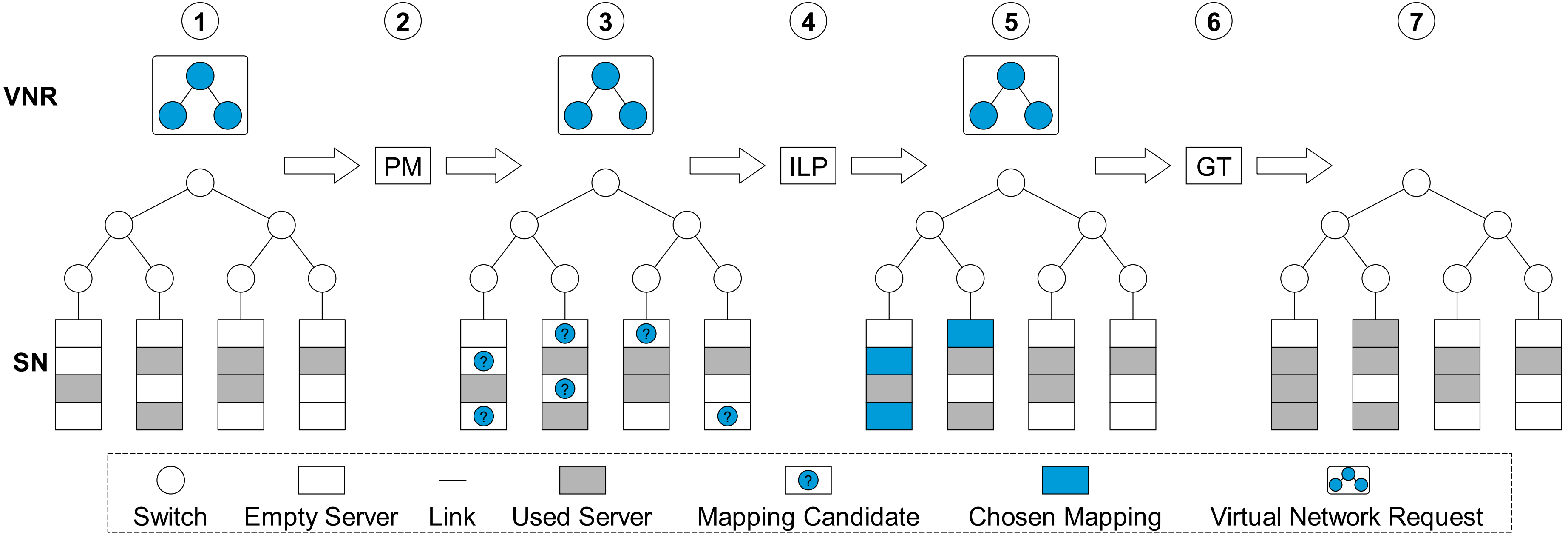
Dept. of Computer Science (adjunct Professor)

www.es.tu-darmstadt.de

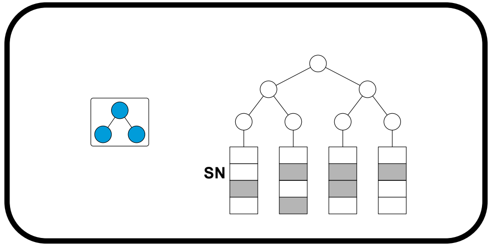
Sebastian Ehmes
Maximilian Kratz

This work has been funded by the German Research Foundation (DFG) within the Collaborative Research Center (CRC) 1053 MAKI.

Virtual Network Embedding (VNE)



Problem Description



➢ **Graph Mapping Problem**

- Input/Output: Graphs
- Constraints
- **Goal**: Optimize a given cost function (objective)
- For every new scenario:
  - Implement an ILP generator "by hand"
  - Connect the ILP generator with the GT framework
- Requires ILP expertise + Highly error-prone

State-Of-The-Art & Related Work

How to solve such problems until now?

