Virtual Network Embeddings:
Theoretical Foundations and Provably Good Algorithms

Matthias Rost
Technische Universität Berlin

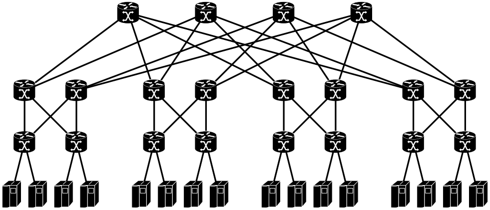
wissenschaftliche Aussprache, 25. März 2019

Virtual Networks

Virtual Networks

Cloud Applications

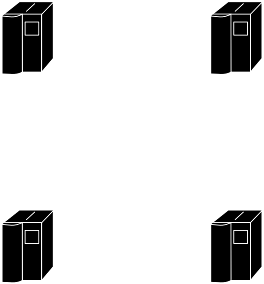
Physical Infrastructure: Data Center



Virtual Networks

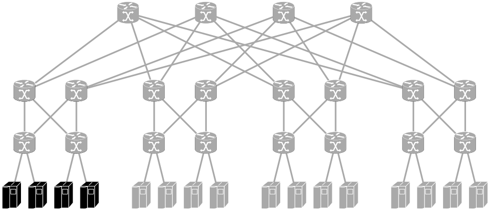
Virtual Machines

1 vCPU 4 vCPU



3 vCPU 1 vCPU

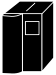
Physical Infrastructure: Data Center



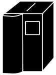
Virtual Networks

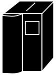
Virtual Machines

1 vCPU

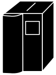


4 vCPU



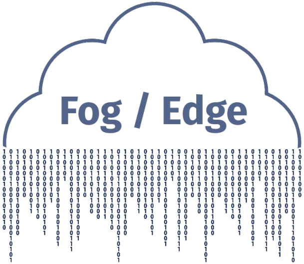


3 vCPU



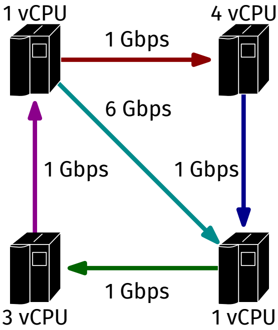
1 vCPU

Network-dependent Applications

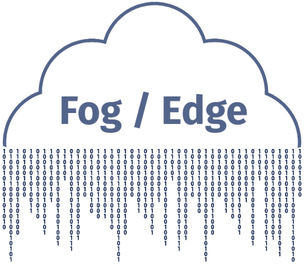


Virtual Networks

Virtual Network

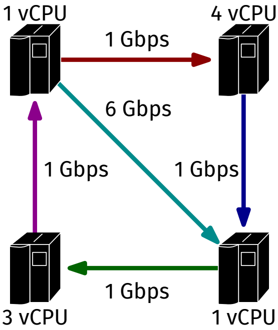


Network-dependent Applications



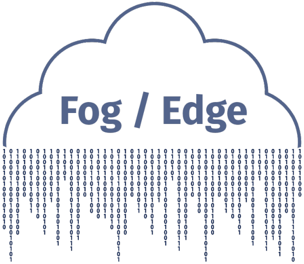
Virtual Networks

Virtual Network



Goal: Predictable Performance

Network-dependent Applications

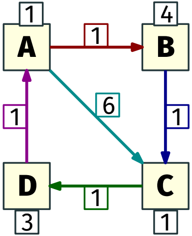


Virtual Network Embeddings

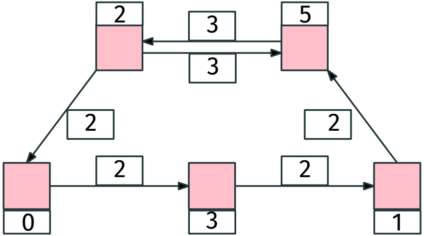
Virtual Network Embeddings

Request
Virtual Network

demand



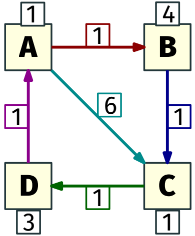
Substrate
Physical Network



capacity

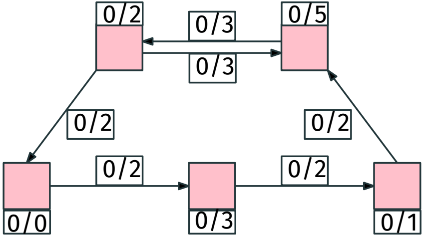
Virtual Network Embeddings

Request
Virtual Network



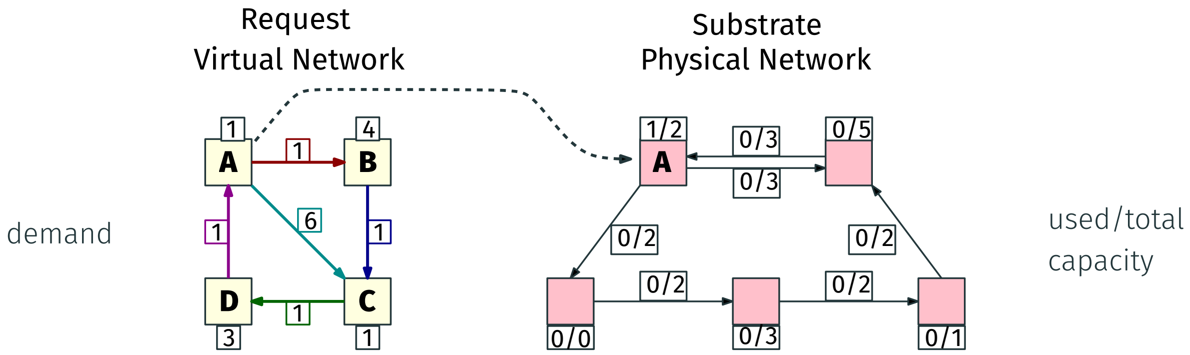
demand

Substrate
Physical Network



used/total
capacity

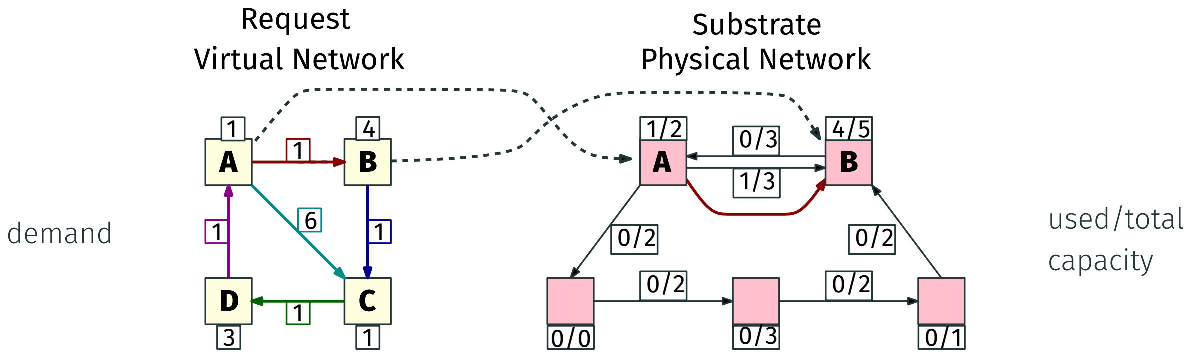
Virtual Network Embeddings



Mapping Properties

- virtual nodes are mapped to substrate nodes

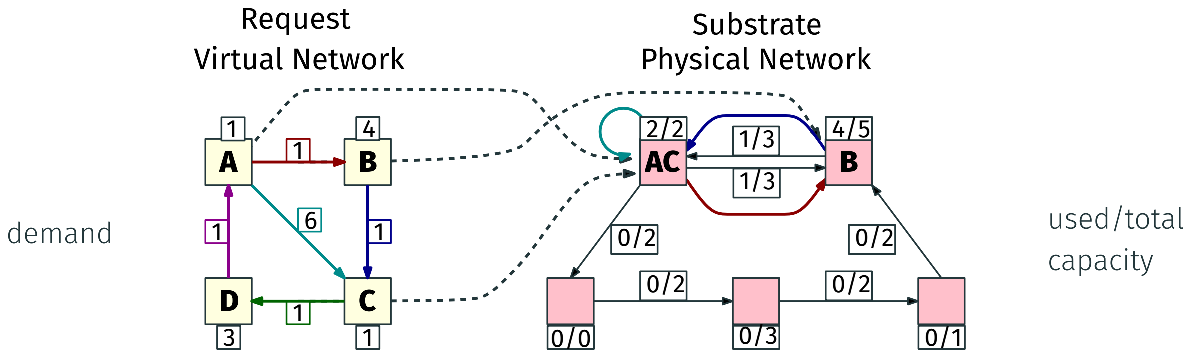
Virtual Network Embeddings



Mapping Properties

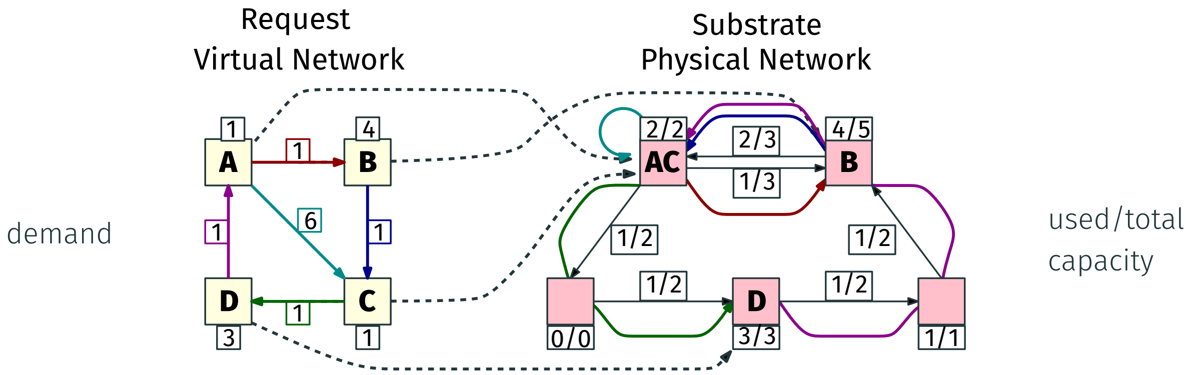
- virtual nodes are mapped to substrate nodes
- virtual edges are mapped to substrate paths

Virtual Network Embeddings



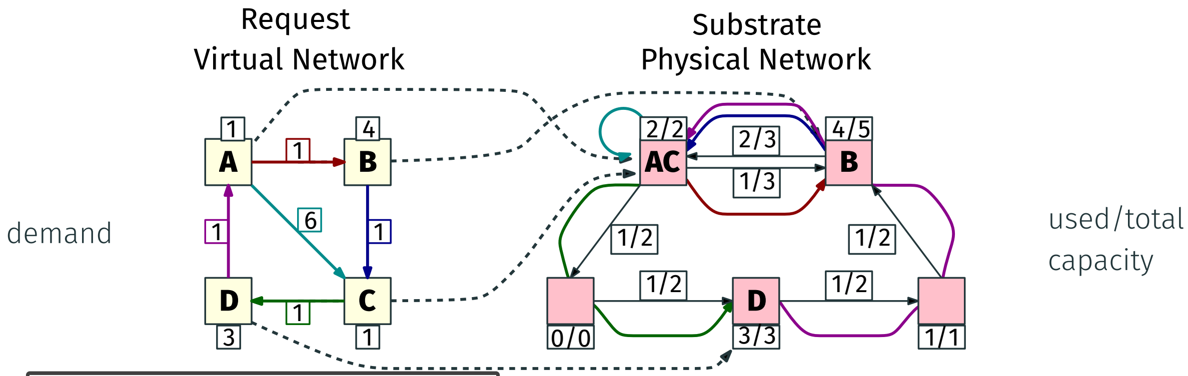
Mapping Properties

- virtual nodes are mapped to substrate nodes
- virtual edges are mapped to substrate paths



