More Hierarchy in Route Planning Using Edge Hierarchies

ATMOS'19 · September 12, 2019

Demian Hespe and Peter Sanders

INSTITUTE OF THEORETICAL INFORMATICS · ALGORITHMICS GROUP


Motivation

- Dijkstra's algorithm is too slow
- Last decade: split up into preprocessing and query phases
- Hierachical Route Planning
  - Roads in the middle of a path are more important than on the ends
- Contraction Hierarchies have one level per node (crossing)

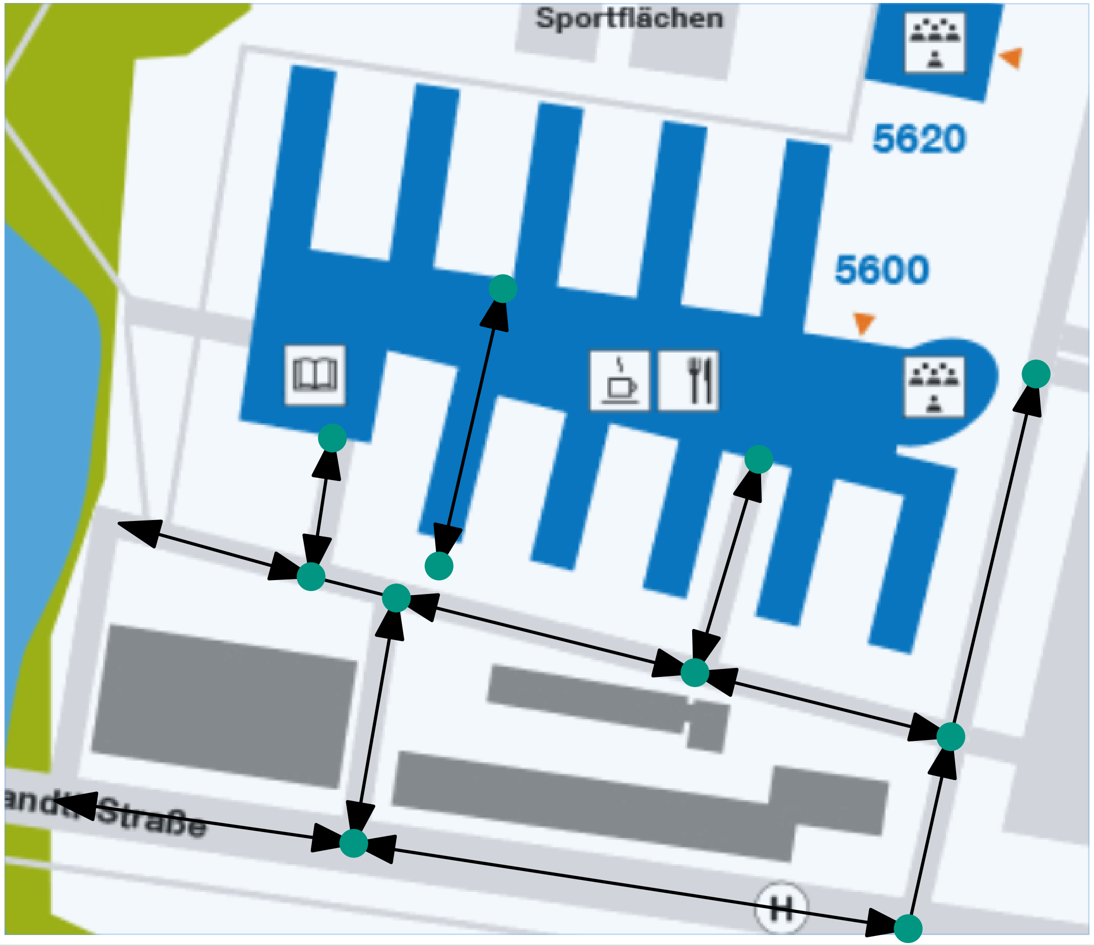
- Today: One level per edge (road)

Problem Definition

- Vertices = Crossings
- Edges = Roads
- Edge weights = metric to optimize

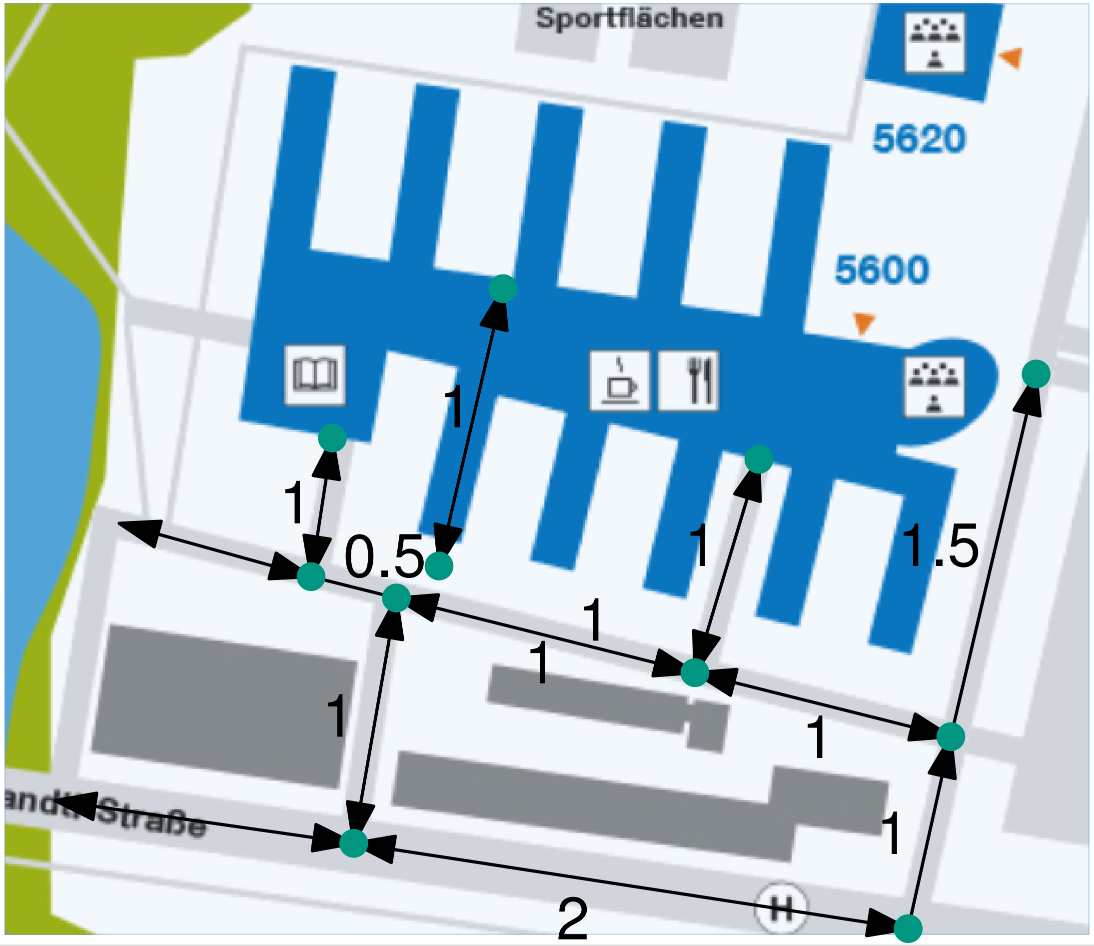
Problem Definition

- Vertices = Crossings
- Edges = Roads
- Edge weights = metric to optimize



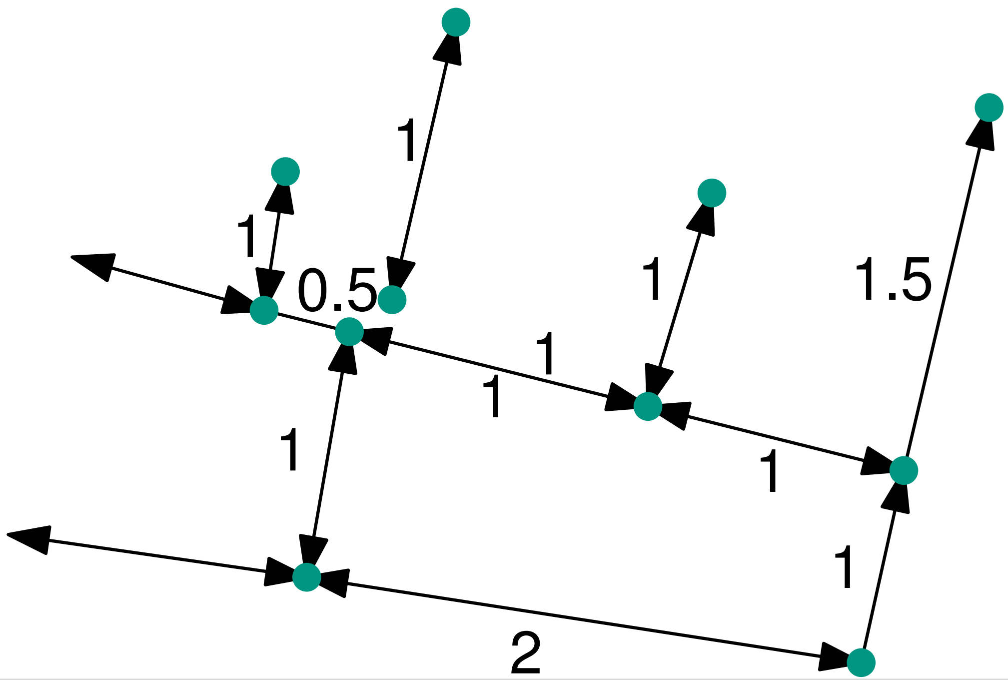
Problem Definition

- Vertices = Crossings
- Edges = Roads
- Edge weights = metric to optimize

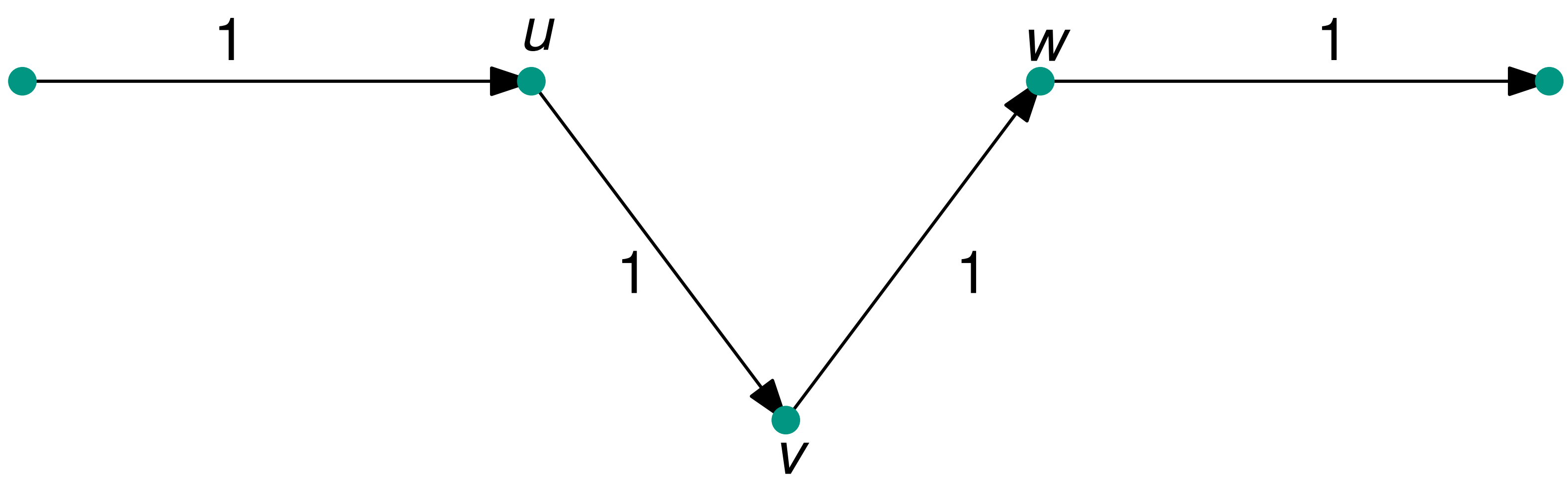


Problem Definition

- Vertices = Crossings
- Edges = Roads
- Edge weights = metric to optimize

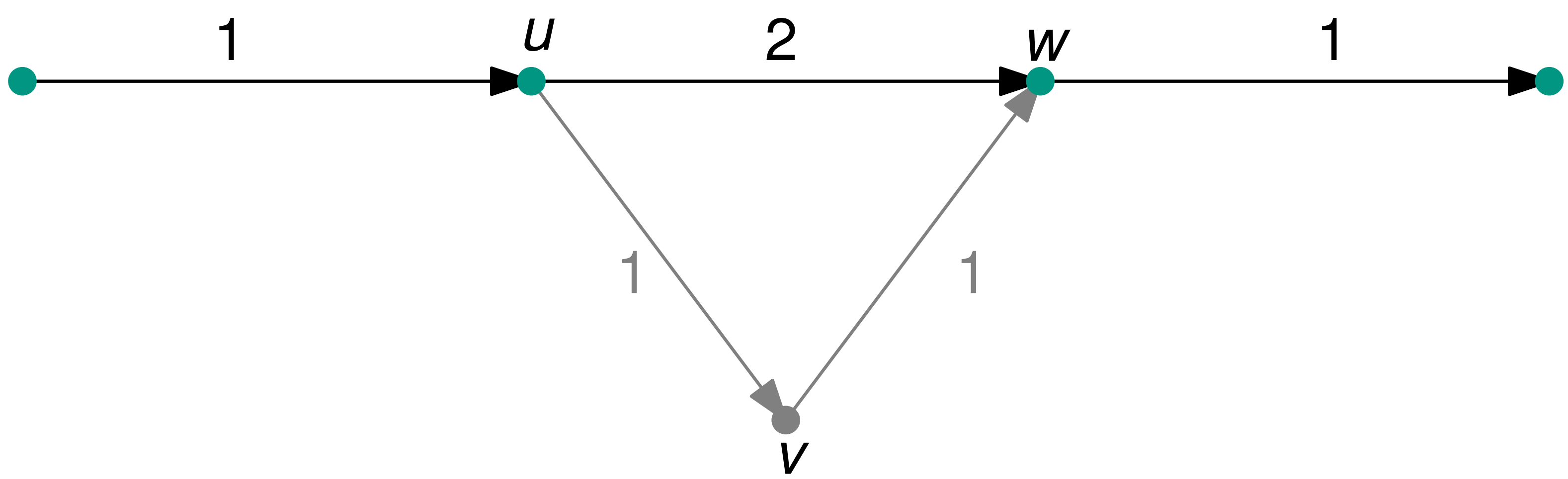


Shortcuts



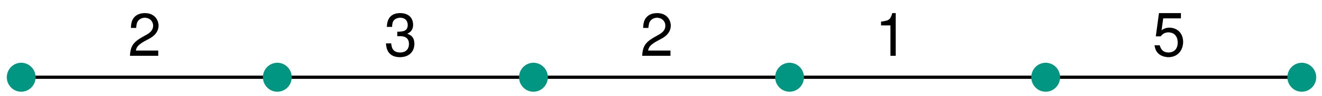
- Add new edge skipping over one vertex
- Distances unchanged
- Unpack by storing midway vertex