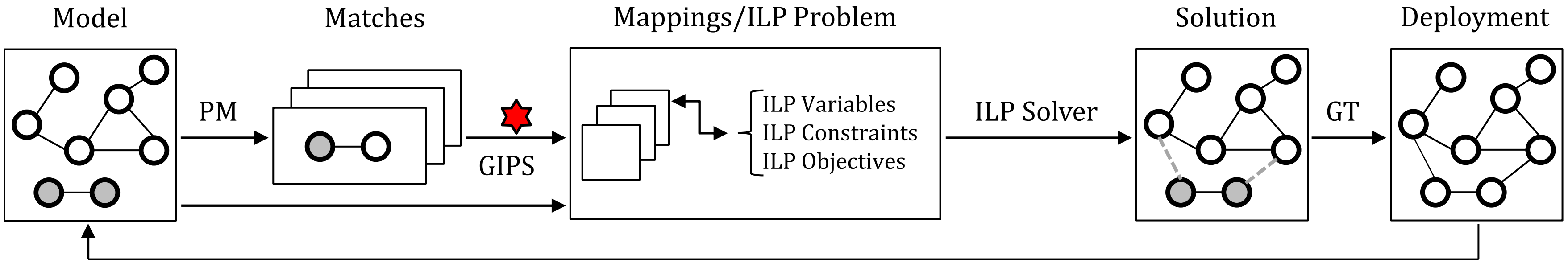
- Build yourself a specific tool for one problem domain
  - For example, with (M)ILP: iDyVE[1]
- Model synchronization tools
  - Triple Graph Grammars (e.g., with eMoflon-IBeX[2] or eMoflon-Neo[3])
  - Janus Transformation Language (JTL)[4]
- Others (example): MOMoT[5]

Our Idea

- Goal: Convenient specifying and efficient solving of graph mapping problems
  - ➢ "Without" ILP expertise
  - ➢ With little implementation effort
- Domain-Specific Language (DSL) for specifications
  - Based on eMoflon-GT
  - Integrates ILP constraints, patterns and GT rules
  - Adds specification of objectives
- Framework generates ILP generators using a given specification
- Combines eMoflon::IBeX-GT, HiPE[6], an ILP generator and an ILP solver