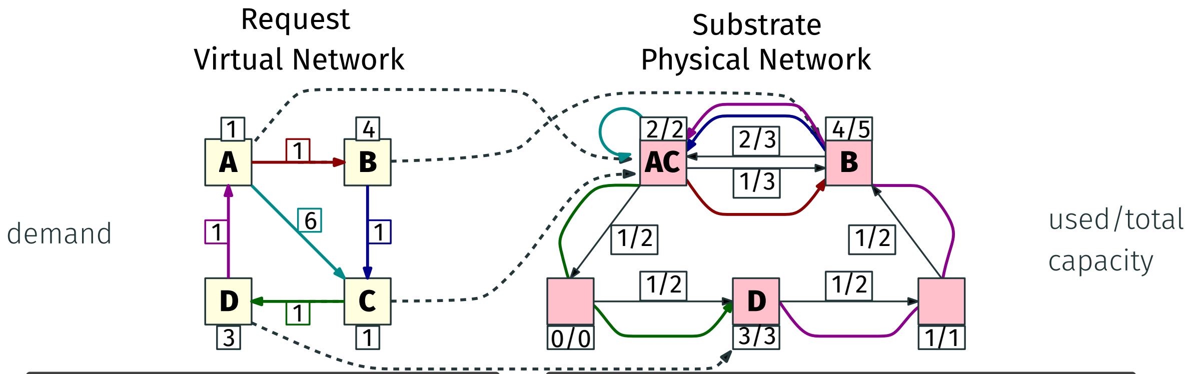
Mapping Properties
▸ virtual nodes are mapped to substrate nodes
▸ virtual edges are mapped to substrate paths

Virtual Network Embeddings



Feasibility
capacities are not exceeded

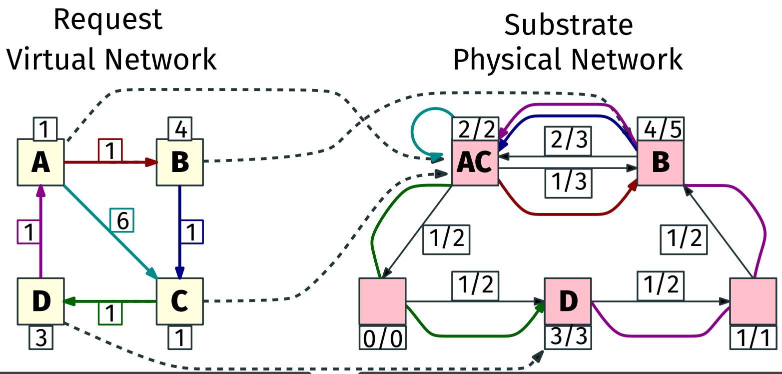
Virtual Network Embeddings



Feasibility
capacities are not exceeded

Use-Case Specific Validity
additional requirements may be imposed

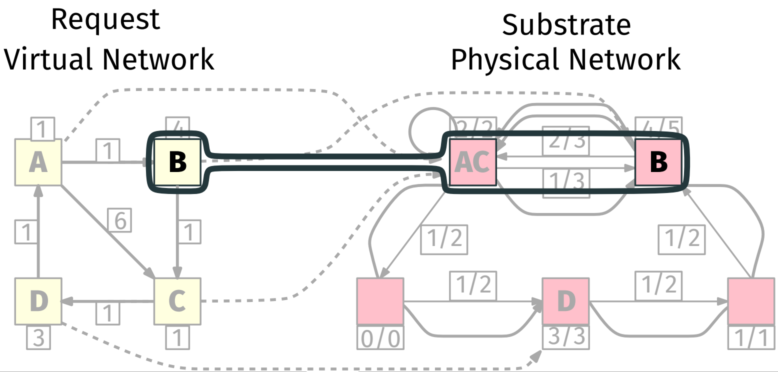
Virtual Network Embeddings



Feasibility
capacities are not exceeded

Use-Case Specific Validity
▶ Node Placement Restrictions ▶ Routing Restrictions ▶ Latency Restrictions

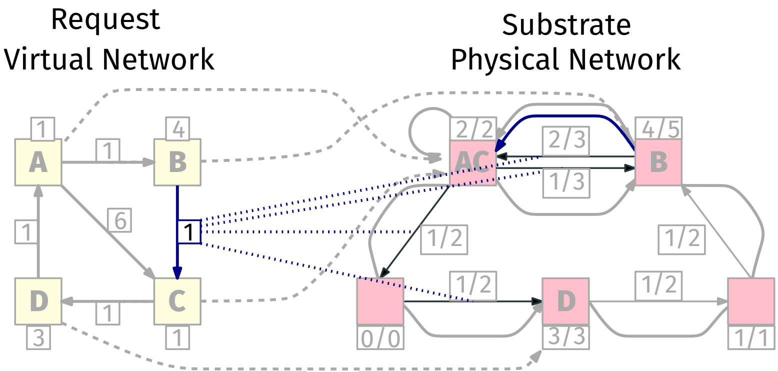
Virtual Network Embeddings



Feasibility
capacities are not exceeded

Use-Case Specific Validity
▶ Node Placement Restrictions ▶ Routing Restrictions ▶ Latency Restrictions

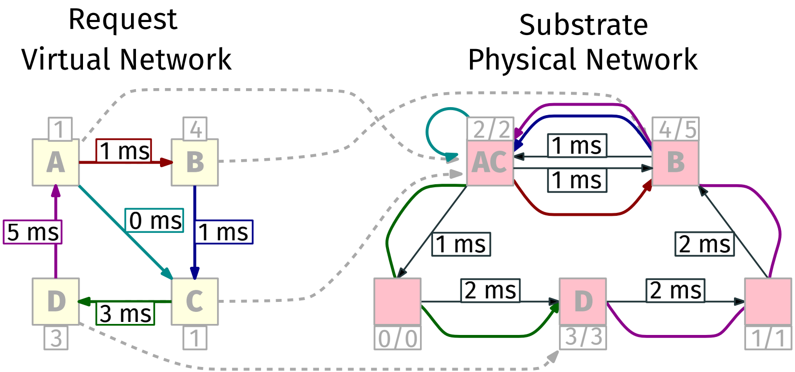
Virtual Network Embeddings



Feasibility
capacities are not exceeded

Use-Case Specific Validity
▸ Node Placement Restrictions ▸ Routing Restrictions ▸ Latency Restrictions

Virtual Network Embeddings



Feasibility
capacities are not exceeded

Use-Case Specific Validity
▶ Node Placement Restrictions ▶ Routing Restrictions ▶ Latency Restrictions

The Embedding Problem Zoo
& Related Work

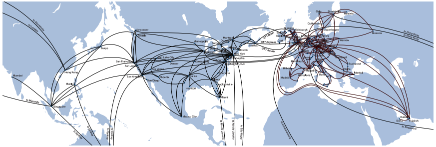
The Embedding Problem Zoo & Related Work

Classification

Restrictions

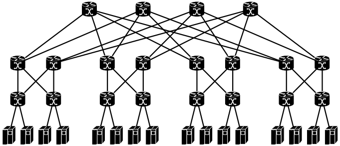
Network Type

Node Placement



Routing

Latencies



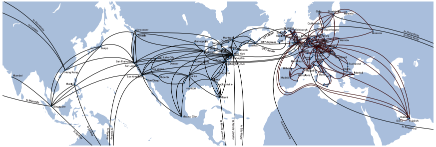
The Embedding Problem Zoo & Related Work

Classification

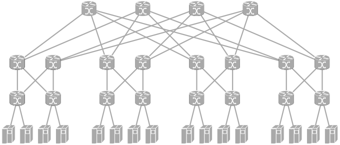
Restrictions

Network Type

Node Placement



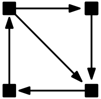
Routing



Latencies

Virtual Networks ≈ 2006

general request topologies



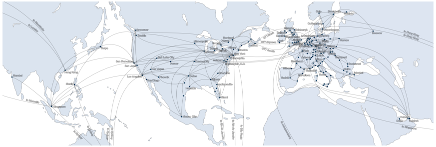
The Embedding Problem Zoo & Related Work

Classification

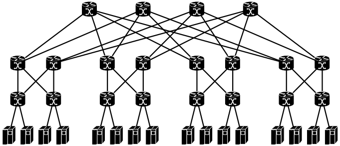
Restrictions

Network Type

Node Placement



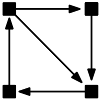
Routing



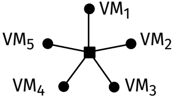
Latencies

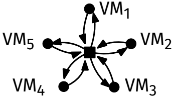
Virtual Networks ≈ 2006

general request topologies



Virtual Clusters ≈ 2011





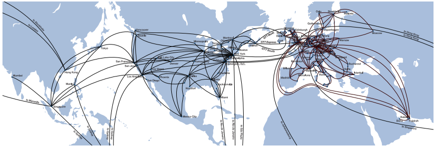
The Embedding Problem Zoo & Related Work

Classification

Restrictions

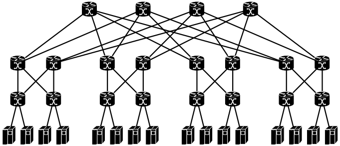
Network Type

Node Placement



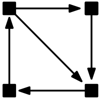
Routing

Latencies

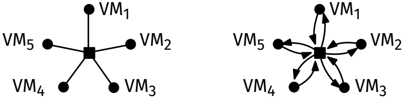


Virtual Networks ≈ 2006

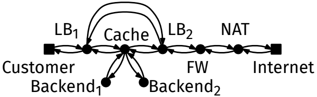
general request topologies



Virtual Clusters ≈ 2011



Service Chains ≈ 2012

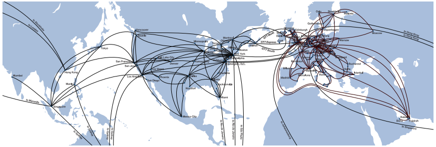


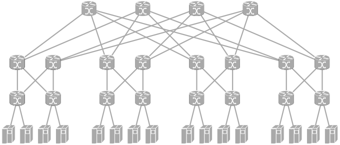
The Embedding Problem Zoo & Related Work

Classification

Restrictions | Node Placement | Routing | Latencies

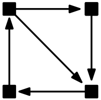
Network Type



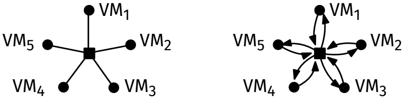


Virtual Networks ≈ 2006

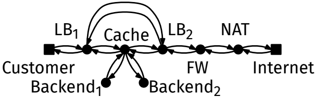
general request topologies



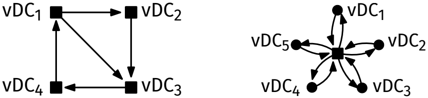
Virtual Clusters ≈ 2011



Service Chains ≈ 2012



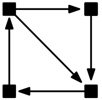
Virtual Data Centers ≈ 2013



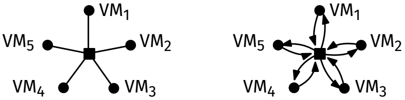
The Embedding Problem Zoo & Related Work

Virtual Networks ≈ 2006

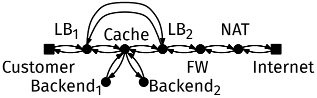
general request topologies



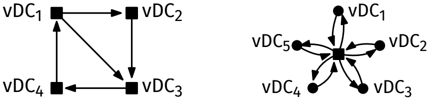
Virtual Clusters ≈ 2011



Service Chains ≈ 2012



Virtual Data Centers ≈ 2013

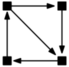


Finding ‘good’ embeddings is the **core algorithmic challenge**.

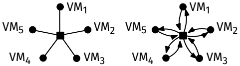
The Embedding Problem Zoo & Related Work

Virtual Networks ≈ 2006

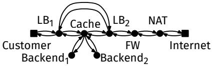
general topologies



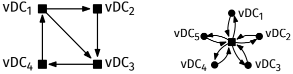
Virtual Clusters ≈ 2011



Service Chains ≈ 2012



Virtual Data Centers ≈ 2013



Finding 'good' embeddings is the **core algorithmic challenge**.

Objectives

Online Setting – Request Sequence

- min. resource usage / costs
- min. resource fragmentation

...

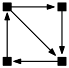
Offline Setting – Multiple Requests

- max. profit via admission control
- min. resource consumption

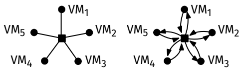
...