Shortcuts

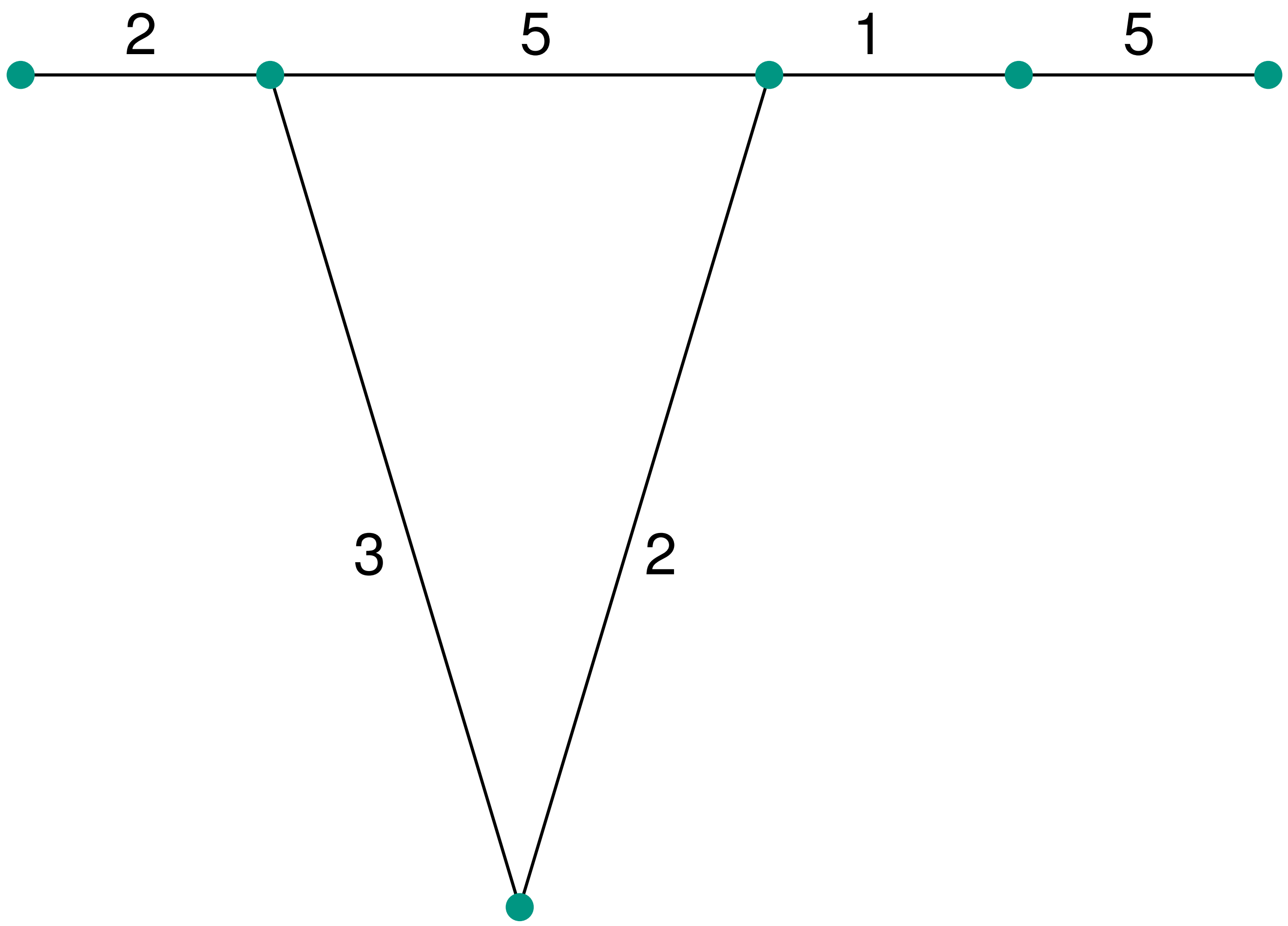


- Add new edge skipping over one vertex
- Distances unchanged
- Unpack by storing midway vertex

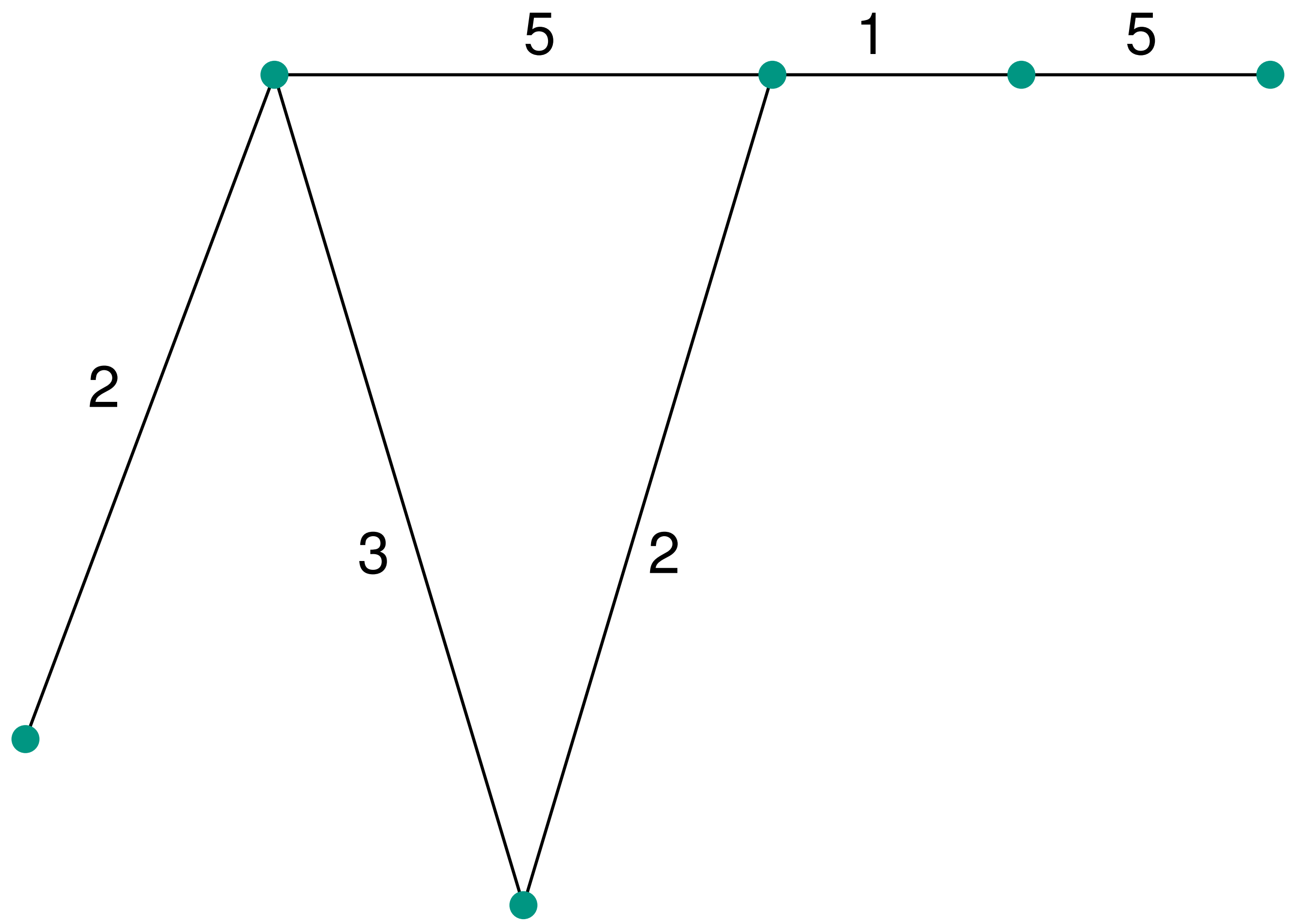
Contraction Hierarchies (CHs) [GSSV2012]



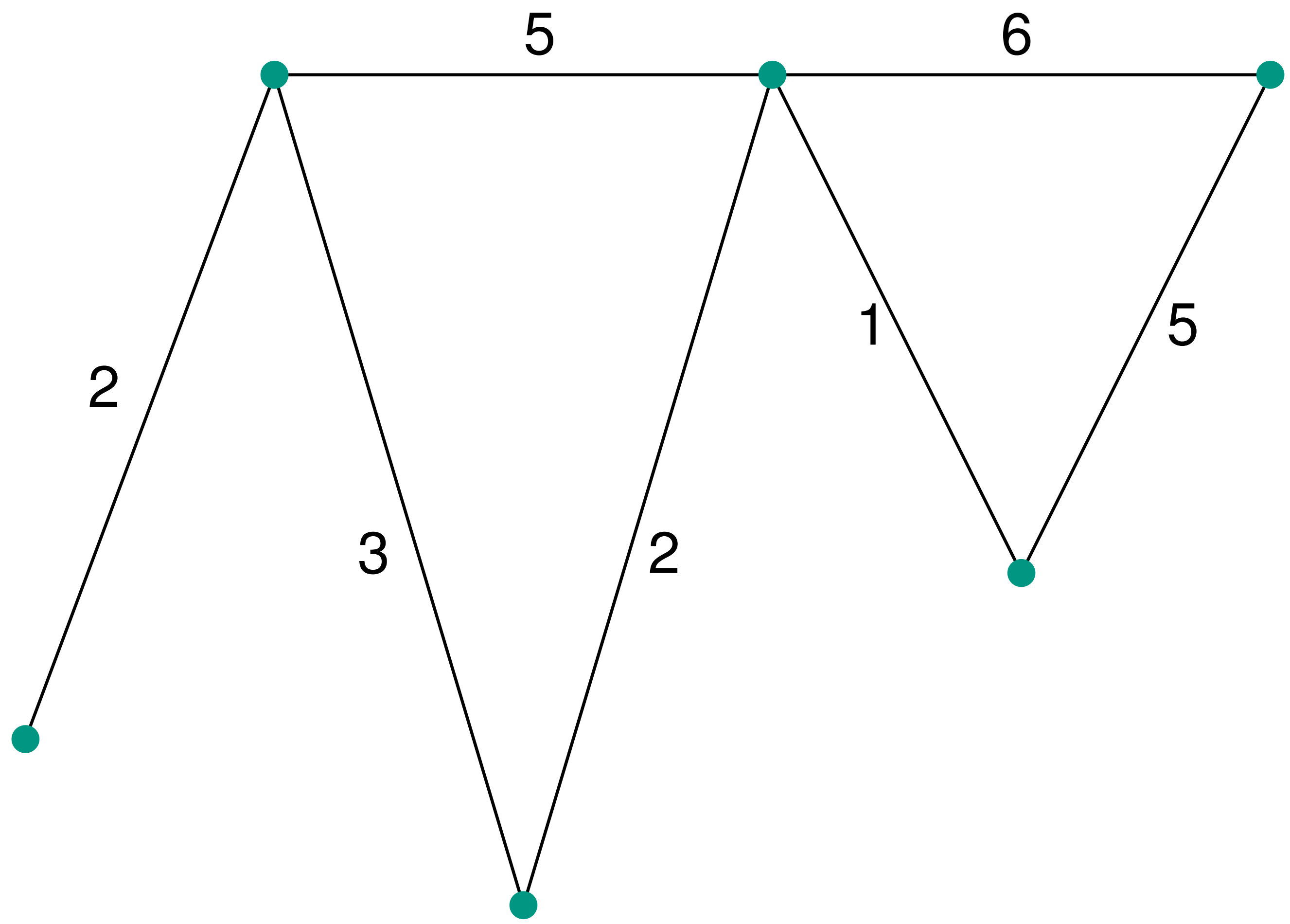
Contraction Hierarchies (CHs) [GSSV2012]



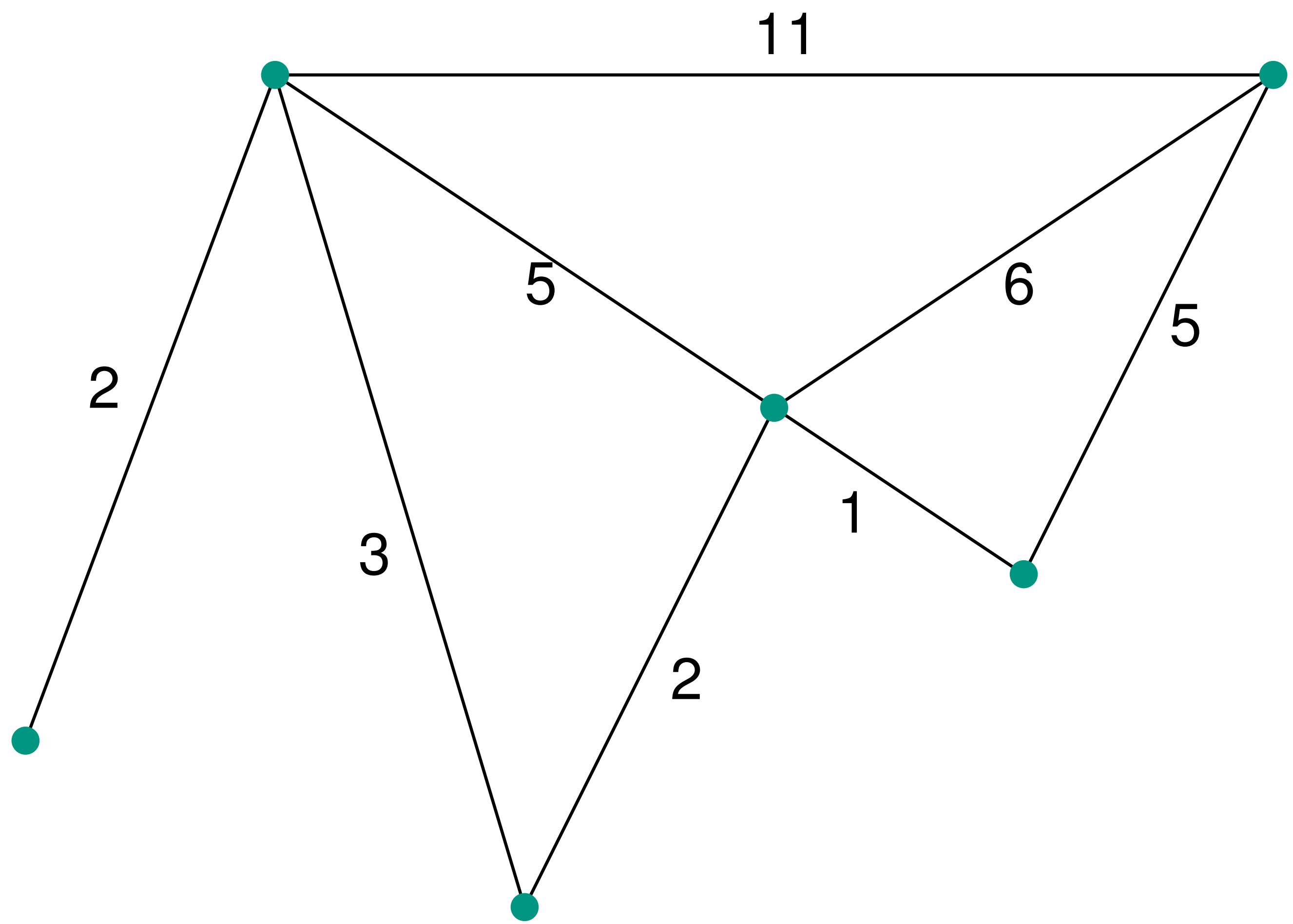
Contraction Hierarchies (CHs) [GSSV2012]