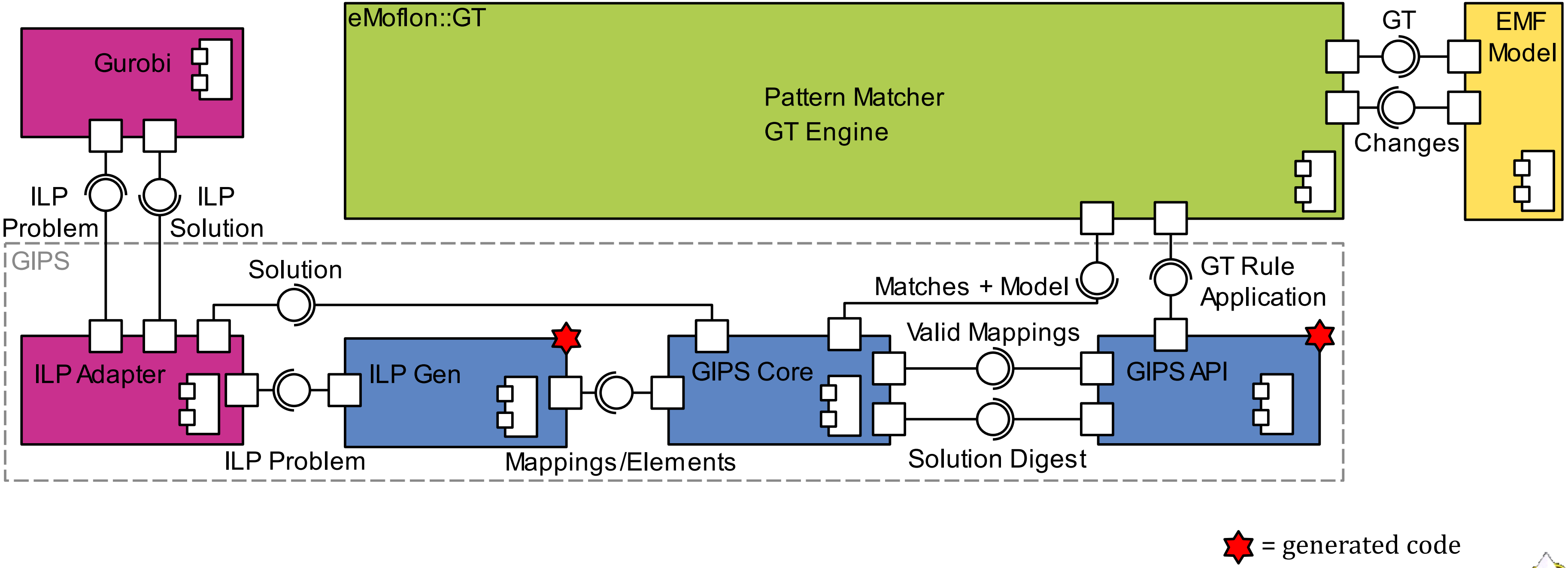
➢ A tool to build other tools

Our Idea: GIPS (**G**raph-Based **ILP** **P**roblem **S**pecification Tool)



