Counting and sampling minimum cuts in weighted planar graphs

Ivona Bezáková

(Rochester Institute of Technology)

Based on joint works with Adam Friedlander and Zach Langley.

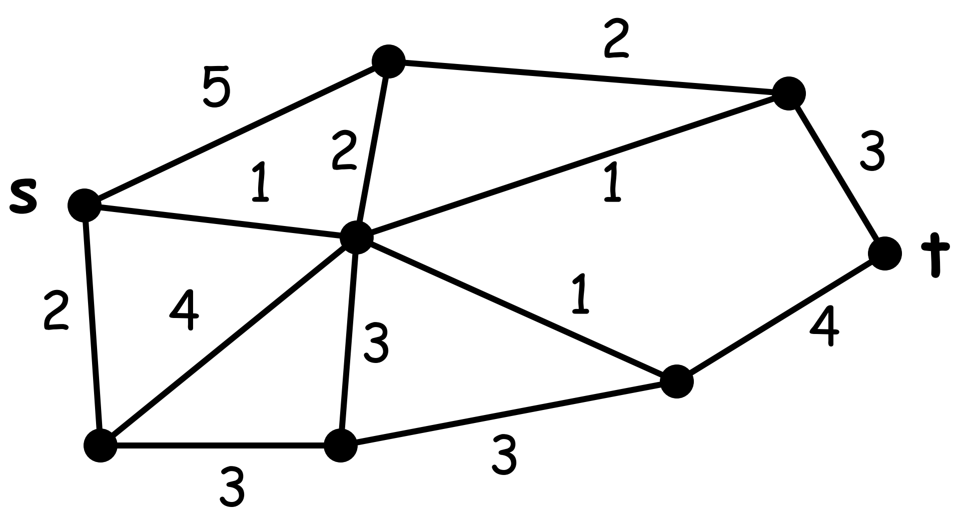
Discrete Lattice Models in Mathematics, Physics, and Computing
MSRI, Berkeley, January 13, 2012

Problem 1: Counting minimum (s,t)-cuts

Input: a positively weighted (directed) planar graph $G=(V,E)$ and two vertices s,t

Output: the number of minimum (s,t)-cuts of G

Example:



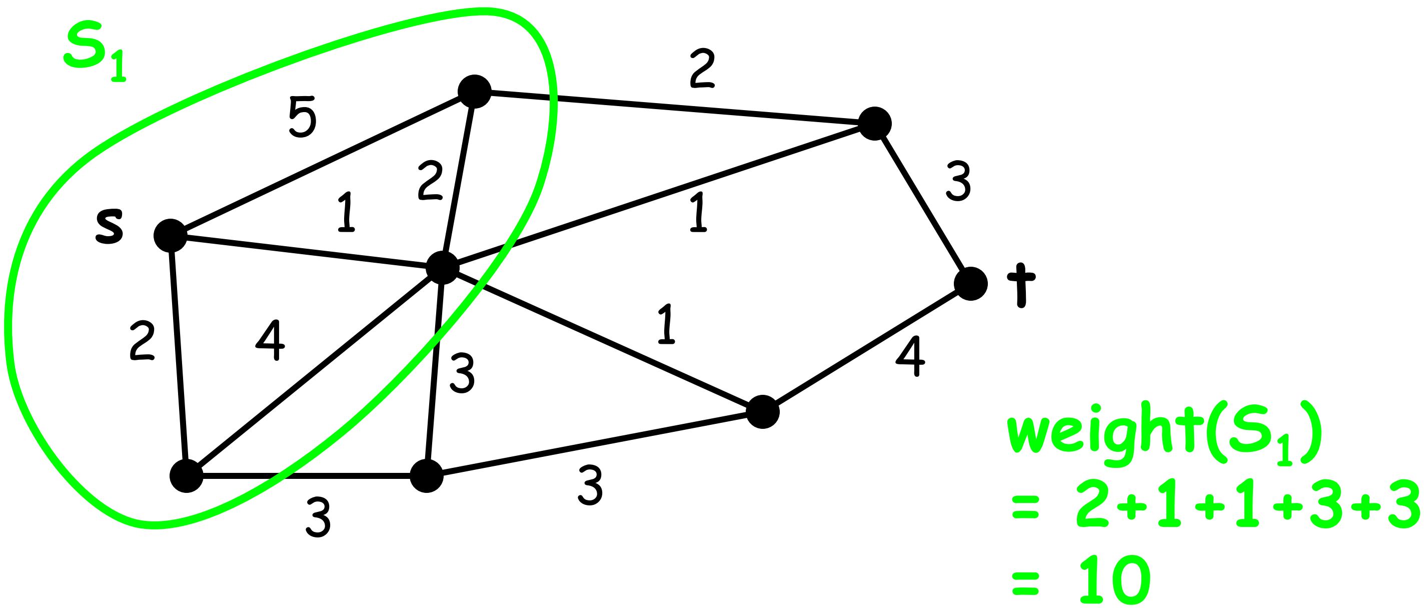
Recall: (s,t)-cut is a set $S \subseteq V$ containing s but not t; its weight is the sum of edge weights out of S

Problem 1: Counting minimum (s,t)-cuts

Input: a positively weighted (directed) planar graph $G=(V,E)$ and two vertices s,t

Output: the number of minimum (s,t)-cuts of G

Example:



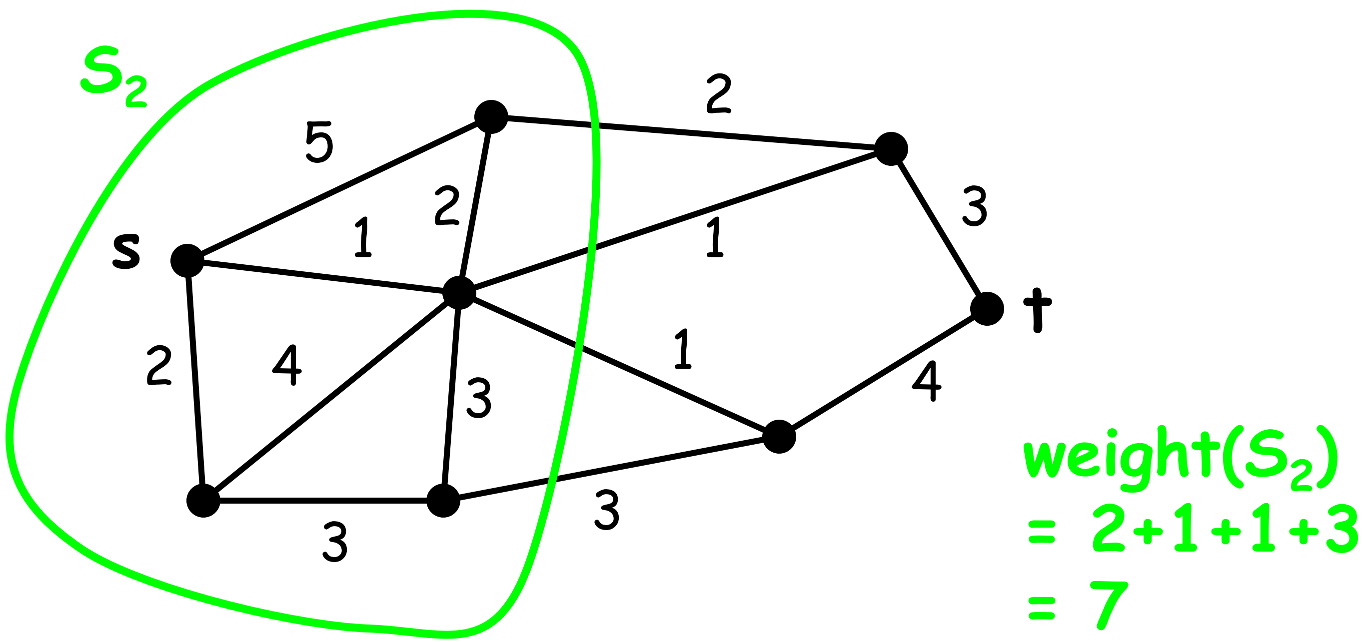
Recall: (s,t)-cut is a set $S \subseteq V$ containing s but not t; its weight is the sum of edge weights out of S