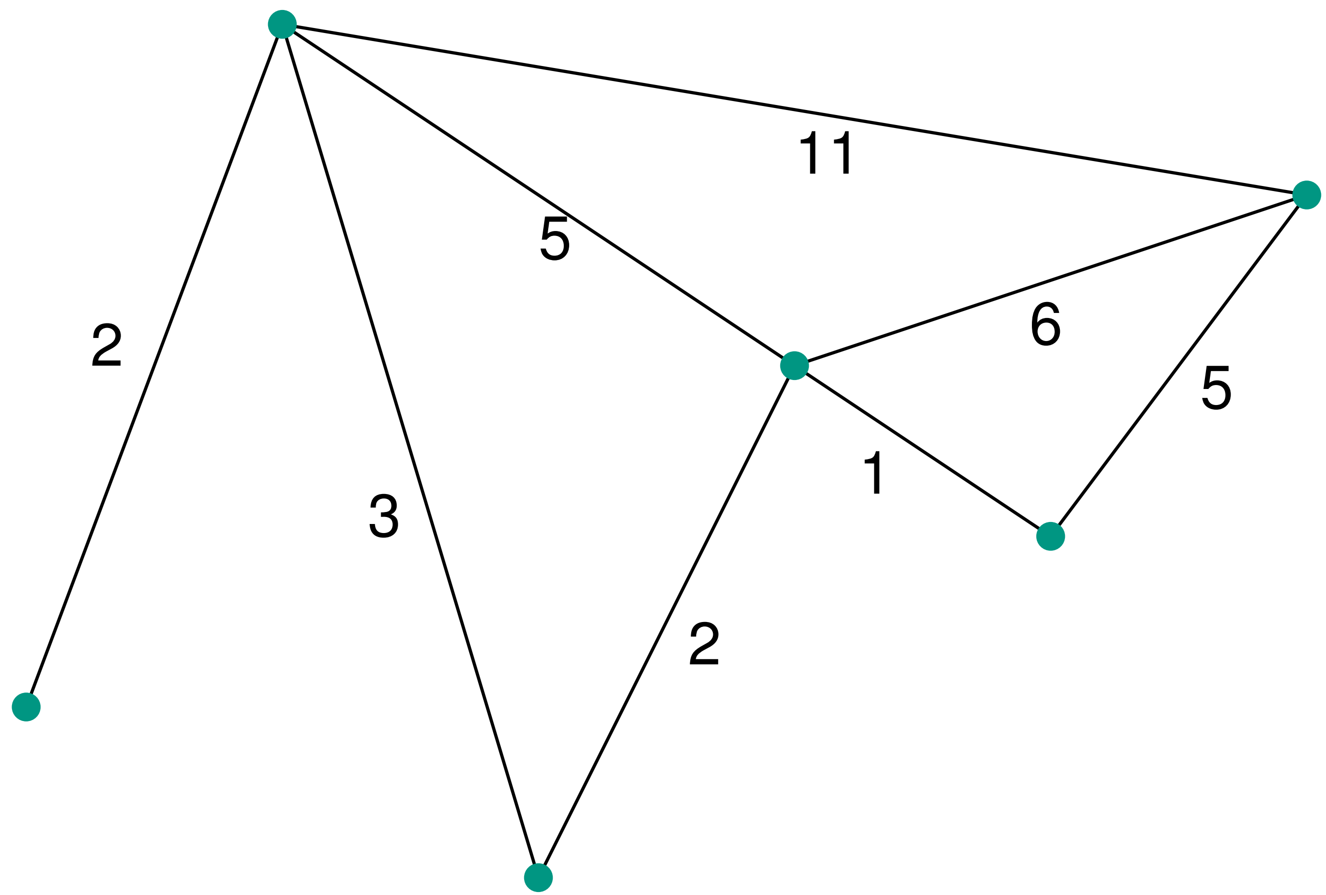
Contraction Hierarchies (CHs) [GSSV2012]



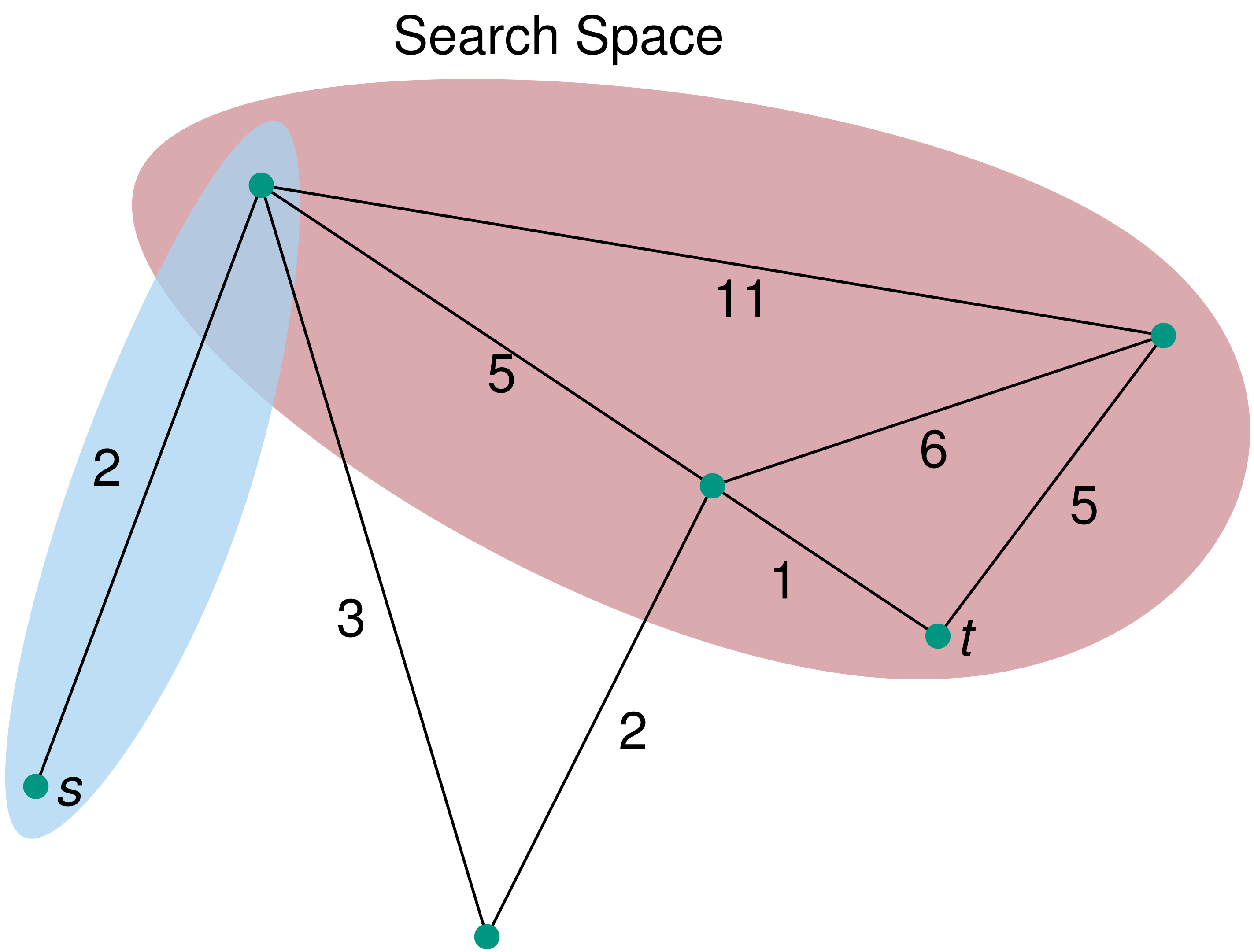
Contraction Hierarchies (CHs) [GSSV2012]



Contraction Hierarchies (CHs) [GSSV2012]

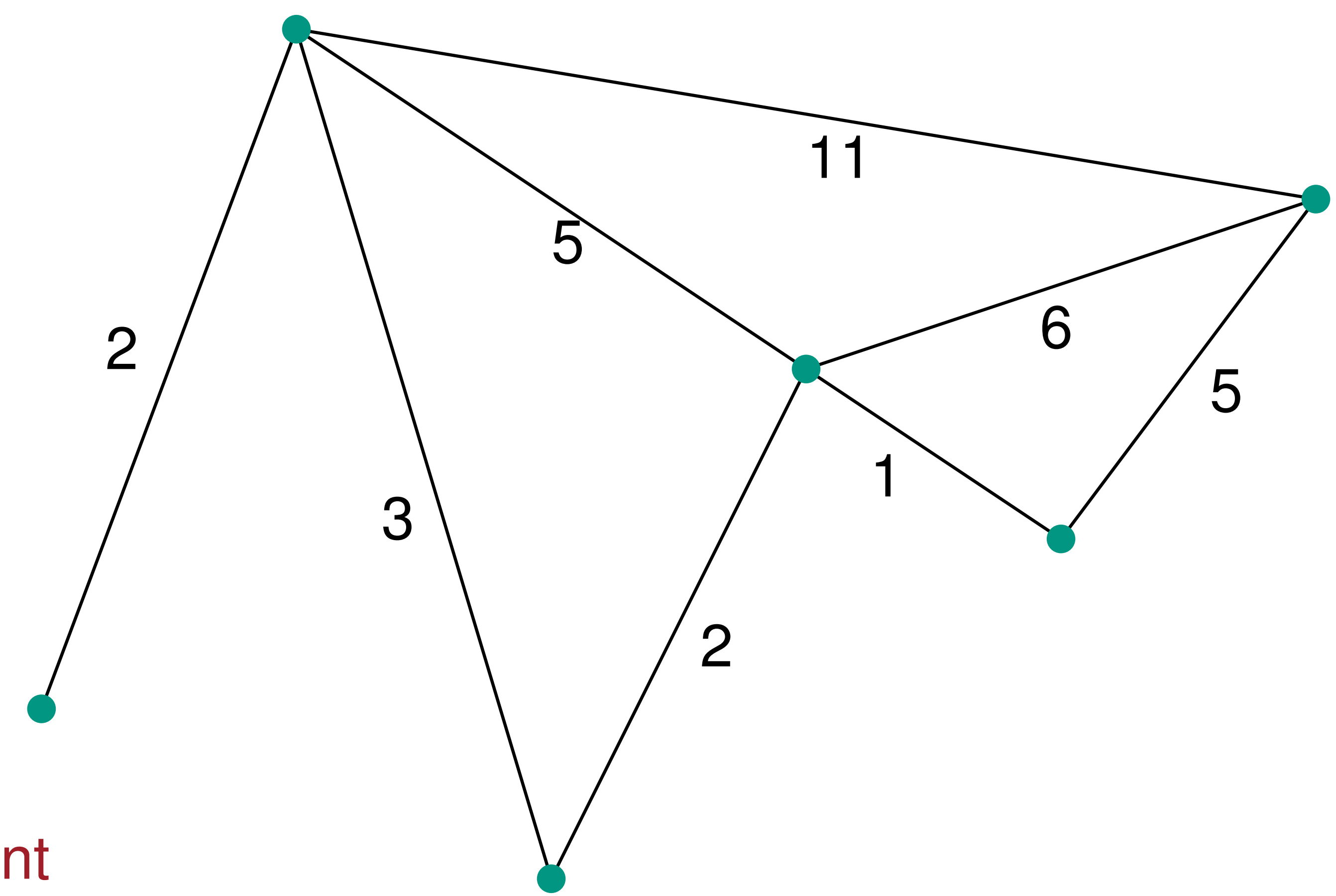


Contraction Hierarchies (CHs) [GSSV2012]



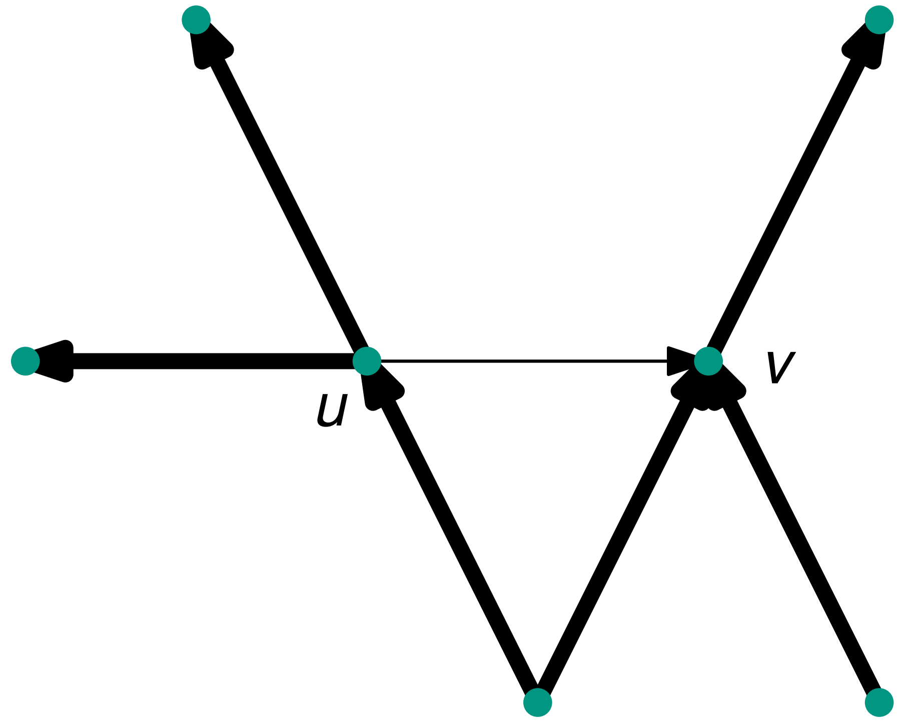
Contraction Hierarchies (CHs) [GSSV2012]

Important

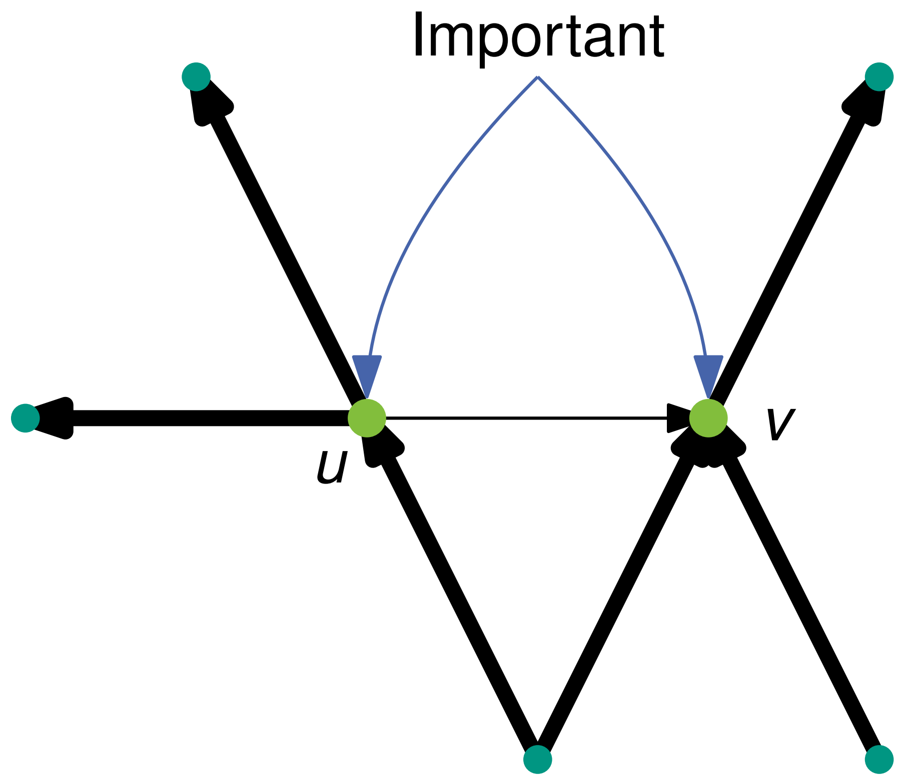


Unimportant

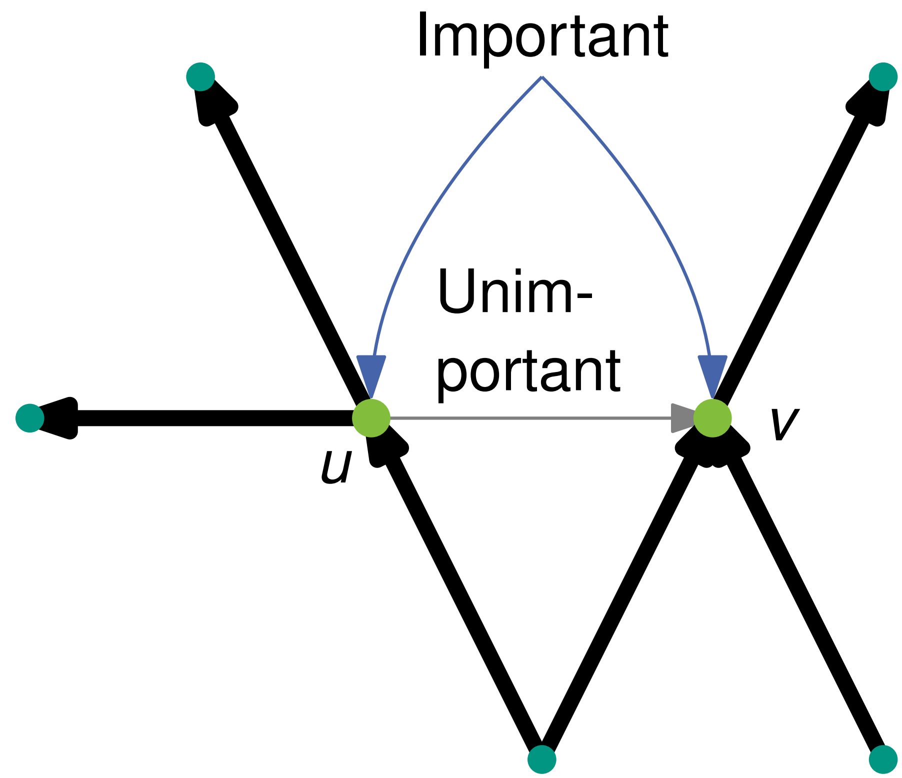
Why Edge Hierarchies?