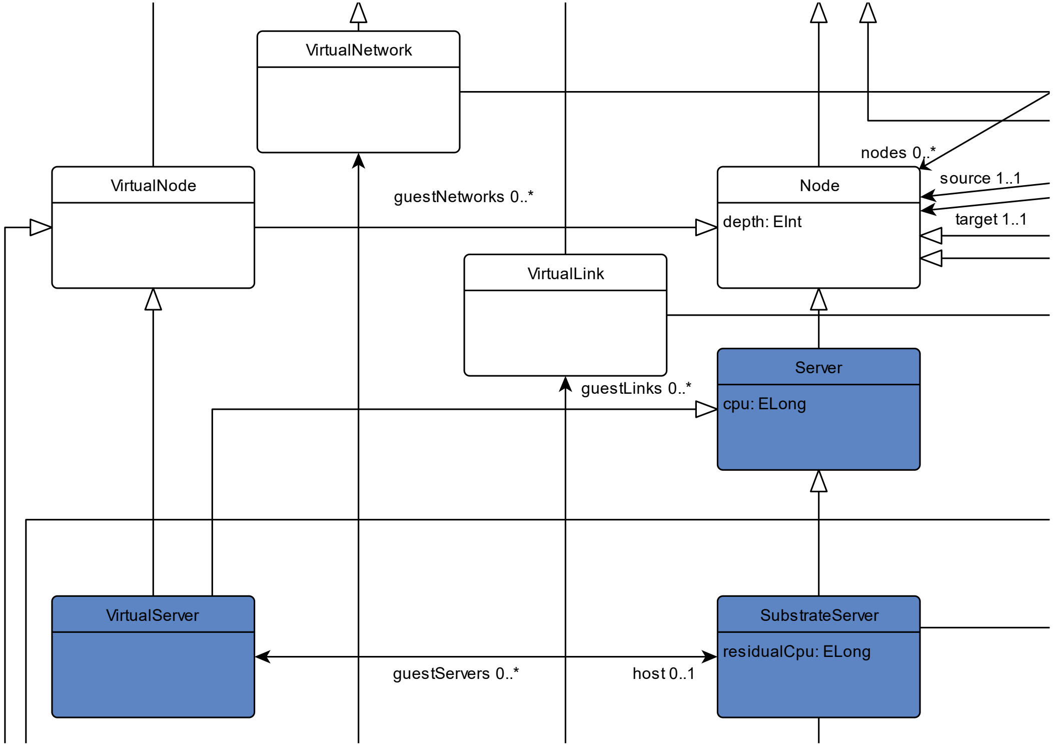
✶ = generated code

GIPS: Architecture



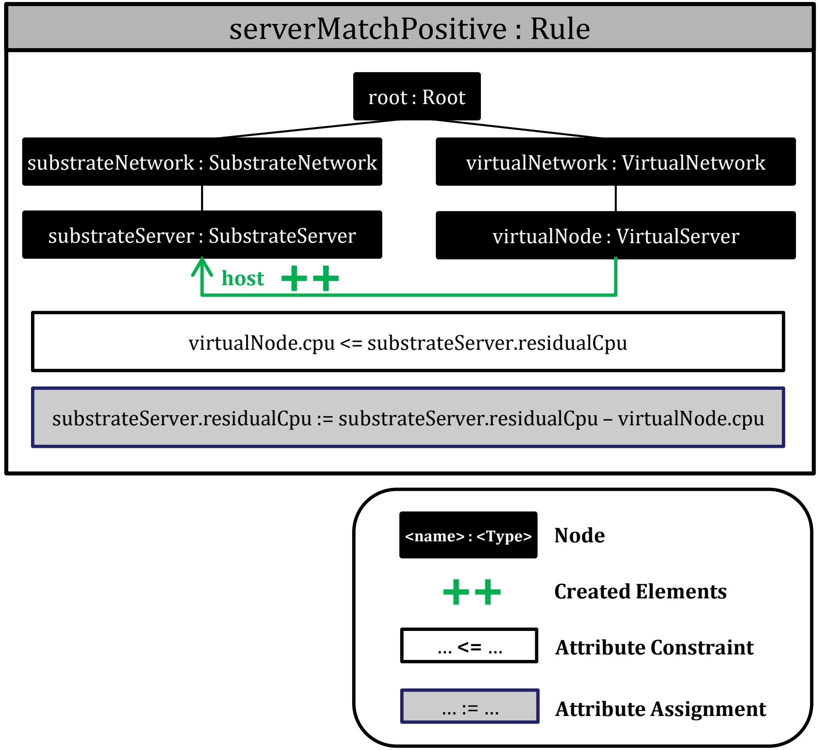
★ = generated code

Example: MdVNE – Metamodel



GIPSL (GIPS Language) as Example

```
rule serverMatchPositive {
    root: Root {
        -networks -> substrateNetwork
        -networks -> virtualNetwork
    }
    substrateServer: SubstrateServer {
        .residualCpu := substrateServer.residualCpu - virtualNode.cpu
        ++ -guestServers -> virtualNode
    }
    virtualNode: VirtualServer {
        ++ -host -> substrateServer
    }
    substrateNetwork: SubstrateNetwork {
        -nodes -> substrateServer
    }
    virtualNetwork: VirtualNetwork {
        -nodes -> virtualNode
    }
    # virtualNode.cpu <= substrateServer.residualCpu
}
when serverNotMapped
```



serverMatchPositive : Rule

root : Root

substrateNetwork : SubstrateNetwork

virtualNetwork : VirtualNetwork

substrateServer : SubstrateServer

virtualNode : VirtualServer

host ++

virtualNode.cpu <= substrateServer.residualCpu

substrateServer.residualCpu := substrateServer.residualCpu – virtualNode.cpu

<name> : <Type> Node

++ Created Elements

… <= … Attribute Constraint

… := … Attribute Assignment

GIPSL (GIPS Language) as Example

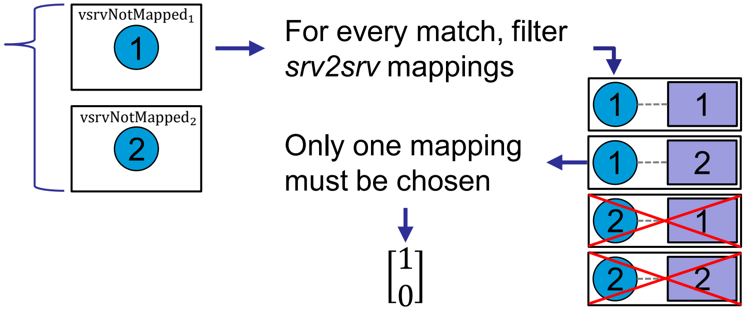
```
mapping srv2srv with serverMatchPositive;

constraint -> pattern::vsrvNotMapped {
    mappings.srv2srv
              ->filter(m | m.nodes().virtualNode == self.nodes().virtualServer)
                             ->sum(m | m.value()) == 1
}

constraint -> class::SubstrateServer {
    mappings.srv2srv
              ->filter(m | m.nodes().substrateServer == self)
                             ->sum(m | m.nodes().virtualNode.cpu)
                                           <= self.residualCpu
}

objective srvObj -> mapping::srv2srv {
    (self.nodes().substrateServer.residualCpu / self.nodes().substrateServer.cpu)
               * self.value()
}
```