Virtual Networks ≈ 2006

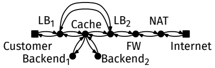
general topologies



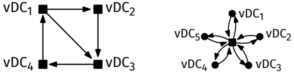
Virtual Clusters ≈ 2011



Service Chains ≈ 2012



Virtual Data Centers ≈ 2013



Finding 'good' embeddings is the **core algorithmic challenge**.

Algorithmic Solution Approaches

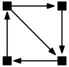
heuristics

exact algorithms

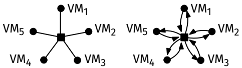
The Embedding Problem Zoo & Related Work

Virtual Networks ≈ 2006

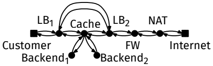
general topologies



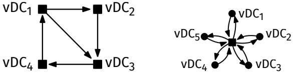
Virtual Clusters ≈ 2011



Service Chains ≈ 2012



Virtual Data Centers ≈ 2013



Finding 'good' embeddings is the **core algorithmic challenge**.

Algorithmic Solution Approaches

quality guarantees

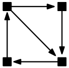
none — optimality

heuristics — exact algorithms

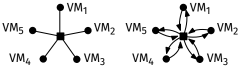
The Embedding Problem Zoo & Related Work

Virtual Networks ≈ 2006

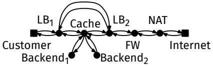
general topologies



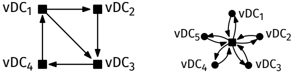
Virtual Clusters ≈ 2011



Service Chains ≈ 2012



Virtual Data Centers ≈ 2013



Finding ‘good’ embeddings is the **core algorithmic challenge**.

Algorithmic Solution Approaches

quality guarantees

none — **optimality**

heuristics — **exact algorithms**

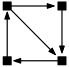
polynomial — exponential

runtime

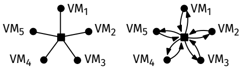
The Embedding Problem Zoo & Related Work

Virtual Networks ≈ 2006

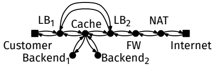
general topologies



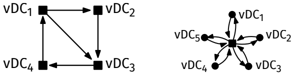
Virtual Clusters ≈ 2011



Service Chains ≈ 2012



Virtual Data Centers ≈ 2013



Finding 'good' embeddings is the **core algorithmic challenge**.

Algorithmic Solution Approaches

quality guarantees

none — optimality

heuristics — **approximations** — exact algorithms

polynomial — exponential

runtime

Finding ‘good’ embeddings is the **core algorithmic challenge**.

Algorithmic Solution Approaches

quality guarantees

none — optimality

heuristics **approximations** **exact algorithms**

polynomial — exponential

runtime

quality guarantees + bounded runtime ⇝ predictable algorithm performance

The Embedding Problem Zoo & Related Work

Finding ‘good’ embeddings is the **core algorithmic challenge**.

Algorithmic Solution Approaches

quality guarantees

none — optimality

heuristics | **approximations** | exact algorithms

polynomial — exponential

runtime

quality guarantees + bounded runtime $\rightsquigarrow$ predictable algorithm performance

Until now: only heuristics and exact algorithms known.

Thesis Overview

Thesis Overview

Prime Goals

1. Development of *efficient* approximation algorithms.

Virtual Network Embedding Problem

Thesis Overview

Prime Goals

1. Development of *efficient* approximation algorithms.

Virtual Network Embedding Problem

Computational Complexity

- Study structural hardness and *inapproximability*.

Thesis Overview

Prime Goals

1. Development of *efficient* approximation algorithms.

Virtual Network Embedding Problem

Computational Complexity

- Study structural hardness and *inapproximability*.

· IFIP Networking '18 (best paper)
· IEEE/ACM TON (major rev.)

Thesis Overview

Prime Goals

1. Development of *efficient* approximation algorithms.

Virtual Network Embedding Problem

Computational Complexity

- Study structural hardness and *inapproximability*.

• IFIP Networking '18 (best paper)
• IEEE/ACM TON (major rev.)

Problem Relaxations

- Find appropriate *tractable model* relaxations.

Thesis Overview

Prime Goals

1. Development of *efficient* approximation algorithms.

Virtual Network Embedding Problem

Computational Complexity

- Study structural hardness and *inapproximability*.

· IFIP Networking '18 (best paper)
· IEEE/ACM TON (major rev.)

Problem Relaxations

- Find appropriate *tractable model* relaxations.

Offline Approximations

- Use relaxations to obtain ***first*** approximations.

Thesis Overview

Prime Goals

1. Development of *efficient* approximation algorithms.
2. Bridge gap between theory and practice: *efficient* heuristics.

Virtual Network Embedding Problem

Computational Complexity

- Study structural hardness and *inapproximability*.

- IFIP Networking '18 (best paper)
- IEEE/ACM TON (major rev.)

Problem Relaxations

- Find appropriate *tractable model* relaxations.

Offline Approximations

- Use relaxations to obtain ***first*** approximations.

Heuristics for Offline Profit VNEP and Evaluation

- Derive heuristics and exploratively evaluate performance.

Thesis Overview

Prime Goals

1. Development of *efficient* approximation algorithms.
2. Bridge gap between theory and practice: *efficient* heuristics.

Virtual Network Embedding Problem

Computational Complexity

- Study structural hardness and *inapproximability*.

· IFIP Networking '18 (best paper)
· IEEE/ACM TON (major rev.)

Problem Relaxations

- Find appropriate *tractable model* relaxations.

Offline Approximations

- Use relaxations to obtain ***first*** approximations.

· IFIP Networking '18 · arXiv '18 · IEEE/ACM TON (minor rev.) · ACM CCR '19

Heuristics for Offline Profit VNEP and Evaluation

- Derive heuristics and exploratively evaluate performance.

Thesis Overview

Prime Goals

1. Development of *efficient* approximation algorithms.
2. Bridge gap between theory and practice: *efficient* heuristics.

Virtual Network Embedding Problem

Computational Complexity

- Study structural hardness and *inapproximability*.

· IFIP Networking '18 (best paper)
· IEEE/ACM TON (major rev.)

Problem Relaxations

- Find appropriate *tractable model* relaxations.

Offline Approximations

- Use relaxations to obtain ***first*** approximations.

· IFIP Networking '18 · arXiv '18 · IEEE/ACM TON (minor rev.) · ACM CCR '19

Heuristics for Offline Profit VNEP and Evaluation

- Derive heuristics and exploratively evaluate performance.

Specific Embeddings & Embedding Models

· ACM CCR '15

Virtual Clusters

- Study problem and optimize resource usage.

· IEEE IPDPS '14

Temporal VNEP

- Identify ways to harness temporal flexibilities.

Computational Complexity of the VNEP

Reminder: 3-SAT and $\mathcal{NP}$-Completeness

3-SAT-Formula ϕ

$\phi = \bigwedge_{\mathcal{C}_i \in \mathcal{C}_\phi} \mathcal{C}_i$ with $\mathcal{C}_i \in \mathcal{C}_\phi$ being disjunctions of at most 3 literals.

Example 3-SAT formula ϕ over literals $\mathcal{L}_\phi = \{x_1, x_2, x_3, x_4\}$

$$\phi = \underbrace{(x_1 \vee x_2 \vee x_3)}_{\mathcal{C}_1} \wedge \underbrace{(\bar{x}_1 \vee x_2 \vee x_4)}_{\mathcal{C}_2} \wedge \underbrace{(x_2 \vee \bar{x}_3 \vee x_4)}_{\mathcal{C}_3}$$

Definition of 3-SAT

Decide whether formula ϕ can be satisfied.

Decision Problems

Output is ‘yes’ or ‘no’.

Theorem: Karp [1972]

3-SAT is $\mathcal{NP}$-complete.

3-SAT-Formula ϕ

$\phi = \bigwedge_{\mathcal{C}_i \in \mathcal{C}_\phi} \mathcal{C}_i$ with $\mathcal{C}_i \in \mathcal{C}_\phi$ being disjunctions of at most 3 literals.

Example 3-SAT formula ϕ over literals $\mathcal{L}_\phi = \{x_1, x_2, x_3, x_4\}$

$$\phi = \underbrace{(x_1 \vee x_2 \vee x_3)}_{\mathcal{C}_1} \wedge \underbrace{(\bar{x}_1 \vee x_2 \vee x_4)}_{\mathcal{C}_2} \wedge \underbrace{(x_2 \vee \bar{x}_3 \vee x_4)}_{\mathcal{C}_3}$$

Definition of 3-SAT

Decide whether formula ϕ can be satisfied.

Decision Problems

Output is ‘yes’ or ‘no’.

Theorem: Karp [1972]

3-SAT is $\mathcal{NP}$-complete.

Reminder: 3-SAT and $\mathcal{NP}$-Completeness

3-SAT-Formula ϕ

$\phi = \bigwedge_{\mathcal{C}_i \in \mathcal{C}_\phi} \mathcal{C}_i$ with $\mathcal{C}_i \in \mathcal{C}_\phi$ being disjunctions of at most 3 literals.

Example 3-SAT formula ϕ over literals $\mathcal{L}_\phi = \{x_1, x_2, x_3, x_4\}$

$$\phi = \underbrace{(x_1 \vee x_2 \vee x_3)}_{\mathcal{C}_1} \wedge \underbrace{(\bar{x}_1 \vee x_2 \vee x_4)}_{\mathcal{C}_2} \wedge \underbrace{(x_2 \vee \bar{x}_3 \vee x_4)}_{\mathcal{C}_3}$$

Definition of 3-SAT

Decide whether formula ϕ can be satisfied.

Decision Problems

Output is 'yes' or 'no'.

Theorem: Karp [1972]

3-SAT is $\mathcal{NP}$-complete.

Proving $\mathcal{NP}$-completeness of the VNEP

1. Show that VNEP lies in $\mathcal{NP}$ ($\checkmark$).
2. Reduce 3-SAT to VNEP.

Outline of Reduction Framework

3-SAT instance $\phi \longmapsto$ VNEP instance $(G_{r(\phi)}, G_{S(\phi)},$ restrictions$)$

ϕ satisfiable? $\rightleftarrows$ feasible embedding of $G_{r(\phi)}$ on $G_{S(\phi)}$ under restrictions?

Our Reduction Framework

Proving $\mathcal{NP}$-completeness of the VNEP

1. Show that VNEP lies in $\mathcal{NP}$ ($\checkmark$).
2. Reduce 3-SAT to VNEP.

Outline of Reduction Framework

3-SAT instance $\phi \longmapsto$ VNEP instance $(G_{r(\phi)}, G_{S(\phi)},$ restrictions$)$

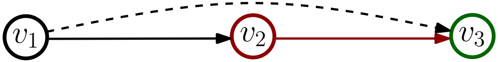
ϕ satisfiable? $\rightleftarrows$ feasible embedding of $G_{r(\phi)}$ on $G_{S(\phi)}$ under restrictions?

Our Reduction Framework

Input: 3-SAT formula $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4) \wedge (x_2 \vee \bar{x}_3 \vee x_4)$

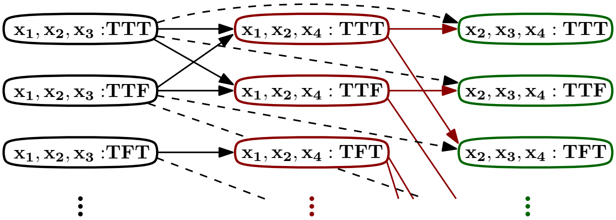
Request $G_{r(\phi)}$

- one virtual node per clause
- edge (v_i, v_j) when $\mathcal{C}_i$ introduces literal used by $\mathcal{C}_j$ for $i < j$

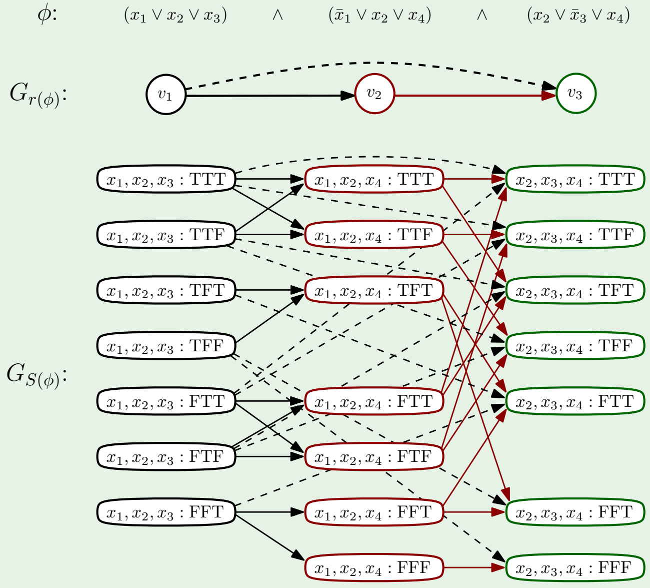


Substrate $G_{S(\phi)}$

- 7 substrate nodes per clause: **represent satisfying assignments**
- edges as for request, **only when assignments agree**



Complete Picture



Our Reduction Framework

Outline of Reduction Framework

3-SAT instance $\phi \longmapsto$ VNEP instance $(G_{r(\phi)}, G_{S(\phi)},$ restrictions$)$

ϕ satisfiable? $\Longleftrightarrow$ feasible embedding of $G_{r(\phi)}$ on $G_{S(\phi)}$ under restrictions?