Why Edge Hierarchies?

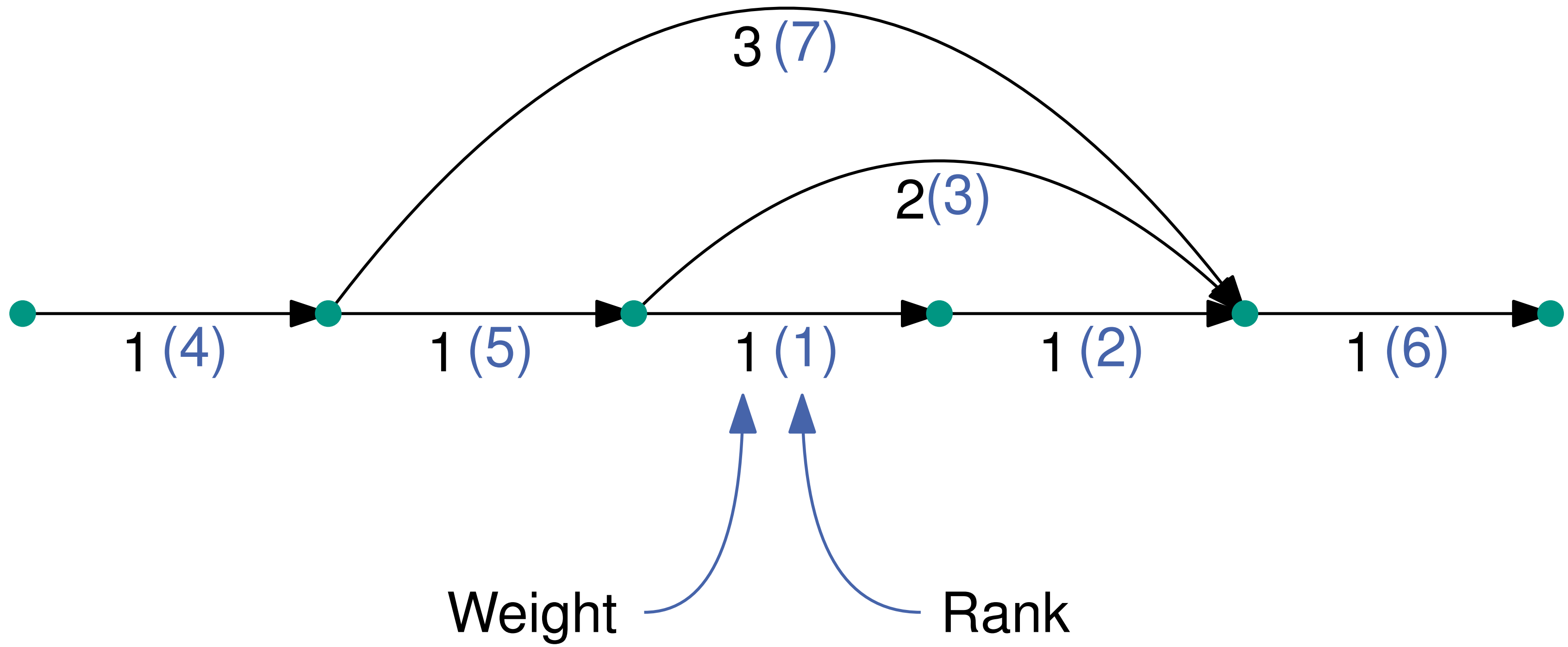


Why Edge Hierarchies?