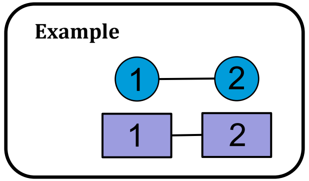
Matches of pattern *vsrvNotMapped*

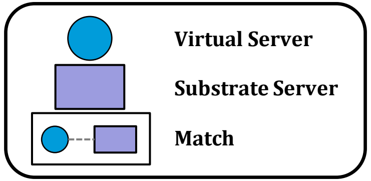


For every match, filter *srv2srv* mappings

Only one mapping must be chosen

$$\begin{bmatrix}1\\0\end{bmatrix}$$





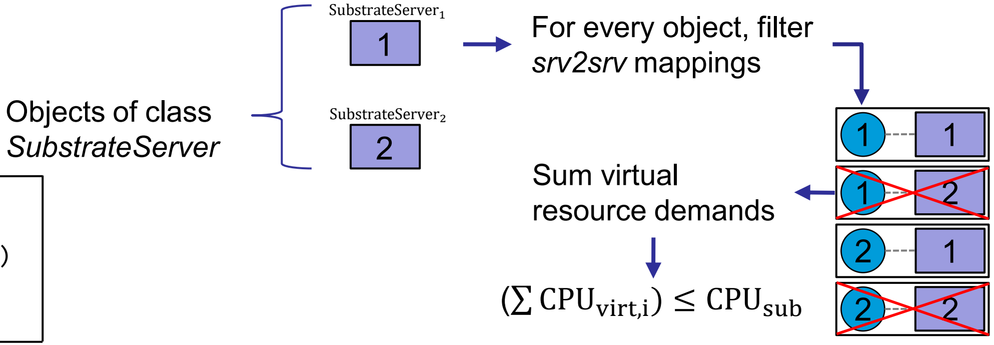
GIPSL (GIPS Language) as Example

```
mapping srv2srv with serverMatchPositive;

constraint -> pattern::vsrvNotMapped {
    mappings.srv2srv
              ->filter(m | m.nodes().virtualNode == self.nodes().virtualServer)
                             ->sum(m | m.value()) == 1
}

constraint -> class::SubstrateServer {
    mappings.srv2srv
              ->filter(m | m.nodes().substrateServer == self)
                             ->sum(m | m.nodes().virtualNode.cpu)
                                           <= self.residualCpu
}

objective srvObj -> mapping::srv2srv {
    (self.nodes().substrateServer.residualCpu / self.nodes().substrateServer.cpu)
               * self.value()
}
```

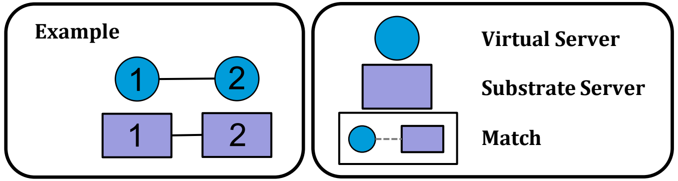


Objects of class *SubstrateServer*

For every object, filter *srv2srv* mappings

Sum virtual resource demands

$(\sum \mathrm{CPU}_{\mathrm{virt},i}) \leq \mathrm{CPU}_{\mathrm{sub}}$



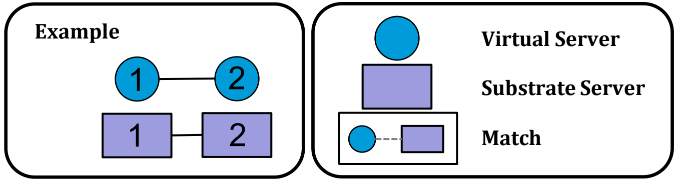
GIPSL (GIPS Language) as Example

```
mapping srv2srv with serverMatchPositive;

constraint -> pattern::vsrvNotMapped {
    mappings.srv2srv
              ->filter(m | m.nodes().virtualNode == self.nodes().virtualServer)
                            ->sum(m | m.value()) == 1
}

constraint -> class::SubstrateServer {
    mappings.srv2srv
              ->filter(m | m.nodes().substrateServer == self)
                            ->sum(m | m.nodes().virtualNode.cpu)
                                          <= self.residualCpu
}

objective srvObj -> mapping::srv2srv {
    (self.nodes().substrateServer.residualCpu / self.nodes().substrateServer.cpu)
          * self.value()
}
```