Our Reduction Framework

Outline of Reduction Framework

3-SAT instance $\phi \longmapsto$ VNEP instance $(G_{r(\phi)}, G_{S(\phi)},$ restrictions$)$

ϕ satisfiable? $\rightleftarrows$ feasible embedding of $G_{r(\phi)}$ on $G_{S(\phi)}$ under restrictions?

Base Lemma

Formula ϕ is satisfiable **if and only if** there exists a mapping of $G_{r(\phi)}$ on $G_{S(\phi)}$, s.t.

(1) each virtual node v_i is mapped to a (satisfying) substrate node of the i-th clause, and

(2) all virtual edges are mapped on exactly one substrate edge.

Computational Complexity Results

Base Lemma

Formula ϕ is satisfiable **if and only if** there exists a mapping of $G_{r(\phi)}$ on $G_{S(\phi)}$, s.t.

(1) each virtual node v_i is mapped to a (satisfying) substrate node of the i-th clause, and

(2) all virtual edges are mapped on exactly one substrate edge.

Theorem: Decision VNEP is $\mathcal{NP}$-complete under *node placement and routing restrictions*

Proof: via application of base lemma.

Base Lemma

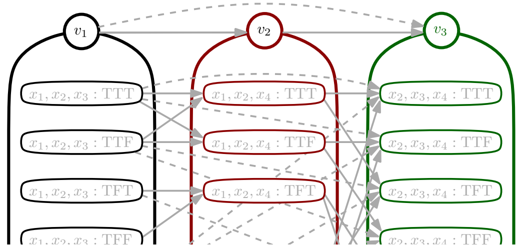
Formula ϕ is satisfiable **if and only if** there exists a mapping of $G_{r(\phi)}$ on $G_{S(\phi)}$, s.t.

(1) each virtual node v_i is mapped to a (satisfying) substrate node of the i-th clause, and

(2) all virtual edges are mapped on exactly one substrate edge.

Theorem: Decision VNEP is $\mathcal{NP}$-complete under *node placement and routing restrictions*

Node placement restrictions enforce (1)



Base Lemma

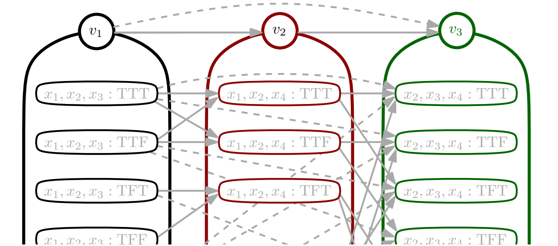
Formula ϕ is satisfiable **if and only if** there exists a mapping of $G_{r(\phi)}$ on $G_{S(\phi)}$, s.t.

(1) each virtual node v_i is mapped to a (satisfying) substrate node of the i-th clause, and

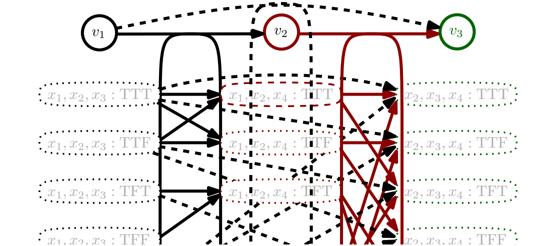
(2) all virtual edges are mapped on exactly one substrate edge.

Theorem: Decision VNEP is $\mathcal{NP}$-complete under *node placement and routing restrictions*

Node placement restrictions enforce (1)



Routing restrictions enforce (2)



Computational Complexity Results

Base Lemma

Formula ϕ is satisfiable **if and only if** there exists a mapping of $G_{r(\phi)}$ on $G_{S(\phi)}$, s.t.

(1) each virtual node v_i is mapped to a (satisfying) substrate node of the i-th clause, and

(2) all virtual edges are mapped on exactly one substrate edge.

Theorem: Decision VNEP is $\mathcal{NP}$-complete under *node placement and routing restrictions*

What about other restrictions?

Base Lemma

Formula ϕ is satisfiable **if and only if** there exists a mapping of $G_{r(\phi)}$ on $G_{S(\phi)}$, s.t.

(1) each virtual node v_i is mapped to a (satisfying) substrate node of the i-th clause, and

(2) all virtual edges are mapped on exactly one substrate edge.

Theorem: VNEP is $\mathcal{NP}$-complete under ...

Node Restrictions		Edge Restrictions
capacities	&	capacities
capacities	&	routing
placement	&	capacities
placement	&	routing
placement	&	latencies

Computational Complexity Results

Base Lemma

Formula ϕ is satisfiable **if and only if** there exists a mapping of $G_{r(\phi)}$ on $G_{S(\phi)}$, s.t.

(1) each virtual node v_i is mapped to a (satisfying) substrate node of the i-th clause, and

(2) all virtual edges are mapped on exactly one substrate edge.

Theorem: VNEP is $\mathcal{NP}$-complete under ...

Node Restrictions		Edge Restrictions
capacities	&	capacities
capacities	&	routing
placement	&	capacities
placement	&	routing
placement	&	latencies

Theorem: $\mathcal{NP}$-Completeness remains if ...

substrate is acyclic ***and***
request is acyclic ***and***
request is planar ***and***
request has max degree 12.

Computational Complexity Results

Theorem: VNEP is $\mathcal{NP}$-complete under ...

Node Restrictions		Edge Restrictions
capacities	&	capacities
capacities	&	routing
placement	&	capacities
placement	&	routing
placement	&	latencies

Theorem: $\mathcal{NP}$-Completeness remains if ...

substrate is acyclic ***and***
request is acyclic ***and***
request is planar ***and***
request has max degree 12.

Practical Implications?

Computational Complexity Results

Theorem: VNEP is $\mathcal{NP}$-complete under ...

Node Restrictions		Edge Restrictions
capacities	&	capacities
capacities	&	routing
placement	&	capacities
placement	&	routing
placement	&	latencies

Theorem: $\mathcal{NP}$-Completeness remains if ...

substrate is acyclic ***and***
request is acyclic ***and***
request is planar ***and***
request has max degree 12.

Practical Implications ...

- Finding a feasible embedding is in general not possible in polynomial-time[a].
- The VNEP is *inapproximable* under any objective even for a single request[a].
- Computing valid mappings is already hard!

[a] unless $\mathcal{P} = \mathcal{NP}$ holds

Computational Complexity Results

Theorem: VNEP is $\mathcal{NP}$-complete under ...

Node Restrictions		Edge Restrictions
capacities	&	capacities
capacities	&	routing
placement	&	capacities
placement	&	routing
placement	&	latencies

Theorem: $\mathcal{NP}$-Completeness remains if ...

substrate is acyclic ***and***
request is acyclic ***and***
request is planar ***and***
request has max degree 12.

Practical Implications ...

- Finding a feasible embedding is in general not possible in polynomial-time[a].
- The VNEP is *inapproximable* under any objective even for a single request[a].
- Computing valid mappings is already hard!

[a] unless $\mathcal{P} = \mathcal{NP}$ holds

Computational Complexity Results

Theorem: VNEP is $\mathcal{NP}$-complete under ...

Node Restrictions		Edge Restrictions
capacities	&	capacities
capacities	&	routing
placement	&	capacities
placement	&	routing
placement	&	latencies

Theorem: $\mathcal{NP}$-Completeness remains if ...

substrate is acyclic *and*
request is acyclic *and*
request is planar *and*
request has max degree 12.

Practical Implications ...

- Finding a feasible embedding is in general not possible in polynomial-time[a].
- The VNEP is *inapproximable* under any objective even for a single request[a].
- Computing valid mappings is already hard!

[a] unless $\mathcal{P} = \mathcal{NP}$ holds

Computational Complexity Results

- Finding a feasible embedding is in general not possible in polynomial-time[a].
- The VNEP is *inapproximable* under any objective even for a single request[a].
- Computing valid mappings is already hard!

[a] unless $\mathcal{P} = \mathcal{NP}$ holds

Additional Inapproximability Results

- Intractability even for *approximate* solutions when relaxing either
  - node capacities or latencies by factor $2 - \varepsilon$, or
  - edge capacities by factor $\log n / \log\log n$, with $n =$ number of substrate nodes [a].

[a] unless $\mathcal{NP} \subseteq \mathcal{BP}$-*TIME*$(\bigcup_{d \geq 1} n^{d \log\log n})$

Problem Relaxations

Relaxation: Valid Mappings

VNEP is $\mathcal{NP}$-complete under ...		
Node Restrictions		Edge Restrictions
$\vdots$	&	$\vdots$
placement	&	routing
placement	&	latencies

Relaxation: Valid Mappings

VNEP is $\mathcal{NP}$-complete under ...		
Node Restrictions		Edge Restrictions
$\vdots$	&	$\vdots$
placement	&	routing
placement	&	latencies

Validity restrictions are non-negotiable.

Relaxation: Valid Mappings

VNEP is $\mathcal{NP}$-complete under ...		
Node Restrictions		Edge Restrictions
placement	&	routing

Focus

Computing valid mappings under node placement and routing restrictions.

VNEP is $\mathcal{NP}$-complete under ...

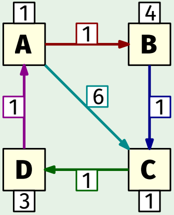
Node Restrictions & Edge Restrictions

placement & routing

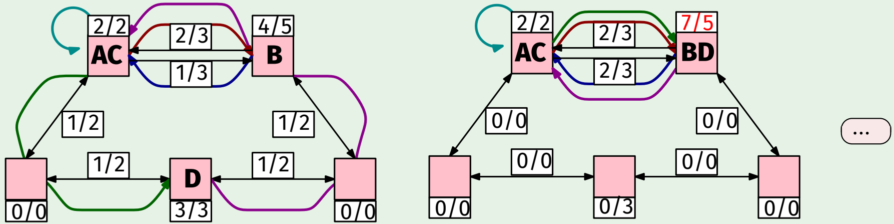
Focus

Computing valid mappings under node placement and routing restrictions.

Request r



Valid Mappings $\mathcal{M}_r = \{m_r^1, m_r^2, m_r^3, \ldots\}$



Valid mappings do not necessarily respect capacity constraints!

Relaxation: Valid Mappings

Valid Mapping Problem (VMP)

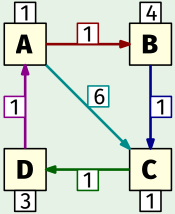
Find valid mapping $m_r \in \mathcal{M}_r$ of least cost:

$$c_S(m_r) = \sum_{x \in G_S} c_S(x) \cdot A(m_r, x)$$

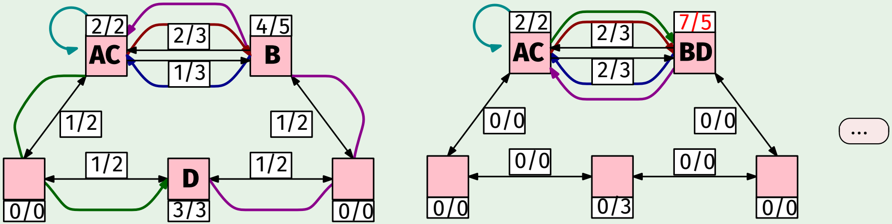
Focus

Computing valid mappings under node placement and routing restrictions.

Request r



Valid Mappings $\mathcal{M}_r = \{m_r^1, m_r^2, m_r^3, \ldots\}$

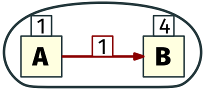


Valid mappings do not necessarily respect capacity constraints!

