Walking Through Middleboxes

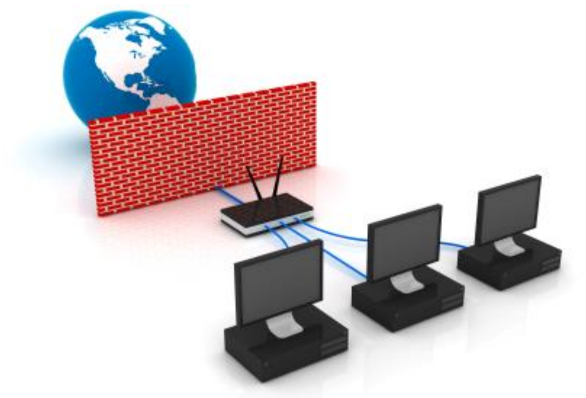
Klaus-T. Foerster, University of Vienna

25 Jul 2018 @ Cornell University. Host: Nate Foster

Walking Through ~~Middleboxes~~ Ithaca

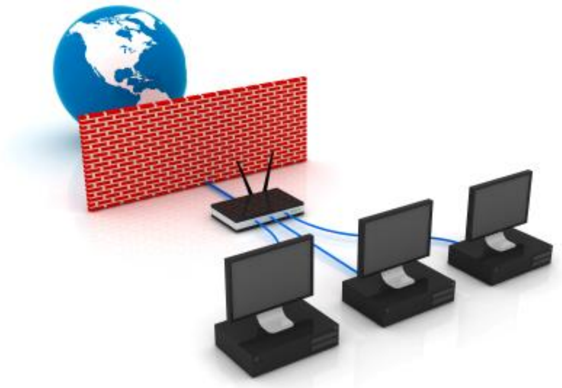
Practical Motivation: Middleboxes

- Classical case: Routing inflexible
  - Place at the edge / along the route

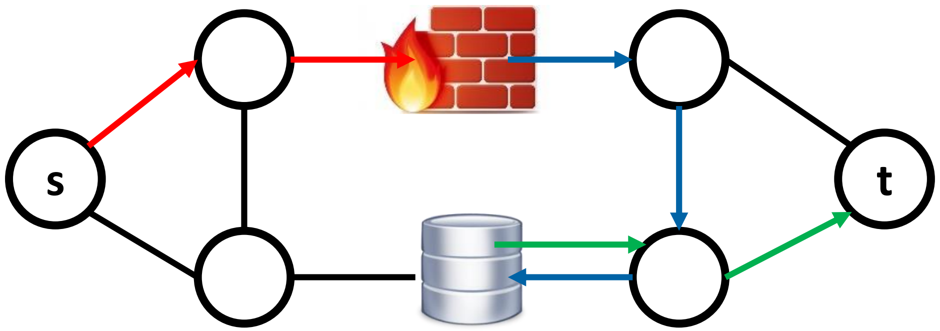


Practical Motivation: Middleboxes

- Classical case: Routing inflexible
  - Place at the edge / along the route

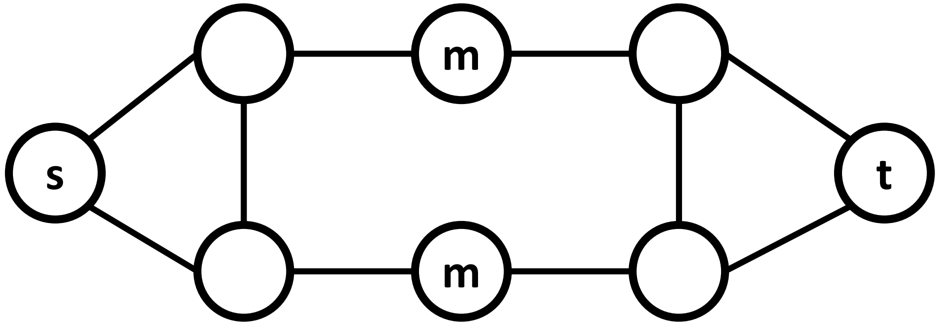


- Software-Defined Networking paradigm:
  - Arbitrary routes

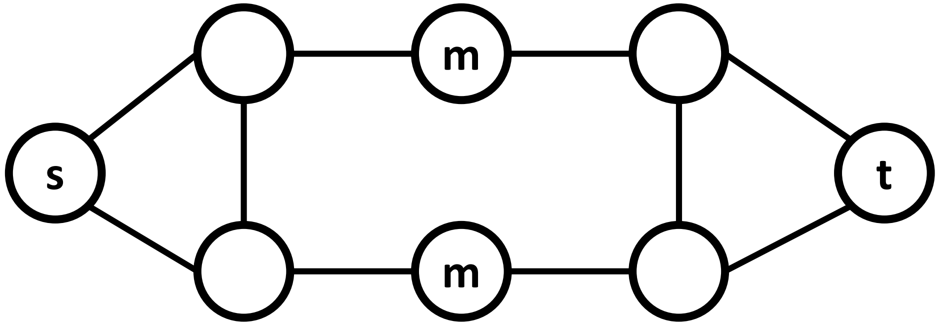


- Also via: Source routing
- To some extent: Segment routing

Walking Through Middleboxes



Walking Through Middleboxes

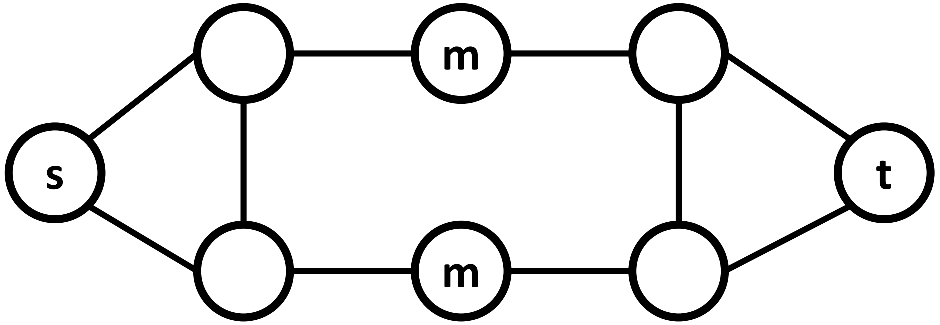


This talk: *Algorithms to find routes*

Related: *Verification / What-if analysis of waypoint properties in MPLS routing*

- Unlike general SDN, MPLS can be modeled via push-down automata
- S. Schmid and J. Srba: "*Polynomial-Time What-If Analysis for Prefix-Manipulating MPLS Networks*", INFOCOM 2018.

Walking Through Middleboxes



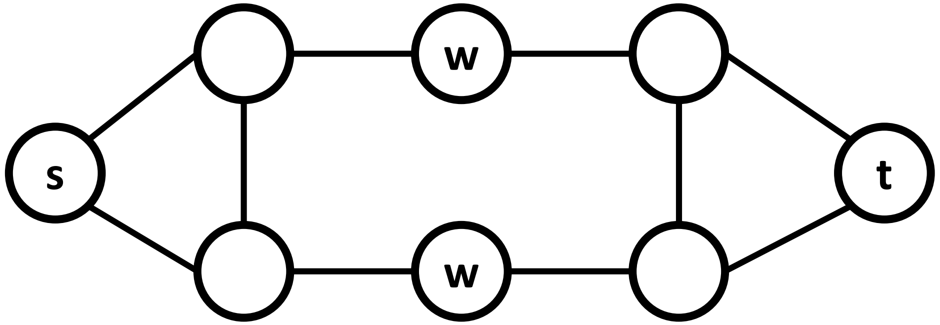
This talk: *Algorithms to find routes*

Related: *Verification / What-if analysis of waypoint properties in MPLS routing*

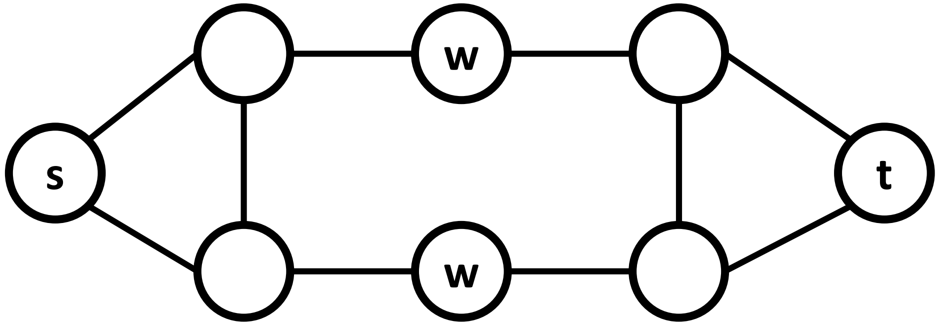
- Unlike general SDN, MPLS can be modeled via push-down automata
- S. Schmid and J. Srba: "*Polynomial-Time What-If Analysis for Prefix-Manipulating MPLS Networks*", INFOCOM 2018.

Also already some work on joint problem with waypoint placement

Walking Through ~~Middleboxes~~ Waypoints



Walking Through Waypoints

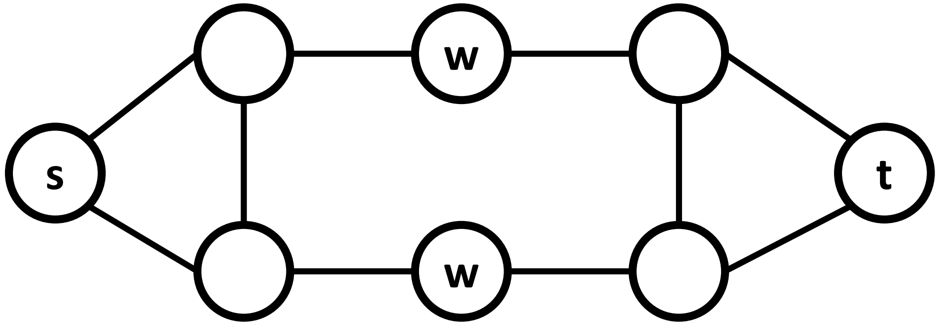


- Given: Undirected graph with waypoints
- Find: Shortest s-t walk through all waypoints

Walking Through Waypoints

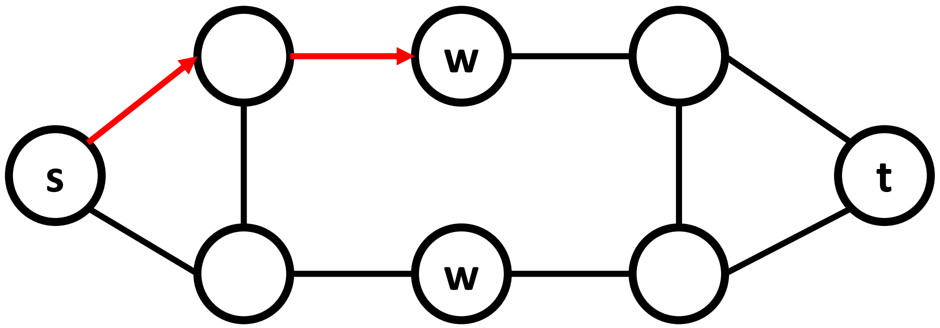
Later also: directed

- Given: Undirected graph with waypoints
- Find: Shortest s-t walk through all waypoints



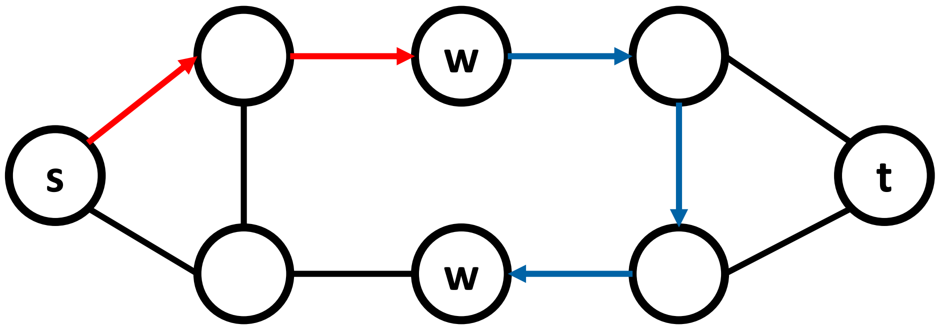
universität wien

Walking Through Waypoints



- Given: Undirected graph with waypoints
- Find: Shortest s-t walk through all waypoints

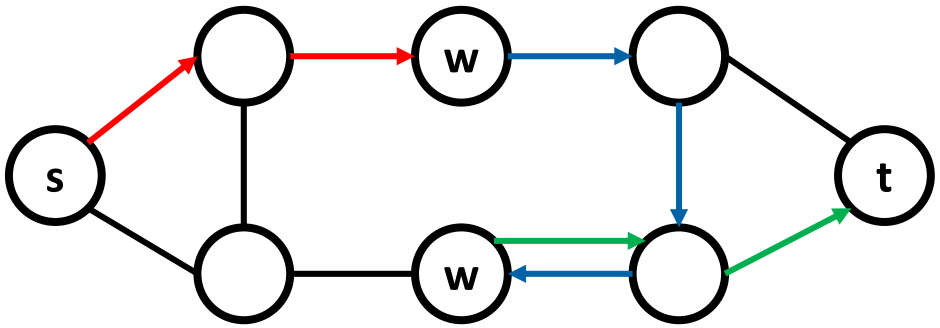
Walking Through Waypoints



- Given: Undirected graph with waypoints
- Find: Shortest s-t walk through all waypoints

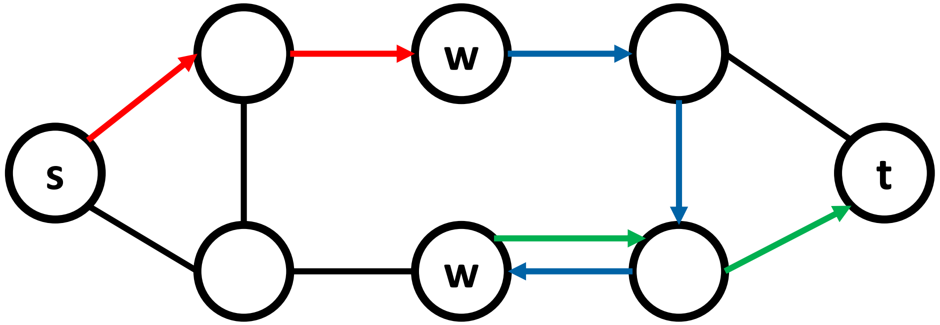
universität wien

Walking Through Waypoints



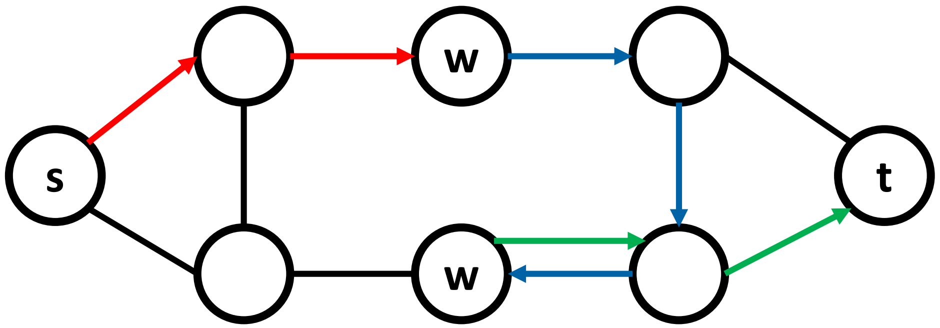
- Given: Undirected graph with waypoints
- Find: Shortest s-t walk through all waypoints

Walking Through Waypoints



- Given: Undirected graph with waypoints
- Find: Shortest s-t walk through all waypoints
- Twist: Respect capacities

Walking Through Waypoints

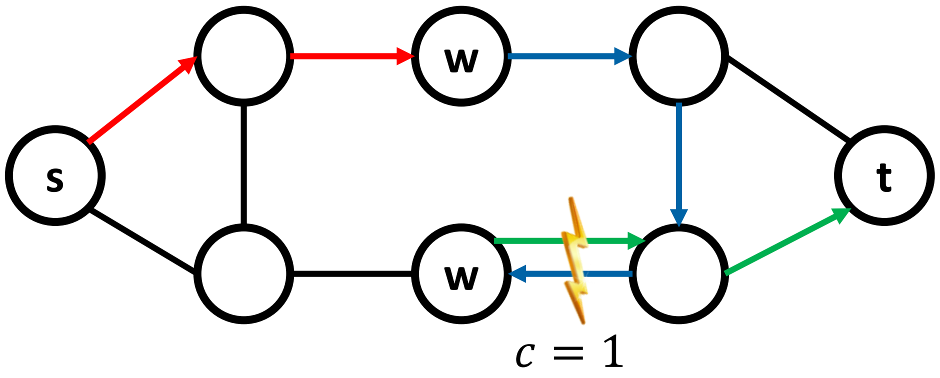


$c = 1$

- Given: Undirected graph with waypoints
- Find: Shortest s-t walk through all waypoints
- Twist: Respect capacities