Generated ILP Problem (Example)

```
\ LP format - for model browsing. Use MPS format to capture full model detail.
Minimize
  1 srv2srv#3 + 1 srv2srv#2 + 1 srv2srv#1 + 1 srv2srv#0 + $LINK_OBJECTIVES

Subject To
  $PATH_CONSTRAINTS $SWITCH_CONSTRAINTS $NETWORK_CONSTRAINTS
  PatternConstraint3OnvsrvNotMapped_0: srv2srv#2 + srv2srv#1 = 1
  PatternConstraint3OnvsrvNotMapped_1: srv2srv#3 + srv2srv#0 = 1
  TypeConstraint4OnSubstrateServer_0: 10 srv2srv#3 + 10 srv2srv#1 <= 10
  TypeConstraint4OnSubstrateServer_1: 10 srv2srv#2 + 10 srv2srv#0 <= 10

BOUNDS

Binaries
  srv2srv#3 srv2srv#2 srv2srv#1 srv2srv#0
  $SWITCH_VARS $LINK_VARS
End
```

```
objective srvObj -> mapping::srv2srv {
    (self.nodes().substrateServer.residualCpu / self.nodes().substrateServer.cpu)
        * self.value()
}
```

```
constraint -> pattern::vsrvNotMapped {
    mappings.srv2srv
        ->filter(m | m.nodes().virtualNode == self.nodes().virtualServer)
            ->sum(m | m.value()) == 1
}
```

```
constraint -> class::SubstrateServer {
    mappings.srv2srv
        ->filter(m | m.nodes().substrateServer == self)
            ->sum(m | m.nodes().virtualNode.cpu)
                <= self.residualCpu
}
```

Generated ILP Problem (Example)

```
\ LP format - for model browsing. Use MPS format to capture full model detail.
Minimize
  1 srv2srv#3 + 1 srv2srv#2 + 1 srv2srv#1 + 1 srv2srv#0 + $LINK_OBJECTIVES

Subject To
  $PATH_CONSTRAINTS $SWITCH_CONSTRAINTS $NETWORK_CONSTRAINTS
  PatternConstraint3OnvsrvNotMapped_0: srv2srv#2 + srv2srv#1 = 1
  PatternConstraint3OnvsrvNotMapped_1: srv2srv#3 + srv2srv#0 = 1
  TypeConstraint4OnSubstrateServer_0: 10 srv2srv#3 + 10 srv2srv#1 <= 10
  TypeConstraint4OnSubstrateServer_1: 10 srv2srv#2 + 10 srv2srv#0 <= 10

BOUNDS

Binaries
  srv2srv#3 srv2srv#2 srv2srv#1 srv2srv#0
  $SWITCH_VARS $LINK_VARS
End
```

Target function

Map virtual servers once

CPU resource constraints

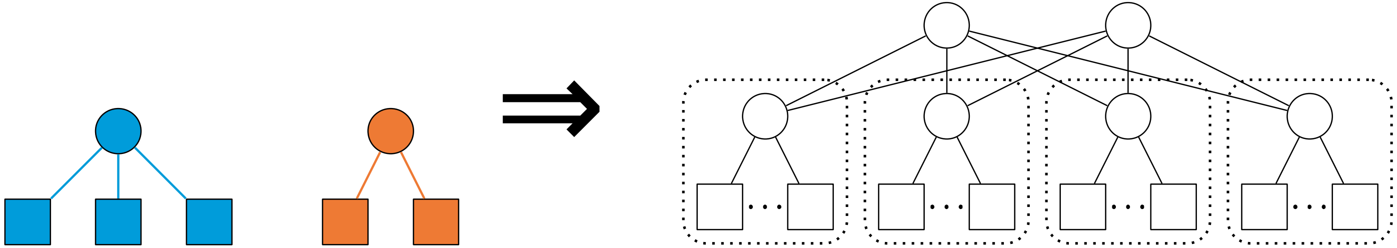
Mapping variables = binaries

Evaluation – Research Questions

1. How does our approach compare itself to "hand-crafted" approaches with respect to problem solving **runtime performance**?
2. How much **effort** (for example, *Lines Of Code* (LOC), *Number Of Characters* (NOC)) can really be saved by using our approach to generate an ILP generator compared to implementing one "by hand"?