Solving the Valid Mapping Problem: DYNVMP Intuition

DYNVMP algorithm: solve VMP via *dynamic programming*.

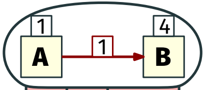
DynVMP algorithm: solve VMP via *dynamic programming*.



Start with simplest request: single edge.

Solving the Valid Mapping Problem: DynVMP Intuition

DynVMP algorithm: solve VMP via *dynamic programming*.

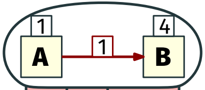


A	B	cost
u	u	
u	v	
v	u	
v	v	

Key Idea:
Enumerate node mappings inside bag.

Solving the Valid Mapping Problem: DynVMP Intuition

DynVMP algorithm: solve VMP via *dynamic programming*.



A	B	cost
u	u	
u	v	
v	u	
v	v	

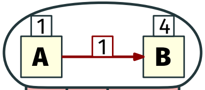
Key Idea:
Enumerate node mappings inside bag.

Validity

- nodes: trivial
- edges: graph-search

Solving the Valid Mapping Problem: DynVMP Intuition

DynVMP algorithm: solve VMP via *dynamic programming*.



A	B	cost
u	u	
u	v	
v	u	
v	v	

Key Idea:
Enumerate node mappings inside bag.

Validity

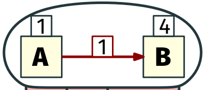
- nodes: trivial
- edges: graph-search

Mapping Costs

- nodes: trivial
- edges: shortest-paths (using allowed edges)

Solving the Valid Mapping Problem: DynVMP Intuition

DynVMP algorithm: solve VMP via *dynamic programming*.



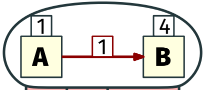
A	B	cost
u	u	1
u	v	
v	u	
v	v	

Key Idea:
Enumerate node mappings inside bag.

Validity	Mapping Costs
▸ nodes: trivial	▸ nodes: trivial
▸ edges: graph-search	▸ edges: shortest-paths (using allowed edges)

Solving the Valid Mapping Problem: DynVMP Intuition

DynVMP algorithm: solve VMP via *dynamic programming*.



A	B	cost
u	u	1
u	v	2
v	u	
v	v	

Key Idea:
Enumerate node mappings inside bag.

Validity

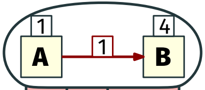
- nodes: trivial
- edges: graph-search

Mapping Costs

- nodes: trivial
- edges: shortest-paths (using allowed edges)

Solving the Valid Mapping Problem: DynVMP Intuition

DynVMP algorithm: solve VMP via *dynamic programming*.



A	B	cost
u	u	1
u	v	2
v	u	∞
v	v	

Key Idea:
Enumerate node mappings inside bag.

Validity	Mapping Costs
▸ nodes: trivial	▸ nodes: trivial
▸ edges: graph-search	▸ edges: shortest-paths (using allowed edges)

Solving the Valid Mapping Problem: DynVMP Intuition

DynVMP algorithm: solve VMP via *dynamic programming*.

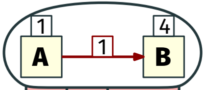
Key Idea:
Enumerate node mappings inside bag.

Validity

- nodes: trivial
- edges: graph-search

Mapping Costs

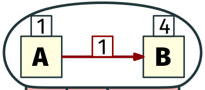
- nodes: trivial
- edges: shortest-paths (using allowed edges)



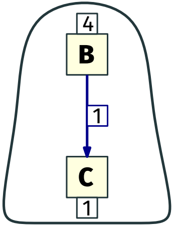
A	B	cost
u	u	1
u	v	2
v	u	∞
v	v	5

Solving the Valid Mapping Problem: DynVMP Intuition

DynVMP algorithm: solve VMP via *dynamic programming*.



A	B	cost
u	u	1
u	v	2
v	u	∞
v	v	5



Key Idea:
Enumerate node mappings inside bag.

Validity

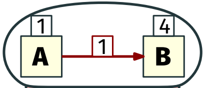
- nodes: trivial
- edges: graph-search

Mapping Costs

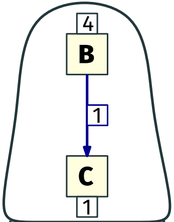
- nodes: trivial
- edges: shortest-paths (using allowed edges)

Solving the Valid Mapping Problem: DynVMP Intuition

DynVMP algorithm: solve VMP via *dynamic programming*.



A	B	cost
u	u	1
u	v	2
v	u	∞
v	v	5



B	C	cost
u	u	∞
u	v	∞
v	u	3
v	v	1

Key Idea:
Enumerate node mappings inside bag.

Validity

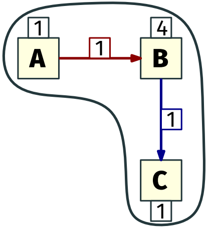
- nodes: trivial
- edges: graph-search

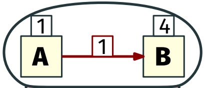
Mapping Costs

- nodes: trivial
- edges: shortest-paths (using allowed edges)

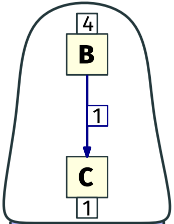
Solving the Valid Mapping Problem: DYNVMP Intuition

DYNVMP algorithm: solve VMP via *dynamic programming*.





A	B	cost
u	u	1
u	v	2
v	u	∞
v	v	5



B	C	cost
u	u	∞
u	v	∞
v	u	3
v	v	1

Key Idea:
Enumerate node mappings inside bag.

Validity

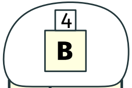
- nodes: trivial
- edges: graph-search

Mapping Costs

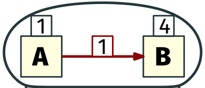
- nodes: trivial
- edges: shortest-paths (using allowed edges)

Solving the Valid Mapping Problem: DynVMP Intuition

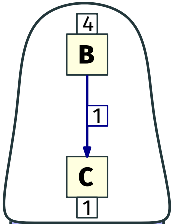
DynVMP algorithm: solve VMP via *dynamic programming*.



B	cost



A	B	cost
u	u	1
u	v	2
v	u	∞
v	v	5



B	C	cost
u	u	∞
u	v	∞
v	u	3
v	v	1

Key Idea:
Enumerate node mappings inside bag.

Validity

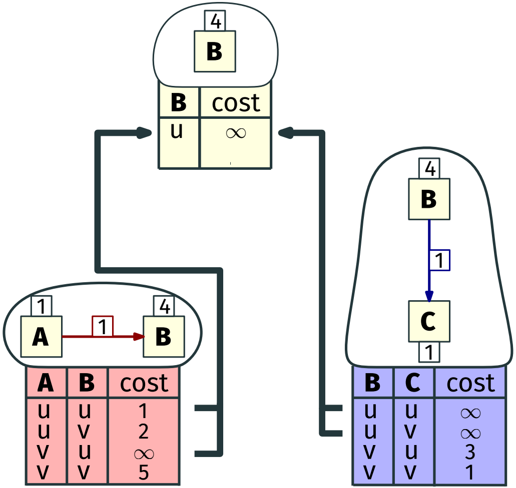
- nodes: trivial
- edges: graph-search

Mapping Costs

- nodes: trivial
- edges: shortest-paths (using allowed edges)

Solving the Valid Mapping Problem: DYNVMP Intuition

DYNVMP algorithm: solve VMP via *dynamic programming*.



B	cost
u	∞

A	B	cost
u	u	1
u	v	2
v	u	∞
v	v	5

B	C	cost
u	u	∞
u	v	∞
v	u	3
v	v	1

Key Idea:
Enumerate node mappings inside bag.

Validity

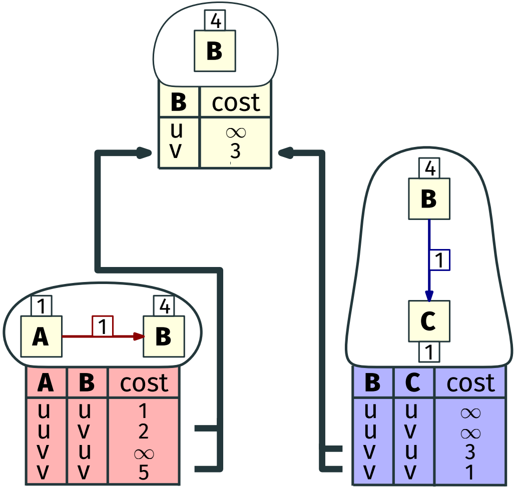
- nodes: trivial
- edges: graph-search

Mapping Costs

- nodes: trivial
- edges: shortest-paths (using allowed edges)

Solving the Valid Mapping Problem: DynVMP Intuition

DynVMP algorithm: solve VMP via *dynamic programming*.



B	cost
u	∞
v	3

A	B	cost
u	u	1
u	v	2
v	u	∞
v	v	5

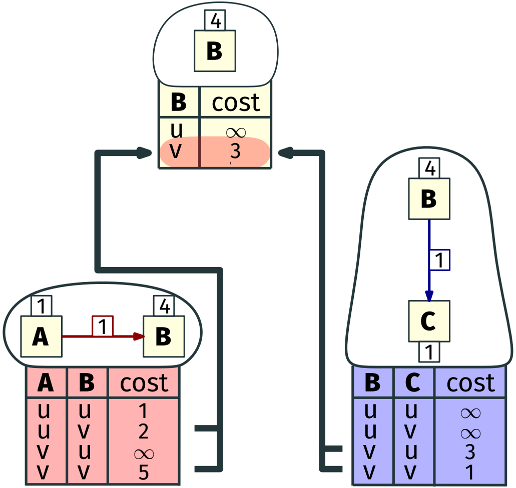
B	C	cost
u	u	∞
u	v	∞
v	u	3
v	v	1

Key Idea:
Enumerate node mappings inside bag.

Validity	Mapping Costs
▸ nodes: trivial	▸ nodes: trivial
▸ edges: graph-search	▸ edges: shortest-paths (using allowed edges)

Solving the Valid Mapping Problem: DynVMP Intuition

DynVMP algorithm: solve VMP via *dynamic programming*.

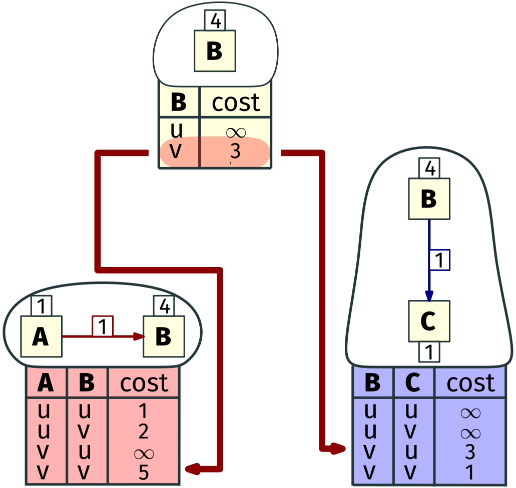


Key Idea:
Enumerate node mappings inside bag.

Validity	Mapping Costs
▸ nodes: trivial	▸ nodes: trivial
▸ edges: graph-search	▸ edges: shortest-paths (using allowed edges)

Solving the Valid Mapping Problem: DynVMP Intuition

DynVMP algorithm: solve VMP via *dynamic programming*.

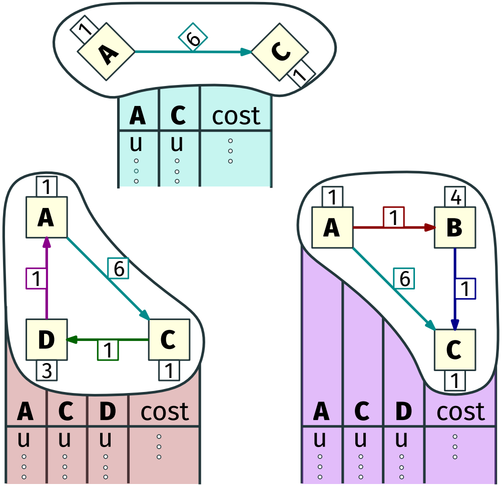


Key Idea:
Enumerate node mappings inside bag.