Problem 1: Counting minimum (s,t)-cuts

Input: a positively weighted (directed) planar graph G=(V,E) and two vertices s,t

Output: the number of minimum (s,t)-cuts of G

Example:



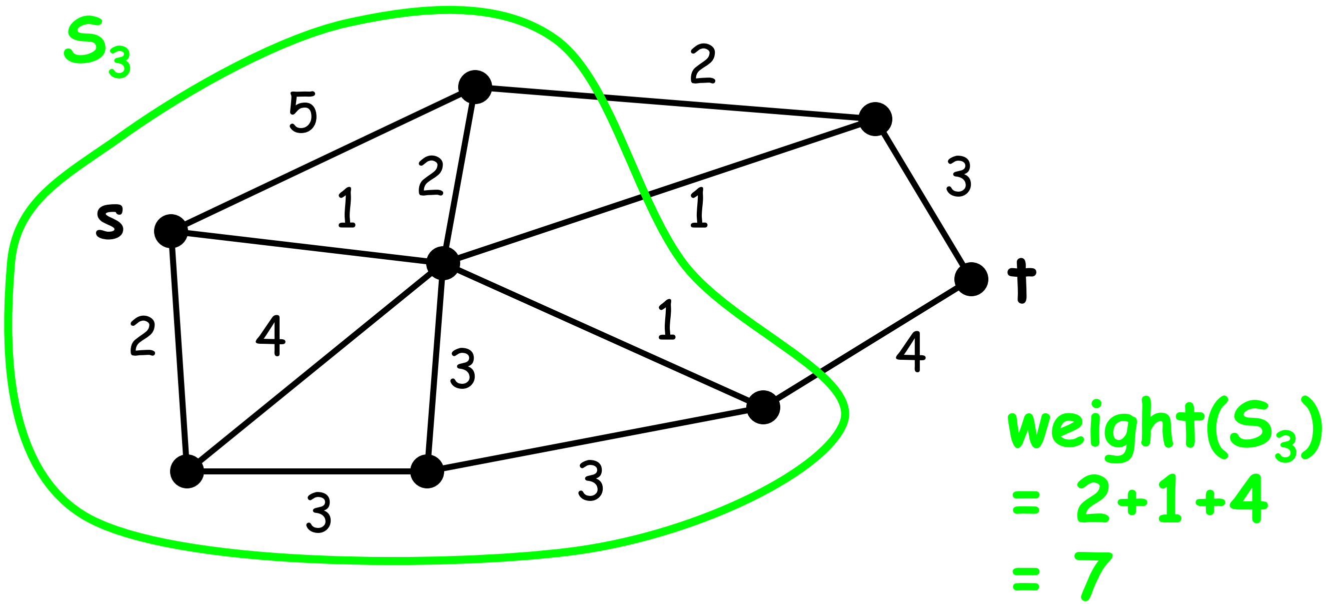
Recall: (s,t)-cut is a set $S \subseteq V$ containing s but not t; its weight is the sum of edge weights out of S

Problem 1: Counting minimum (s,t)-cuts

Input: a positively weighted (directed) planar graph $G=(V,E)$ and two vertices s,t

Output: the number of minimum (s,t)-cuts of G

Example:



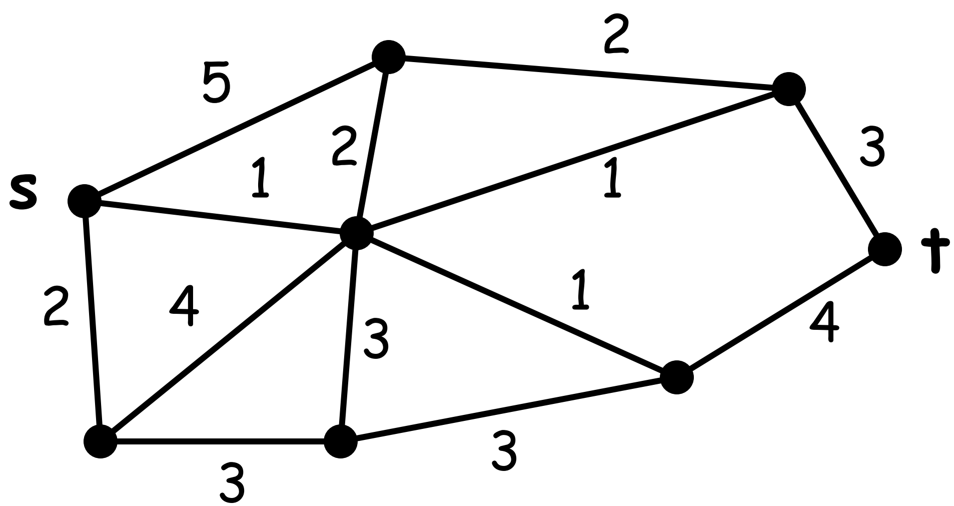
Recall: (s,t)-cut is a set $S \subseteq V$ containing s but not t; its weight is the sum of edge weights out of S

Problem 1: Counting minimum (s,t)-cuts

Input: a positively weighted (directed) planar graph $G=(V,E)$ and two vertices s,t

Output: the number of minimum (s,t)-cuts of G

Example:



Min (s,t)-cut weight: 7

Number of min (s,t)-cuts: 5

Motivation & related work

Earliest cut-counting works: **network reliability problems**

- Min-cuts / Disconnecting two vertices:
  - number of min (s,t)-cuts useful in estimating the probability of disconnecting the network, e.g., [Colbourn '05]
  - efficient poly-time counting in (unweighted) planar (multi-) graphs when s,t are on the same face [Ball and Provan '83]