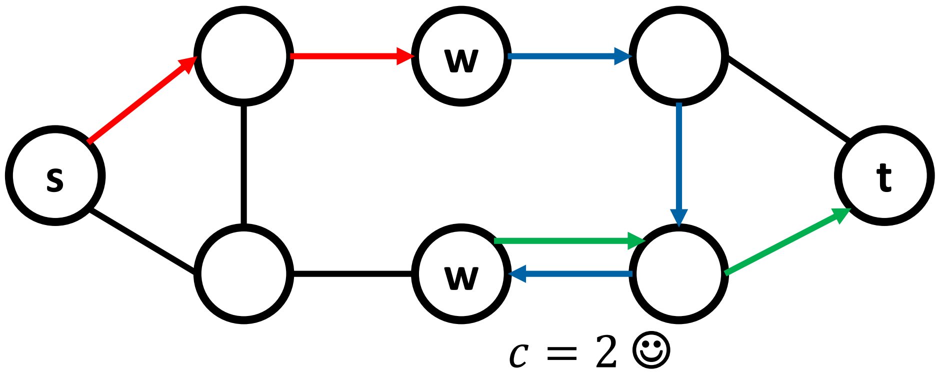
universität
wien

Walking Through Waypoints



- Given: Undirected graph with waypoints
- Find: Shortest s-t walk through all waypoints
- Twist: Respect capacities

Walking Through Waypoints



- Given: Undirected graph with waypoints
- Find: Shortest s-t walk through all waypoints
- Twist: Respect capacities

Model Variants

- **Ordered** Waypoint Routing:
  - S. Akhoondian Amiri, K.-T. Foerster, R. Jacob, and S. Schmid, "*Charting the Algorithmic Complexity of Waypoint Routing*", ACM SIGCOMM Computer Communication Review, vol. 48, no. 1, 2018.
  - S. Akhoondian Amiri, K.-T. Foerster, M. Parham, R. Jacob, and S. Schmid, "*Waypoint Routing in Special Networks*", IFIP Networking 2018

- **Unordered** Waypoint Routing
  - S. Akhoondian Amiri, K.-T. Foerster, and S. Schmid, "*Walking through Waypoints*", LATIN 2018

Model Variants

- **Ordered** Waypoint Routing:
  - S. Akhoondian Amiri, K.-T. Foerster, R. Jacob, and S. Schmid, "*Charting the Algorithmic Complexity of Waypoint Routing*", ACM SIGCOMM Computer Communication Review, vol. 48, no. 1, 2018.
  - S. Akhoondian Amiri, K.-T. Foerster, M. Parham, R. Jacob, and S. Schmid, "*Waypoint Routing in Special Networks*", IFIP Networking 2018

- **Unordered** Waypoint Routing
  - S. Akhoondian Amiri, K.-T. Foerster, and S. Schmid, "*Walking through Waypoints*", LATIN 2018

- We start with **joint** case: **One** waypoint

Model Variants

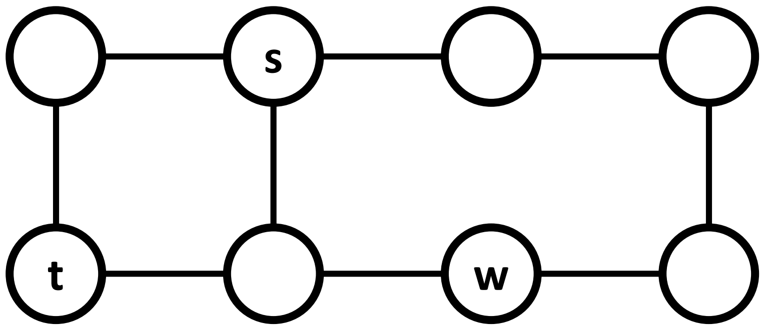
- **Ordered** Waypoint Routing:
  - S. Akhoondian Amiri, K.-T. Foerster, R. Jacob, and S. Schmid, "*Charting the Algorithmic Complexity of Waypoint Routing*", ACM SIGCOMM Computer Communication Review, vol. 48, no. 1, 2018.
  - S. Akhoondian Amiri, K.-T. Foerster, M. Parham, R. Jacob, and S. Schmid, "*Waypoint Routing in Special Networks*", IFIP Networking 2018

- **Unordered** Waypoint Routing
  - S. Akhoondian Amiri, K.-T. Foerster, and S. Schmid, "*Walking through Waypoints*", LATIN 2018

- We start with **joint** case: **One** waypoint
  - Followed by ordered and unordered

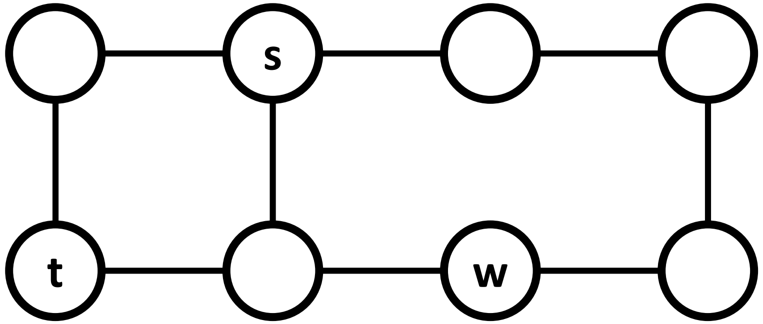
Walking Through One Waypoint

- Already non-trivial



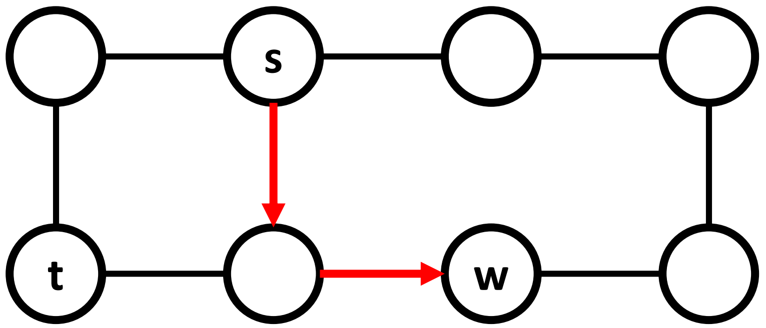
Walking Through One Waypoint

- Already non-trivial
- For simplicity: unit demand, unit capacity



Walking Through One Waypoint

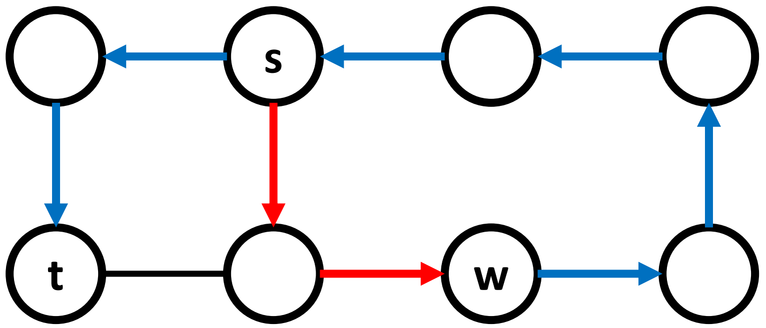
- Already non-trivial
- For simplicity: unit demand, unit capacity



- Greedy fails: choose shortest path from **s** to **w**…

Walking Through One Waypoint

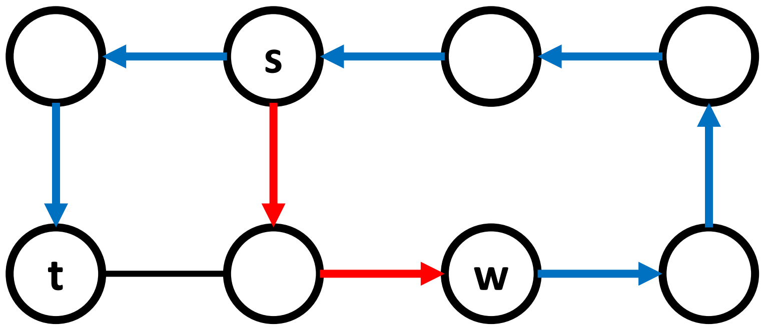
- Already non-trivial
- For simplicity: unit demand, unit capacity



- Greedy fails: … now needs long path from **w** to **t**

Walking Through One Waypoint

- Already non-trivial
- For simplicity: unit demand, unit capacity

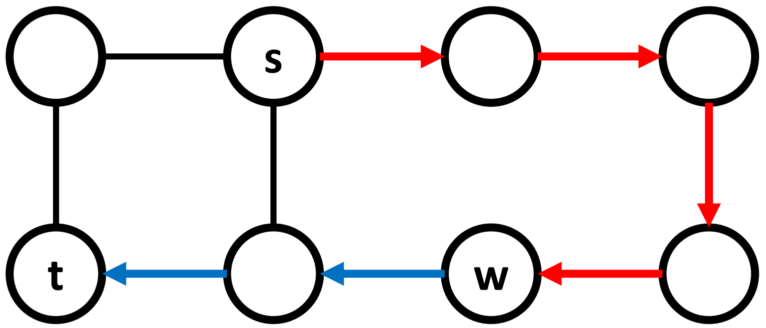


- Greedy fails: total length: 2+6=8

universität wien

Walking Through One Waypoint

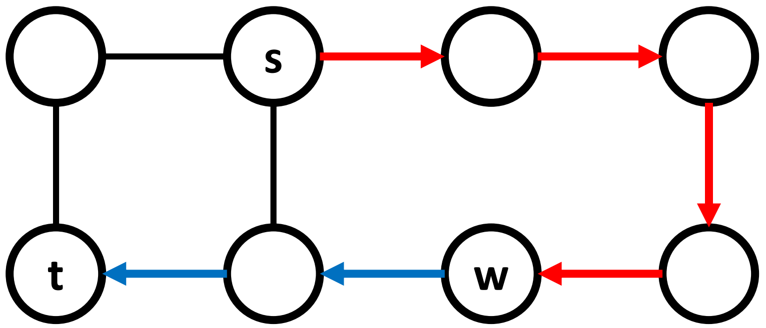
- Already non-trivial
- For simplicity: unit demand, unit capacity



- Greedy fails: total length: 2+6=8
- Optimal: Jointly optimize: 4+2=6

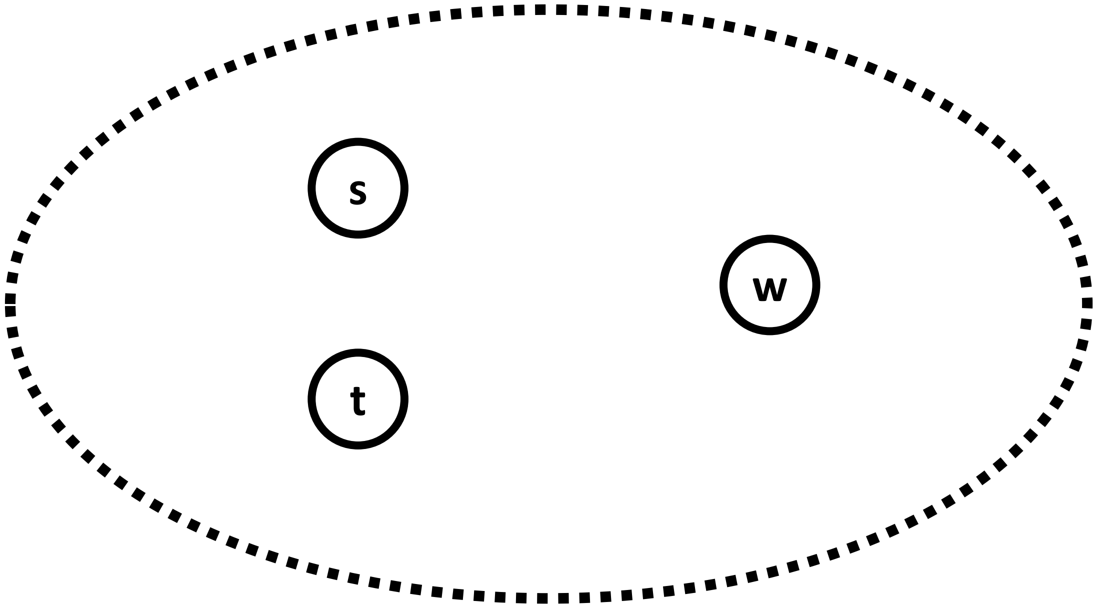
Walking Through One Waypoint

- Already non-trivial
- For simplicity: unit demand, unit capacity

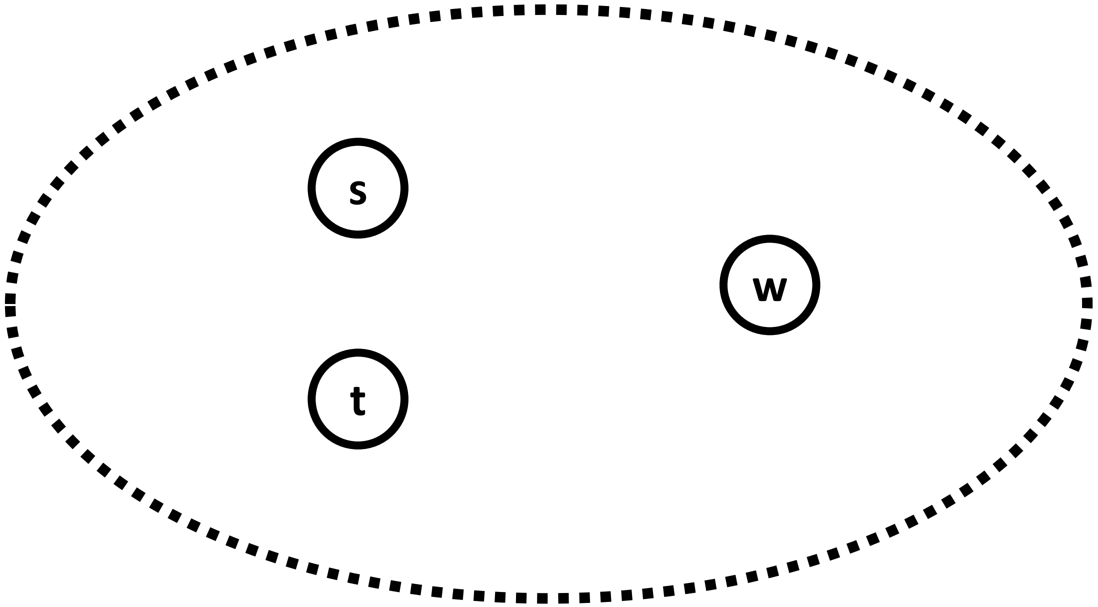


- Greedy fails: total length: 2+6=8.
- Optimal: Jointly optimize: 4+2=6. **How hard can it be?**