Validity	Mapping Costs
▸ nodes: trivial	▸ nodes: trivial
▸ edges: graph-search	▸ edges: shortest-paths (using allowed edges)

Solving the Valid Mapping Problem: DynVMP Intuition

DynVMP algorithm: solve VMP via *dynamic programming*.

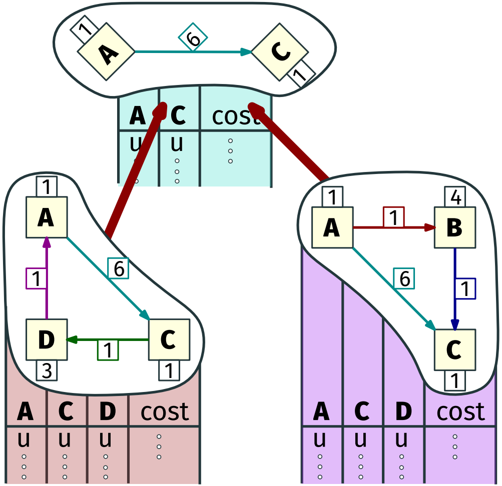


Key Idea:
Enumerate node mappings inside bag.

Validity	Mapping Costs
▸ nodes: trivial	▸ nodes: trivial
▸ edges: graph-search	▸ edges: shortest-paths (using allowed edges)

Solving the Valid Mapping Problem: DynVMP Intuition

DynVMP algorithm: solve VMP via *dynamic programming*.

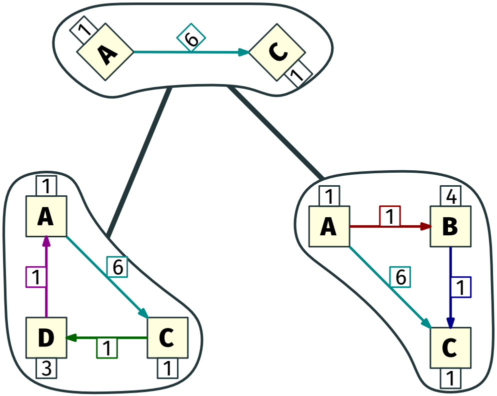


Key Idea:
Enumerate node mappings inside bag.

Validity	Mapping Costs
▸ nodes: trivial	▸ nodes: trivial
▸ edges: graph-search	▸ edges: shortest-paths (using allowed edges)

Solving the Valid Mapping Problem: DynVMP Intuition

DynVMP algorithm: solve VMP via *dynamic programming*.



Tree Decomposition

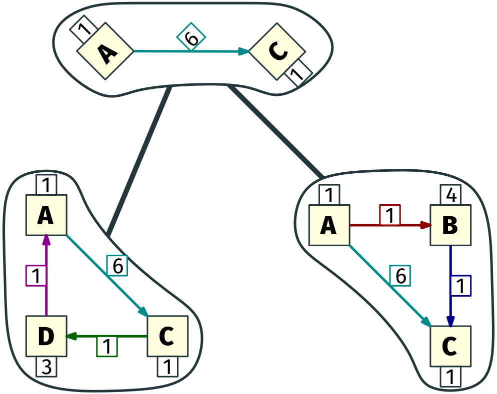
Key Idea:
Enumerate node mappings inside bag.

Validity	**Mapping Costs**
▸ nodes: trivial	▸ nodes: trivial
▸ edges: graph-search	▸ edges: shortest-paths (using allowed edges)

Dynamic programming works when guided by **tree decomposition** $\mathcal{T}_r$.

Solving the Valid Mapping Problem: DynVMP Intuition

DynVMP algorithm: solve VMP via *dynamic programming*.



Tree Decomposition

Key Idea:
Enumerate node mappings inside bag.

Validity

- nodes: trivial
- edges: graph-search

Mapping Costs

- nodes: trivial
- edges: shortest-paths (using allowed edges)

Dynamic programming works when guided by **tree decomposition** $\mathcal{T}_r$.

Runtime is exponential in max. bag size: the **treewidth** $\mathrm{tw}(\mathcal{T}_r)$.

Excursion: Tree Decompositions and Treewidth

- Important concept in theoretical computer science $\rightarrow$ parametrized complexity theory
- Finding tree decomposition of minimal width is $\mathcal{NP}$-hard but *fixed-parameter tractable*.

Excursion: Tree Decompositions and Treewidth

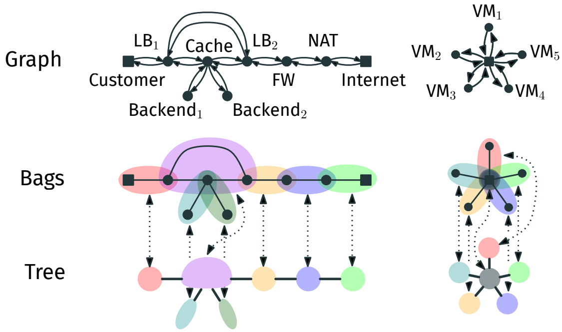
- Important concept in theoretical computer science $\rightarrow$ parametrized complexity theory
- Finding tree decomposition of minimal width is $\mathcal{NP}$-hard but *fixed-parameter tractable*.

Graph Class	Treewidth
trees	1
cacti	2
series-parallel	2
(1-)outerplanar	2
k-outerplanar	$k+1$
planar	*unbounded*

Excursion: Tree Decompositions and Treewidth

- Important concept in theoretical computer science → parametrized complexity theory
- Finding tree decomposition of minimal width is $\mathcal{NP}$-hard but *fixed-parameter tractable*.

Graph Class	Treewidth
trees	1
cacti	2
series-parallel	2
(1-)outerplanar	2
k-outerplanar	$k+1$
planar	*unbounded*



Important request topologies have small treewidth.

Theorem: Correctness & Runtime – Node Placement & Routing

DynVMP solves the VMP in XP-time $\mathcal{O}(|V_r|^3 \cdot |V_S|^{2 \cdot \mathrm{tw}(\mathcal{T}_r)+3})$.

DynVMP: Overview of Results

Theorem: Correctness & Runtime – Node Placement & Routing

DynVMP solves the VMP in **XP**-time $\mathcal{O}(|V_r|^3 \cdot |V_S|^{2 \cdot \mathrm{tw}(\mathcal{T}_r)+3})$.

Theorem: Generalization – Node Placement & Routing & Latencies

DynVMP: Overview of Results

Theorem: Correctness & Runtime – Node Placement & Routing

DynVMP solves the VMP in **XP**-time $\mathcal{O}(|V_r|^3 \cdot |V_S|^{2 \cdot \mathrm{tw}(\mathcal{T}_r)+3})$.

Theorem: Generalization – Node Placement & Routing & Latencies

The DynVMP algorithm finds a $(1+\varepsilon_{\mathsf{LCSP}})$-optimal valid mapping, if one exists.

The **XP**-runtime is bounded by $\mathcal{O}(|V_r|^2 \cdot (|V_r| \cdot |V_S|^{2 \cdot \mathrm{tw}(\mathcal{T}_r)+2} + \mathsf{time}_{\mathsf{LCSP}}(\varepsilon_{\mathsf{LCSP}})))$.

DynVMP: Overview of Results

Theorem: Correctness & Runtime – Node Placement & Routing

DynVMP solves the VMP in **XP**-time $\mathcal{O}(|V_r|^3 \cdot |V_S|^{2 \cdot \mathrm{tw}(\mathcal{T}_r)+3})$.

Theorem: Generalization – Node Placement & Routing & Latencies

The DynVMP algorithm finds a $(1+\varepsilon_{\text{LCSP}})$-optimal valid mapping, if one exists.
The **XP**-runtime is bounded by $\mathcal{O}(|V_r|^2 \cdot (|V_r| \cdot |V_S|^{2 \cdot \mathrm{tw}(\mathcal{T}_r)+2} + \text{time}_{\text{LCSP}}(\varepsilon_{\text{LCSP}})))$.

Important Observation

- The VNEP reduces to the VMP, when any valid mapping is also feasible.
- DynVMP solves the *online VNEP* **optimally / approximatively** in this setting.

DynVMP: Overview of Results

Theorem: Correctness & Runtime – Node Placement & Routing

DynVMP solves the VMP in **XP**-time $\mathcal{O}(|V_r|^3 \cdot |V_S|^{2 \cdot \mathrm{tw}(\mathcal{T}_r)+3})$.

Theorem: Generalization – Node Placement & Routing & Latencies

The DynVMP algorithm finds a $(1+\varepsilon_{\mathsf{LCSP}})$-optimal valid mapping, if one exists.
The **XP**-runtime is bounded by $\mathcal{O}(|V_r|^2 \cdot (|V_r| \cdot |V_S|^{2 \cdot \mathrm{tw}(\mathcal{T}_r)+2} + \mathsf{time}_{\mathsf{LCSP}}(\varepsilon_{\mathsf{LCSP}})))$.

Important Observation

- The VNEP reduces to the VMP, when any valid mapping is also feasible.
- DynVMP solves the *online VNEP* **optimally / approximatively** in this setting.

Key Application: Solving the *Fractional* Offline VNEP

Next Relaxation: allowing **convex combinations** of **valid** mappings

Solving the Fractional Offline VNEP

Next Relaxation: allowing **convex combinations** of **valid** mappings

Offline VNEP – request set $\mathcal{R} = \{r_1, r_2, \ldots\}$

Profit Variant

- Profit for requests $b_r > 0$
- **Task:** Embed **subset** of requests *feasibly* maximizing the attained profit.

Cost Variant

- Resource costs $c_S : G_S \to \mathbb{R}_{\geq 0}$
- **Task:** Find *feasible* embeddings for all requests minimizing cost.

Next Relaxation: allowing **convex combinations** of **valid** mappings

Offline VNEP – request set $\mathcal{R} = \{r_1, r_2, \ldots\}$

Profit Variant

- Profit for requests $b_r > 0$
- **Task:** Embed **subset** of requests *feasibly* maximizing the attained profit.

Cost Variant

- Resource costs $c_S : G_S \to \mathbb{R}_{\geq 0}$
- **Task:** Find *feasible* embeddings for all requests minimizing cost.

Solving the Fractional Offline VNEP

Next Relaxation: allowing **convex combinations** of **valid** mappings

Offline VNEP – request set $\mathcal{R} = \{r_1, r_2, \ldots\}$

Profit Variant

- Profit for requests $b_r > 0$
- **Task:** Embed **subset** of requests *feasibly* maximizing the attained profit.

Cost Variant

- Resource costs $c_S : G_S \to \mathbb{R}_{\geq 0}$
- **Task:** Find *feasible* embeddings for all requests minimizing cost.