Other, recent, motivation: **computer vision**

- e.g.: [Boykov & Veksler '06], [Boykov, Veksler, & Zabih '01], [Vicente, Kolmogorov, & Rother '08], [Zeng, Samaras, Chen, & Peng '08]

Motivation & related work

Other, recent, motivation: **computer vision**

- the simplest case: image segmentation where image represented by a planar graph

- user selects two points, the graph cut represents the segmentation

- currently in use only min-cut algorithms (optimization version), using an arbitrary min-cut

http://path.upmc.edu/cases/case123.html

- many advantages of counting (and the related sampling) versions, e.g.:
    - statistical tests
    - user can choose from several cuts
    - can be used to compute the partition function that can be used to estimate model parameters

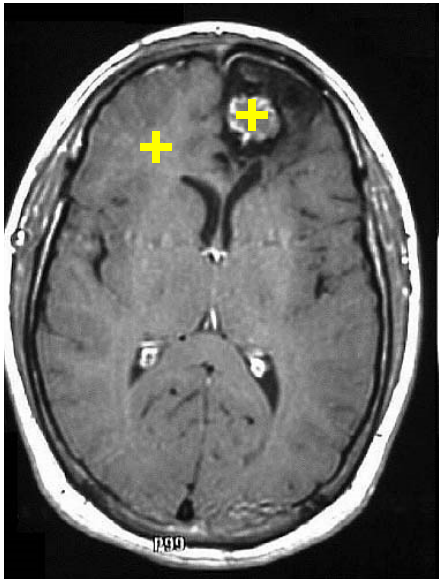
Motivation & related work

Other, recent, motivation: **computer vision**

- the simplest case: image segmentation where image represented by a planar graph

- user selects two points, the graph cut represents the segmentation

- currently in use only min-cut algorithms (optimization version), using an arbitrary min-cut



http://path.upmc.edu/cases/case123.html

- many advantages of counting (and the related sampling) versions, e.g.:

    - statistical tests

    - user can choose from several cuts

    - can be used to compute the partition function that can be used to estimate model parameters

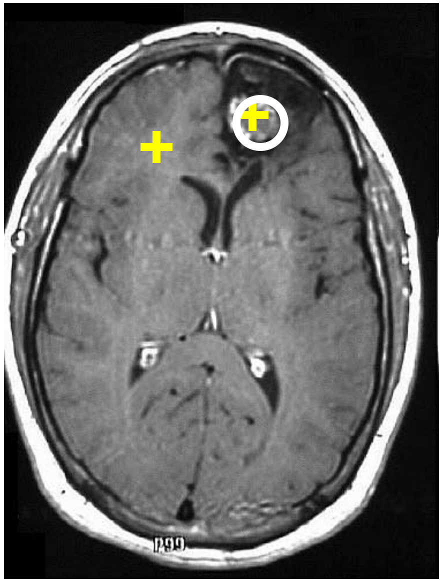
Motivation & related work

Other, recent, motivation: **computer vision**

- the simplest case: image segmentation where image represented by a planar graph

- user selects two points, the graph cut represents the segmentation

- currently in use only min-cut algorithms (optimization version), using an arbitrary min-cut



http://path.upmc.edu/cases/case123.html

- many advantages of counting (and the related sampling) versions, e.g.:

  - statistical tests

  - user can choose from several cuts

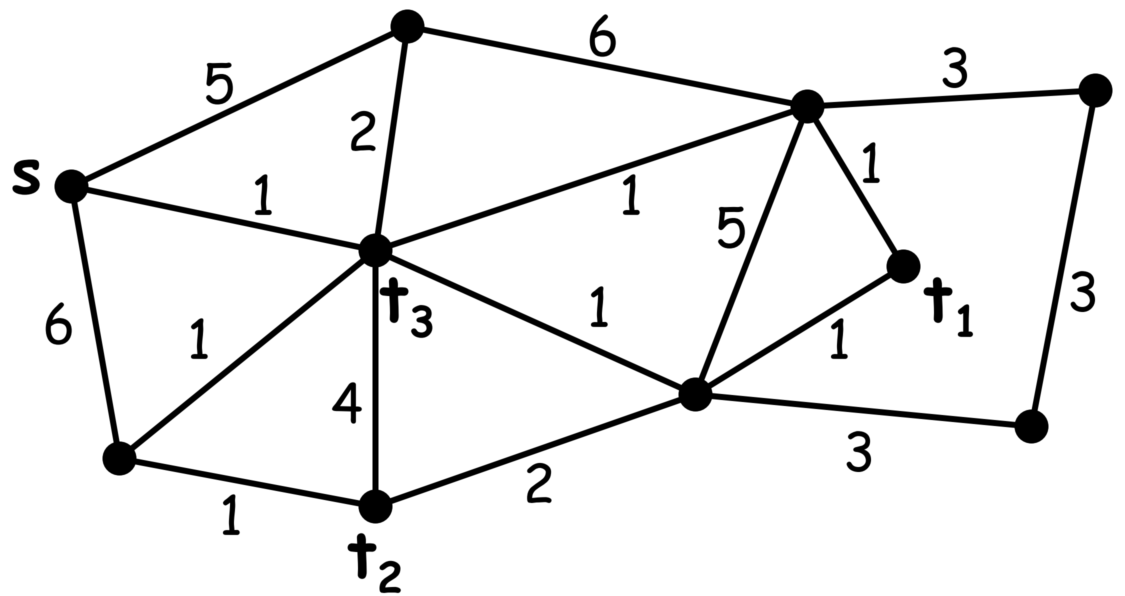
  - can be used to compute the partition function that can be used to estimate model parameters

Problem 2: Single-source-multi-sink cuts

Input: a positively weighted planar graph $G=(V,E)$,
a source vertex s and a set of sink vertices T ($s \notin T$)

Output: the number of contiguous minimum (s,T)-cuts

Example:

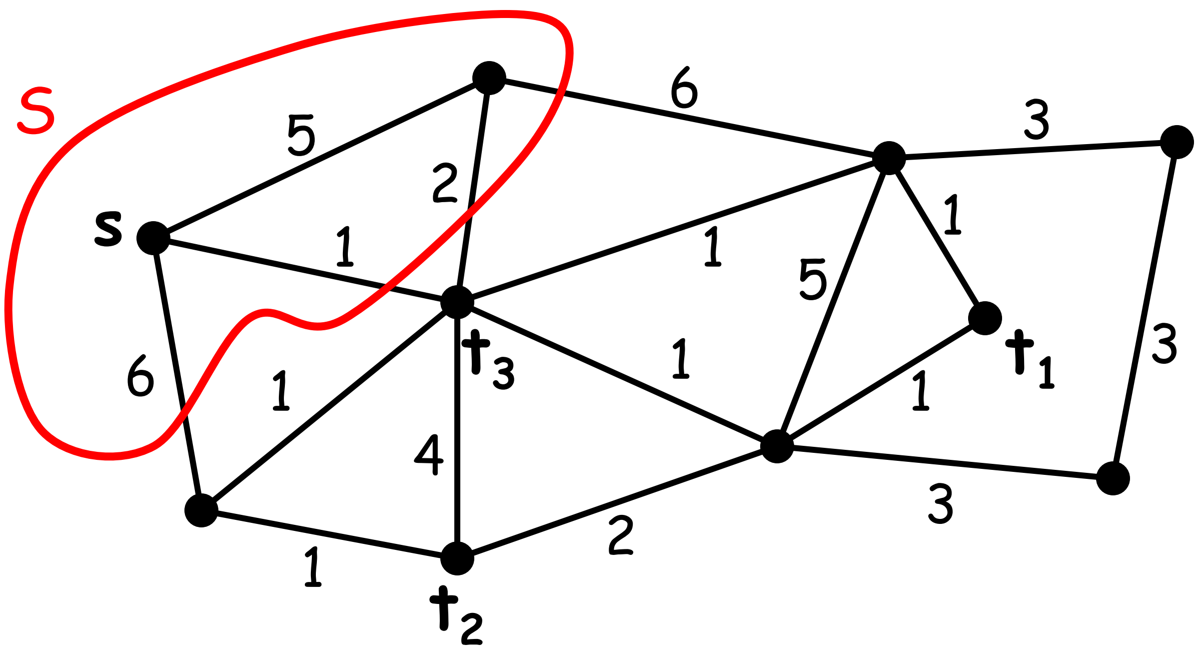


Problem 2: Single-source-multi-sink cuts

Input: a positively weighted planar graph $G=(V,E)$,
a source vertex s and a set of sink vertices T ($s \notin T$)

Output: the number of contiguous minimum (s,T)-cuts

Example:



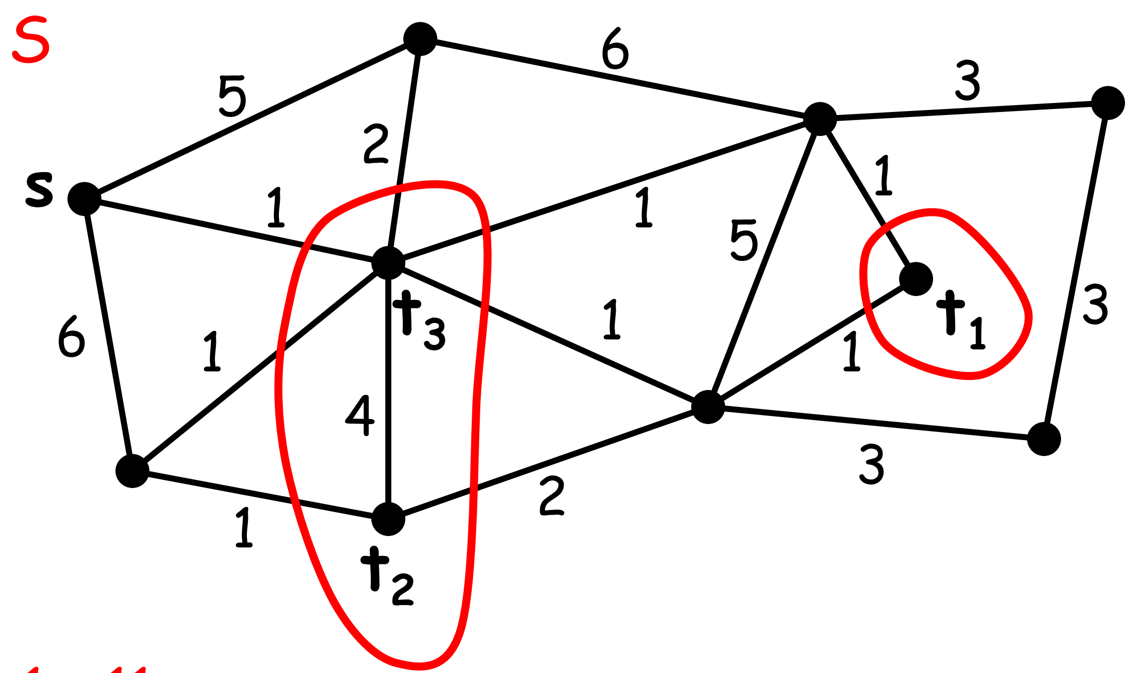
Weight(S) = 6+1+2+6 = 15

Problem 2: Single-source-multi-sink cuts

Input: a positively weighted planar graph $G=(V,E)$,
a source vertex s and a set of sink vertices T ($s \notin T$)

Output: the number of contiguous minimum (s,T)-cuts

Example:



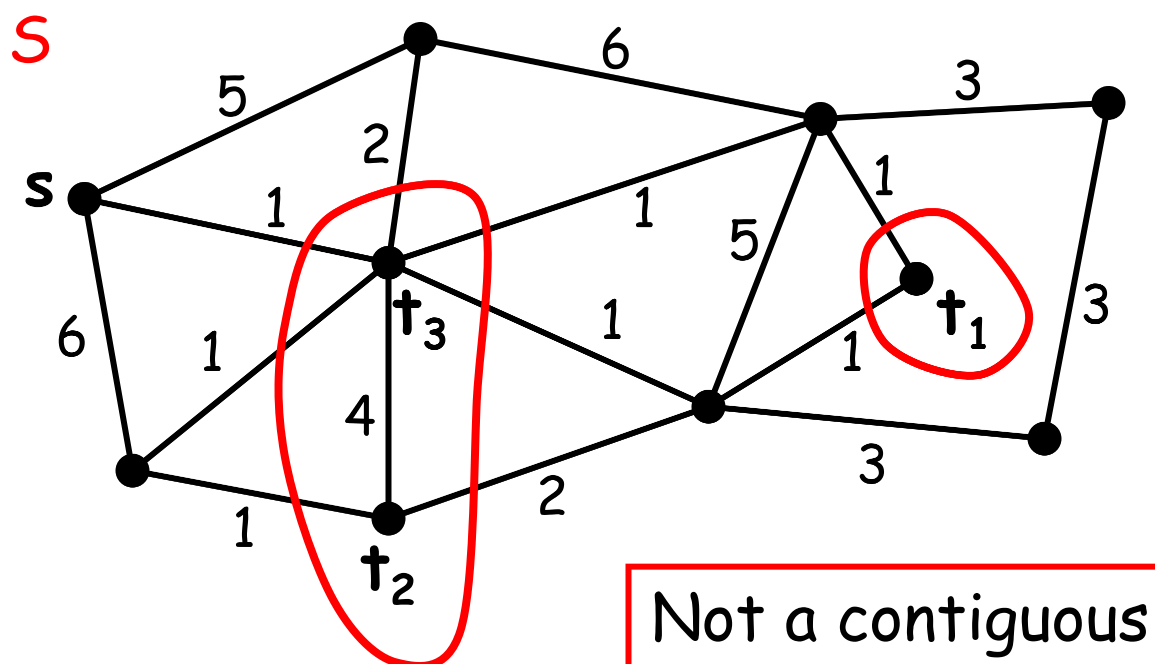
Weight(S) =
1+1+1+2+1+1+2+1+1 = 11

Problem 2: Single-source-multi-sink cuts

Input: a positively weighted planar graph $G=(V,E)$,
a source vertex s and a set of sink vertices T ($s \notin T$)