Joint Optimization



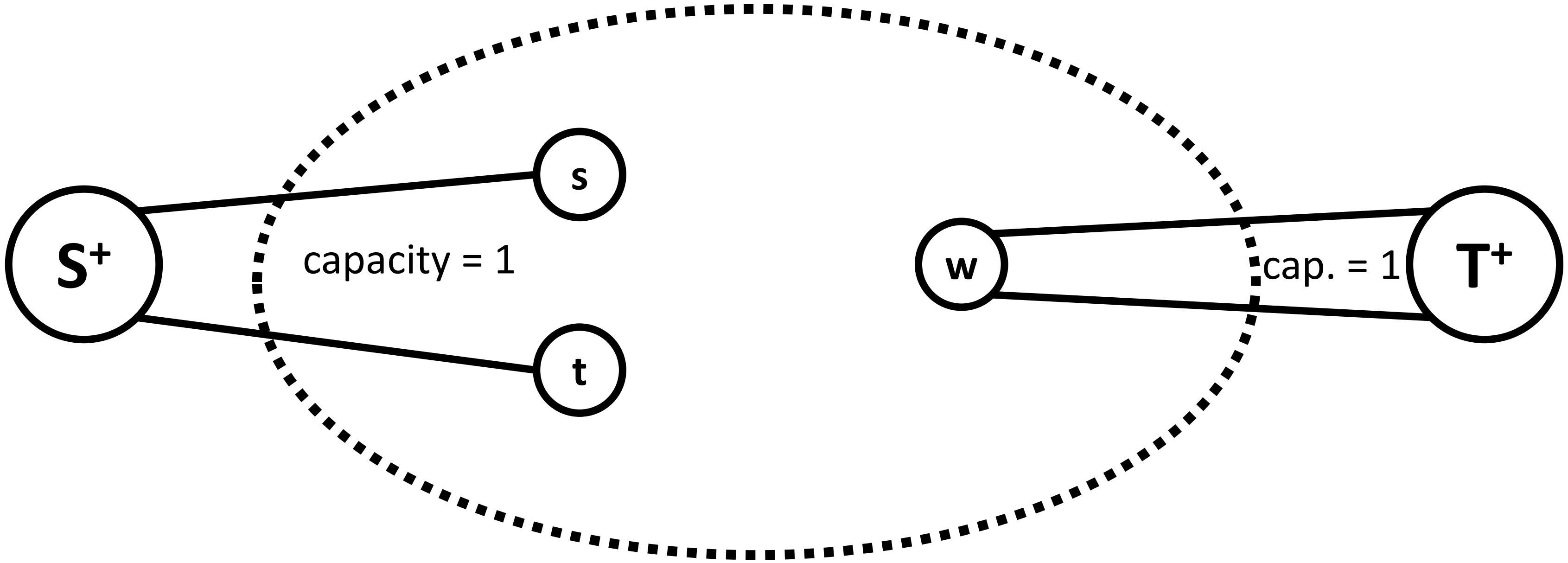
Joint Optimization



Idea: Single-Source Min-Cost Integral Flow has polynomial runtime

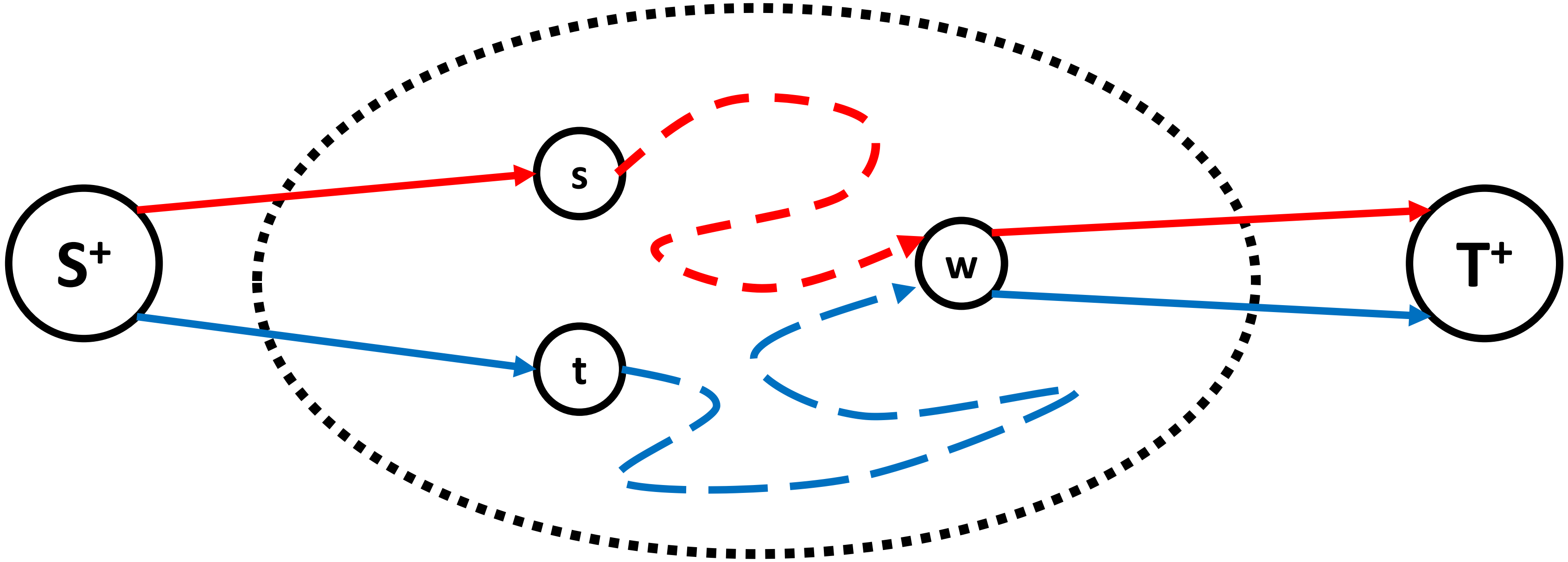
universität wien

Joint Optimization



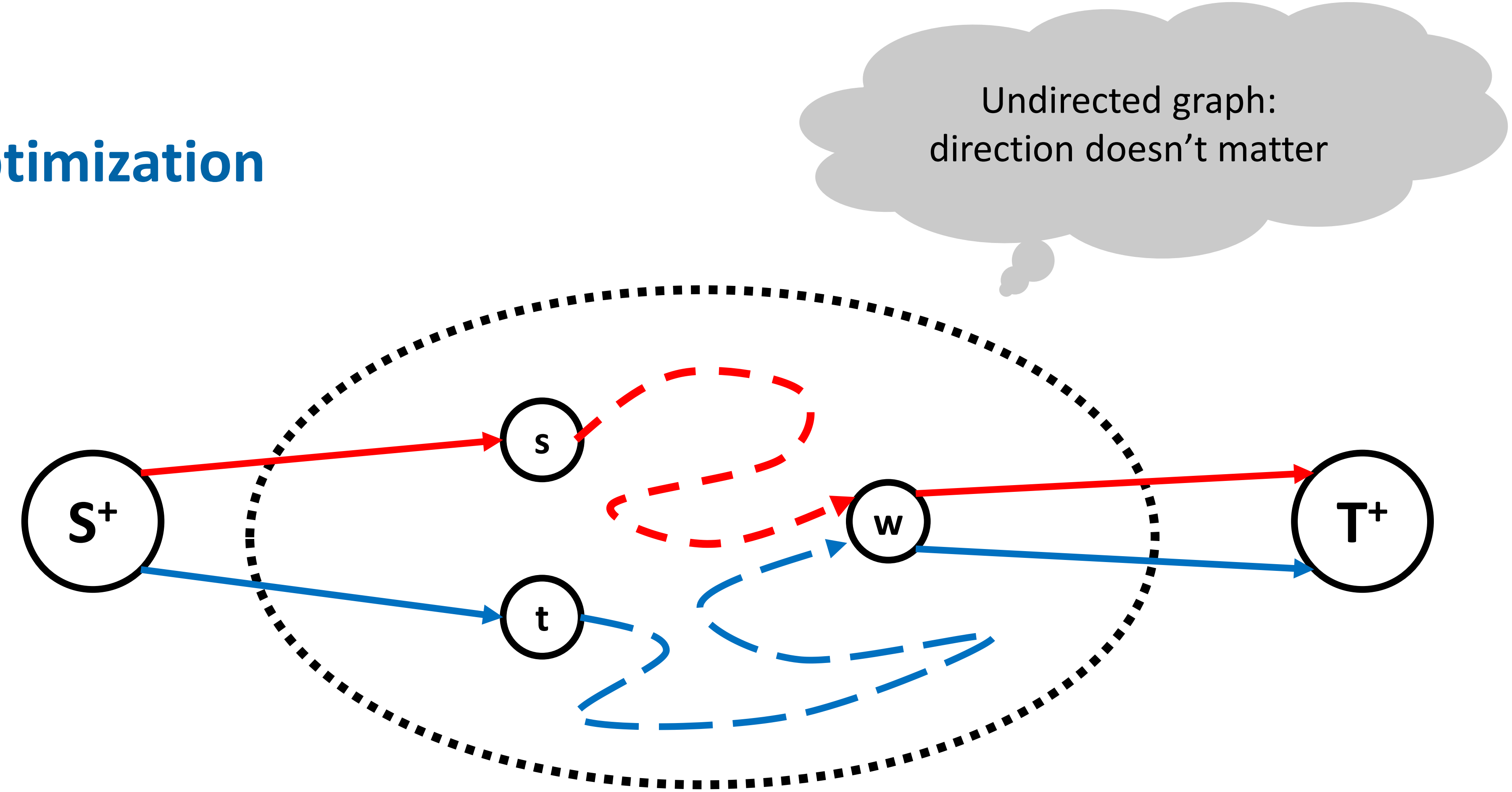
Idea: Single-Source Min-Cost Integral Flow has polynomial runtime

Joint Optimization



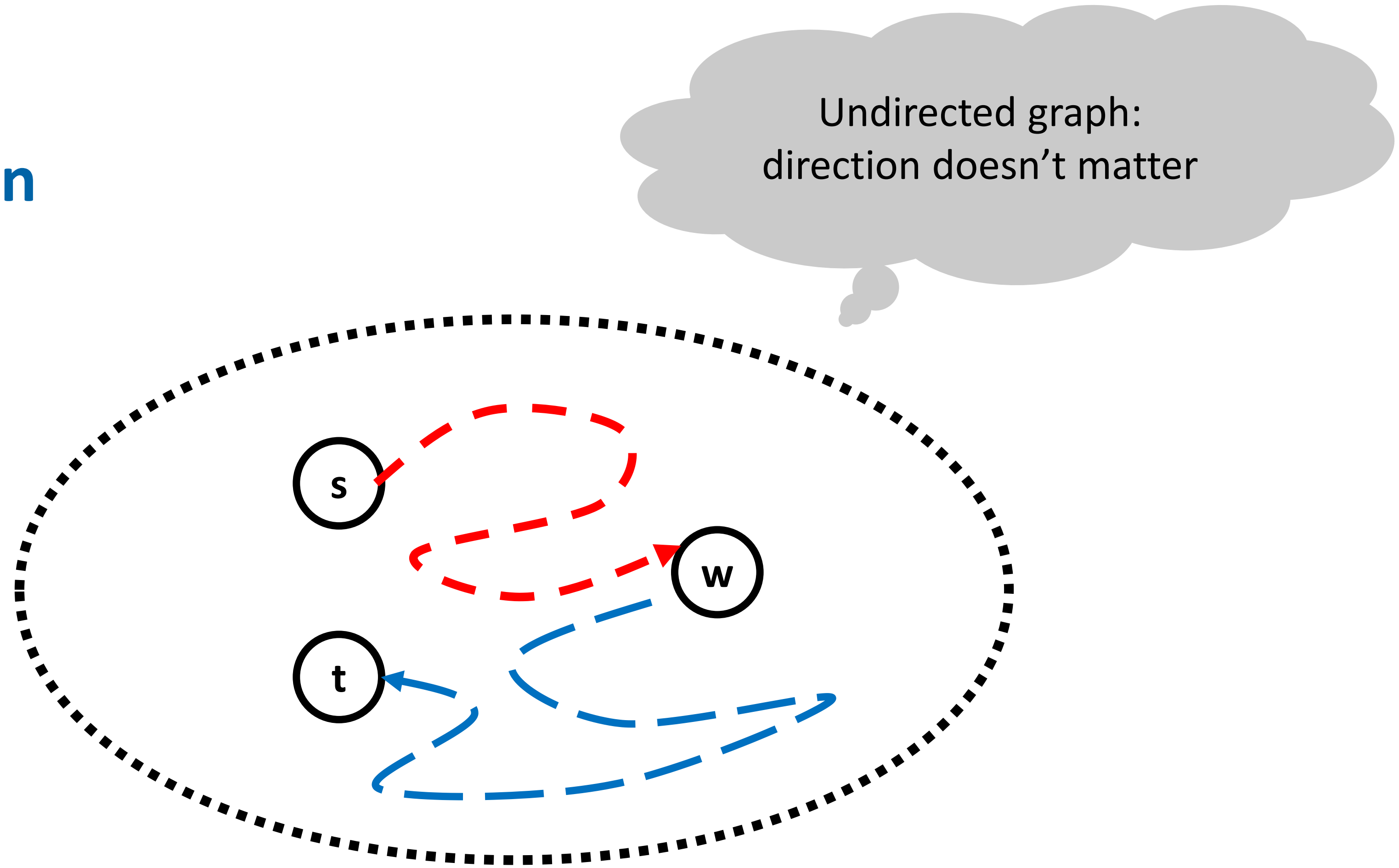
Idea: Single-Source Min-Cost Integral Flow has polynomial runtime

Joint Optimization



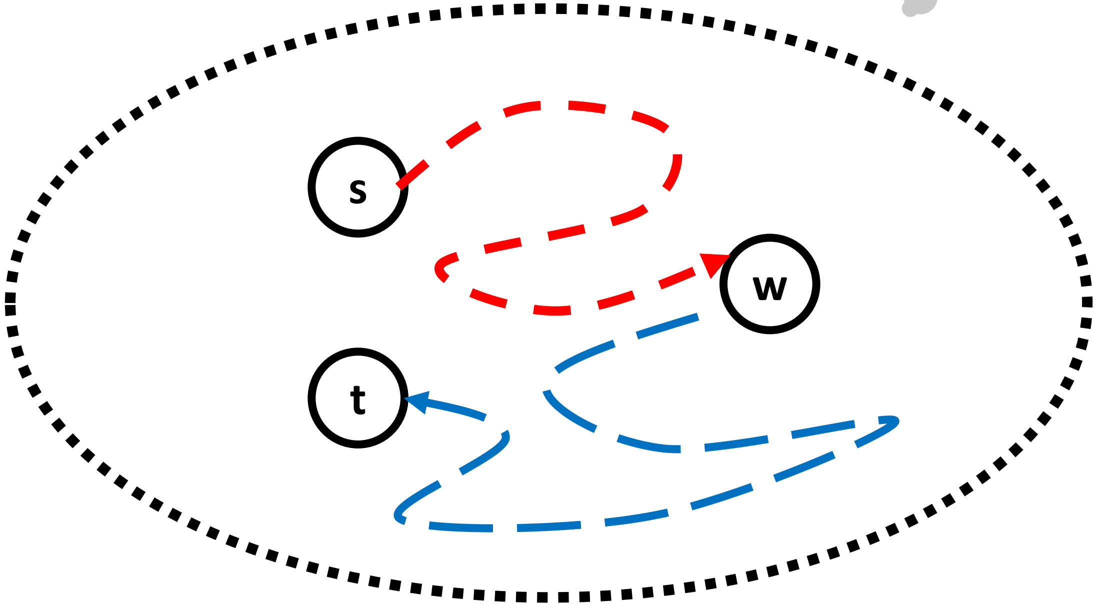
Idea: Single-Source Min-Cost Integral Flow has polynomial runtime

Joint Optimization



Joint Optimization

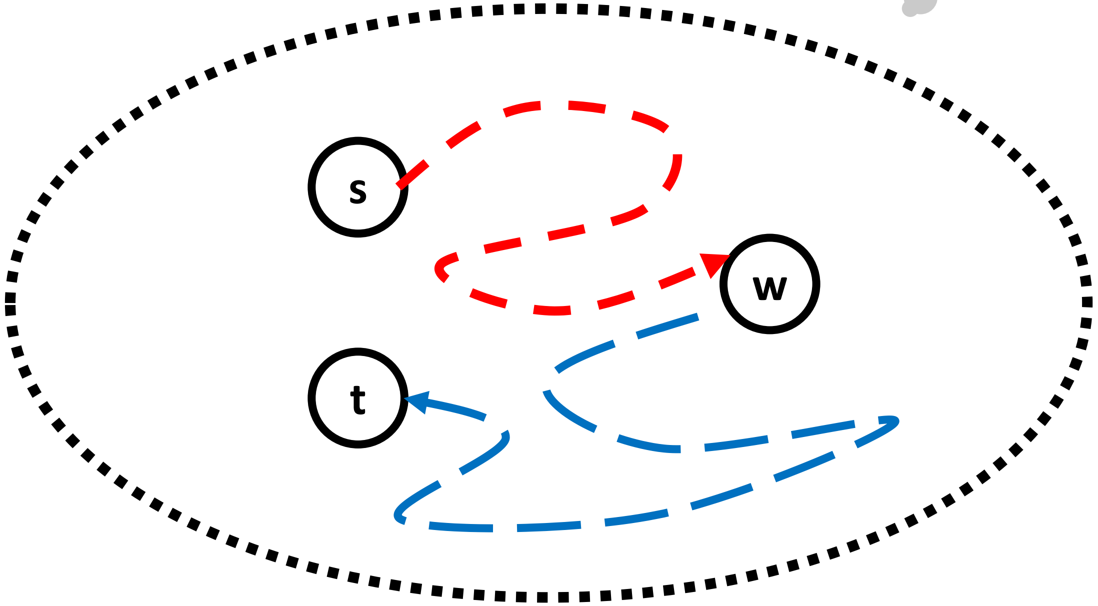
Undirected graph:
direction doesn't matter



Runtime: O((|V|log|E|)(|E| + |V|log|V|))