Next Relaxation: allowing **convex combinations** of **valid** mappings

Fractional Offline VNEP: Linear Program (LP) for Profit

- Selection of k-th mapping: $$f_r^k \in [0,1] \qquad \forall r \in \mathcal{R}, m_r^k \in \mathcal{M}_r \qquad (1)$$
- Select at most 'one' mapping: $$\sum_{m_r^k \in \mathcal{M}_r} f_r^k \leq 1 \qquad \forall r \in \mathcal{R} \qquad (2)$$
- Enforce capacities: $$\sum_{r \in \mathcal{R}} \sum_{m_r^k \in \mathcal{M}_r} A(m_r^k, x) \cdot f_r^k \leq d_S(x) \qquad \forall x \in G_S \qquad (3)$$
- Maximize the profit: $$\max \sum_{r \in \mathcal{R}} \sum_{m_r^k \in \mathcal{M}_r} b_r \cdot f_r^k \qquad (4)$$

Solving the Fractional Offline VNEP

Fractional Offline VNEP: Linear Program (LP) for Profit

- Selection of k-th mapping: $$f_r^k \in [0,1] \qquad \forall r \in \mathcal{R}, m_r^k \in \mathcal{M}_r \qquad (1)$$
- Select at most 'one' mapping: $$\sum_{m_r^k \in \mathcal{M}_r} f_r^k \leq 1 \qquad \forall r \in \mathcal{R} \qquad (2)$$
- Enforce capacities: $$\sum_{r \in \mathcal{R}} \sum_{m_r^k \in \mathcal{M}_r} A(m_r^k, x) \cdot f_r^k \leq d_S(x) \qquad \forall x \in G_S \qquad (3)$$
- Maximize the profit: $$\max \sum_{r \in \mathcal{R}} \sum_{m_r^k \in \mathcal{M}_r} b_r \cdot f_r^k \qquad (4)$$

XP-Tractable via Column Generation

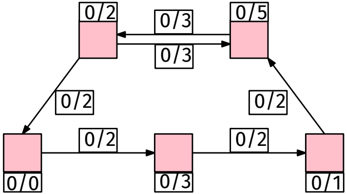
- Dual LP has finitely many variables but exponential number of constraints.
- Dual constraints can be separated using DynVMP algorithm.
  $\rightsquigarrow$ runtime $\mathcal{O}\left(\mathsf{poly}\left(\sum_{r \in \mathcal{R}} |V_r|^3 \cdot |V_S|^{2 \cdot \mathrm{tw}(\mathcal{T}_r)+3}\right)\right)$ due to Grötschel et al. [1988]

Offline Approximation Algorithms

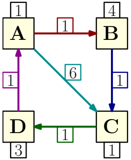
Randomized Rounding: Intuition

Example

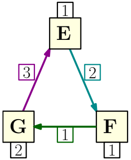
Substrate Network



Request r_1: 100€



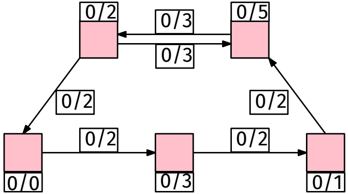
Request r_2: 50€



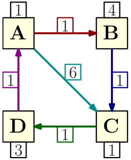
Randomized Rounding: Intuition

Example

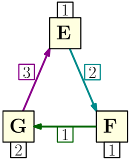
Substrate Network



Request r_1: 100€



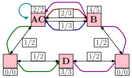
Request r_2: 50€



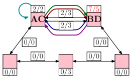
Example Solution to Linear Program: Profit 133€

Variables of r_1 (profit: 100€)

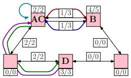
$f_1^1 = 0.5$



$f_1^2 = 0.3$



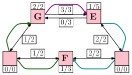
$f_1^3 = 0.2$



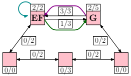
...

Variables of r_2 (profit: 50€)

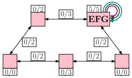
$f_2^1 = 0.5$



$f_2^2 = 0.16$



$f_2^3 = 0$

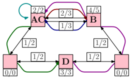


...

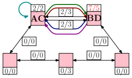
Example Solution to Linear Program: Profit 133€

Variables of r_1 (profit: 100€)

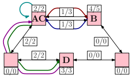
$f_1^1 = 0.5$



$f_1^2 = 0.3$



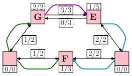
$f_1^3 = 0.2$



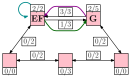
...

Variables of r_2 (profit: 50€)

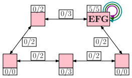
$f_2^1 = 0.5$



$f_2^2 = 0.16$



$f_2^3 = 0$



...

Idea: Treat weights as probabilities!

Algorithm: RoundingProcedure

Input : LP solution

foreach $r \in \mathcal{R}$ **do**

    choose m_r^k with probability f_r^k

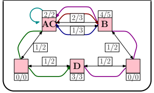
end

Randomized Rounding: Intuition

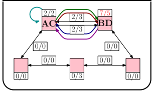
Example Solution to Linear Program: Profit 133€

Variables of r_1 (profit: 100€)

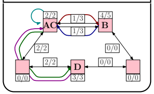
$f_1^1 = 0.5$



$f_1^2 = 0.3$



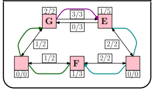
$f_1^3 = 0.2$



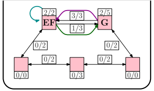
...

Variables of r_2 (profit: 50€)

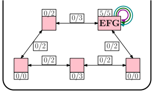
$f_2^1 = 0.5$



$f_2^2 = 0.16$



$f_2^3 = 0$



...

Idea: Treat weights as probabilities!

Algorithm: RoundingProcedure

Input : LP solution

foreach $r \in \mathcal{R}$ **do**

  choose m_r^k with probability f_r^k

end

Rounding Outcomes

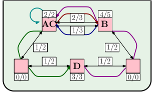
Iter.	Req. 1	Req. 2	Profit	max Load

Randomized Rounding: Intuition

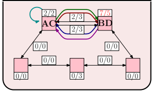
Example Solution to Linear Program: Profit 133€

Variables of r_2 (profit: 50€)

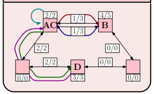
$f_1^1 = 0.5$



$f_1^2 = 0.3$



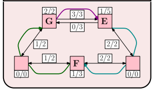
$f_1^3 = 0.2$



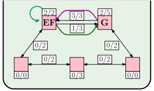
...

Variables of r_2 (profit: 50€)

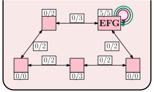
$f_2^1 = 0.5$



$f_2^2 = 0.16$



$f_2^3 = 0$



...

Idea: Treat weights as probabilities!

Algorithm: RoundingProcedure

Input : LP solution

foreach $r \in \mathcal{R}$ **do**

  choose m_r^k with probability f_r^k

end

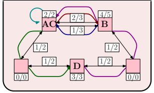
Rounding Outcomes

Iter.	Req. 1	Req. 2	Profit	max Load
1	m_1^1	m_2^2	150€	200%

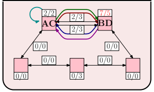
Example Solution to Linear Program: Profit 133€

Variables of request 1

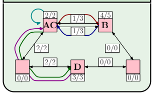
$f_1^1 = 0.5$



$f_1^2 = 0.3$



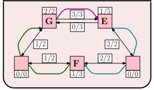
$f_1^3 = 0.2$



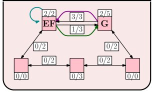
...

Variables of r_2 (profit: 50€)

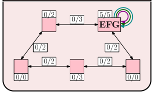
$f_2^1 = 0.5$



$f_2^2 = 0.16$



$f_2^3 = 0$



...

Idea: Treat weights as probabilities!

Algorithm: RoundingProcedure

Input : LP solution

foreach $r \in \mathcal{R}$ **do**

  choose m_r^k with probability f_r^k

end

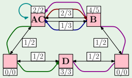
Rounding Outcomes

Iter.	Req. 1	Req. 2	Profit	max Load
1	m_1^1	m_2^2	150€	200%
2	m_1^3	$\emptyset$	100€	100%

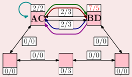
Example Solution to Linear Program: Profit 133€

Variables of request 1

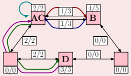
$f_1^1 = 0.5$



$f_1^2 = 0.3$



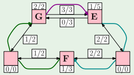
$f_1^3 = 0.2$



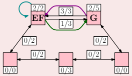
...

Variables of r_2 (profit: 50€)

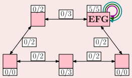
$f_2^1 = 0.5$



$f_2^2 = 0.16$



$f_2^3 = 0$



...

Idea: Treat weights as probabilities!

Algorithm: RoundingProcedure

Input : LP solution

foreach $r \in \mathcal{R}$ **do**

  choose m_r^k with probability f_r^k

end

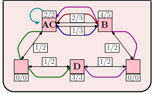
Rounding Outcomes

Iter.	Req. 1	Req. 2	Profit	max Load
1	m_1^1	m_2^2	150€	200%
2	m_1^3	$\emptyset$	100€	100%
3	m_1^1	m_2^1	150€	200%

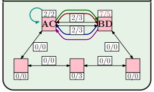
Example Solution to Linear Program: Profit 133€

Variables of request 1

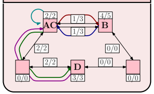
$f_1^1 = 0.5$



$f_1^2 = 0.3$



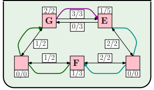
$f_1^3 = 0.2$



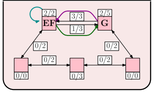
...

Variables of r_2 (profit: 50€)

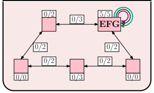
$f_2^1 = 0.5$



$f_2^2 = 0.16$



$f_2^3 = 0$



...

Idea: Treat weights as probabilities!

Algorithm: RoundingProcedure

Input : LP solution

foreach $r \in \mathcal{R}$ **do**

  choose m_r^k **with probability** f_r^k

end

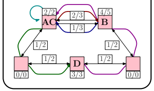
Rounding Outcomes

Iter.	Req. 1	Req. 2	Profit	max Load
1	m_1^1	m_2^2	150€	200%
2	m_1^3	$\emptyset$	100€	100%
3	m_1^1	m_2^1	150€	200%
4	m_1^2	m_2^1	150€	200%

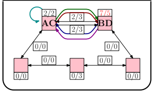
Example Solution to Linear Program: Profit 133€

Variables of request 1

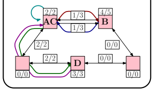
$f_1^1 = 0.5$



$f_1^2 = 0.3$



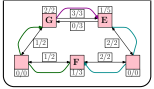
$f_1^3 = 0.2$



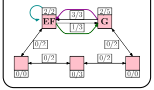
...

Variables of r_2 (profit: 50€)

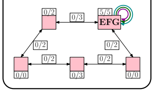
$f_2^1 = 0.5$



$f_2^2 = 0.16$



$f_2^3 = 0$



...

Idea: Treat weights as probabilities!

Algorithm: RoundingProcedure

Input : LP solution

foreach $r \in \mathcal{R}$ **do**

  choose m_r^k with probability f_r^k

end

Rounding Outcomes

Iter.	Req. 1	Req. 2	Profit	max Load
1	m_1^1	m_2^2	150€	200%
2	m_1^3	$\emptyset$	100€	100%
3	m_1^1	m_2^1	150€	200%
4	m_1^2	m_2^1	150€	200%
⋮	⋮	⋮	⋮	⋮

Algorithm: VNEP Approximation (Profit)

```
// perform preprocessing
```

compute *optimal* LP solution

do

  solution $\leftarrow$ ROUNDINGPROCEDURE

while $\left(\begin{array}{l}\textit{not } (\alpha, \beta, \gamma)\text{-approximate} \\ \text{and rounding tries not exceeded}\end{array}\right)$

Algorithm: RoundingProcedure

Input : LP solution

foreach $r \in \mathcal{R}$ **do**

  choose m_r^k **with probability** f_r^k

end

Algorithm: VNEP Approximation (Profit)

```
// perform preprocessing
```

compute *optimal* LP solution

do

  solution ← RoundingProcedure

while $\left(\begin{array}{l}\textit{not } (\alpha, \beta, \gamma)\text{-approximate} \\ \text{and rounding tries not exceeded}\end{array}\right)$

Proof via Chernoff & Hoeffding bounds.

Main Theorem: (XP-)Approximation for the Virtual Network Embedding Problem

The Algorithm returns (α, β, γ)-approximate solutions of at least an α fraction of the optimal profit, and allocations on nodes and edges within factors of β and γ of the original capacities, respectively, *with high probability*.

XP-Approximation Algorithms

Algorithm: VNEP Approximation (Profit)

```
// perform preprocessing
```

compute *optimal* LP solution

do

  solution ← RoundingProcedure

while

$\left(\begin{array}{l}\textit{not } (\alpha,\beta,\gamma)\text{-approximate} \\ \text{and rounding tries not exceeded}\end{array}\right)$

Definition of Parameters

$\alpha = 1/3$	(relative achieved profit)
$\beta = (1 + \varepsilon \cdot \sqrt{2 \cdot \Delta(V_S) \cdot \log \lvert V_S \rvert})$	(max node load)
$\gamma = (1 + \varepsilon \cdot \sqrt{2 \cdot \Delta(E_S) \cdot \log \lvert E_S \rvert})$	(max edge load)
$\varepsilon = \max_{r \in \mathcal{R}, x \in R_S} d_{\max}(r,x)/d_S(x) \leq 1$	(max demand/capacity)
$\Delta(X) = \max_{x \in X} \sum_{r \in \mathcal{R}} (A_{\max}(r,x)/d_{\max}(r,x))^2$	(sum over $\mathcal{R}$ of squared max (total / single) alloc)

Main Theorem: (XP-)Approximation for the Virtual Network Embedding Problem

The Algorithm returns (α, β, γ)-approximate solutions of at least an α fraction of the optimal profit, and allocations on nodes and edges within factors of β and γ of the original capacities, respectively, *with high probability*.