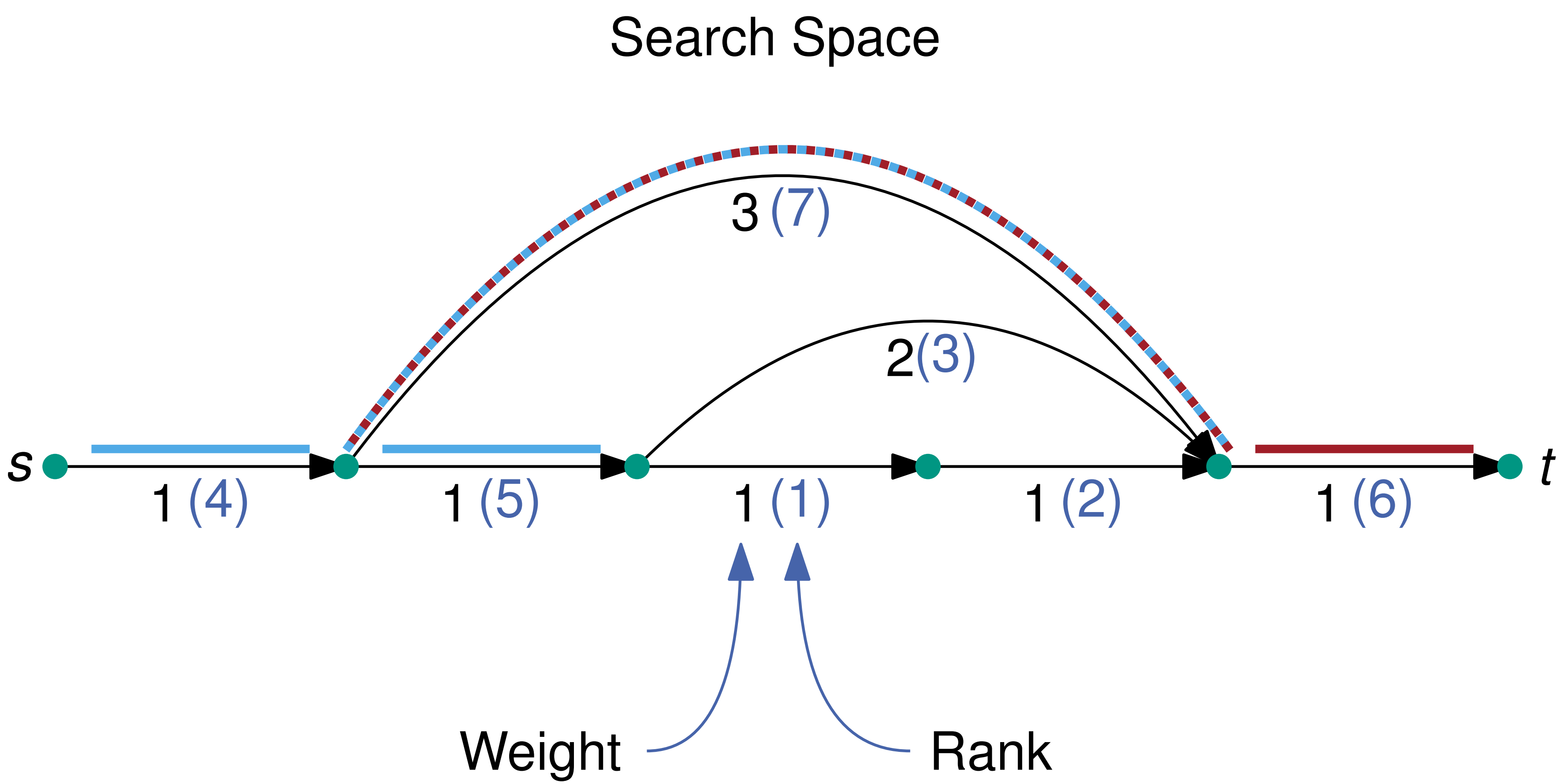


Edge Hierarchy Queries

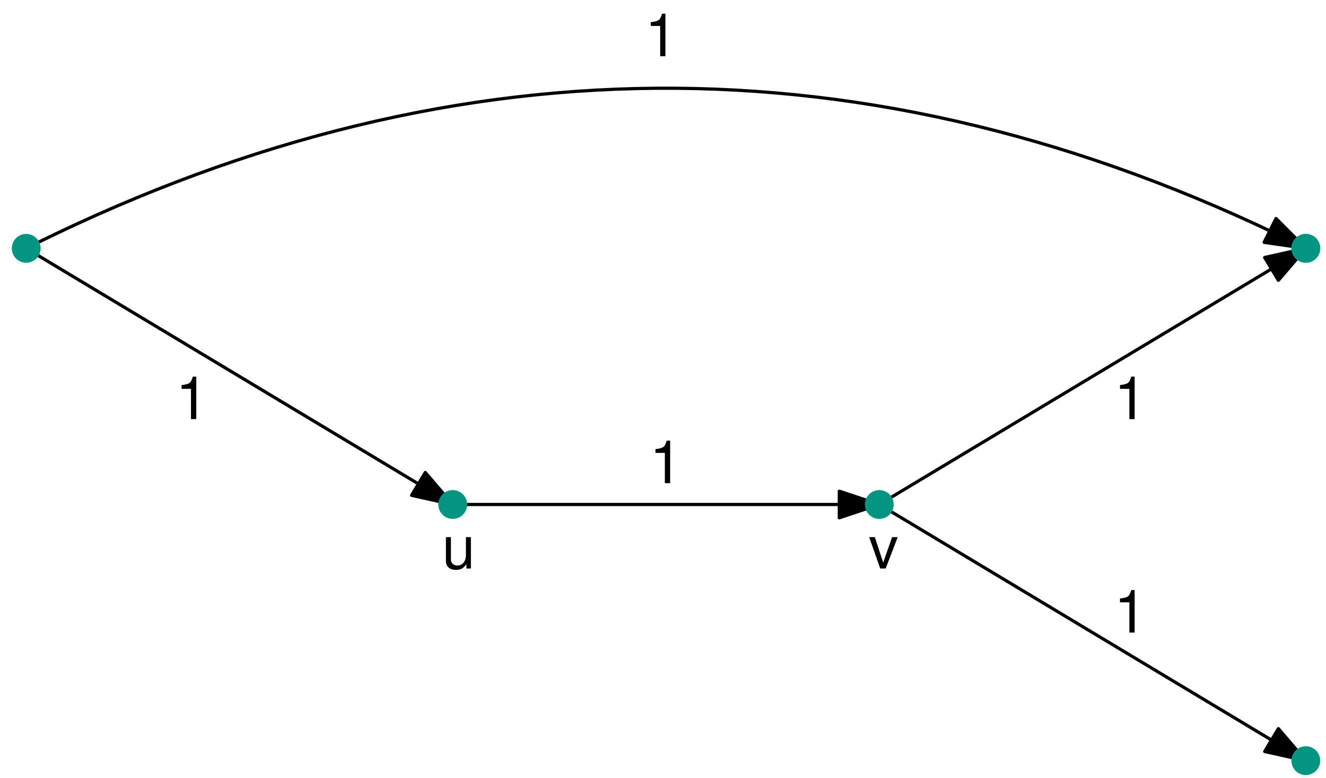
Edge Hierarchy Queries



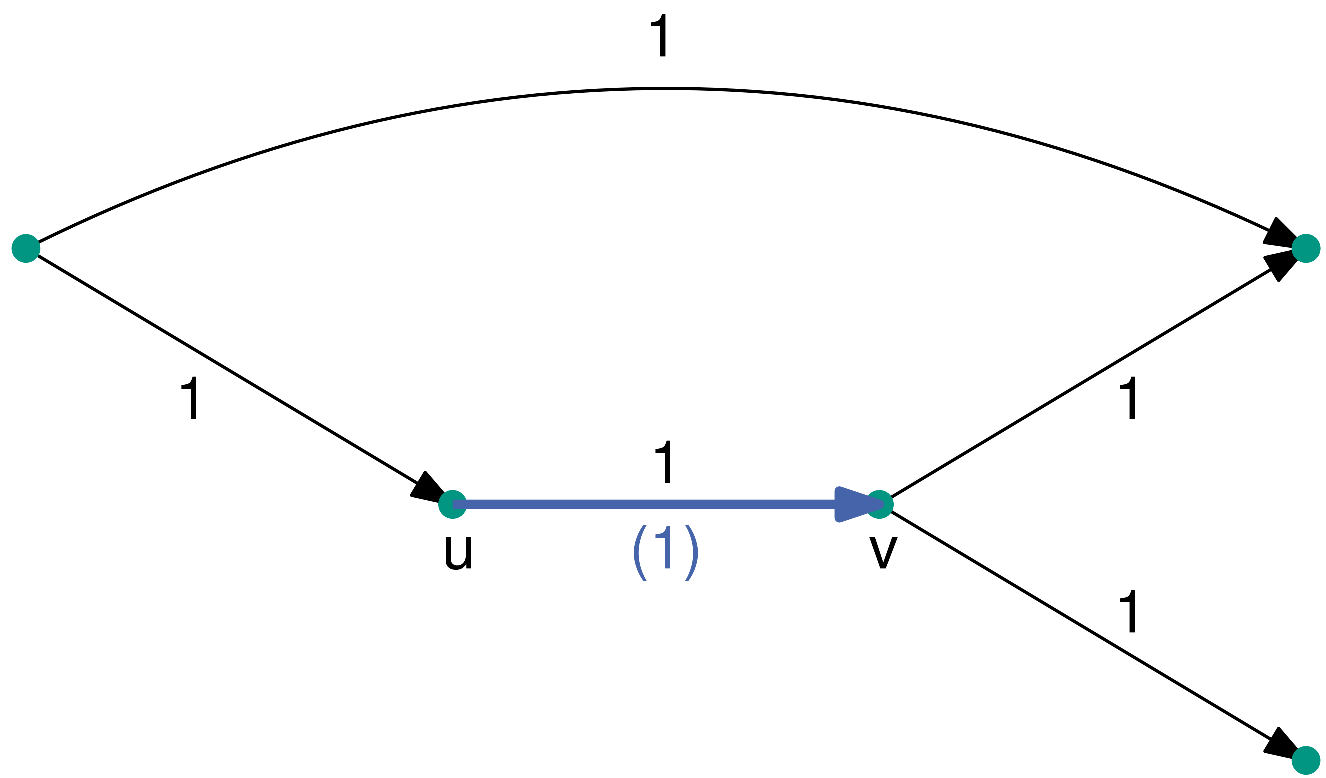
Edge Hierarchy Queries



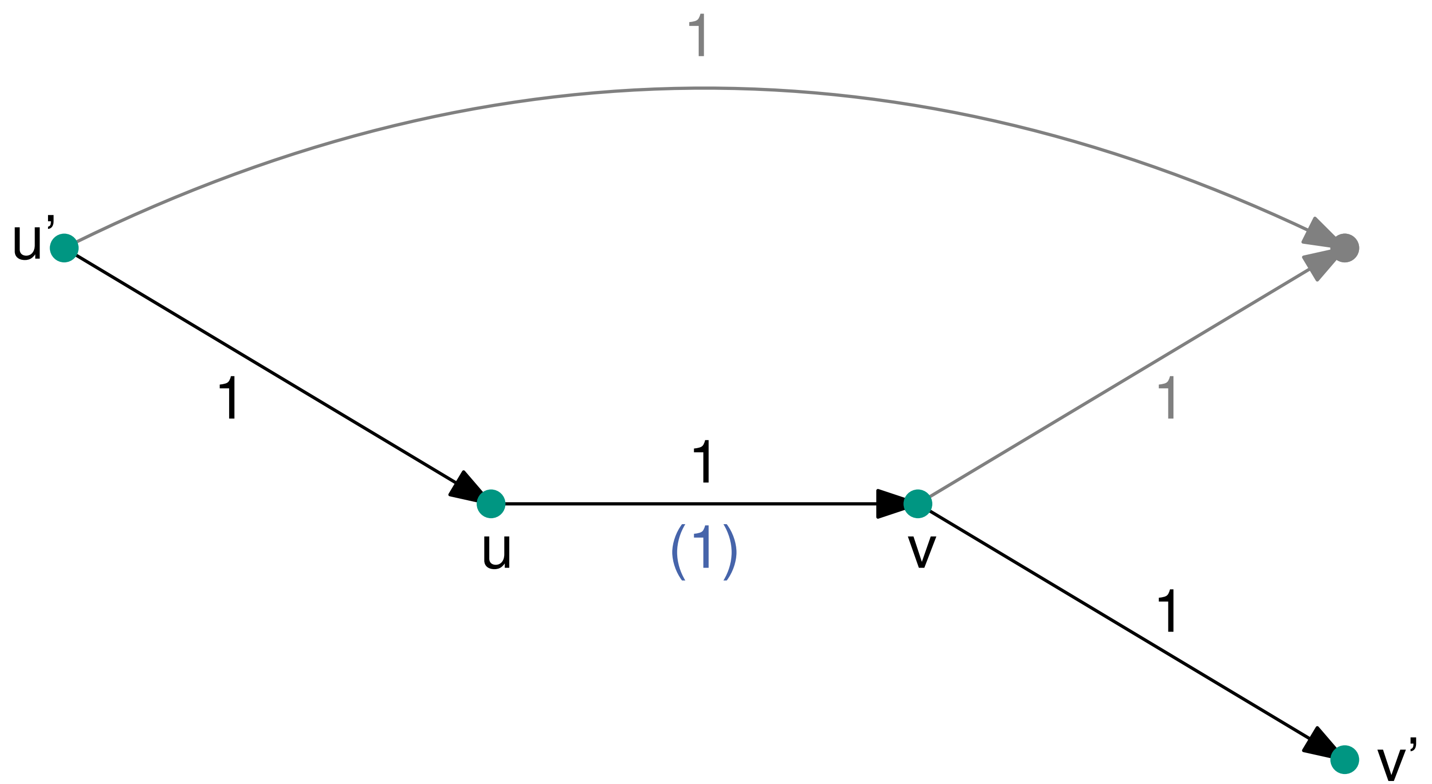
EH Edge Ranking



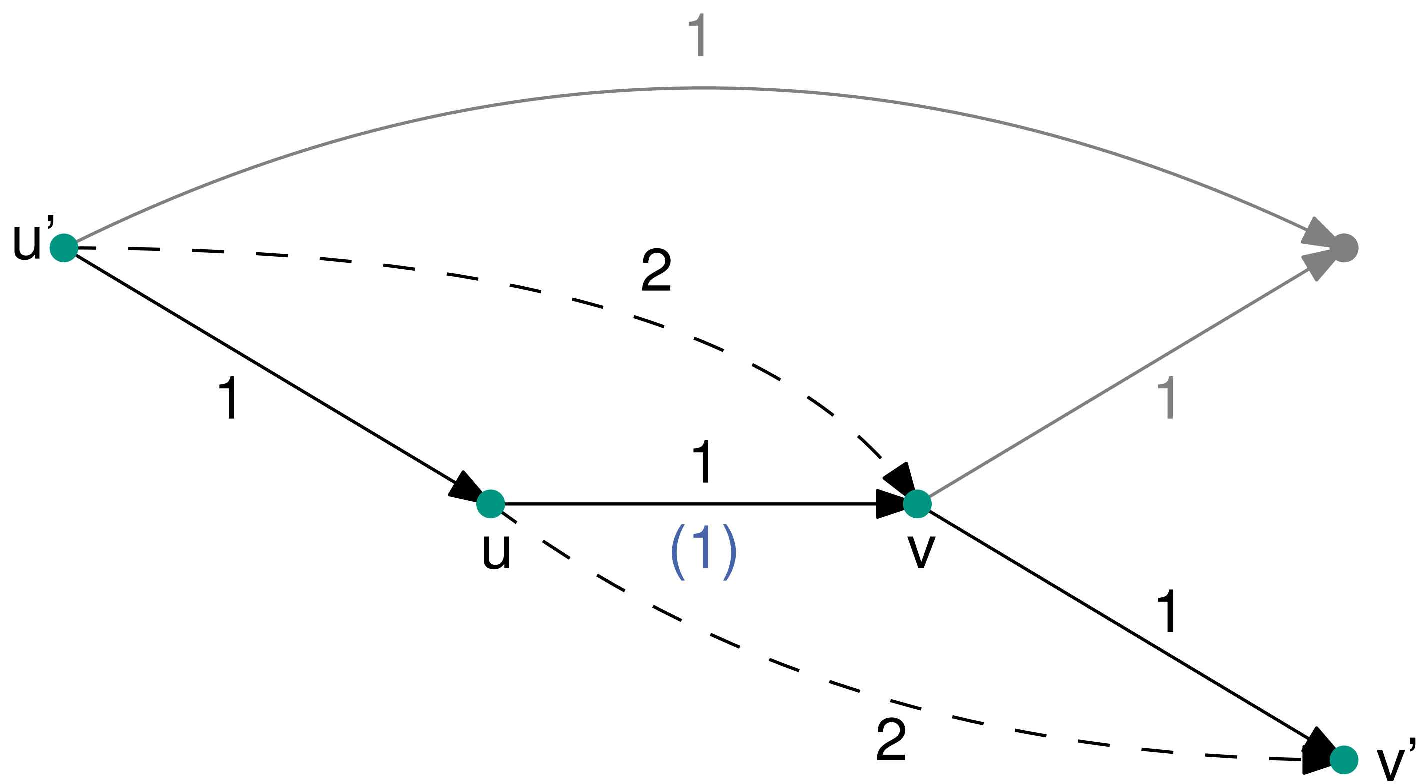
EH Edge Ranking



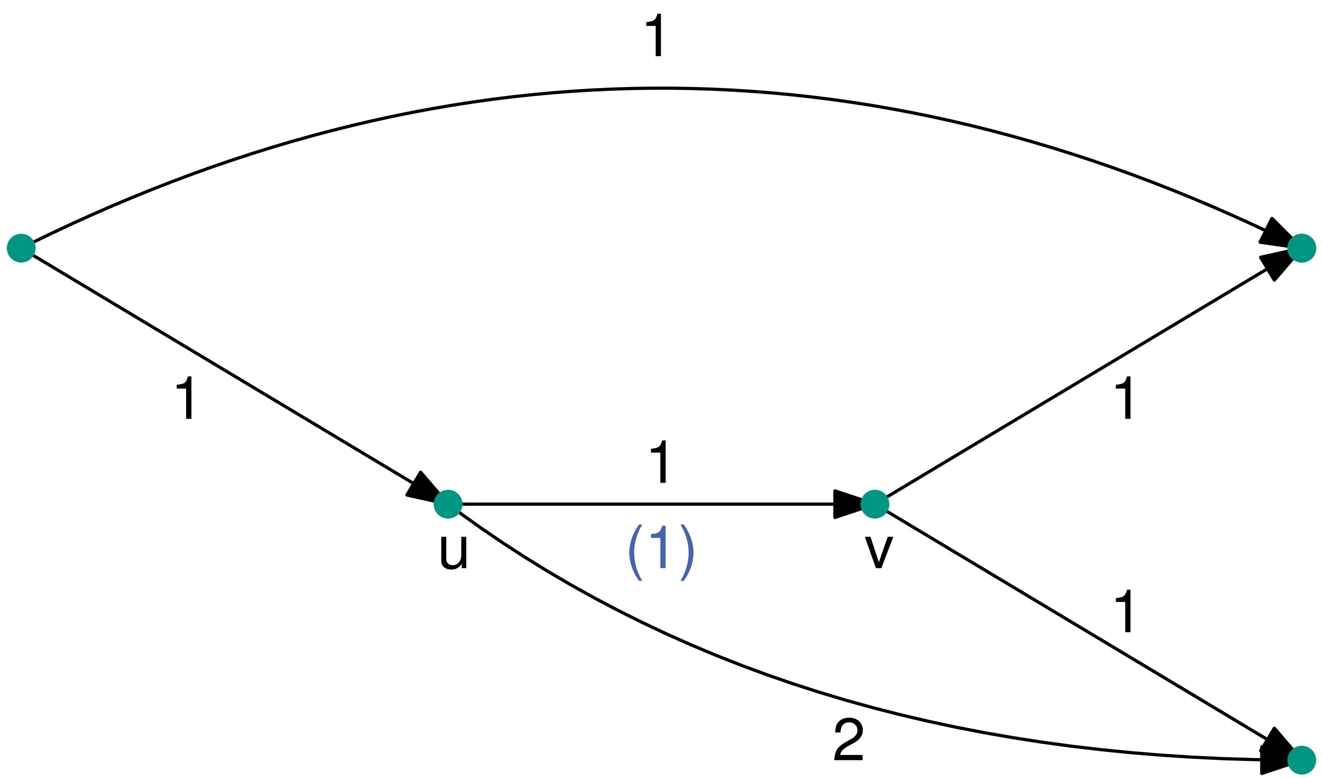
EH Edge Ranking



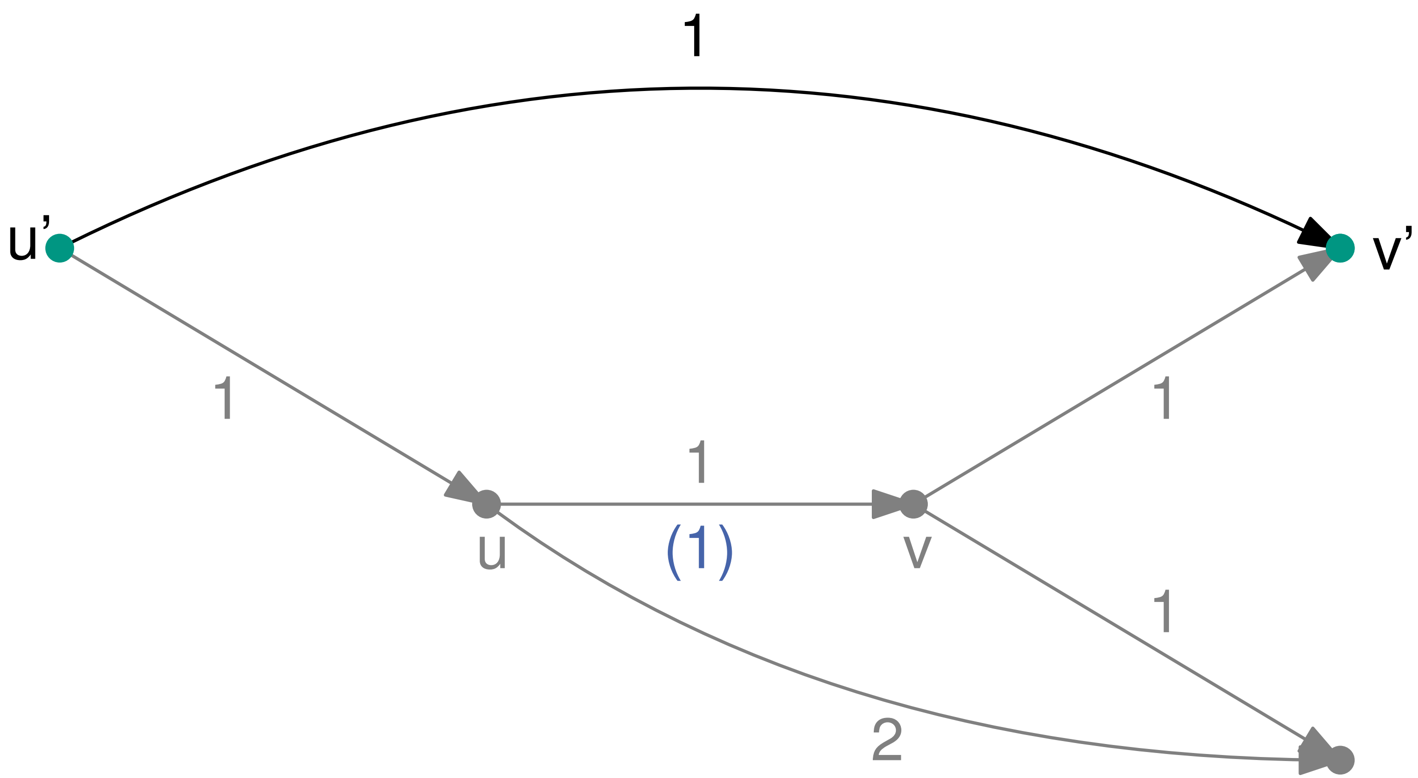
EH Edge Ranking



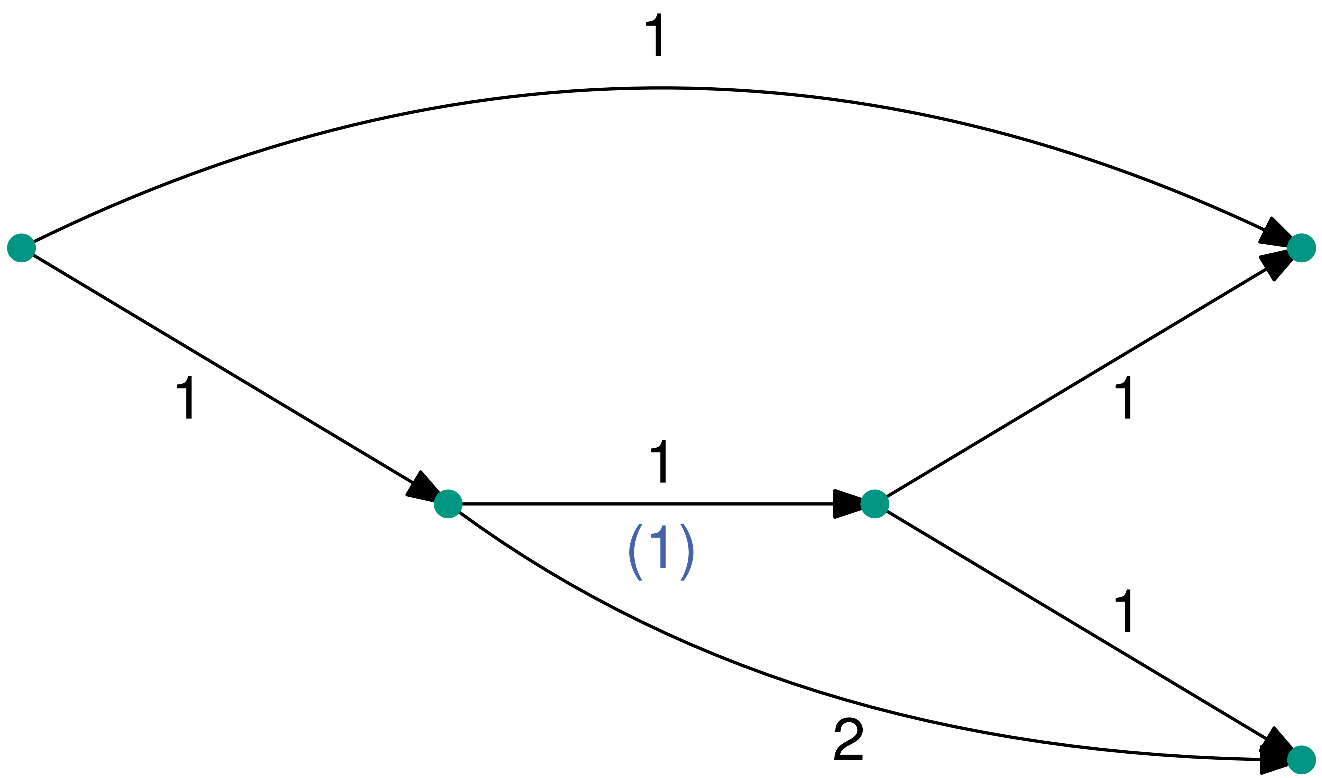
EH Edge Ranking



EH Edge Ranking



EH Edge Ranking



Construction Algorithm

Karlsruhe Institute of Technology

Algorithm 1: BuildEdgeHierarchy

```
currentRank ← 0;
while Unranked edges remain do
    Pick unranked edge (u, v);
    r(u, v) ← currentRank++ ;
    for all unranked edges (u′, u) do
        for all unranked edges (v, v′) do
            if (u′, u, v, v′) is shortest path then
                Add shortcut (u′, v) or (u, v′);
            end
        end
    end
end
```