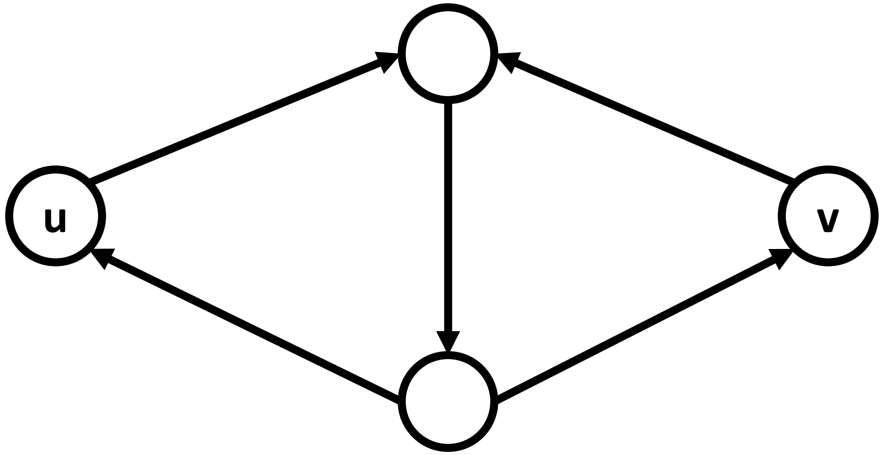
Joint Optimization

Undirected graph:
direction doesn't matter

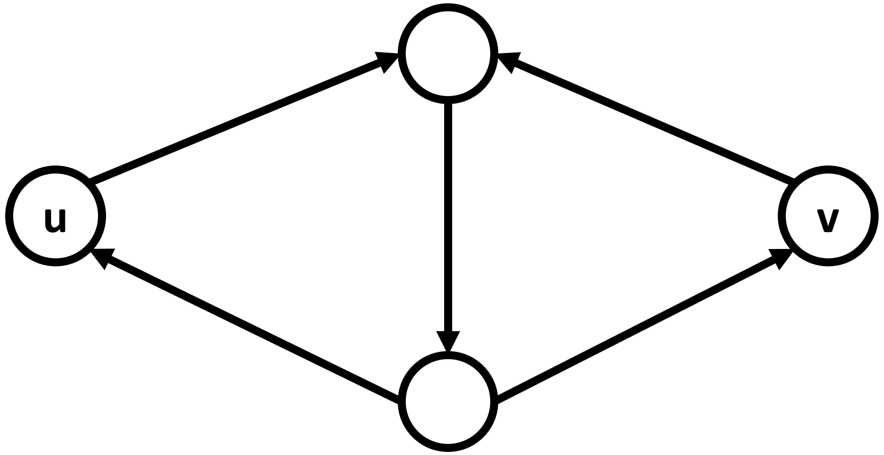


Runtime: O((|V|log|E|)(|E| + |V|log|V|)) **Faster runtime?**

Use Directed Algorithms



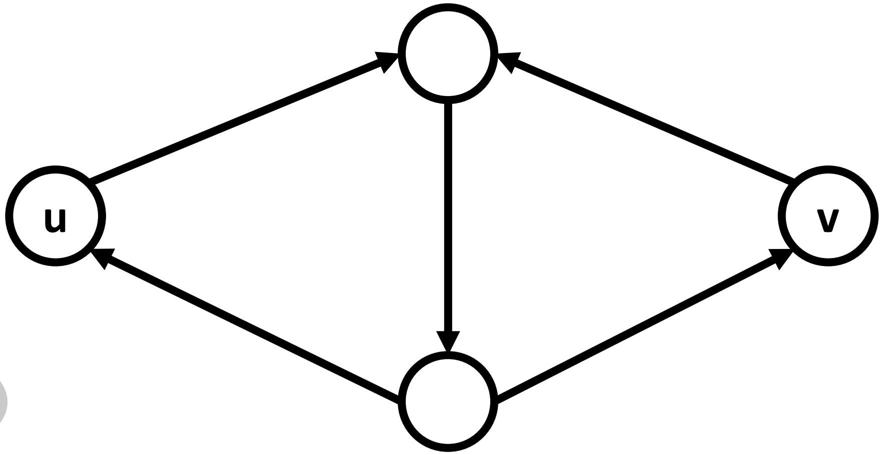
Use Directed Algorithms



Algorithm by Suurballe and Tarjan '84: Two directed edge-disjoint s-t-paths in $O(|E|\log_{(1+|E|/|V|)}|V|)$

universität wien

Use Directed Algorithms



“Trick”: Split edges of high capacity into many parallel ones of capacity 1

Algorithm by Suurballe and Tarjan ‘84: Two directed edge-disjoint s-t-paths in $O(|E|\log_{(1+|E|/|V|)}|V|)$

How Hard is the Directed Case?

universität wien

How Hard is the Directed Case?

- 2 edge-disjoint paths in directed graphs: NP-hard

Suurballe & Tarjan required shared source and shared destination

How Hard is the Directed Case?

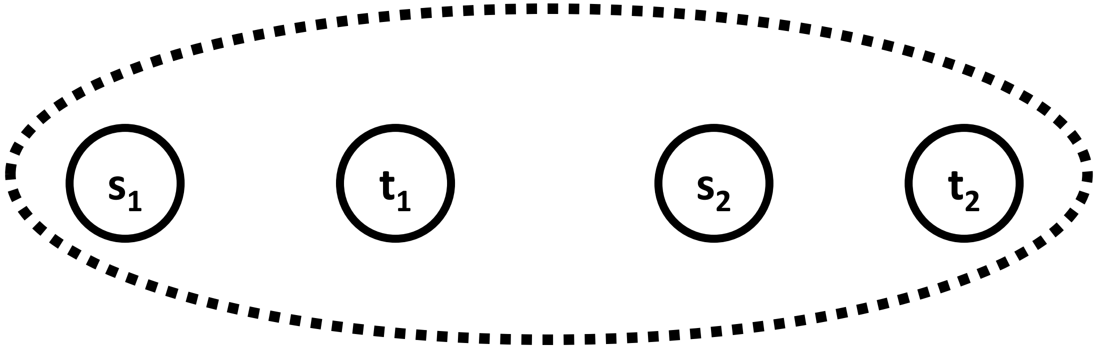
- 2 edge-disjoint paths in directed graphs: NP-hard

How Hard is the Directed Case?

- 2 edge-disjoint paths in directed graphs: NP-hard
  - But for 1 waypoint?

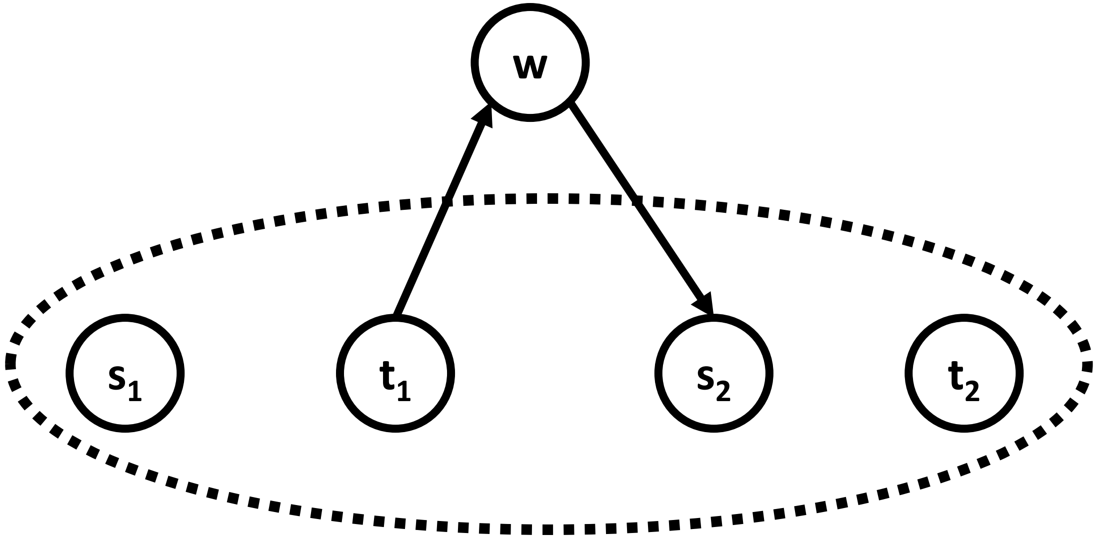
How Hard is the Directed Case?

- 2 edge-disjoint paths in directed graphs: NP-hard
  - But for 1 waypoint?



How Hard is the Directed Case?

- 2 edge-disjoint paths in directed graphs: NP-hard
  - But for 1 waypoint? NP-hard as well ☹



Brief Summary: One Waypoint

- Undirected case: Nearly linear runtime ☺
- Directed case: NP-hard ☹

Brief Summary: One Waypoint

- Undirected case: Nearly linear runtime ☺
  - What if middlebox alters flow size?

- Directed case: NP-hard ☹

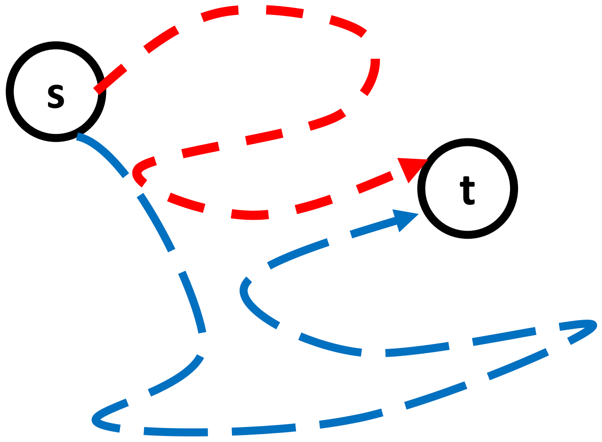
Altered Flow Size

Altered Flow Size

- The following problem is NP-hard:
  - Is the max integral s-t flow 2 or 3, when the flow is 2-splittable?

Altered Flow Size

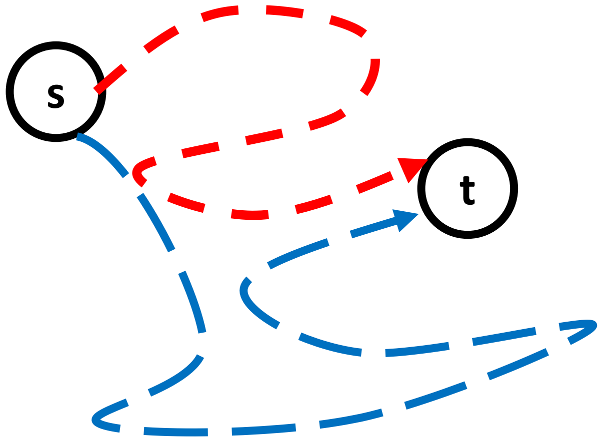
- The following problem is NP-hard:
  - Is the max integral s-t flow 2 or 3, when the flow is 2-splittable?



Can one of the two flows have a size of 2?

Altered Flow Size

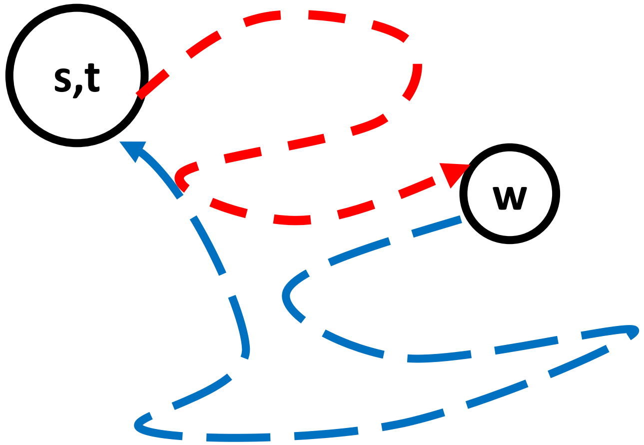
- The following problem is NP-hard:
  - Is the max integral s-t flow 2 or 3, when the flow is 2-splittable?
    - Adapt to waypoint routing



Can one of the two flows have a size of 2?

Altered Flow Size

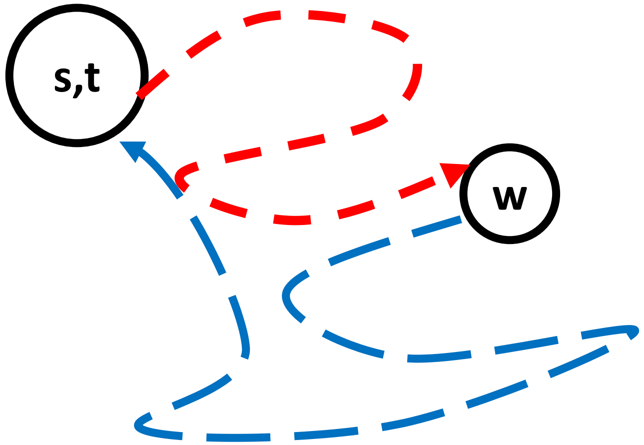
- The following problem is NP-hard:
  - Is the max integral s-t flow 2 or 3, when the flow is 2-splittable?
    - Adapt to waypoint routing



Can one of the two flows have a size of 2?

Altered Flow Size

- The following problem is NP-hard:
  - Is the max integral s-t flow 2 or 3, when the flow is 2-splittable?
    - Adapt to waypoint routing: Again NP-hard ☹



Can one of the two flows have a size of 2?

Brief Summary II: One Waypoint

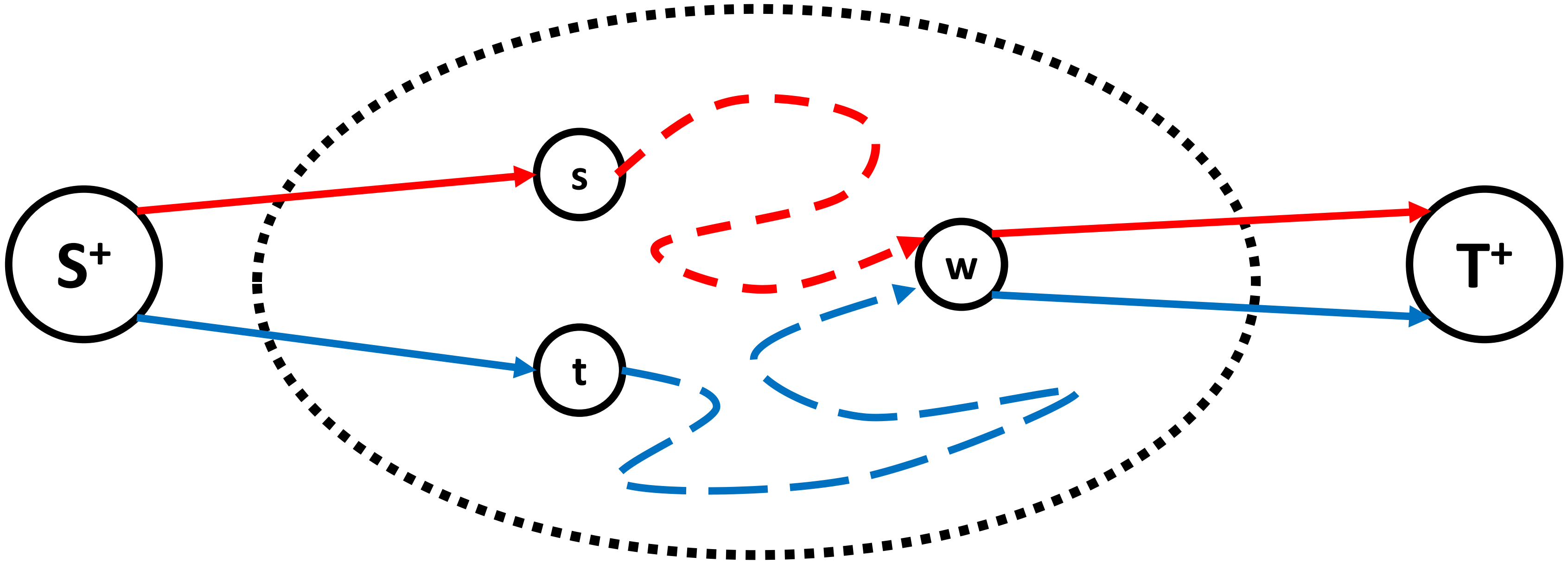
- Undirected case: Nearly linear runtime ☺
  - What if middlebox alters flow size? NP-hard ☹

- Directed case: NP-hard ☹

Brief Summary II: One Waypoint

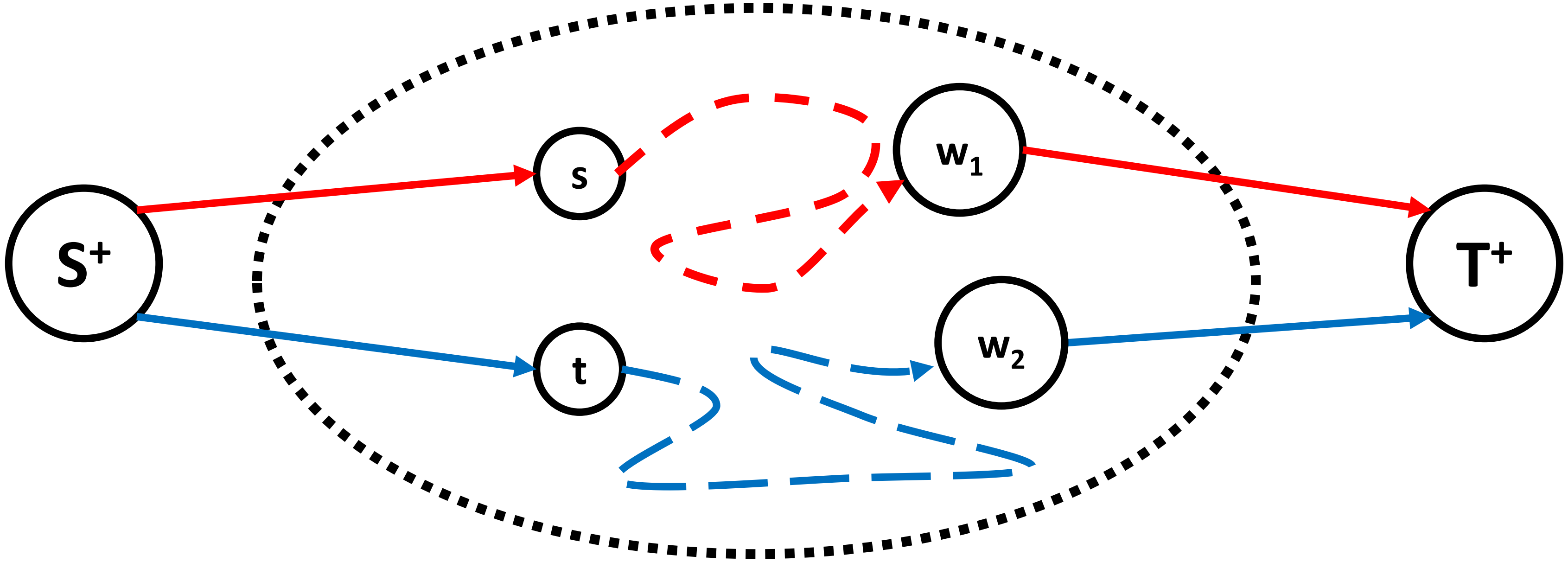
- Undirected case: Nearly linear runtime ☺
  - What if middlebox alters flow size? NP-hard ☹
- Directed case: NP-hard ☹
- How about more than one waypoint in the undirected case?