XP-Approximation Algorithms

Algorithm: VNEP Approximation (Profit)

`// perform preprocessing`

compute *optimal* LP solution

do

solution ← RoundingProcedure

while

$\left(\begin{array}{l}\textit{not } (\alpha, \beta, \gamma)\text{-approximate} \\ \text{and rounding tries not exceeded}\end{array}\right)$

Definition of Parameters

$$
\begin{aligned}
\alpha &= 1/3 && \text{(relative achieved profit)} \\
\beta &= (1 + \varepsilon \cdot \sqrt{2 \cdot \Delta(V_S) \cdot \log |V_S|}) && \text{(max node load)} \\
\gamma &= (1 + \varepsilon \cdot \sqrt{2 \cdot \Delta(E_S) \cdot \log |E_S|}) && \text{(max edge load)} \\
\varepsilon &= \max_{r \in \mathcal{R}, x \in R_S} d_{\max}(r,x)/d_S(x) \leq 1 && \text{(max demand/capacity)}
\end{aligned}
$$

$$
\Delta(X) = \max_{x \in X} \sum_{r \in \mathcal{R}} (A_{\max}(r,x)/d_{\max}(r,x))^2 \quad \left(\begin{array}{c}\text{sum over } \mathcal{R} \text{ of squared} \\ \text{max (total / single) alloc}\end{array}\right)
$$

Worst-Case Analysis

$$
\beta \in \mathcal{O}(\varepsilon \cdot \max_{r \in \mathcal{R}} |V_r| \cdot \sqrt{|\mathcal{R}| \cdot \log |V_S|}) \qquad \gamma \in \mathcal{O}(\varepsilon \cdot \max_{r \in \mathcal{R}} |E_r| \cdot \sqrt{|\mathcal{R}| \cdot \log |E_S|})
$$

XP-Approximation Algorithms

Algorithm: VNEP Approximation (Profit)

```
// perform preprocessing
```

compute *optimal* LP solution

do

solution $\leftarrow$ RoundingProcedure

while

$\left(\text{\textit{not} } (\alpha, \beta, \gamma)\text{-approximate and rounding tries not exceeded} \right)$

Definition of Parameters

$\alpha = 1/3$	(relative achieved profit)
$\beta = (1 + \varepsilon \cdot \sqrt{2 \cdot \Delta(V_S) \cdot \log \lvert V_S \rvert})$	(max node load)
$\gamma = (1 + \varepsilon \cdot \sqrt{2 \cdot \Delta(E_S) \cdot \log \lvert E_S \rvert})$	(max edge load)
$\varepsilon = \max_{r \in \mathcal{R}, x \in R_S} d_{\max}(r,x)/d_S(x) \leq 1$	(max demand/capacity)

$$\Delta(X) = \max_{x \in X} \sum_{r \in \mathcal{R}} (A_{\max}(r,x)/d_{\max}(r,x))^2 \quad \left(\begin{array}{c} \text{sum over } \mathcal{R} \text{ of squared} \\ \text{max (total / single) alloc} \end{array} \right)$$

Worst-Case Analysis

$$\beta \in \mathcal{O}(\varepsilon \cdot \max_{r \in \mathcal{R}} \lvert V_r \rvert \cdot \sqrt{\lvert \mathcal{R} \rvert \cdot \log \lvert V_S \rvert}) \qquad \gamma \in \mathcal{O}(\varepsilon \cdot \max_{r \in \mathcal{R}} \lvert E_r \rvert \cdot \sqrt{\lvert \mathcal{R} \rvert \cdot \log \lvert E_S \rvert})$$

Overview of XP-Approximation Results

Obj.	Setting	Approximation Factors α	$\beta - \beta'$	$\gamma - \gamma'$	Runtime
Cost	$\langle\,\mathbf{VE}\,\vert\,\mathbf{NR}\,\rangle$	2	2	2	$\text{poly}\left(\sum_{r\in\mathcal{R}} \vert V_r\vert^3 \cdot \vert V_S\vert^{2\cdot\mathrm{tw}(\mathcal{T}_r)+3}\right)$
	$\langle\,\mathbf{VE}\,\vert\,\mathbf{NRL}\,\rangle$	$2+2\cdot\varepsilon_{\text{LCSP}}$	$2+2\cdot\varepsilon_{\text{LCSP}}$	$2+2\cdot\varepsilon_{\text{LCSP}}$	$\text{poly}\left(\sum_{r\in\mathcal{R}} \vert V_r\vert^2 \cdot \left(\vert V_r\vert \cdot \vert V_S\vert^{2\cdot\mathrm{tw}(\mathcal{T}_r)+2} + \text{time}_{\text{LCSP}}(\varepsilon_{\text{LCSP}})\right)\right)$
Profit	$\langle\,\mathbf{VE}\,\vert\,\mathbf{NR}\,\rangle$	1/3	1	1	$\text{poly}\left(\sum_{r\in\mathcal{R}} \vert V_r\vert^3 \cdot \vert V_S\vert^{2\cdot\mathrm{tw}(\mathcal{T}_r)+3}\right)$
	$\langle\,\mathbf{VE}\,\vert\,\mathbf{NRL}\,\rangle$	$1/(3+3\cdot\varepsilon_{\text{LCSP}})$	$1+\varepsilon_{\text{LCSP}}$	$1+\varepsilon_{\text{LCSP}}$	$\text{poly}\left(\sum_{r\in\mathcal{R}} \vert V_r\vert^2 \cdot \left(\vert V_r\vert \cdot \vert V_S\vert^{2\cdot\mathrm{tw}(\mathcal{T}_r)+2} + \text{time}_{\text{LCSP}}(\varepsilon_{\text{LCSP}})\right)\right)$

- $\beta' = \varepsilon \cdot \sqrt{2 \cdot \Delta(V_S) \cdot \log |V_S|}$, $\gamma' = \varepsilon \cdot \sqrt{2 \cdot \Delta(V_S) \cdot \log |V_S|}$
- $\varepsilon_{\text{LCSP}} > 0$ must hold; $\text{time}_{\text{LCSP}}(\varepsilon_{\text{LCSP}})$ is polynomial in $1/\varepsilon_{\text{LCSP}}$ (and input)

Derived Heuristics & Evaluation

Derived & Benchmark Heuristics

Derived Heuristics: Key Ideas

- **Goal: feasibility**
  - **reactive**: discard rounded mapping upon violation
  - **proactive**: forbid mappings violating capacities and recompute LP
- Sample solutions
- Different request orders

Benchmark Heuristics: WiNE / ViNE

- ViNE – single request mapping
  - uses randomized rounding to map virtual nodes
  - realizes edges via shortest paths
- WiNE – offline adaptation
  - greedy: process according to profit

Computational Evaluation: Setup

Substrate: GEANT

Explorative Study – Vast Parameter Space

Treewidth:	1, 2, 3, 4
Number of requests:	40, 60, 80, 100
Node-Resource Factor (NRF):	0.2, 0.4, 0.6, 0.8, 1.0
Edge-Resource Factor (ERF):	0.25, 0.5, 1.0, 2.0, 4.0
Instances per combination:	15

Requests

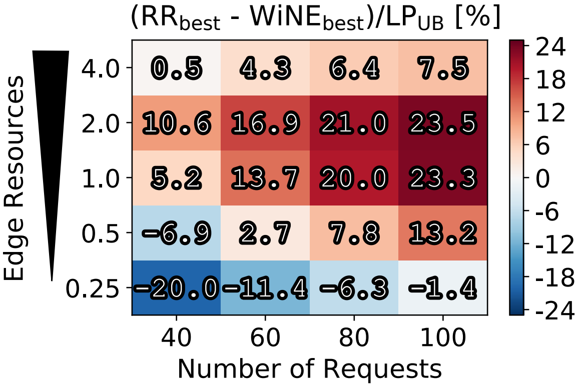
- #nodes uniformly chosen from $\{5, \ldots, 15\}$
- topology: random but with specific treewidth
- node mappings restricted to 10 nodes

Code available:

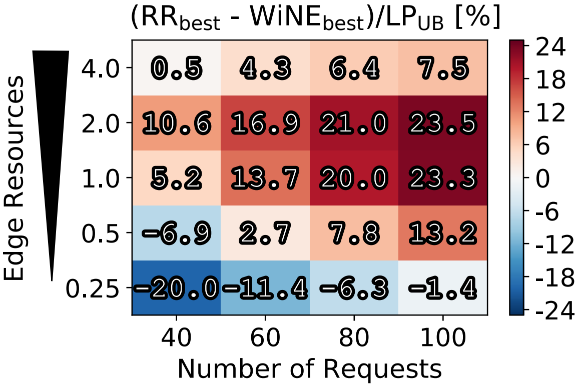
`https://github.com/vnep-approx/`

Investigate qualitative potential of randomized rounding heuristics.

Investigate qualitative potential of randomized rounding heuristics.



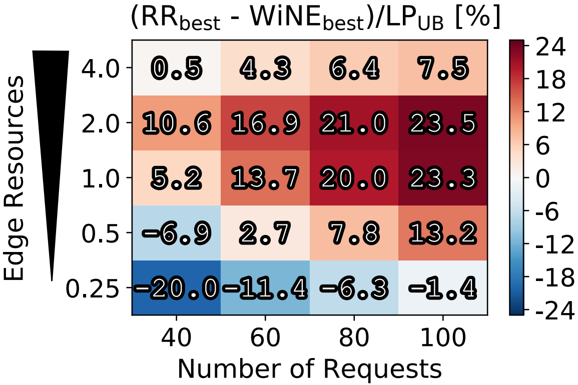
Investigate qualitative potential of randomized rounding heuristics.



Observations

- overall 6.53% (rel.) improvement
- outperforms WiNE in > 99.9%:
  - ERFs 1.0, 2.0 and 80, 100 requests
- performance increases with #requests

Investigate qualitative potential of randomized rounding heuristics.



Observations

- overall 6.53% (rel.) improvement
- outperforms WiNE in $> 99.9\%$:
  - ERFs 1.0, 2.0 and 80, 100 requests
- performance increases with #requests

Efficiency – Total Heuristic Runtime

- $< 200s$ for treewidths 1, 2, 3
- $< 1000s$ for treewidth 4

Conclusion

Thesis Overview and Contributions

Virtual Network Embedding Problem

Computational Complexity

- $\mathcal{NP}$-completeness in various settings
- *Structural* hardness
  - VMP is $\mathcal{NP}$-complete
  - even planar requests

Problem Relaxations

- Valid Mapping Problem
  - DynVMP Algorithm
- Fractional VNEP
  - Column Generation LP

Offline Approximations

- XP-time approximations
- under *all restrictions*
- *profit* and *cost*

Heuristics for Offline Profit VNEP and Evaluation

- no capacity violations
- can consistently outperform heuristic

Thesis Overview and Contributions

Virtual Network Embedding Problem

Computational Complexity

- $\mathcal{NP}$-completeness in various settings
- *Structural* hardness
  - VMP is $\mathcal{NP}$-complete
  - even planar requests

Problem Relaxations

- Valid Mapping Problem
  - DynVMP Algorithm
- Fractional VNEP
  - Column Generation LP

Offline Approximations

- XP-time approximations
- under *all restrictions*
- *profit* and *cost*

Heuristics for Offline Profit VNEP and Evaluation

- no capacity violations
- can consistently outperform heuristic

Derived *several* novel theoretic results and showed applicability in practice.

Thesis Overview and Contributions

Virtual Network Embedding Problem

Computational Complexity

- $\mathcal{NP}$-completeness in various settings
- *Structural* hardness
  - VMP is $\mathcal{NP}$-complete
  - even planar requests

Problem Relaxations

- Valid Mapping Problem
  - DynVMP Algorithm
- Fractional VNEP
  - Column Generation LP

Offline Approximations

- XP-time approximations
- under *all restrictions*
- *profit* and *cost*

Heuristics for Offline Profit VNEP and Evaluation

- no capacity violations
- can consistently outperform heuristic

Derived *several* novel theoretic results and showed applicability in practice.

Specific Embeddings & Embedding Models

Virtual Clusters

- *optimal* algorithm for resource minimization
- *hose*-model $\rightsquigarrow$ bandwidth reduction

Temporal VNEP

- *incorporation* of *scheduling* aspects
- Mixed-Integer Programs to harness flexibility