Output: the number of contiguous minimum (s,T)-cuts

Example:



Weight(S) =
1+1+1+2+1+1+2+1+1 = 11

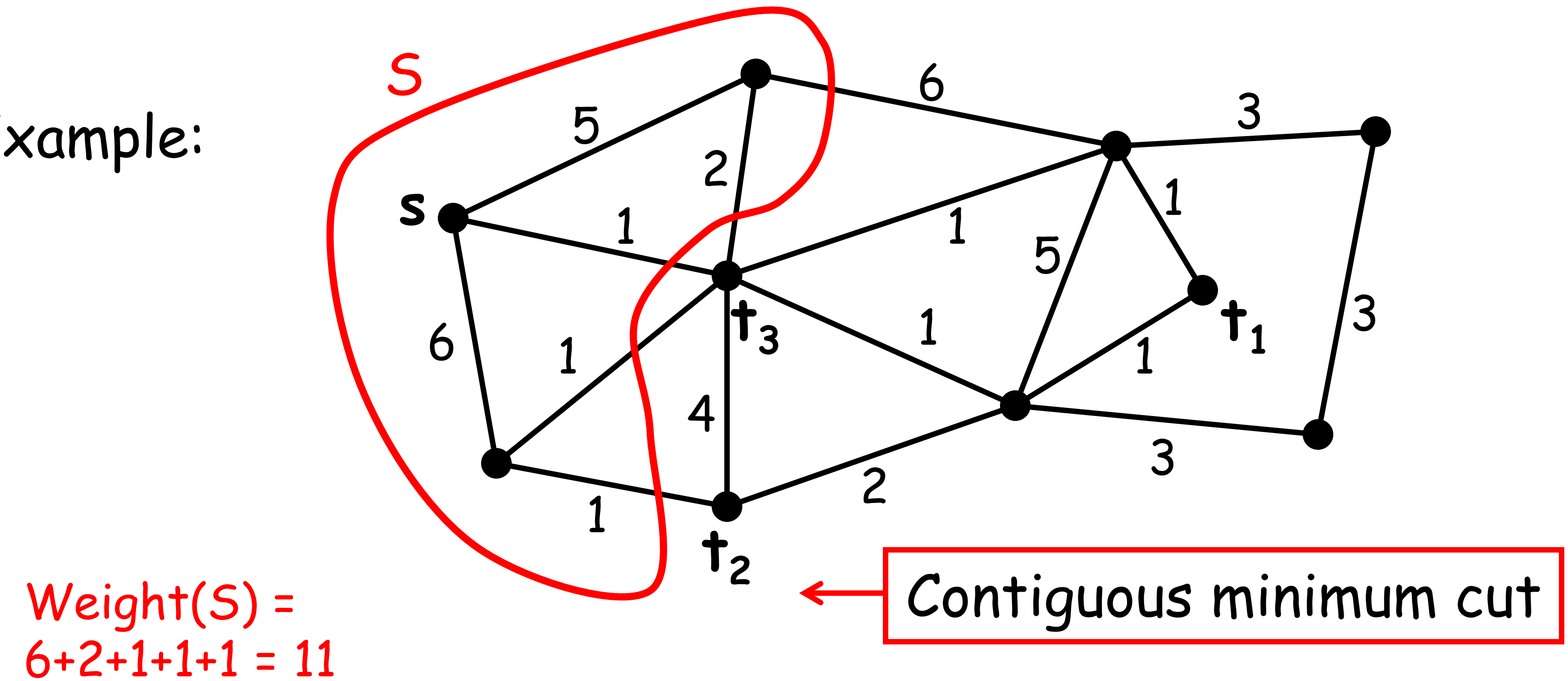
Not a contiguous cut !

Problem 2: Single-source-multi-sink cuts

Input: a positively weighted planar graph $G=(V,E)$,
a source vertex s and a set of sink vertices T ($s \notin T$)

Output: the number of contiguous minimum (s,T)-cuts

Example:

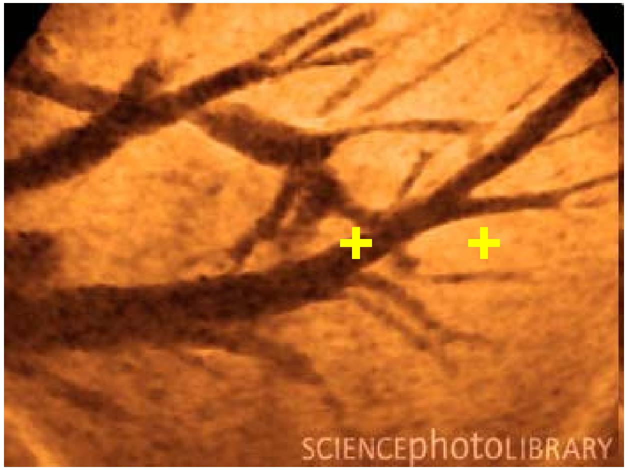


Motivation & related work – con't

Back to image segmentation:

- what about thin objects ?

    - user selects several points to guide the segmentation algorithm



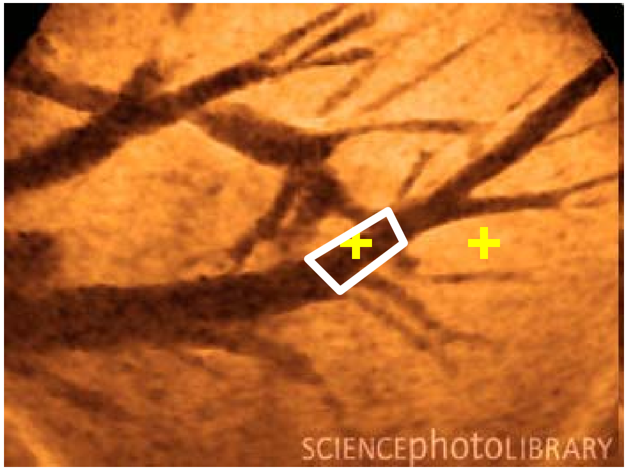
http://www.sciencephoto.com/media/310285/view

Motivation & related work - con't

Back to image segmentation:

- what about thin objects ?