Can We adapt this idea?

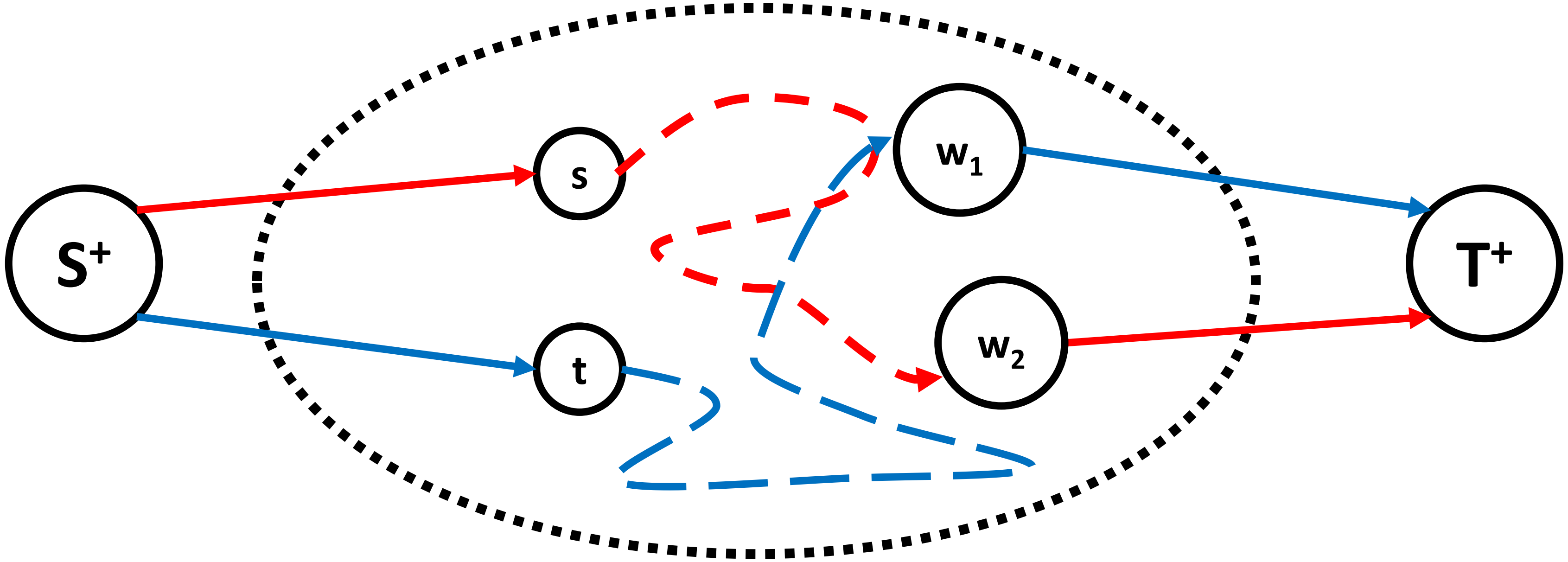


universität wien

Can We adapt this idea?

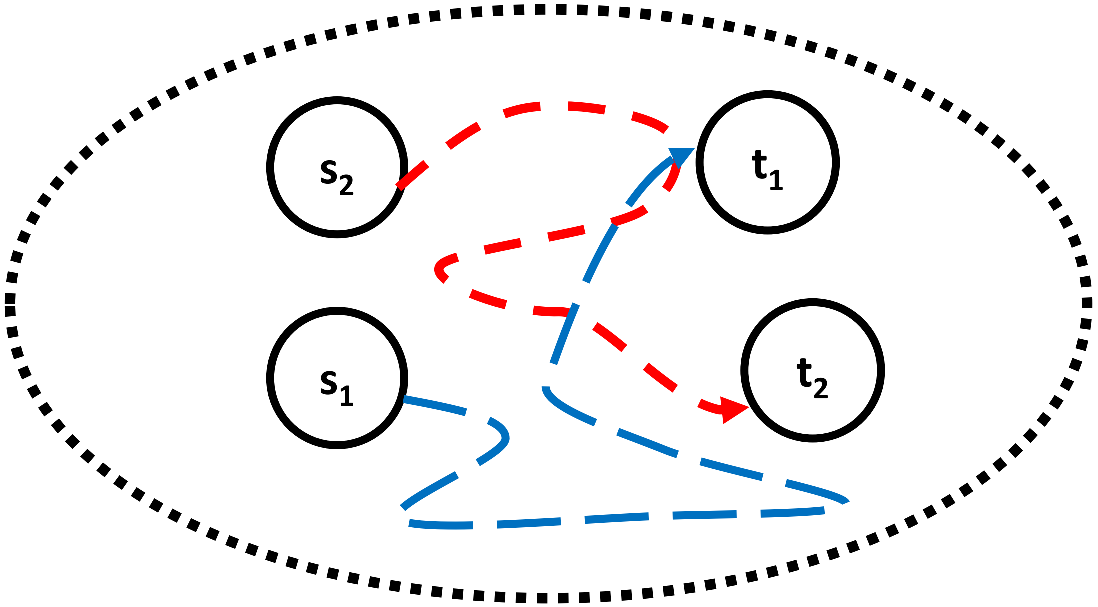


Can We adapt this idea?



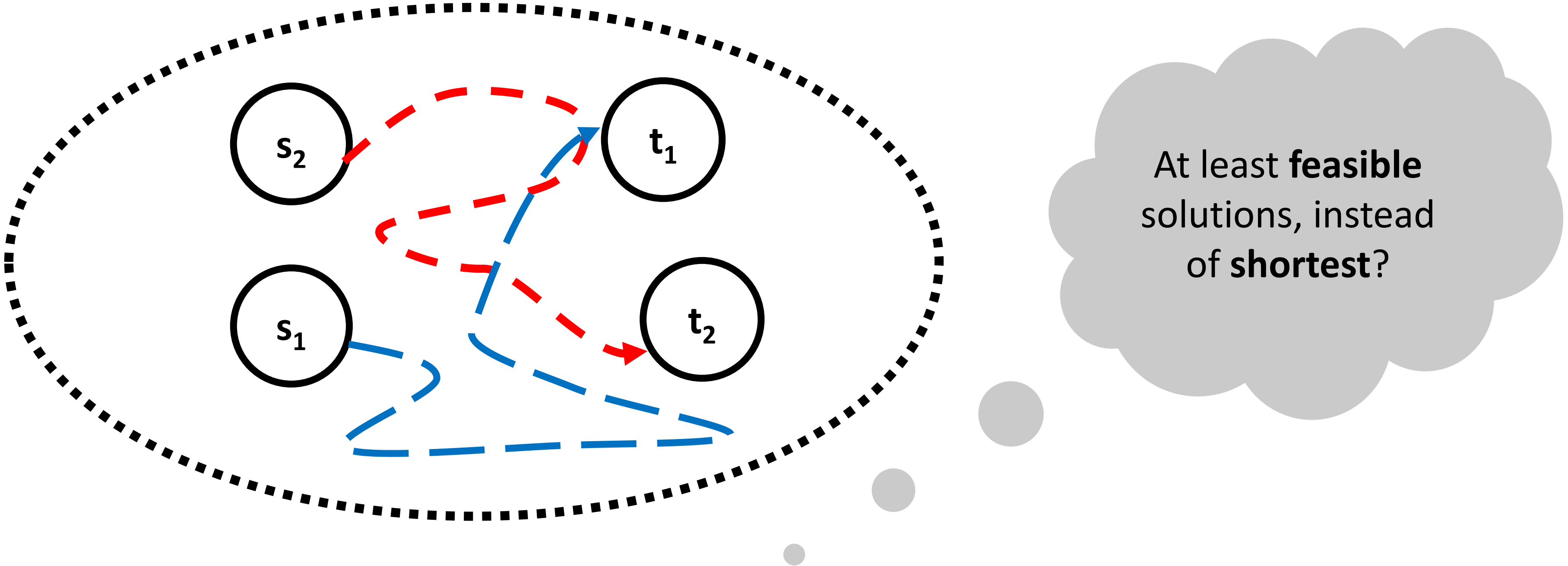
Problematic ☹

Can We adapt this idea?



Related **open** problem: Deterministic algorithm for **shortest 2 edge-disjoint paths**

Can We adapt this idea?



Related **open** problem: Deterministic algorithm for **shortest 2 edge-disjoint paths**

Feasible Solutions: O(1) Waypoints

- $O(1)$ edge-disjoint paths:
  - Has polynomial algorithm!
    - [N. Robertson and P. D. Seymour, "*Graph Minors .XIII. The Disjoint Paths Problem,*" J. Comb. Theory, Ser. B, vol. 63, no. 1, pp. 65–110, 1995.]

Feasible Solutions: O(1) Waypoints

- $O(1)$ edge-disjoint paths:
  - Has polynomial algorithm!
    - [N. Robertson and P. D. Seymour, "*Graph Minors .XIII. The Disjoint Paths Problem,*" J. Comb. Theory, Ser. B, vol. 63, no. 1, pp. 65–110, 1995.]

- Apply to waypoint routing:
  - First path: From source to **first** waypoint
  - Second path: From first waypoint to **second** waypoint
  - …
  - Last path: From **last** waypoint to destination

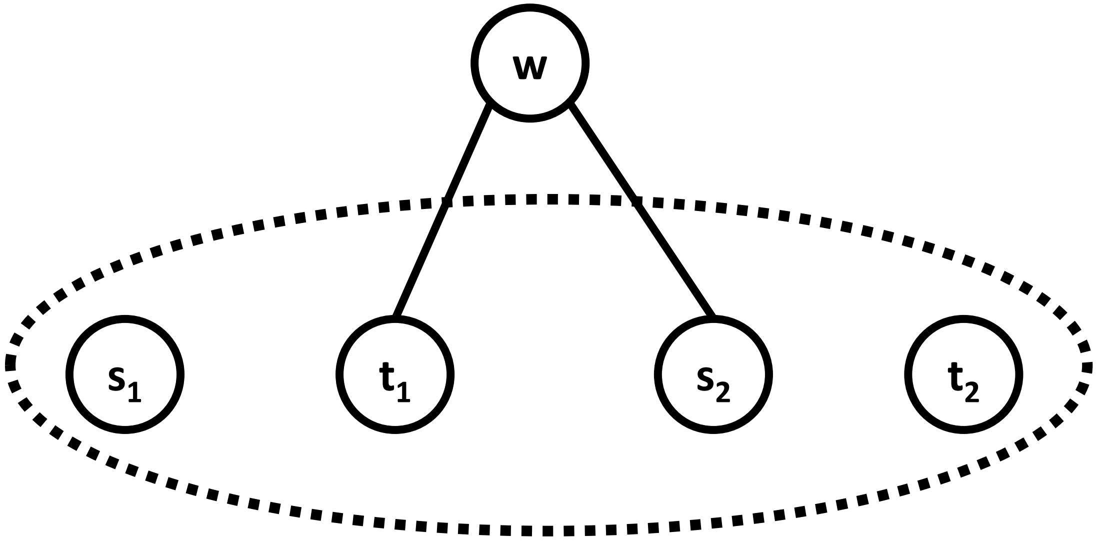
O(n) waypoints

- O(n) edge-disjoint paths? NP-hard

O(n) waypoints

- O(n) edge-disjoint paths? NP-hard

- O(n) waypoints? NP-hard too



Essentially same idea as before

universität wien

Summary: Ordered

<table>
<tr><th></th><th># Waypoints</th><th>Feasible</th><th>Optimal</th><th>Demand Change Feasible</th><th>Optimal</th></tr>
<tr><td rowspan="3">Undirected</td><td>1</td><td colspan="2">P</td><td colspan="2" rowspan="2">Strongly NPC</td></tr>
<tr><td>constant</td><td>P</td><td>?</td></tr>
<tr><td>arbitrary</td><td colspan="4">Strongly NPC</td></tr>
<tr><td rowspan="3">Directed</td><td>1</td><td colspan="4" rowspan="3">Strongly NPC</td></tr>
<tr><td>constant</td></tr>
<tr><td>arbitrary</td></tr>
</table>

TABLE I
OVERVIEW OF THE COMPLEXITY LANDSCAPE FOR WAYPOINT ROUTING IN GENERAL GRAPHS.

We further investigated this open box, parametrized by “treewidth” *tw*

Summary: Ordered

	# Waypoints	Feasible	Optimal	Demand Change Feasible	Optimal
Undirected	1	**P**		**Strongly NPC**	
	constant	**P**	?		
	arbitrary	**Strongly NPC**			
Directed	1	**Strongly NPC**			
	constant				
	arbitrary				

TABLE I
OVERVIEW OF THE COMPLEXITY LANDSCAPE FOR WAYPOINT ROUTING IN GENERAL GRAPHS.

We further investigated this open box, parametrized by “treewidth” *tw*

Summary: Ordered

<table>
<tr><th></th><th># Waypoints</th><th>Feasible</th><th>Optimal</th><th>Demand Change Feasible</th><th>Optimal</th></tr>
<tr><td rowspan="3">Undirected</td><td>1</td><td colspan="2">P</td><td colspan="2" rowspan="2">Strongly NPC</td></tr>
<tr><td>constant</td><td>P</td><td>?</td></tr>
<tr><td>arbitrary</td><td colspan="4">Strongly NPC</td></tr>
<tr><td rowspan="3">Directed</td><td>1</td><td colspan="4" rowspan="3">Strongly NPC</td></tr>
<tr><td>constant</td></tr>
<tr><td>arbitrary</td></tr>
</table>

TABLE I
OVERVIEW OF THE COMPLEXITY LANDSCAPE FOR WAYPOINT ROUTING IN GENERAL GRAPHS.

# Waypoints	Feasible Algorithms	Known Hardness	Demand Change Optimal Algorithms	Demand Change Hardness
Arbitrary	**P**: Outerplanar ($\mathtt{tw} \leq 2$)	**Strongly NPC**: $\mathtt{tw} \leq 3$	**P**: Tree (equivalent to $\mathtt{tw}$ of 1)	**NPC**: Unicyclic ($\mathtt{tw} \leq 2$)
Constant	**P**: General graphs	**P**: General graphs	**P**: Constant treewidth $\mathtt{tw} \in O(1)$	**Strongly NPC**: General graphs

TABLE II
OVERVIEW OF THE COMPLEXITY LANDSCAPE FOR WAYPOINT ROUTING IN SPECIAL UNDIRECTED GRAPHS.

Walking Through *Unordered* Waypoints

- Can we just use algorithms from the ordered case?