Construction Algorithm

Karlsruhe Institute of Technology

Algorithm 1: BuildEdgeHierarchy

```
currentRank ← 0;
while Unranked edges remain do
    Pick unranked edge (u, v);
    r(u, v) ← currentRank++ ;
    for all unranked edges (u′, u) do
        for all unranked edges (v, v′) do
            if (u′, u, v, v′) is shortest path then
                Add shortcut (u′, v) or (u, v′);
            end
        end
    end
end
```

Construction Algorithm

Karlsruhe Institute of Technology

Algorithm 1: BuildEdgeHierarchy

```
currentRank ← 0;
while Unranked edges remain do
    Pick unranked edge (u, v);
    r(u, v) ← currentRank++ ;
    for all unranked edges (u′, u) do
        for all unranked edges (v, v′) do
            if (u′, u, v, v′) is shortest path then
                Add shortcut (u′, v) or (u, v′);
            end
        end
    end
end
```

Construction Algorithm

Algorithm 1: BuildEdgeHierarchy

```
currentRank ← 0;
while Unranked edges remain do
    Pick unranked edge (u, v);
    r(u, v) ← currentRank++ ;
    for all unranked edges (u′, u) do
        for all unranked edges (v, v′) do
            if (u′, u, v, v′) is shortest path then
                Add shortcut (u′, v) or (u, v′);
            end
        end
    end
end
```

Construction Algorithm

Algorithm 1: BuildEdgeHierarchy

```
currentRank ← 0;
while Unranked edges remain do
    Pick unranked edge (u, v);
    r(u, v) ← currentRank++ ;
    for all unranked edges (u′, u) do
        for all unranked edges (v, v′) do
            if (u′, u, v, v′) is shortest path then
                Add shortcut (u′, v) or (u, v′);
            end
        end
    end
end
```


Institute of Theoretical Informatics
Algorithmics Group

Construction Algorithm

Algorithm 1: BuildEdgeHierarchy

```
currentRank ← 0;
while Unranked edges remain do
    Pick unranked edge (u, v);
    r(u, v) ← currentRank++ ;
    for all unranked edges (u′, u) do
        for all unranked edges (v, v′) do
            if (u′, u, v, v′) is shortest path then
                Add shortcut (u′, v) or (u, v′);
            end
        end
    end
end
```


Institute of Theoretical Informatics
Algorithmics Group

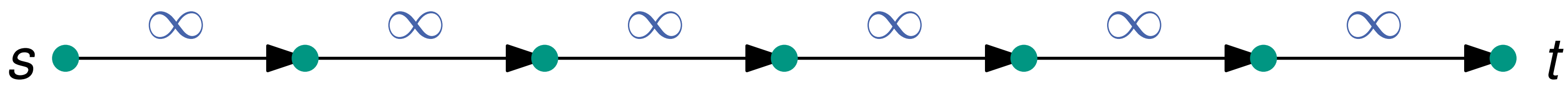
Construction Algorithm

Algorithm 1: BuildEdgeHierarchy

```
currentRank ← 0;
while Unranked edges remain do
    Pick unranked edge (u, v);
    r(u, v) ← currentRank++ ;
    for all unranked edges (u′, u) do
        for all unranked edges (v, v′) do
            if (u′, u, v, v′) is shortest path then
                Add shortcut (u′, v) or (u, v′);
            end
        end
    end
end
```


Institute of Theoretical Informatics
Algorithmics Group

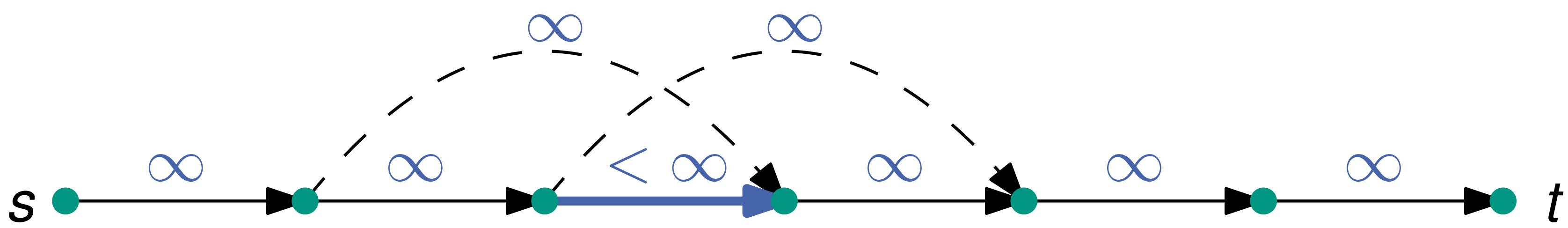
Proof Sketch



- Consider shortest s–t path and edge e being ranked
  - If e is in the middle: shortcut inserted
  - If e is at either end: lowest ranked edge on the path

    $\Rightarrow$ Edges in the middle get higher ranks

Proof Sketch

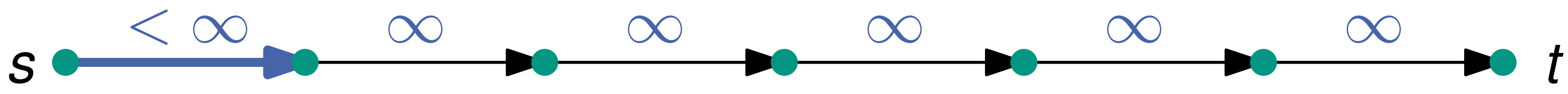


- Consider shortest *s*–*t* path and edge *e* being ranked
  - If *e* is in the middle: shortcut inserted
  - If *e* is at either end: lowest ranked edge on the path

    ⇒ Edges in the middle get higher ranks

Institute of Theoretical Informatics
Algorithmics Group

Proof Sketch



- Consider shortest s–t path and edge e being ranked
  - If e is in the middle: shortcut inserted
  - If e is at either end: lowest ranked edge on the path

  $\Rightarrow$ Edges in the middle get higher ranks

Don't Use the Overlay Graph

Karlsruhe Institute of Technology

Algorithm 2: BuildEdgeHierarchy

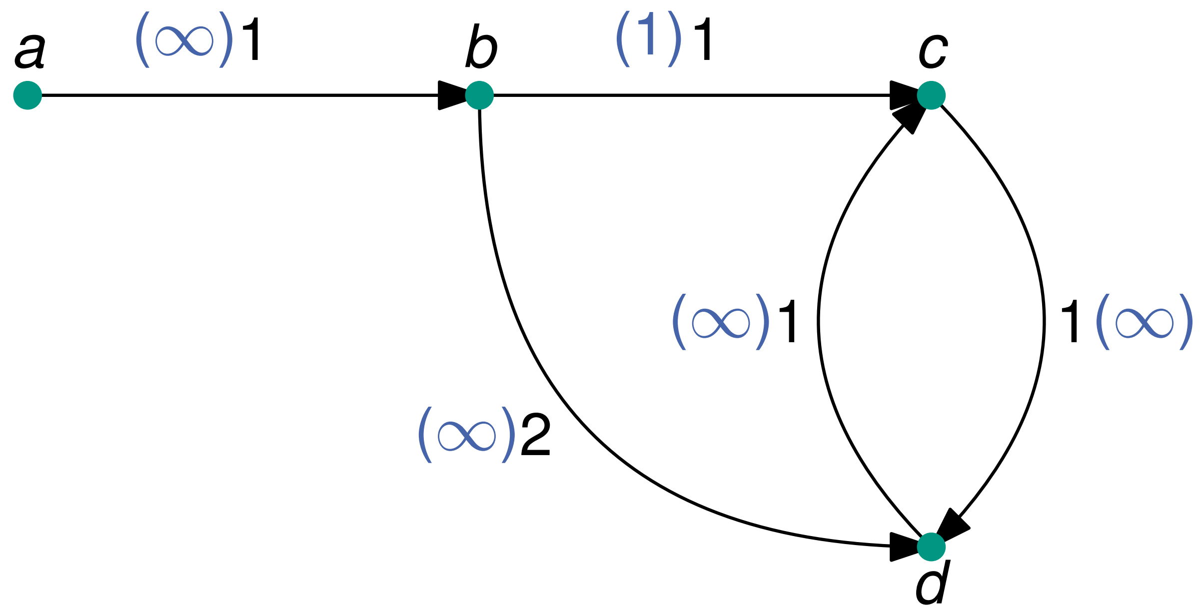
---

currentRank $\leftarrow$ 0;
while Unranked edges remain **do**
  Pick unranked edge (u, v);
  $r(u, v) \leftarrow$ currentRank++ ;
  for all unranked edges (u', u) **do**
    for all unranked edges (v, v') **do**
      if (u', u, v, v') is shortest path **then**
        Add shortcut (u', v) or (u, v');
      end
    end
  end
end

Check whole graph

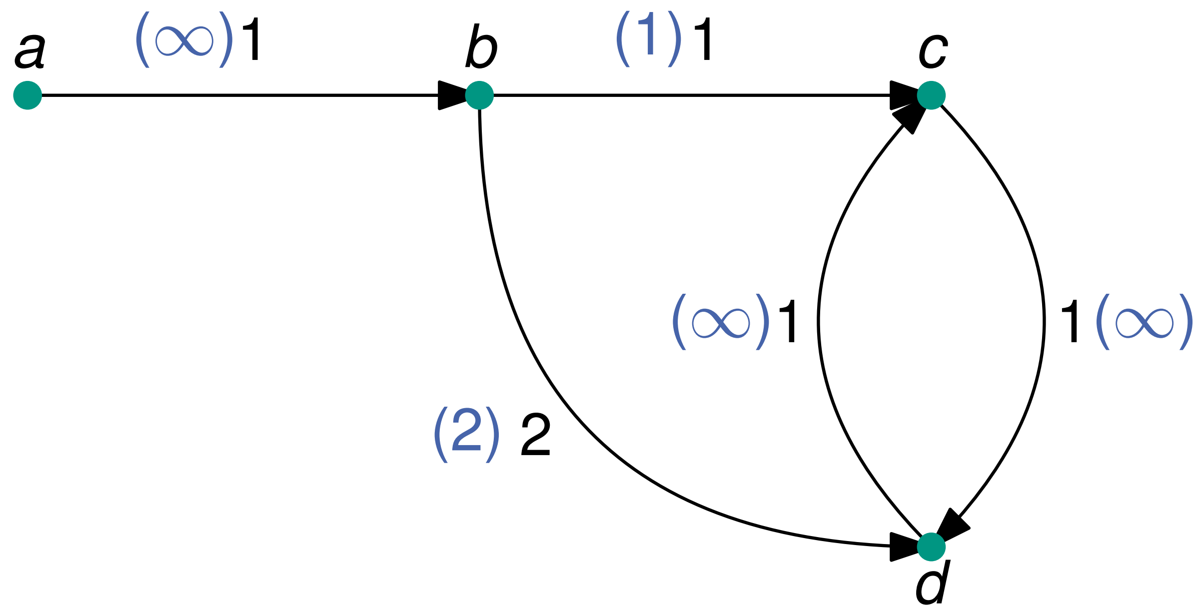
Institute of Theoretical Informatics
Algorithmics Group

Don’t Use the Overlay Graph



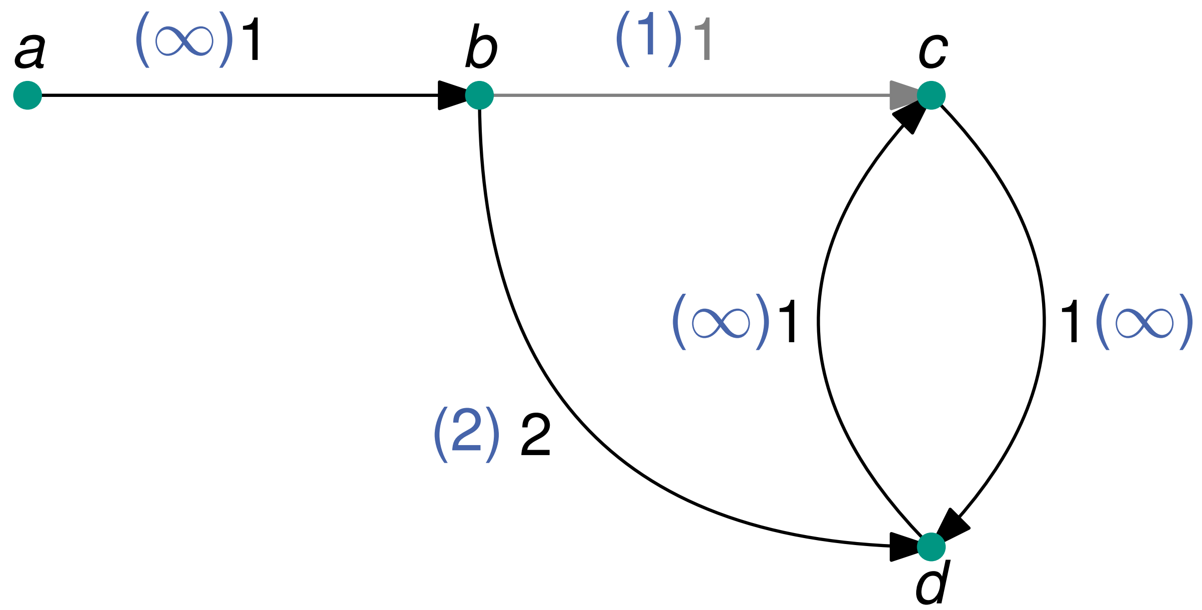
- Overlay graph doesn’t contain shortest paths for all vertex pairs
  - Can use EH query on partially constructed EH

Don't Use the Overlay Graph



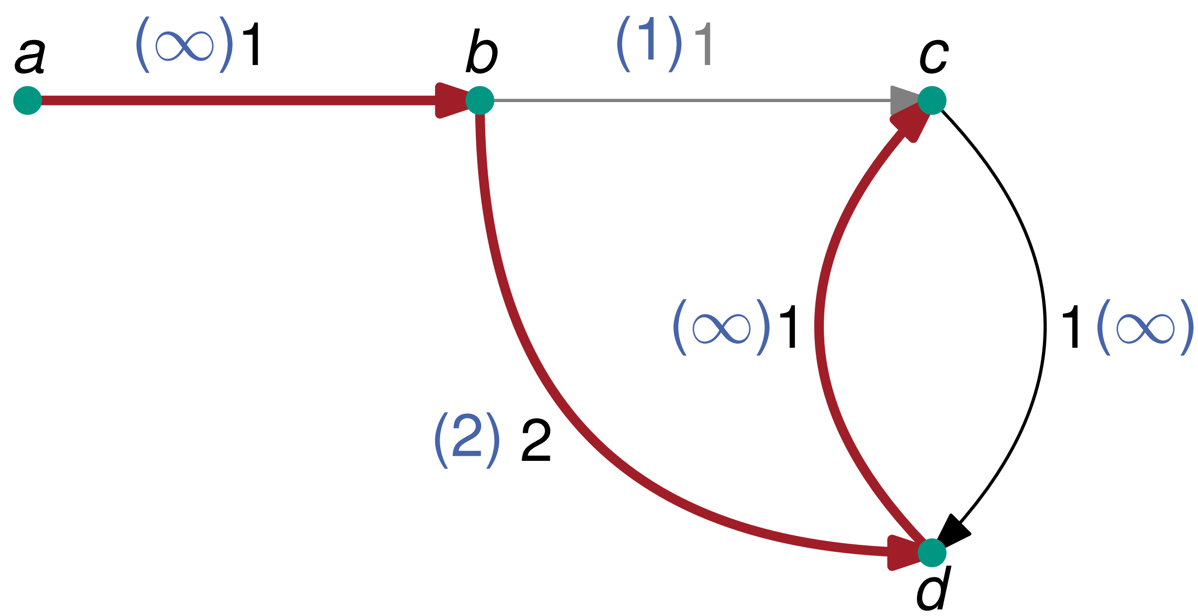
- Overlay graph doesn't contain shortest paths for all vertex pairs
  - Can use EH query on partially constructed EH