Evaluation – Setup

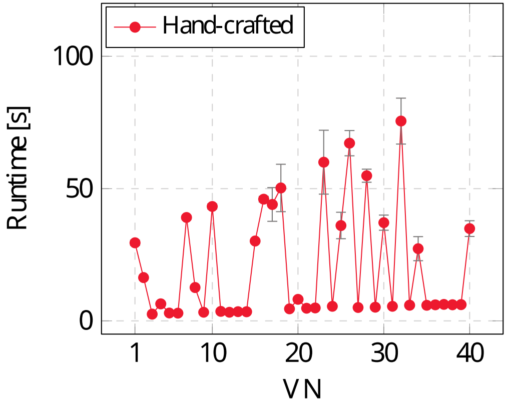
- Task: Virtual Network Embedding (VNE)
- 40 (pseudo-)random generated Virtual Networks (VNs)
  - Embedding takes place one after the other (no batch)
  - No migration
- Data sampled from an industry measurement[7]

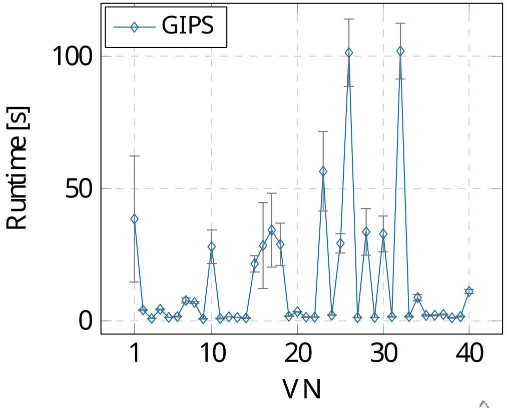


Evaluation – Results

- Both algorithms were able to embed the 40 VNs successfully
- Quality of the embeddings is approximately equal (w.r.t to the objective)
- Runtime GIPS: ~25% lower

Name	LOC	NOC
Manual	~2000	~91000
GIPS	56+29	3884+1251
Saving	~95%	~94%





Future Work – GIPSL

- Extension to support non-binary variables (full ILP + MILP + LP)
  - Idea: Real / integer variable support through parametrized rules
- Automated construction of constraints (from given metamodel, GT rules)
  - For example, as hint or auto completion
- Consideration of inter-rule dependencies
- Output of LP files without starting the solver (e.g., for debugging)
- Further evaluation & testing with different scenarios
  - E.g., test scheduling, peer-2-peer overlay networks, transformation tool contest

References

[1] Stefan Tomaszek, Roland Speith & Andy Schurr (2021): Virtual network embedding: ensuring correctness and optimality by construction using model transformation and integer linear programming techniques. Software and Systems Modeling, pp. 1299–1332, doi:10.1007/s10270-020-00852-z.

[2] eMoflon::IBeX-GT - https://emoflon.org/#emoflonIbex

[3] eMoflon::Neo - https://emoflon.org/#emoflonNeo

[4] Martin Fleck, Javier Troya & ManuelWimmer (2015): Marrying search-based optimization and model transformation technology. In: Proc. of the North American Symposium on Search Based Software Engineering, NasBASE ’15, pp. 1–16.

[5] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo & Alfonso Pierantonio (2011): JTL: A Bidirectional and Change Propagating Transformation Language. Software Language Engineering, pp.183-202, doi:10.1007/978-3-642-19440-5_11

[6] HiPE - https://github.com/HiPE-DevOps/HiPE-Updatesite

[7] Siqi Shen, Vincent Van Beek & Alexandru Iosup (2015): Statistical Characterization of Business-Critical Workloads Hosted in Cloud Datacenters. In: Proc. of the Int. Symposium on Cluster Computing and the Grid, CCGrid ’15, ACM, pp. 465–474, doi:10.1109/CCGrid.2015.60.