Walking Through Unordered Waypoints

- Can we just use algorithms from the ordered case?
  - Problem: Combinatorial explosion of possibilities

Walking Through Unordered Waypoints

- Can we just use algorithms from the ordered case?
  - Problem: Combinatorial explosion of possibilities
    - Already logarithmically many waypoints: not tractable

Walking Through Unordered Waypoints

- Can we just use algorithms from the ordered case?
  - Problem: Combinatorial explosion of possibilities
    - Already logarithmically many waypoints: not tractable

- Let us take a step back

Theoretical Motivation I/II: Subset TSP

- Subset TSP:
  - Shortest tour through k waypoints
  - Difference: *No capacities*
  - E.g., [P. N. Klein & D. Marx , “*A subexponential parameterized algorithm for Subset TSP on planar graphs*”, SODA 2014]

Sometimes k-Cycle is a different problem

Theoretical Motivation II/II: k-Cycle Problem

- Find vertex/edge-disjoint cycle through k waypoints
  - $k \in O(1)$: polynomial algorithm via disjoint path problem
    - [N. Robertson and P. D. Seymour, "*Graph Minors .XIII. The Disjoint Paths Problem,*" J. Comb. Theory, Ser. B, vol. 63, no. 1, pp. 65–110, 1995.]
  - $k \in O\left((\log \log n)^{1/10}\right)$: polynomial algorithm
    - [K. Kawarabayashi, "*An improved algorithm for finding cycles through elements,*" IPCO 2008.]
  - Randomized algorithm with runtime of $2^k n^{O(1)}$ (*shortest* tour)
    - [A. Björklund, T. Husfeld, and N. Taslaman, "*Shortest cycle through specified elements,*" SODA 2012.]

Walking Through Waypoints on Bounded Treewidth Graphs

- Initial ideas:
  - **Capacities $c: E \rightarrow \{1, 2\}$ suffice**
    - Never traverse an edge more than twice
    - From, e.g.,: [Klein & Marx, SODA 2014]
  - **Reduce $s - t$ tours to cycles**
    - Connect s, t to fake vertex
    - Trick does not work for uncapacitated (subset) TSP!
  - Remove weights and capacities by expanding graph ("***Unify***")
    - Only works for integral weights polynomial in input size

undirected graphs

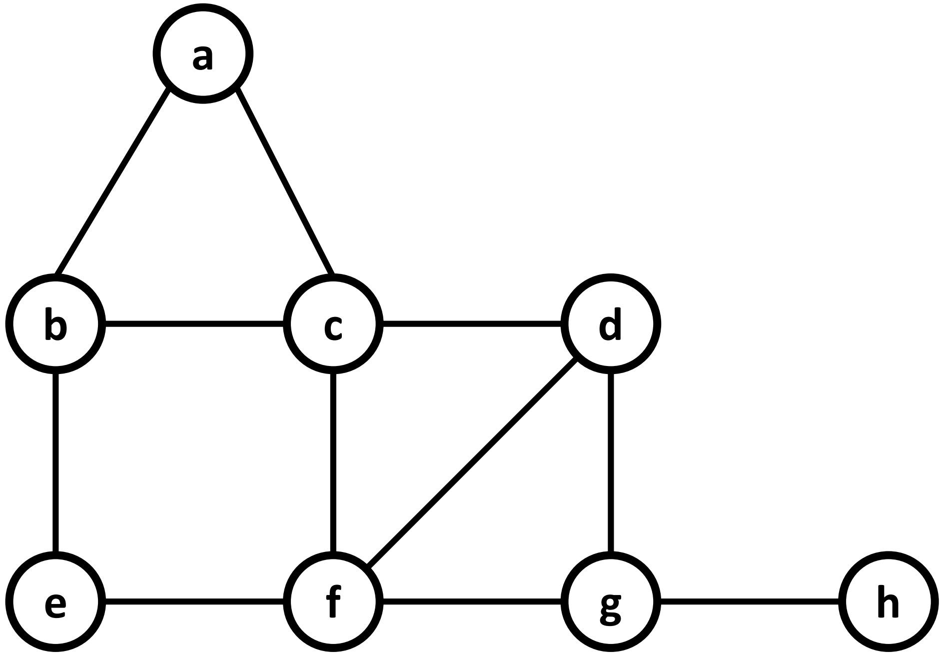
Treewidth

- How much is a graph "*like*" a tree?
  - [Robertson and Seymour, 1984]

- Intuition:
  - A tree is like a tree ☺
  - A complete graph? Not at all.
  - In between?

universität wien

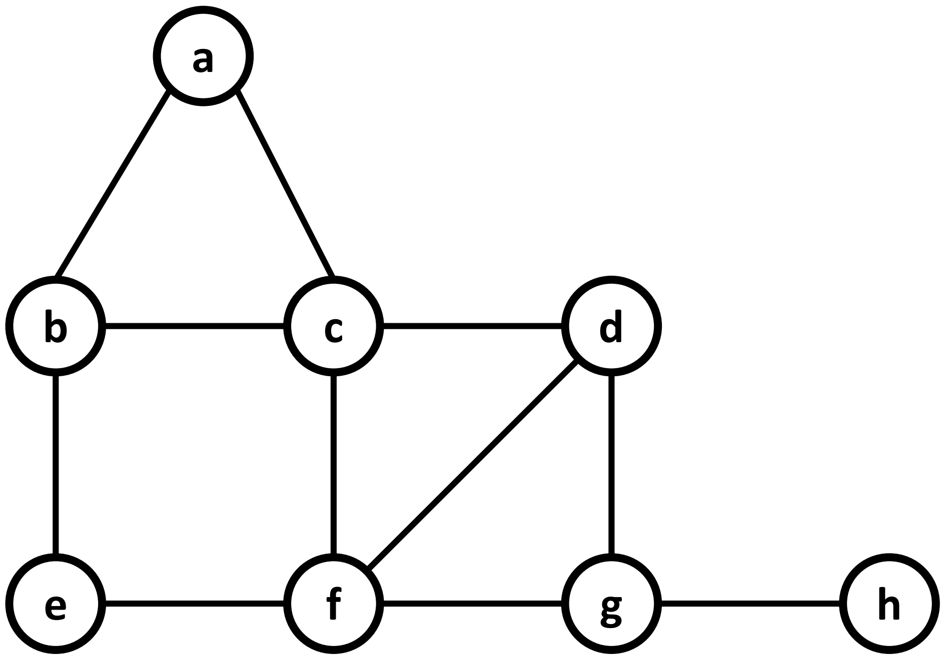
Tree Decomposition $\mathcal{T} = (T, X)$ of a graph G

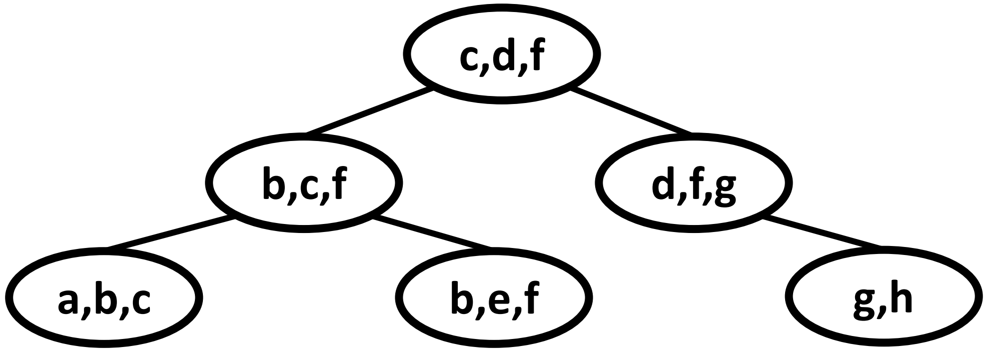


(Example: Based on slides of Dániel Marx)

Tree Decomposition $\mathcal{T} = (T, X)$ of a graph G

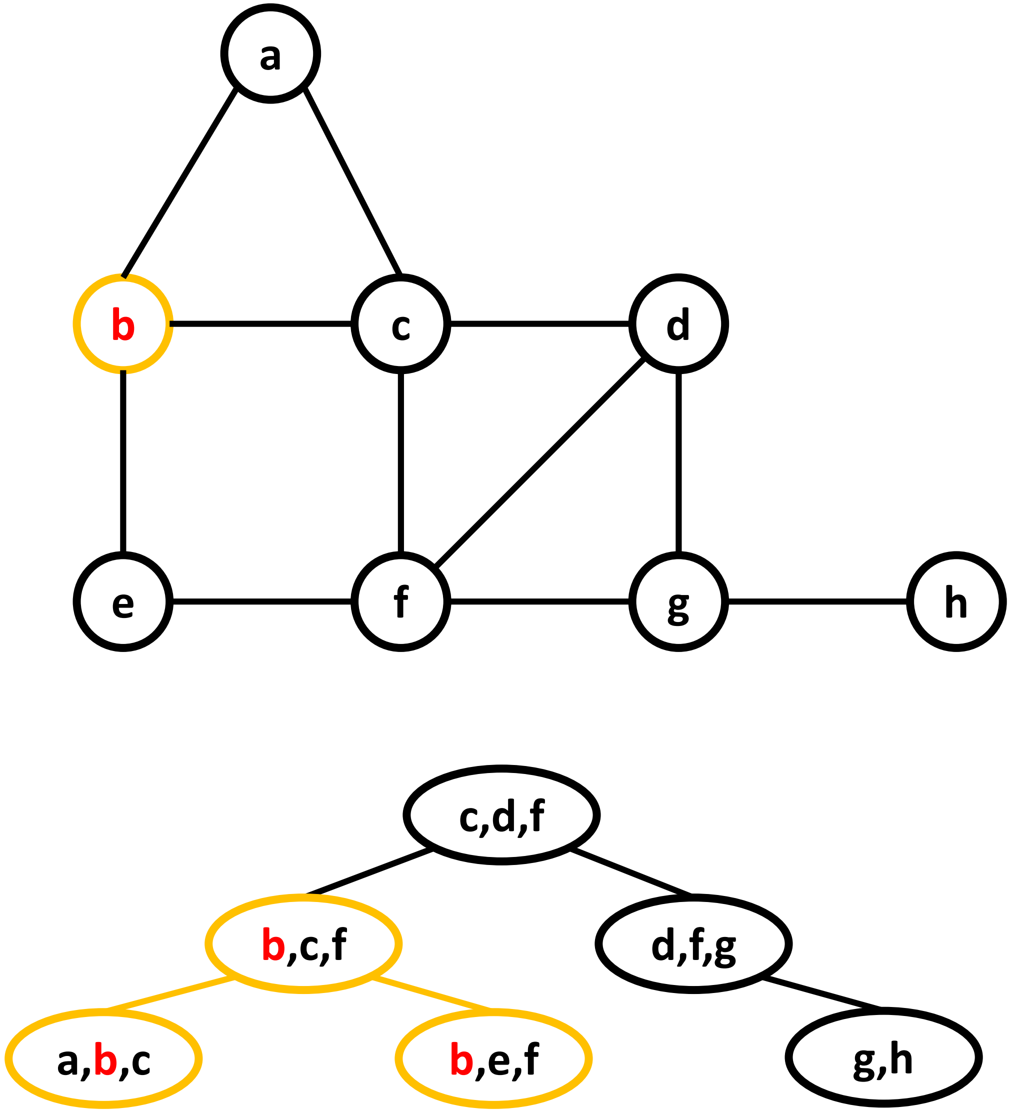
- Bijection between tree T and collection X, where every element of X is a set of vertices of G such that:





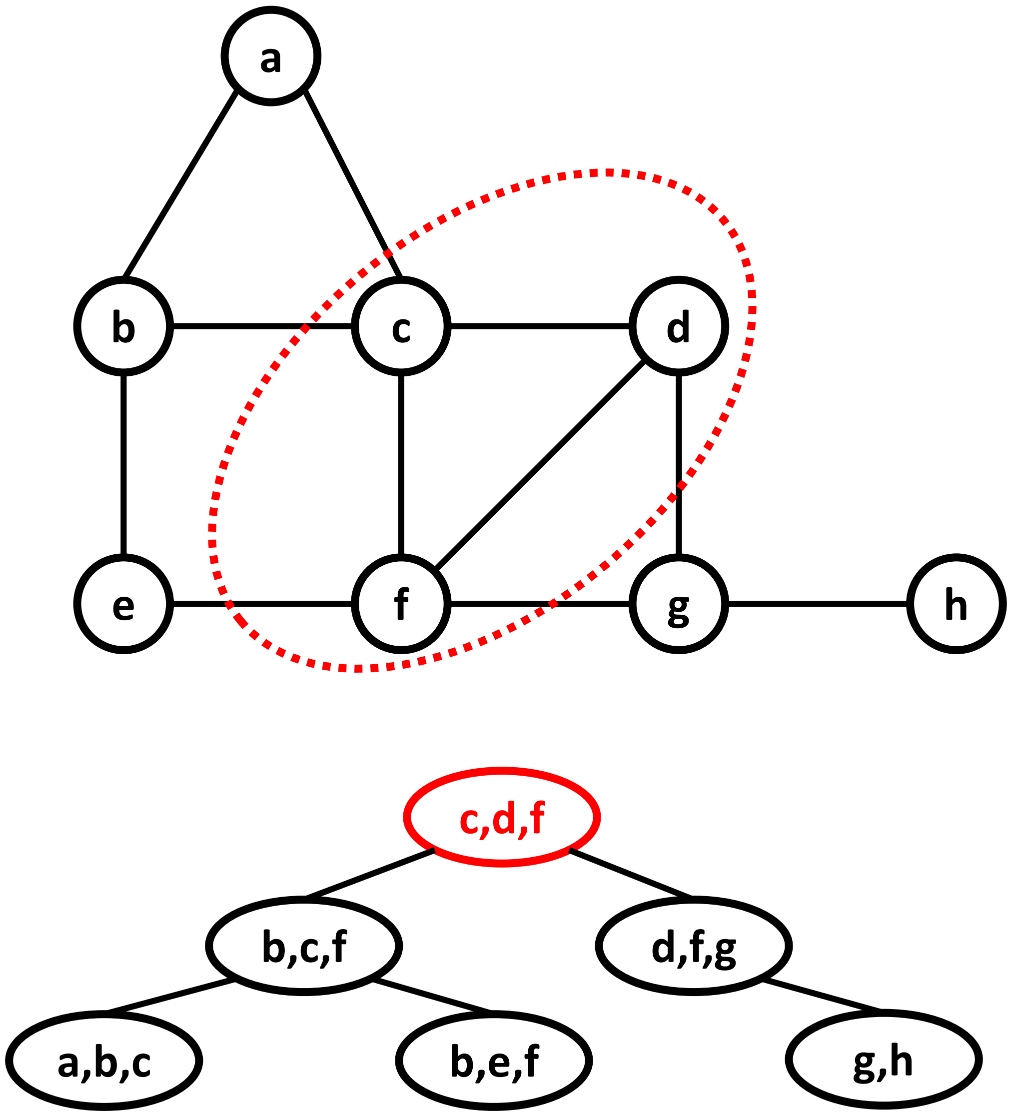
Tree Decomposition $\mathcal{T} = (T, X)$ of a graph G

- Bijection between tree T and collection X, where every element of X is a set of vertices of G such that:
  1. Each graph vertex is contained in at least one tree node (the bag or separator)



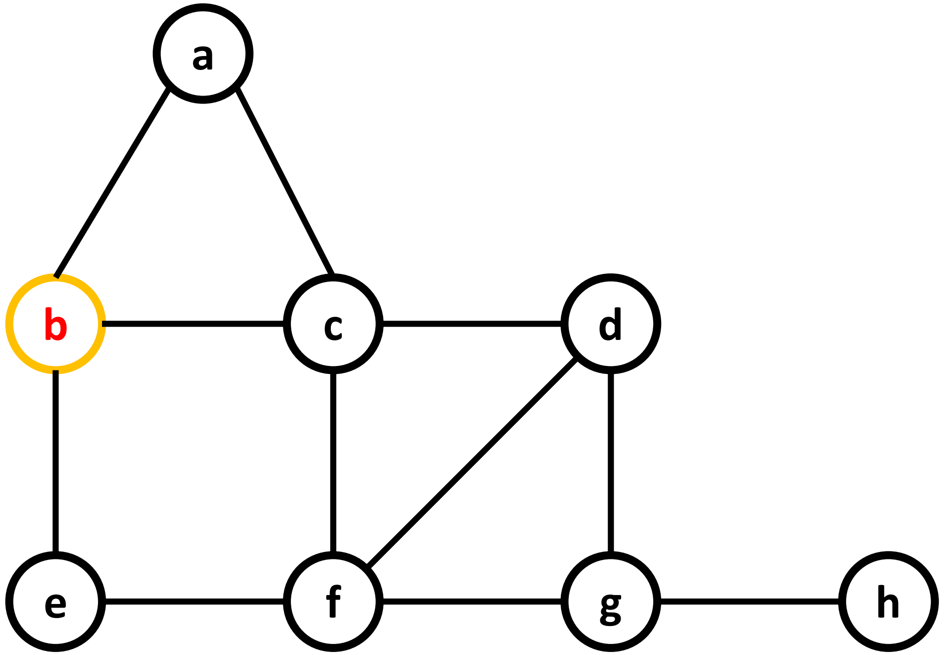
Tree Decomposition $\mathcal{T} = (T, X)$ of a graph G