Don't Use the Overlay Graph



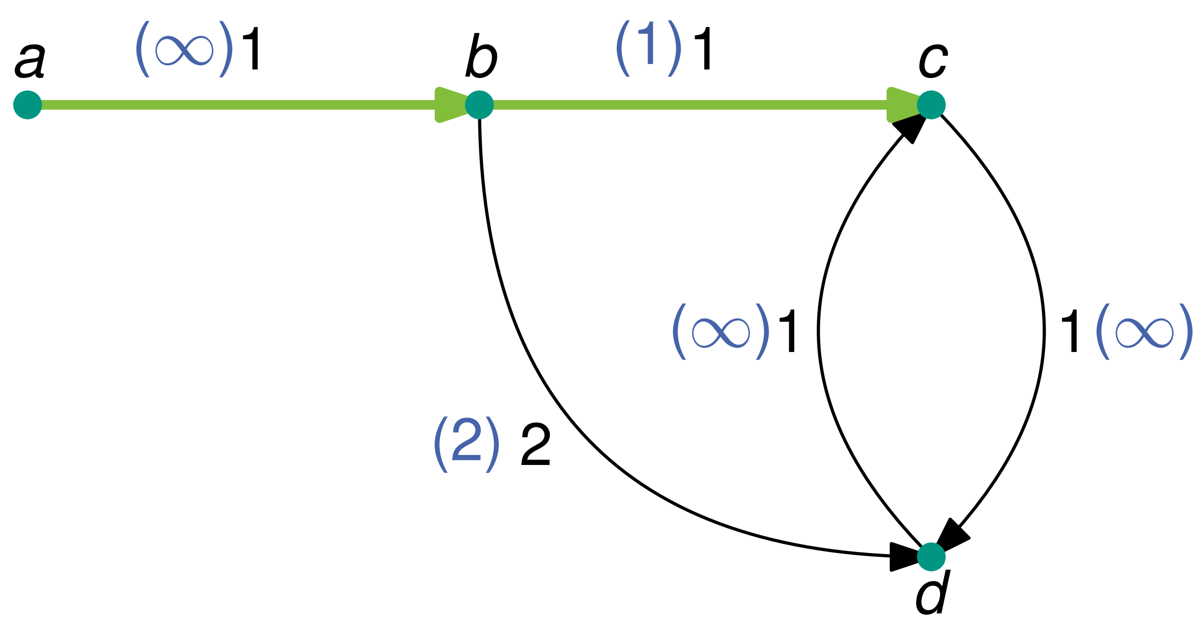
- Overlay graph doesn't contain shortest paths for all vertex pairs
  - Can use EH query on partially constructed EH

Don't Use the Overlay Graph



- Overlay graph doesn't contain shortest paths for all vertex pairs
  - Can use EH query on partially constructed EH

Don’t Use the Overlay Graph



- Overlay graph doesn’t contain shortest paths for all vertex pairs
  - Can use EH query on partially constructed EH

Filling Out the Blanks

Algorithm 3: BuildEdgeHierarchy

currentRank ← 0;
while Unranked edges remain **do**
  Pick unranked edge (u, v);
  $r(u, v) \leftarrow$ currentRank++ ;
  for all unranked edges (u', u) **do**
    for all unranked edges (v, v') **do**
      if (u', u, v, v') is shortest path **then**
        Add shortcut (u', v) or (u, v')
      end
    end
  end
end

Institute of Theoretical Informatics
Algorithmics Group

Filling Out the Blanks

Algorithm 3: BuildEdgeHierarchy

```
currentRank ← 0;
while Unranked edges remain do
    Pick unranked edge (u, v);
    r(u, v) ← currentRank++ ;
    for all unranked edges (u′, u) do
        for all unranked edges (v, v′) do
            if (u′, u, v, v′) is shortest path then
                Add shortcut (u′, v) or (u, v′)
            end
        end
    end
end
```

How?

Institute of Theoretical Informatics
Algorithmics Group

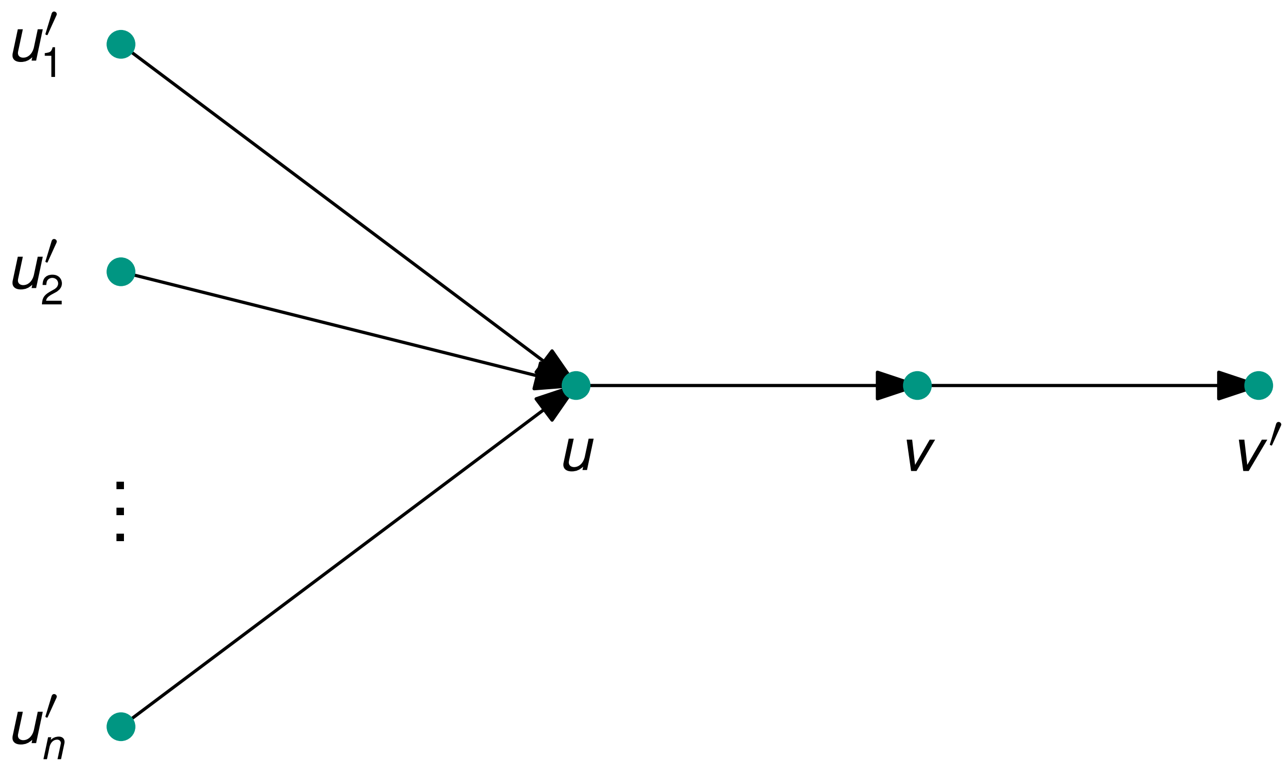
Filling Out the Blanks

Algorithm 3: BuildEdgeHierarchy

currentRank ← 0;
while Unranked edges remain **do**
    Pick unranked edge (u, v);
    $r(u, v)$ ← currentRank++ ;
    for all unranked edges (u', u) **do**
        for all unranked edges (v, v') **do**
            if (u', u, v, v') is shortest path **then**
                Add shortcut (u', v) or (u, v')
            end
        end
    end
end

How?

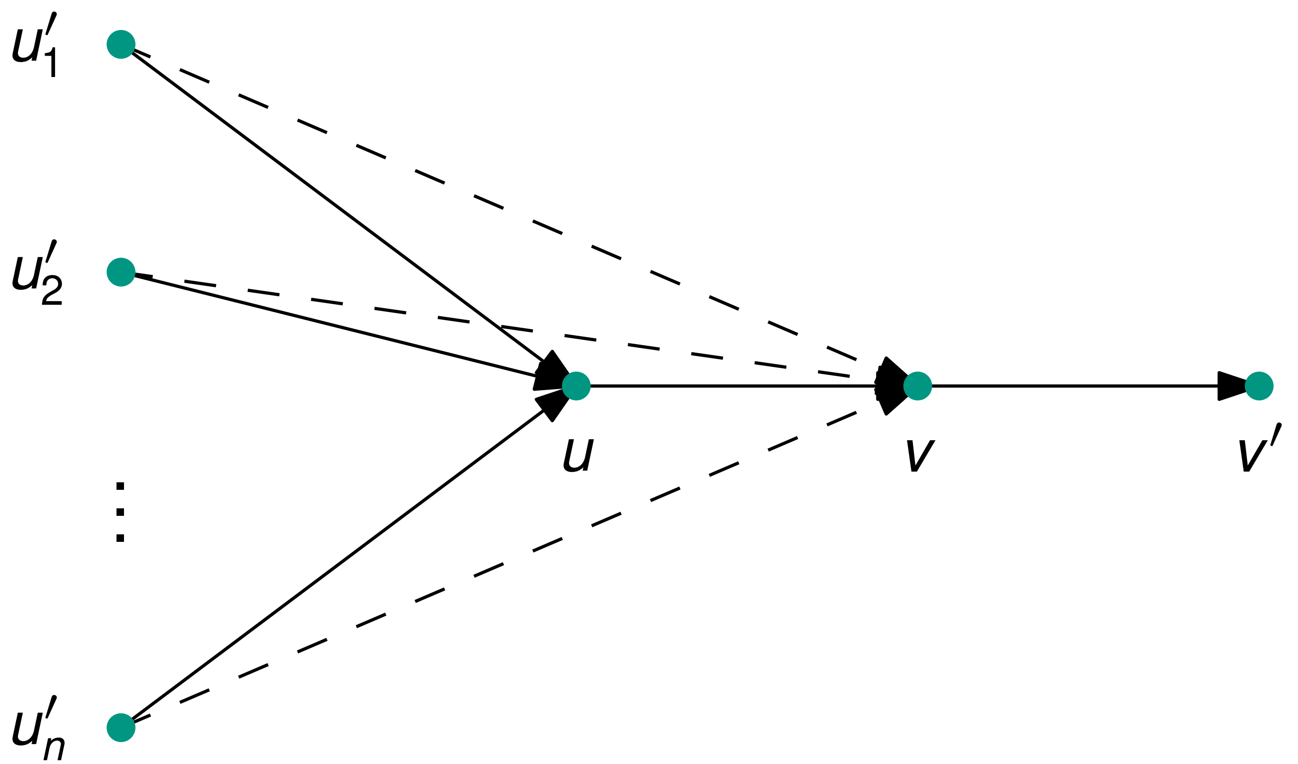
What Can Go Wrong?



- n shortcuts or 1 shortcut

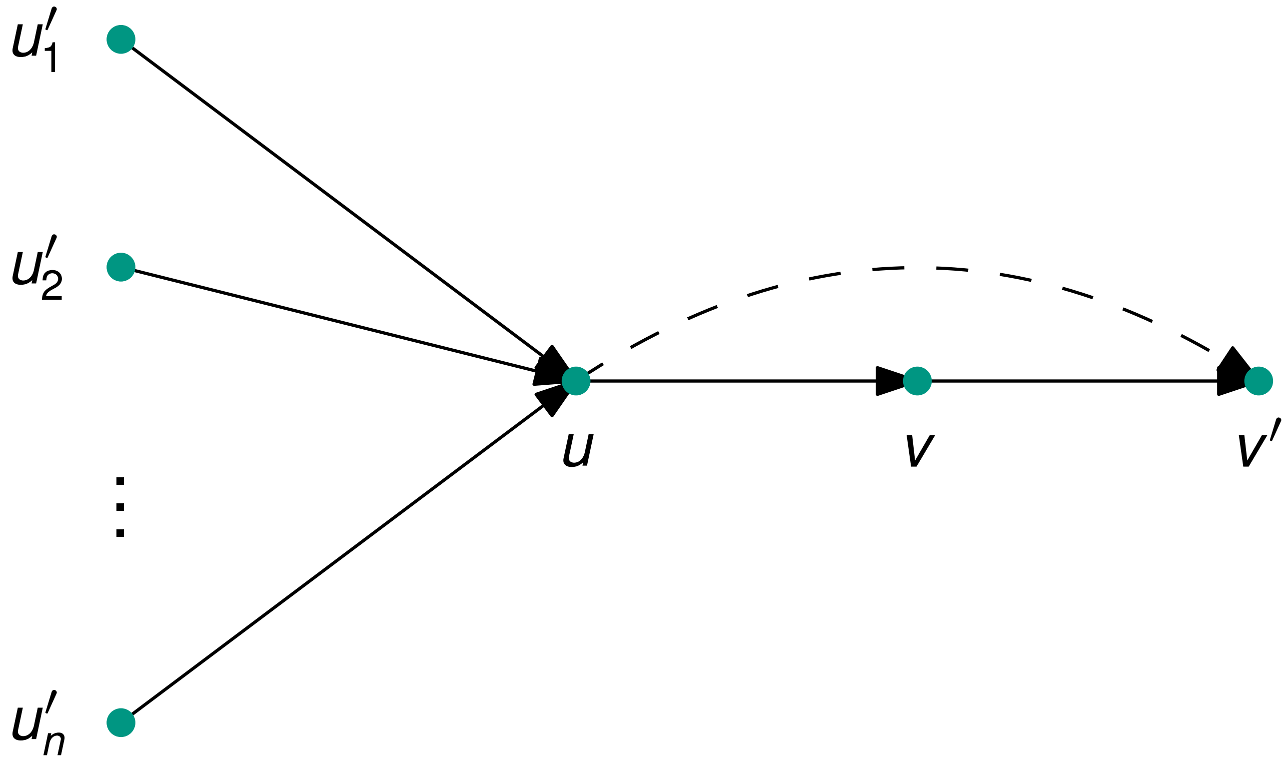
What Can Go Wrong?

Karlsruhe Institute of Technology



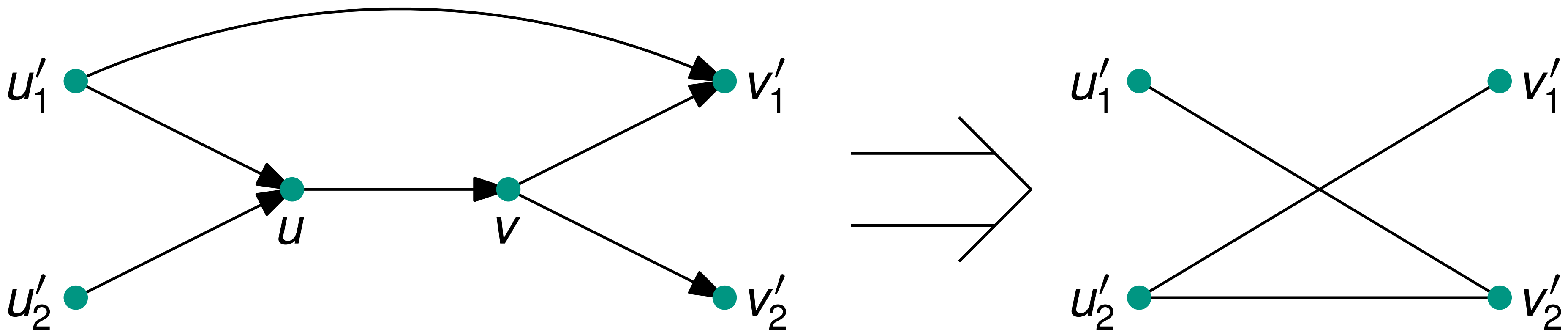
- n shortcuts or 1 shortcut

What Can Go Wrong?