  - user selects several points to guide the segmentation algorithm



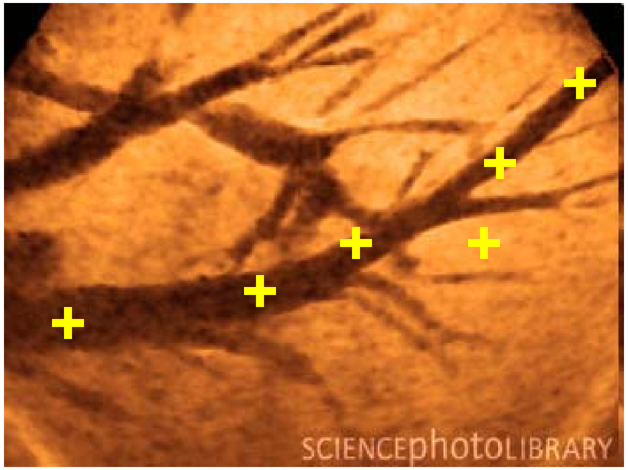
http://www.sciencephoto.com/media/310285/view

Motivation & related work - con't

Back to image segmentation:

- what about thin objects ?

    - user selects several points to guide the segmentation algorithm



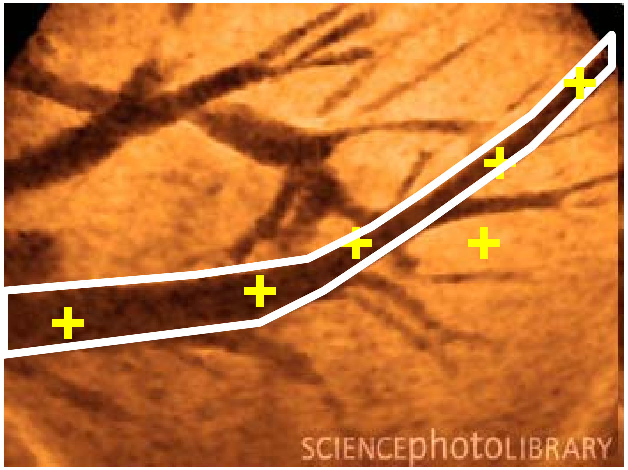
http://www.sciencephoto.com/media/310285/view

Motivation & related work - con't

Back to image segmentation:

- what about thin objects ?

    - user selects several points to guide the segmentation algorithm



http://www.sciencephoto.com/media/310285/view

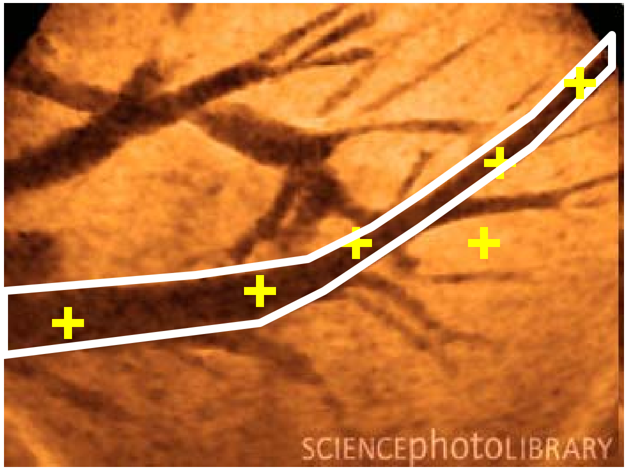
Motivation & related work - con't

Back to image segmentation:

- what about thin objects ?

    - user selects several points to guide the segmentation algorithm

    - contiguity requirements [connectivity priors]



http://www.sciencephoto.com/media/310285/view

Motivation & related work - con't

Recent progress in flow/cut algorithms for planar graphs [optimization problems, not counting/sampling]:

- [Borradaile & Klein '09]: O(n log n) single-source-single-sink acyclic max.flow

- [Borradaile, Sankowski, & Wulff-Nilsen '10]: fast computation of a minimum single-source-single-sink cut, for a sequence of given source-sink pairs

- [Italiano, Nussbaum, Sankowski, & Wulff-Nilsen '11]: undirected planar graphs, algos below O(n log n) time barrier

- [Borradaile, Klein, Mozes, Nussbaum, & Wulff-Nilsen '11]: $O(n \log^3 n)$ multi-source-multi-sinks max. flow

Motivation & related work - con't

Hardness results in general graphs:

- [Ball & Provan '83]: counting minimum (s,t)-cuts is #P-complete

- [Dyer, Goldberg, Greenhill, Jerrum '03]: AP-reduction between counting independent sets in bipartite graphs and counting upper sets of a poset

Our contributions

Thm 1: An $O(dn + n \log n)$ algorithm counting all minimum (s,t)-cuts in weighted planar graphs, where n is the number of vertices, and d is the distance from s to t in the unweighted graph.

Thm 2: An $O(n^3)$ algorithm counting all contiguous minimum single-source multi-sink cuts in weighted planar graphs.

[Can be used to find a contiguous minimum (s,T)-cut.]

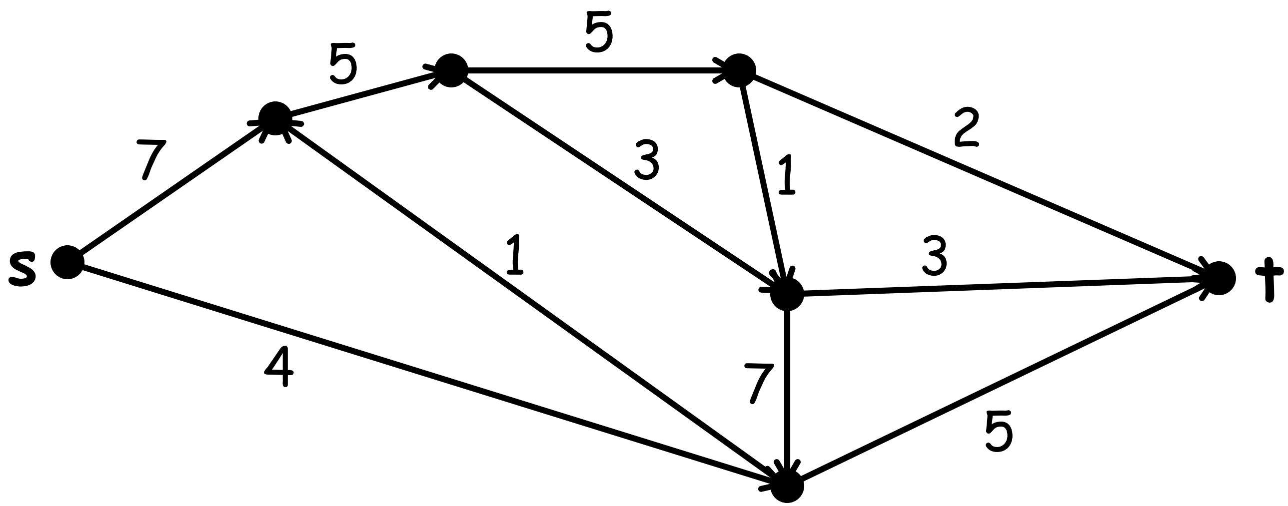
In both cases:

After this preprocessing time, we can produce a uniformly random minimum (s,t)-cut, resp. contiguous minimum (s,T)-cut, in linear time.