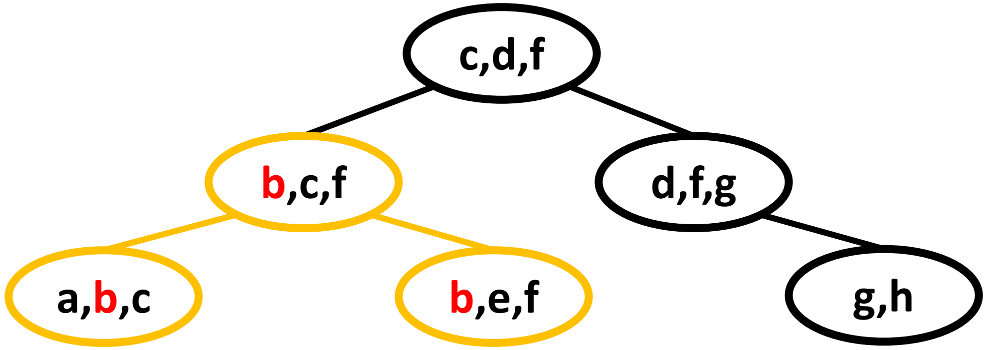
- Bijection between tree T and collection X, where every element of X is a set of vertices of G such that:
  1. Each graph vertex is contained in at least one tree node (the bag or separator)



Tree Decomposition $\mathcal{T} = (T, X)$ of a graph G

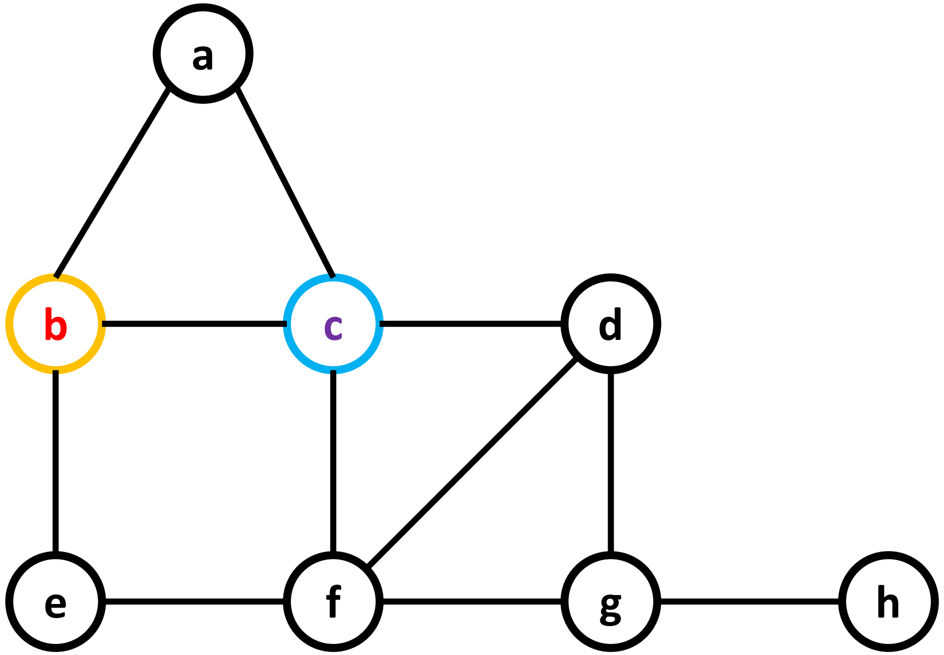
- Bijection between tree T and collection X, where every element of X is a set of vertices of G such that:
  1. Each graph vertex is contained in at least one tree node (the bag or separator)
  2. Tree nodes containing a vertex v form a connected subtree of T

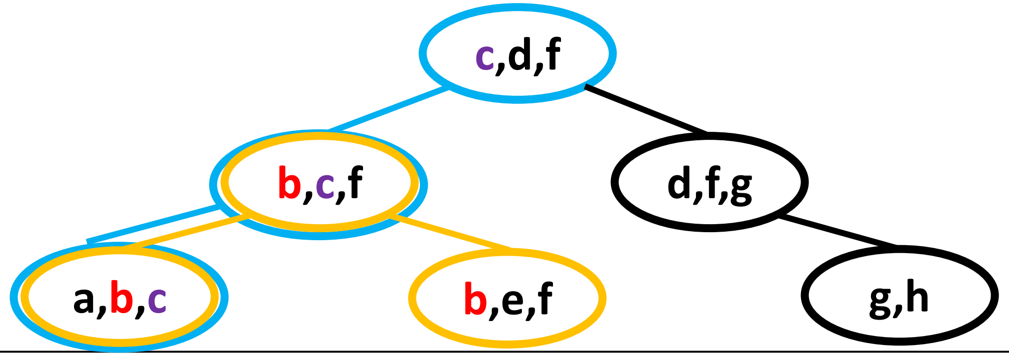




Tree Decomposition $\mathcal{T} = (T, X)$ of a graph G

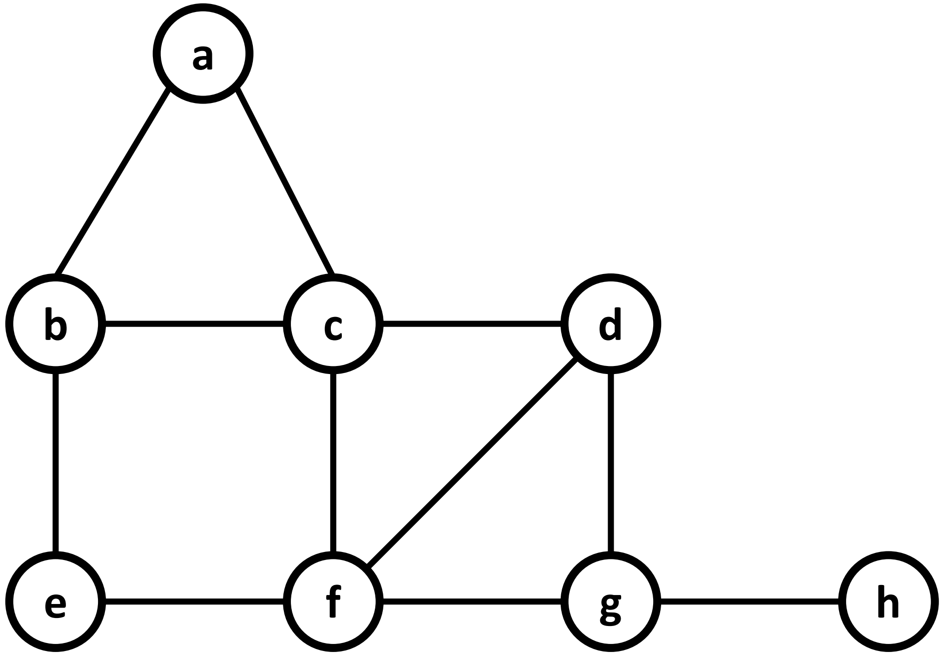
- Bijection between tree T and collection X, where every element of X is a set of vertices of G such that:
  1. Each graph vertex is contained in at least one tree node (the bag or separator)
  2. Tree nodes containing a vertex v form a connected subtree of T
  3. When vertices are adjacent in the graph, then the corresponding subtrees have a node in common

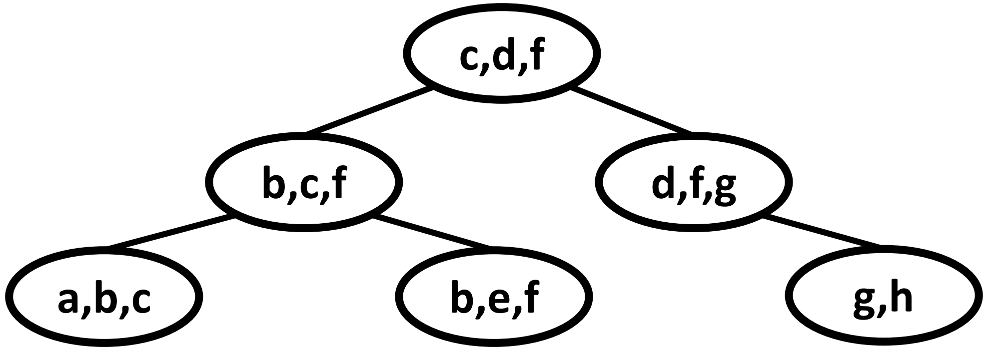




Tree Decomposition $\mathcal{T} = (T, X)$ of a graph G

- Bijection between tree T and collection X, where every element of X is a set of vertices of G such that:
  1. Each graph vertex is contained in at least one tree node (the bag or separator)
  2. Tree nodes containing a vertex v form a connected subtree of T
  3. When vertices are adjacent in the graph, then the corresponding subtrees have a node in common

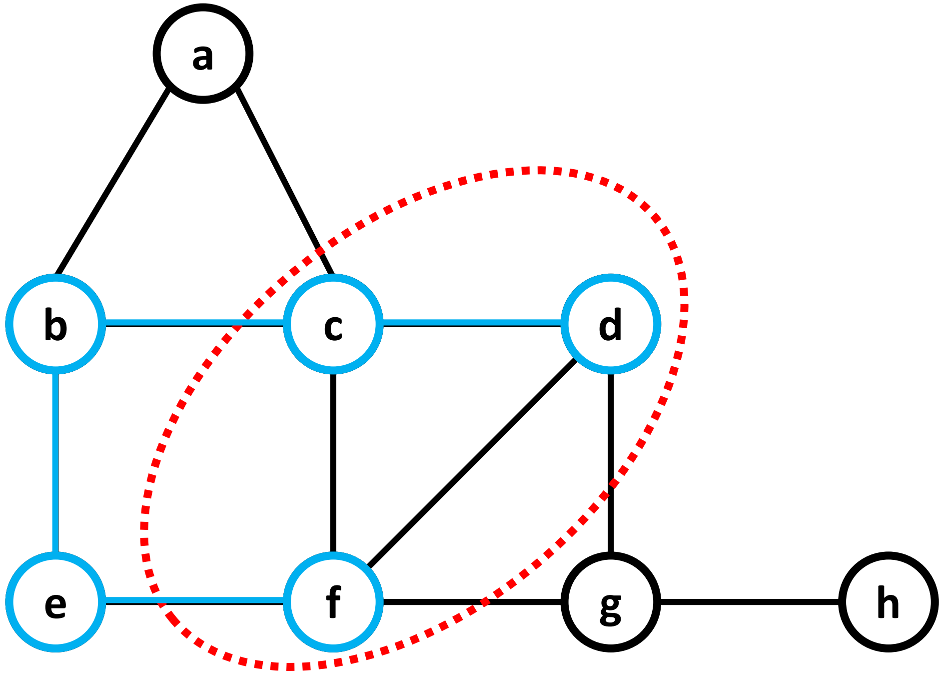


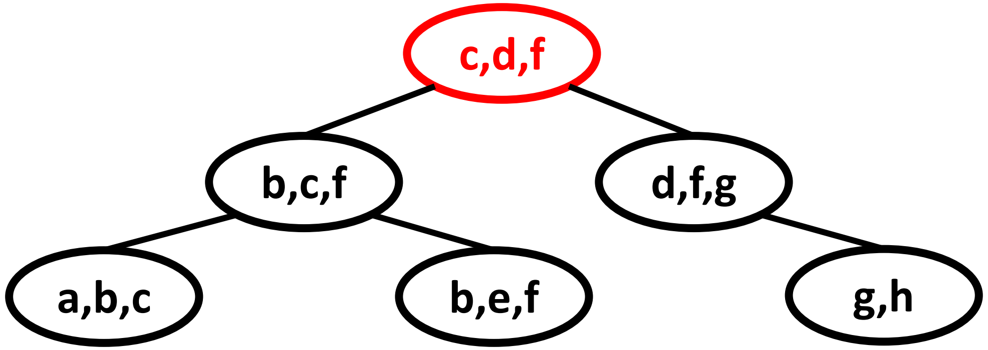


- **Width** of a tree decomposition: largest bag size -1
- **Treewidth**: Minimum width of all tree decompositions

How does a Tree Decomposition help us?

- Recall: nodes represent separators
- Do dynamic programming
  - Don’t care what is “behind” the separator

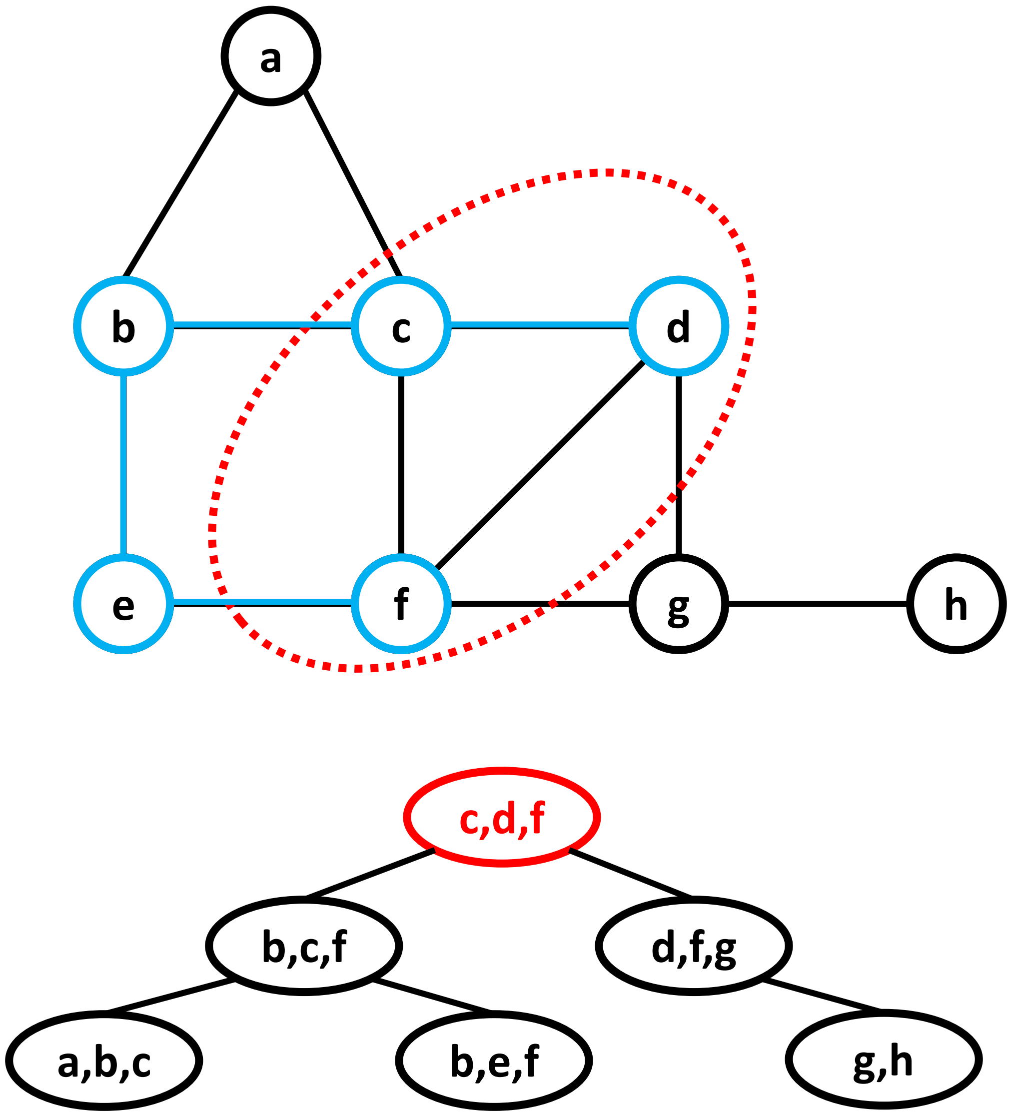




How does a Tree Decomposition help us?