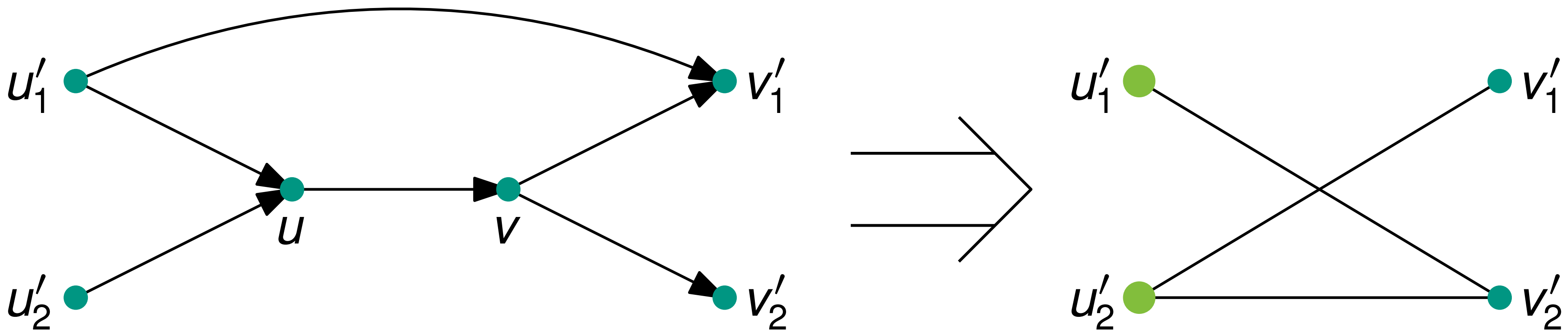
- n shortcuts or 1 shortcut

Minimizing the Number of Shortcuts Added



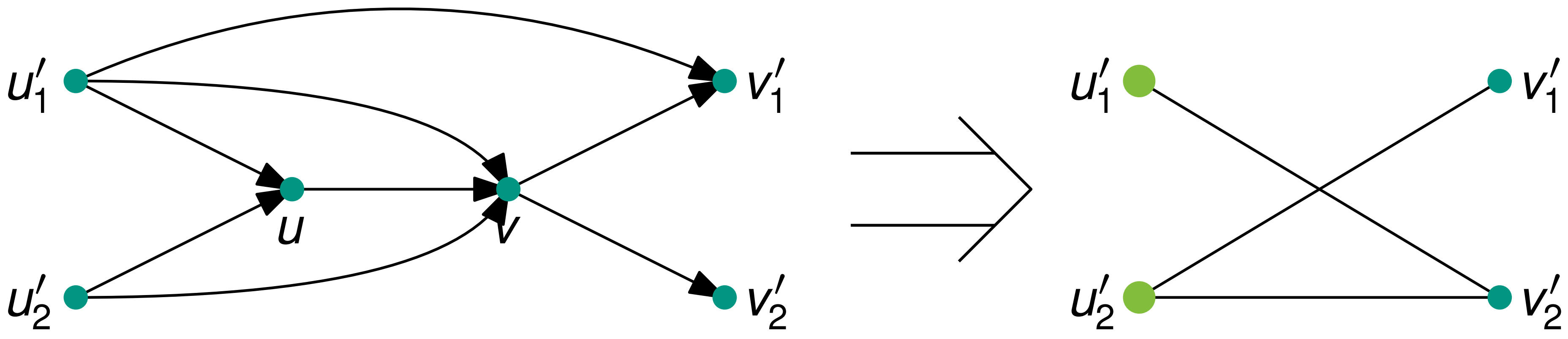
- Build bipartite Minimum Vertex Cover (MVC) problem
- Add shortcuts to vertices in MVC
- König's Theorem $\Rightarrow$ polytime

Minimizing the Number of Shortcuts Added



- Build bipartite Minimum Vertex Cover (MVC) problem
- Add shortcuts to vertices in MVC
- König's Theorem $\Rightarrow$ polytime

Minimizing the Number of Shortcuts Added



- Build bipartite Minimum Vertex Cover (MVC) problem
- Add shortcuts to vertices in MVC
- König's Theorem $\Rightarrow$ polytime

Preprocessing

Travel Time

	Graph	Prepr. [s] EH	Prepr. [s] CH	$\lvert E \rvert$ [M] EH	$\lvert E \rvert$ [M] CH
Orig.	USA	7145	674	104.5	104.0
Orig.	EUROPE	3171	453	70.3	70.3
Turns	USA	45904	15462	270.3	404.3
Turns	EUROPE	17822	4743	194.0	249.1

Distance

	Graph	Prepr. [s] EH	Prepr. [s] CH	$\lvert E \rvert$ [M] EH	$\lvert E \rvert$ [M] CH
Orig.	USA	21041	1537	110.8	110.7
Orig.	EUROPE	14487	2152	79.6	79.6
Turns	USA	58025	30869	251.1	398.3
Turns	EUROPE	24757	13266	172.3	267.2

* EH Preprocessing uses CH queries

Intel Xeon E5-4640, 2.4 GHz

Preprocessing

Travel Time

	Graph	Prepr. [s] EH	Prepr. [s] CH	$\lvert E \rvert$ [M] EH	$\lvert E \rvert$ [M] CH
Orig.	USA	7145	674	104.5	104.0
Orig.	EUROPE	3171	453	70.3	70.3
Turns	USA	45904	15462	270.3	404.3
Turns	EUROPE	17822	4743	194.0	249.1

Distance

	Graph	Prepr. [s] EH	Prepr. [s] CH	$\lvert E \rvert$ [M] EH	$\lvert E \rvert$ [M] CH
Orig.	USA	21041	1537	110.8	110.7
Orig.	EUROPE	14487	2152	79.6	79.6
Turns	USA	58025	30869	251.1	398.3
Turns	EUROPE	24757	13266	172.3	267.2

* EH Preprocessing uses CH queries
Intel Xeon E5-4640, 2.4 GHz

Preprocessing

Travel Time

	Graph	Prepr. [s] EH	Prepr. [s] CH	$|E|$ [M] EH	$|E|$ [M] CH
Orig.	USA	7145	674	104.5	104.0
Orig.	EUROPE	3171	453	70.3	70.3
Turns	USA	45904	15462	270.3	404.3
Turns	EUROPE	17822	4743	194.0	249.1

Distance

	Graph	Prepr. [s] EH	Prepr. [s] CH	$|E|$ [M] EH	$|E|$ [M] CH
Orig.	USA	21041	1537	110.8	110.7
Orig.	EUROPE	14487	2152	79.6	79.6
Turns	USA	58025	30869	251.1	398.3
Turns	EUROPE	24757	13266	172.3	267.2

* EH Preprocessing uses CH queries
Intel Xeon E5-4640, 2.4 GHz

Preprocessing

Travel Time

	Graph	Prepr. [s] EH	Prepr. [s] CH	$\lvert E \rvert$ [M] EH	$\lvert E \rvert$ [M] CH
Orig.	USA	7145	674	104.5	104.0
Orig.	EUROPE	3171	453	70.3	70.3
Turns	USA	45904	15462	270.3	404.3
Turns	EUROPE	17822	4743	194.0	249.1

Distance

	Graph	Prepr. [s] EH	Prepr. [s] CH	$\lvert E \rvert$ [M] EH	$\lvert E \rvert$ [M] CH
Orig.	USA	21041	1537	110.8	110.7
Orig.	EUROPE	14487	2152	79.6	79.6
Turns	USA	58025	30869	251.1	398.3
Turns	EUROPE	24757	13266	172.3	267.2

* EH Preprocessing uses CH queries
Intel Xeon E5-4640, 2.4 GHz

Preprocessing

Travel Time

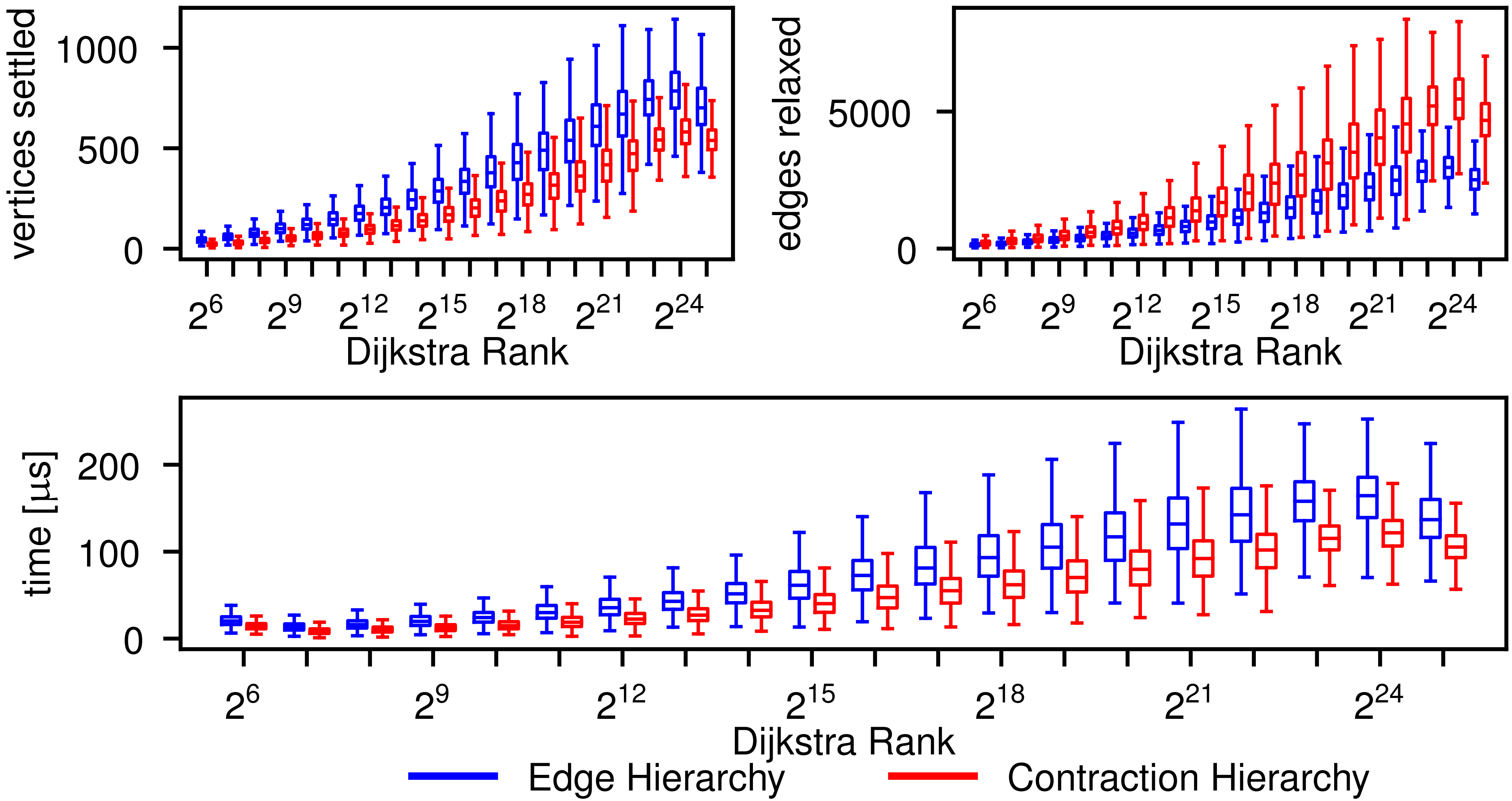
	Graph	Prepr. [s] EH	Prepr. [s] CH	$\lvert E \rvert$ [M] EH	$\lvert E \rvert$ [M] CH
Orig.	USA	7145	674	104.5	104.0
Orig.	EUROPE	3171	453	70.3	70.3
Turns	USA	45904	15462	270.3	404.3
Turns	EUROPE	17822	4743	194.0	249.1

Distance

	Graph	Prepr. [s] EH	Prepr. [s] CH	$\lvert E \rvert$ [M] EH	$\lvert E \rvert$ [M] CH
Orig.	USA	21041	1537	110.8	110.7
Orig.	EUROPE	14487	2152	79.6	79.6
Turns	USA	58025	30869	251.1	398.3
Turns	EUROPE	24757	13266	172.3	267.2

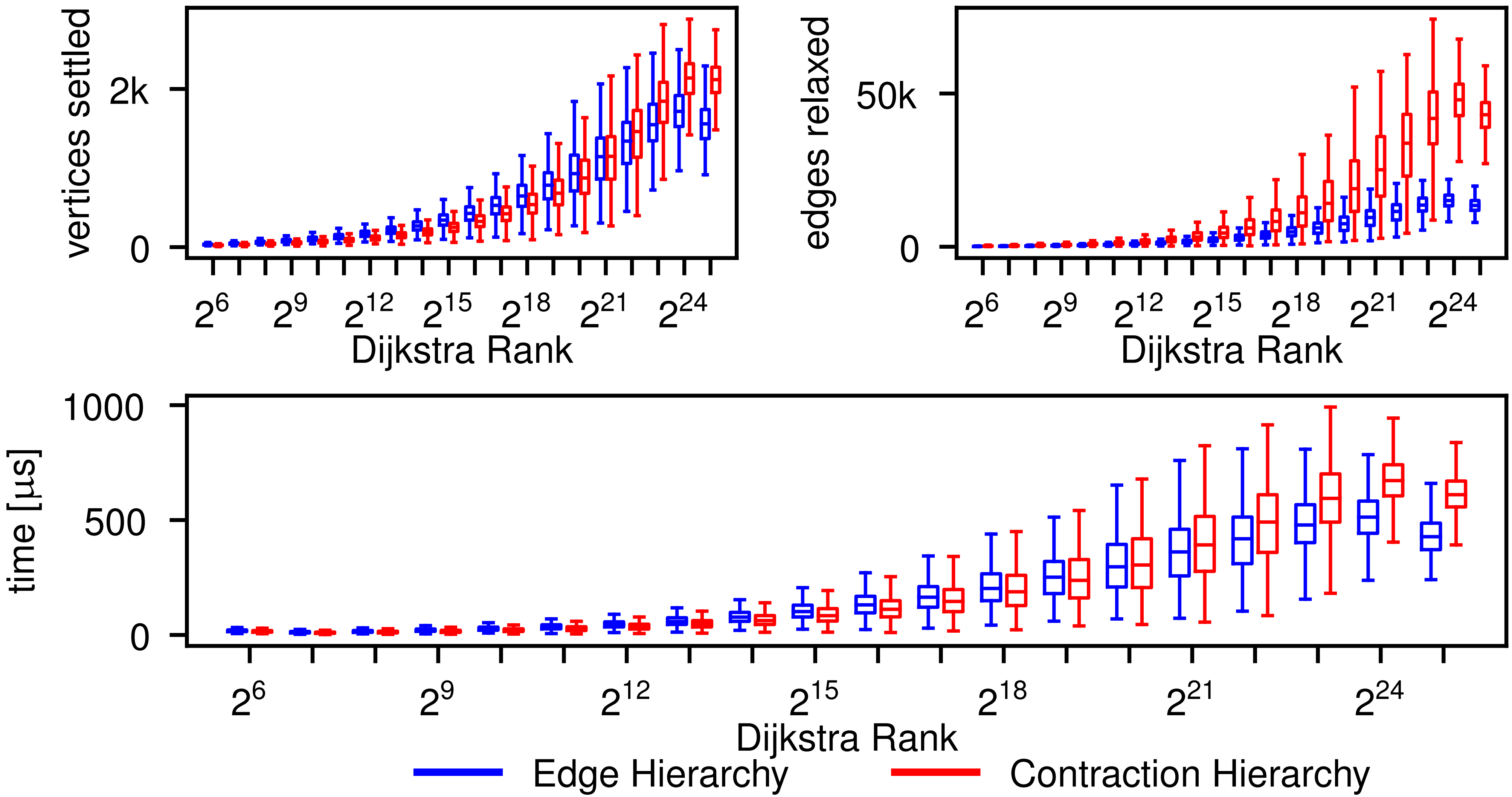
* EH Preprocessing uses CH queries
Intel Xeon E5-4640, 2.4 GHz

Queries (Travel Time + Turns)



Instance: Western Europe by PTV

Queries (Distance + Turns)



Instance: Western Europe by PTV

The End

- **In the Paper**
  - Edge Selection
  - Partial Stall on Demand
- **Future Work**
  - Cache efficient distance labels
  - Applications with more expensive edge relaxations
- **Questions?**

Institute of Theoretical Informatics
Algorithmics Group

References

- Geisberger, R., Sanders, P., Schultes, D., & Vetter, C. (2012). Exact routing in large road networks using contraction hierarchies. Transportation Science, 46(3), 388-404.
- Highway picture by Free-Photos (pixabay.com)
- Small road picture by Jan Walldén
- Alley and road between Italian style houses picture by fshoq.com
- Karlsruhe road network from i11www.iti.kit.edu/teaching/sommer2018/routenplanung/index
- TUM Campusplan from portal.mytum.de/campus/garching