Review of network flows

Flow network:

- a directed graph with positive capacities on the edges, and
- two vertices s (the source) and t (the sink)



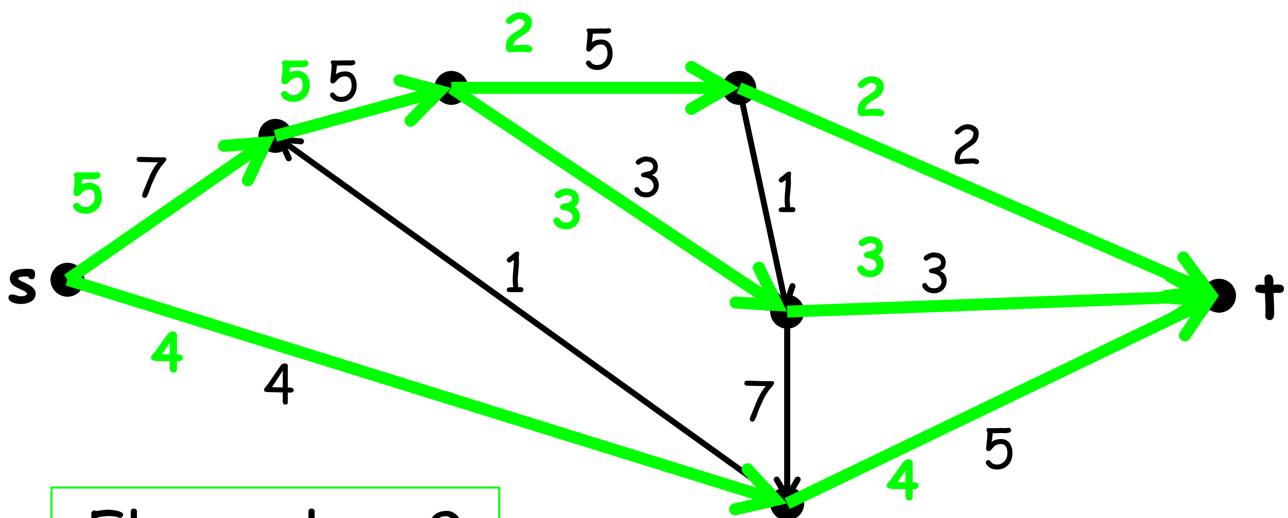
Review of network flows

Flow f: flow amount on every edge satisfying:

- for every edge e: flow amount f(e) $\leq$ capacity c(e), and
- for every vertex v (except s,t):

  flow amount into v = flow amount out of v

- flow value: amount out of s minus amount into s

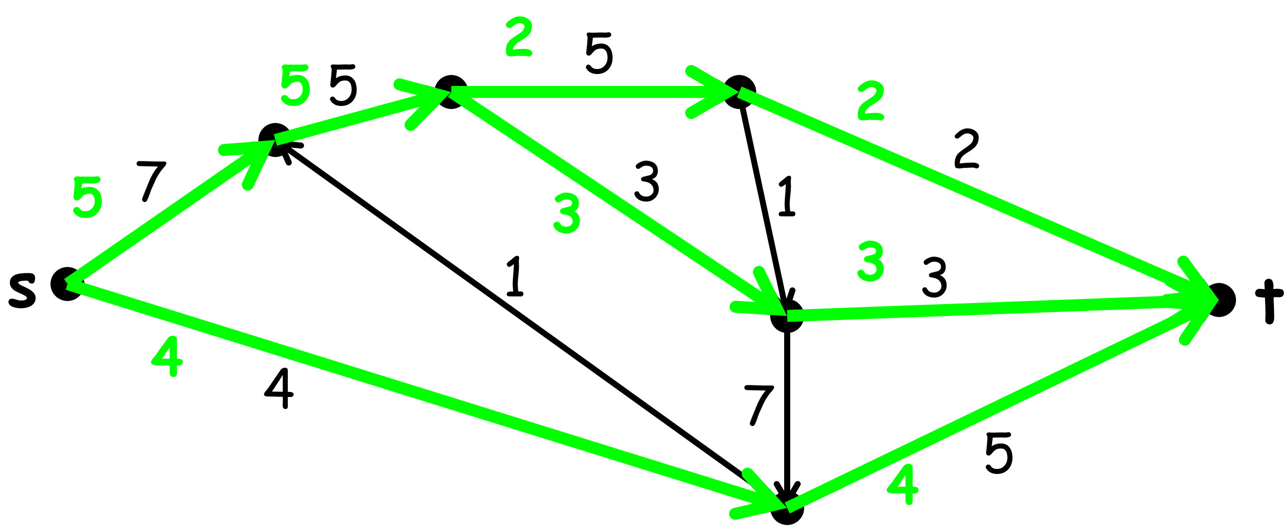


Flow value: 9

Review of network flows

Residual graph of a flow f:

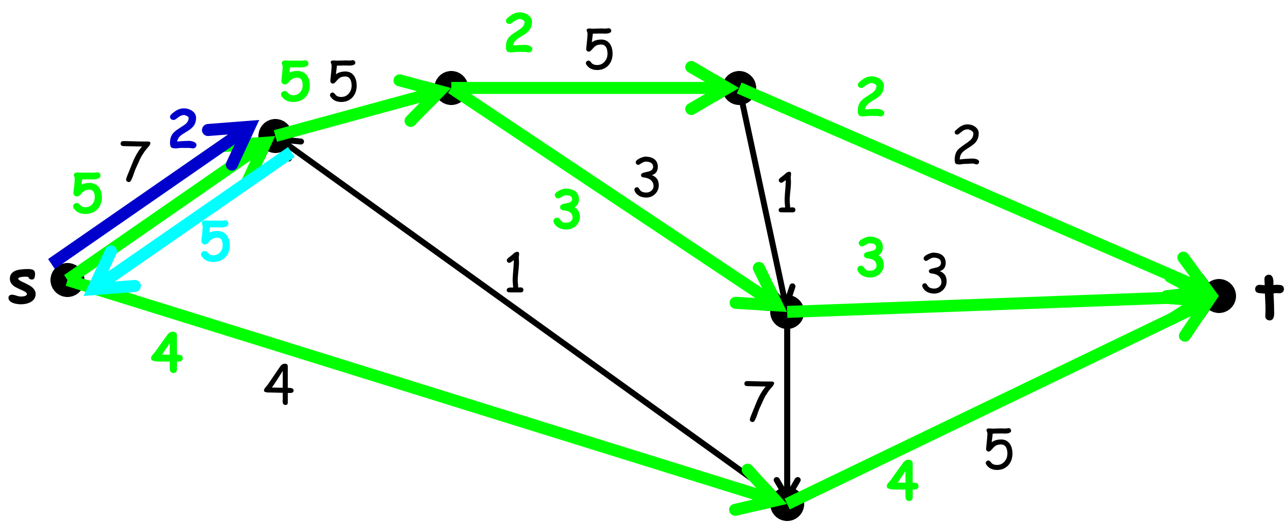
- **forward edges**: weight = capacity – flow
- **backward edges**: weight = flow