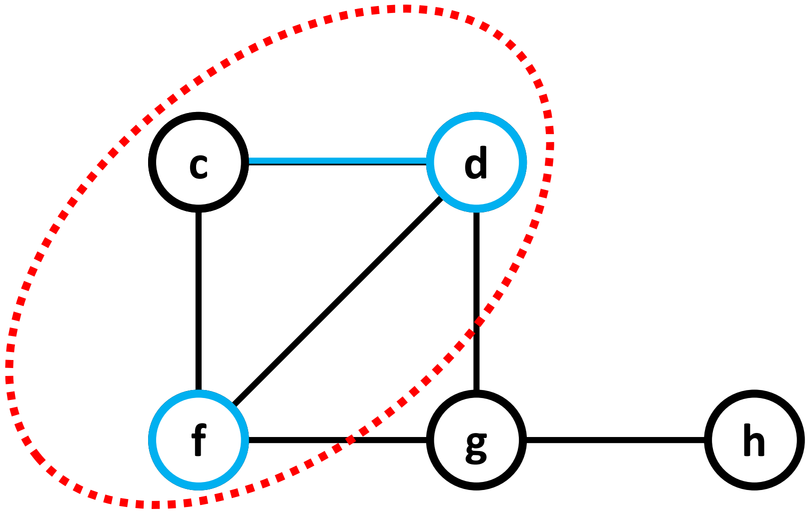
- Recall: nodes represent separators

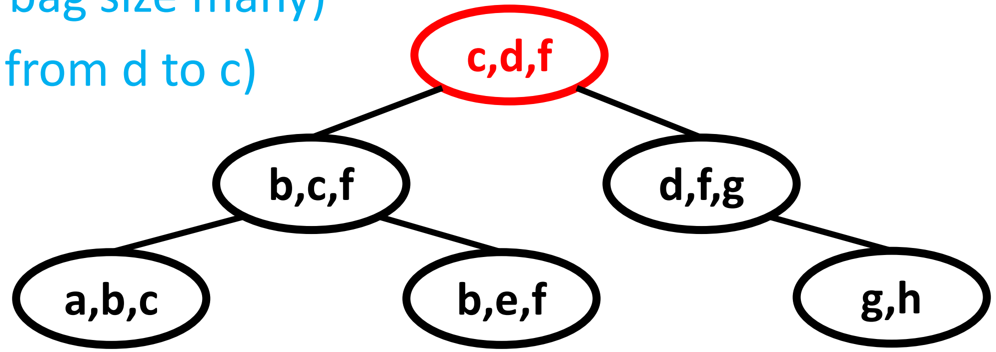
- Do dynamic programming
  - Don’t care what is “behind” the separator
  - Represent subsolutions with a signature



How does a Tree Decomposition help us?

- Recall: nodes represent separators
- Do dynamic programming
  - Don't care what is "behind" the separator
  - Represent subsolutions with a signature
    - Vertex-pairs at start/end of sub-walks (here: (d,f), at most bag size many)
    - Edges between vertices in the bag in the sub-walks (here: from d to c)
    - For each signature, only store min cost subsolution
    - Subsolutions must contain all waypoints in subgraph
- # signatures for treewidth tw: $2^{O(tw^2)}$ per bag at most





Easier Dynamic Programming: Nice Tree Decomposition

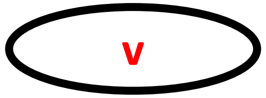
- See for example:
  - [T. Kloks, "*Treewidth, Computations and Approximations*", LNCS 842, 1994.]

Easier Dynamic Programming: Nice Tree Decomposition

- Rooted tree decomposition with 4 node types:

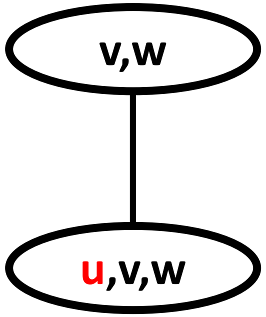
Easier Dynamic Programming: Nice Tree Decomposition

- Rooted tree decomposition with 4 node types:
  - **Leaf** (bag size of 1)



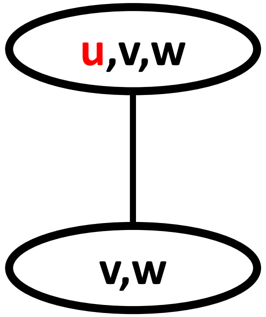
Easier Dynamic Programming: Nice Tree Decomposition

- Rooted tree decomposition with 4 node types:
  - **Leaf** (bag size of 1)
  - **Forget** (1 vertex leaves bag in parent node (only child))



Easier Dynamic Programming: Nice Tree Decomposition

- Rooted tree decomposition with 4 node types:
  - **Leaf** (bag size of 1)
  - **Forget** (1 vertex leaves bag in parent node (only child))
  - **Introduce** (1 vertex enters bag in parent node (only child))

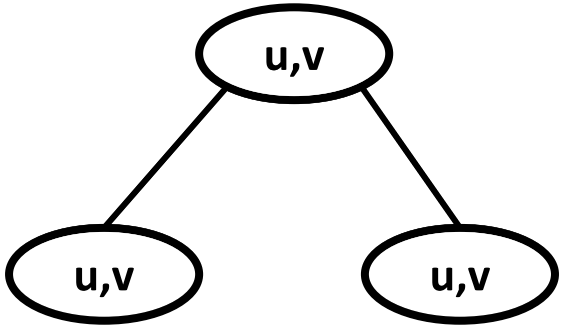


universität wien

DP bottom up

Easier Dynamic Programming: Nice Tree Decomposition

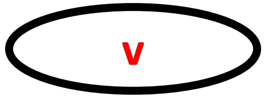
- Rooted tree decomposition with 4 node types:
  - **Leaf** (bag size of 1)
  - **Forget** (1 vertex leaves bag in parent node (only child))
  - **Introduce** (1 vertex enters bag in parent node (only child))
  - **Join** (Both children have same vertices)



- Min tw decomposition? NP-hard ☹
- But: Constant factor approximation in $O(c^{tw} \mathrm{n} \log tw\ )$ [Bodlaender et al., FOCS 2013] ☺
- Every tree decomposition can be made nice in $O(n * tw^2)$ time with $O(n * tw)$ nodes ☺

Leaf Nodes

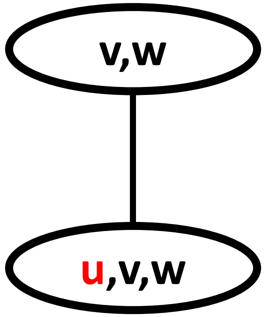
- A leaf node only contains a single vertex v
  - Just enumerate all options ☺
    - (v,v)
    - Empty signature (only valid if v is not a waypoint!)



- Runtime: $O(1)$

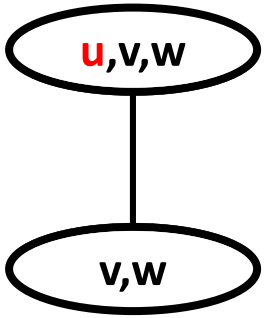
Forget Nodes

- Intuition:
  - For walks from the rest of the graph to reach u, they have to pass through the separator v,w
- In other words:
  - Take the signatures from the child node that
    - Don't contain u as an endpoint
    - Remove all edges incident to u
- Runtime: $2^{O(tw^2)}$



Introduce nodes

- Intuition:
  - In the new subsolutions, u only has neighbors from v,w

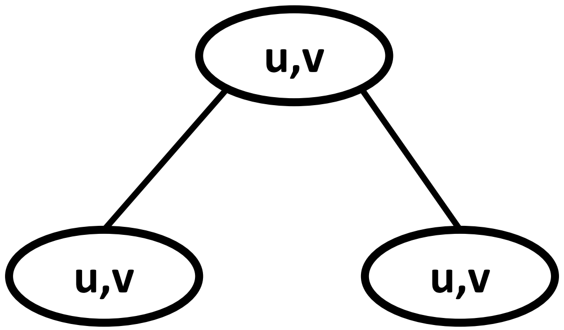


- Rough idea:
  - Take all signatures from the child node
  - Combine with new edges (can extend sub-walks or merge them…)
  - If u is a waypoint don't forget to cover it

- Runtime: $tw^{O(tw^2)}$

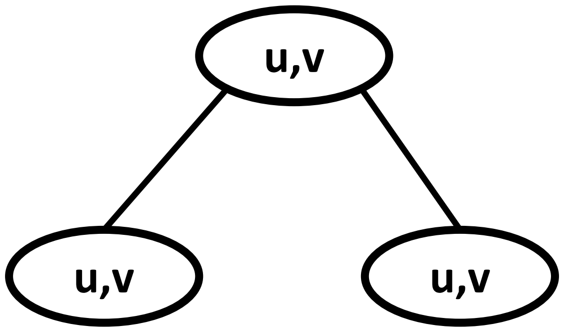
Join nodes

- So far: All node runtimes in $f(tw)$ – independent of V, E
- Classic approach: "*Glue*" subsolutions together, easy! ☺
  - E.g., for Hamiltonian Cycle problem
  - Does not work here ☹
    - (Problem: Too many crossings when "cutting" apart)



Join nodes

- Rather, coming backwards:
  - A valid subsolution of the join node can be separated
- As thus, going forward:
  - Take the edges of both child subsolutions
  - Merge them
  - Create all possible signatures/subsolutions for join node
- Runtime: $|V|^{O(tw)}2^{O(tw^2)}$



Total Runtime so far

- Nice tree decomposition:
  - $O(c^{tw} \mathrm{n} \log tw\ ) + O(n * tw^2)$ for some $c \in \mathbb{N}$

- Individual runtimes per node, $O(n * tw)$ at most:
  - Join: $O(1)$
  - Forget: $2^{O(tw^2)}$
  - Intoduce: $tw^{O(tw^2)}$
  - Join: $|V|^{O(tw)} 2^{O(tw^2)}$

- Total: In Class **XP**, i.e., $|V|^{f(tw)}$

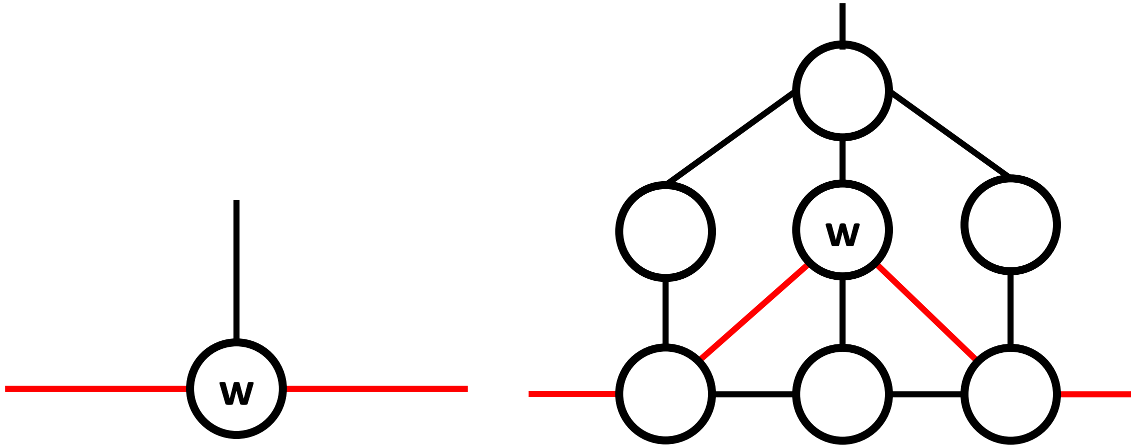
universität wien

Are we done?

- We actually forgot something:
  - Signatures limit #sub-walks to bag size
  - Will this yield an optimal solution?
- Observe: Optimal solution walk induces Eulerian Graph
- We can show, for any (A,B)-vertex-separator:
  - Walk can be separated appropriately
  - (Not shown here as I already talked long enough)

Walking through logarithmically many waypoints on general graphs

- Again: “Unify” graphs
  - All weights and capacities are 1

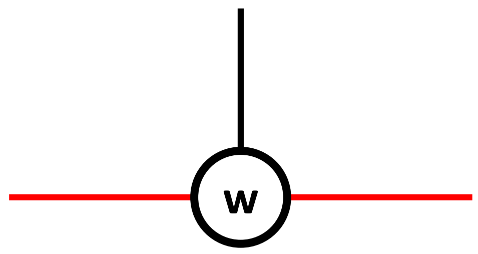


- Idea:
  - Create adapted line graph
  - Apply disjoint k-Cycle algorithms from Kawarabayash & Björklund et al.

- Deterministic and feasible: Polynomial runtime for $|W| \in O\left((\log\log n)^{1/10}\right)$
- Randomized and shortest tour: Runtime of $2^{|W|}n^{O(1)}$ (i.e., $|W| \in O(\log n)$)

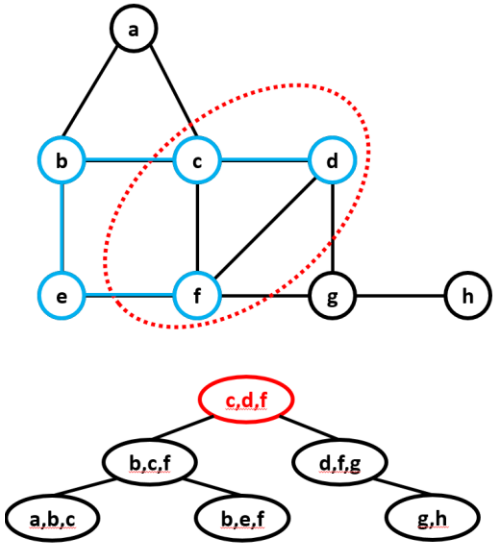
NP-hardness

- Idea:
  - Max degree 3? Enter and leave every node only once ☺
  - Look for problems where Hamiltonian Cycle is NP-hard
  - “Blow up“ number of nodes polynomially

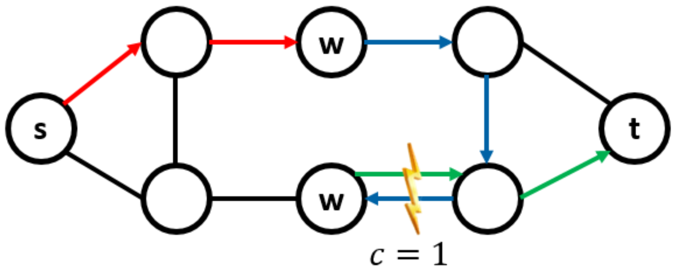


- NP-hard for any fixed constant r on
  - Grid graphs of maximum degree 3
  - 3-regular bipartite planar graphs

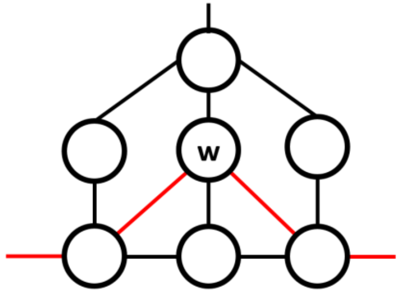
Summary: Unordered



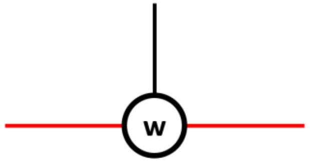
Bounded treewidth: in XP ($n^{O(tw^2)}$)
(i.e., polynomial for constant tw)



Analogy: Capacitated Subset TSP



General graphs & $|W| \in O(\log n)$:
polynomial runtime



Grid graphs & $|W| \in O(n^{1/r})$:
NP-hard

universität wien

Summary: Ordered

<table>
<tr><th></th><th># Waypoints</th><th>Feasible</th><th>Optimal</th><th>Demand Change Feasible</th><th>Optimal</th></tr>
<tr><td rowspan="3">Undirected</td><td>1</td><td colspan="2">P</td><td colspan="2" rowspan="2">Strongly NPC</td></tr>
<tr><td>constant</td><td>P</td><td>?</td></tr>
<tr><td>arbitrary</td><td colspan="4">Strongly NPC</td></tr>
<tr><td rowspan="3">Directed</td><td>1</td><td colspan="4" rowspan="3">Strongly NPC</td></tr>
<tr><td>constant</td></tr>
<tr><td>arbitrary</td></tr>
</table>

TABLE I
OVERVIEW OF THE COMPLEXITY LANDSCAPE FOR WAYPOINT ROUTING IN GENERAL GRAPHS.

universität wien

Summary: Ordered

<table>
<tr><th></th><th># Waypoints</th><th>Feasible</th><th>Optimal</th><th>Demand Change Feasible</th><th>Optimal</th></tr>
<tr><td rowspan="3">Undirected</td><td>1</td><td colspan="2">P</td><td colspan="2" rowspan="2">Strongly NPC</td></tr>
<tr><td>constant</td><td>P</td><td>?</td></tr>
<tr><td>arbitrary</td><td colspan="4">Strongly NPC</td></tr>
<tr><td rowspan="3">Directed</td><td>1</td><td colspan="4" rowspan="3">Strongly NPC</td></tr>
<tr><td>constant</td></tr>
<tr><td>arbitrary</td></tr>
</table>

TABLE I
OVERVIEW OF THE COMPLEXITY LANDSCAPE FOR WAYPOINT ROUTING IN GENERAL GRAPHS.

# Waypoints	Feasible Algorithms	Known Hardness	Demand Change Optimal Algorithms	Demand Change Hardness
Arbitrary	**P**: Outerplanar ($\mathtt{tw} \leq 2$)	**Strongly NPC**: $\mathtt{tw} \leq 3$	**P**: Tree (equivalent to $\mathtt{tw}$ of 1)	**NPC**: Unicyclic ($\mathtt{tw} \leq 2$)
Constant	**P**: General graphs	**P**: General graphs	**P**: Constant treewidth $\mathtt{tw} \in O(1)$	**Strongly NPC**: General graphs

TABLE II
OVERVIEW OF THE COMPLEXITY LANDSCAPE FOR WAYPOINT ROUTING IN SPECIAL UNDIRECTED GRAPHS.

Walking Through Middleboxes

Klaus-T. Foerster, University of Vienna

25 Jul 2018 @ Cornell University. Host: Nate Foster

Thank you ☺