Review of network flows

Residual graph of a flow f:

- **forward edges**: weight = capacity – flow
- **backward edges**: weight = flow

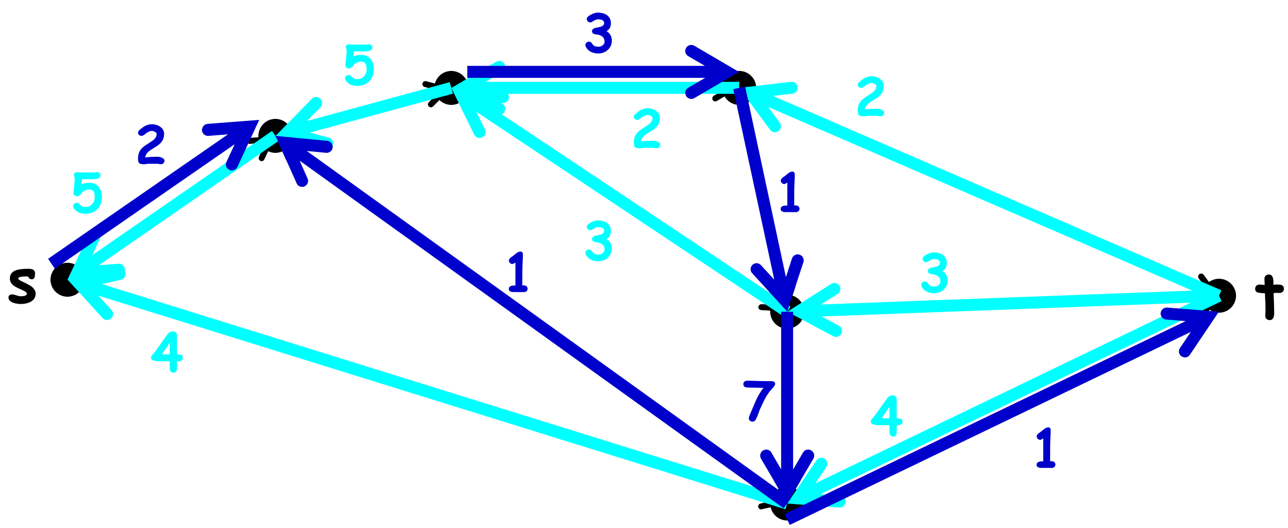


Review of network flows

Residual graph of a flow f:

- **forward edges**: weight = capacity – flow
- **backward edges**: weight = flow

(only edges with positive weight)

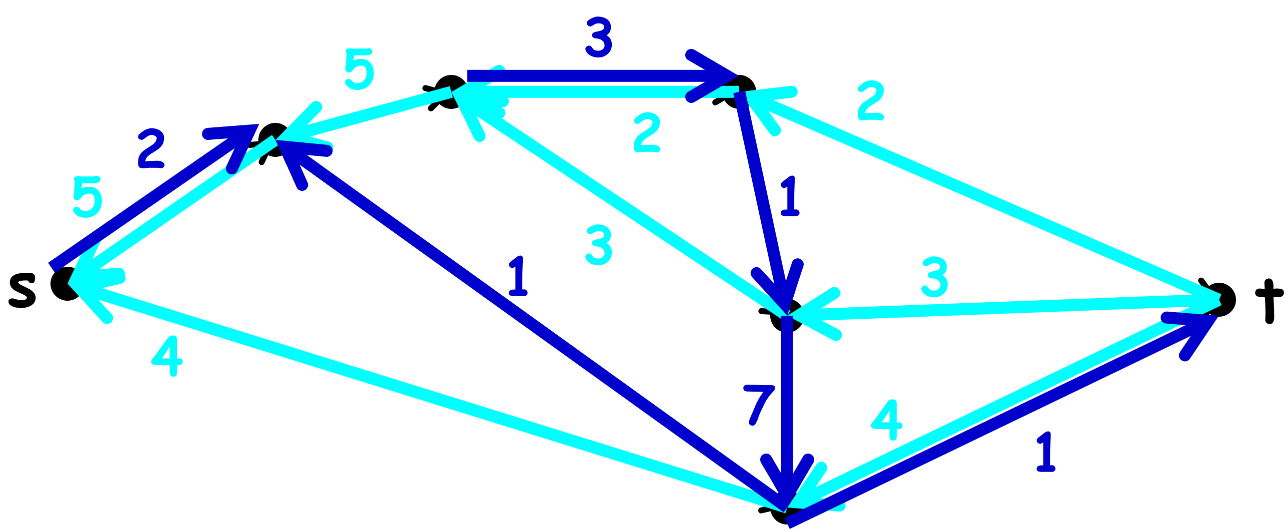


Review of network flows

Ford-Fulkerson Thm:

value of max s-t flow = value of min s-t cut

Note: flow is max iff no s-t path in the residual graph

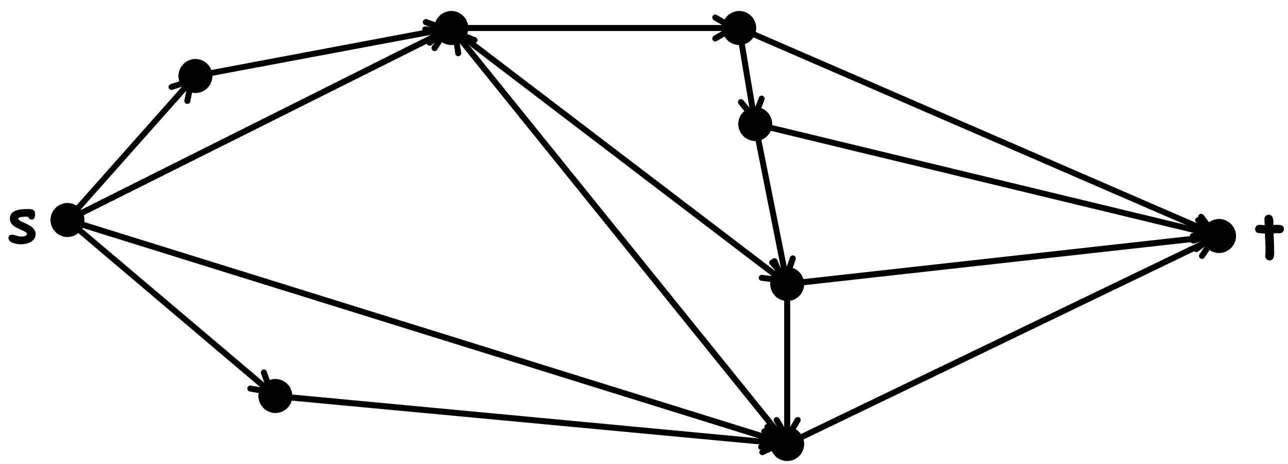


Ball & Provan's reduction

Given an unweighted (multi-)graph and vertices s,t:

1. Find a max flow and construct the residual graph
2. Contract strongly connected components
3. Compute # "forward-cuts" in the DAG

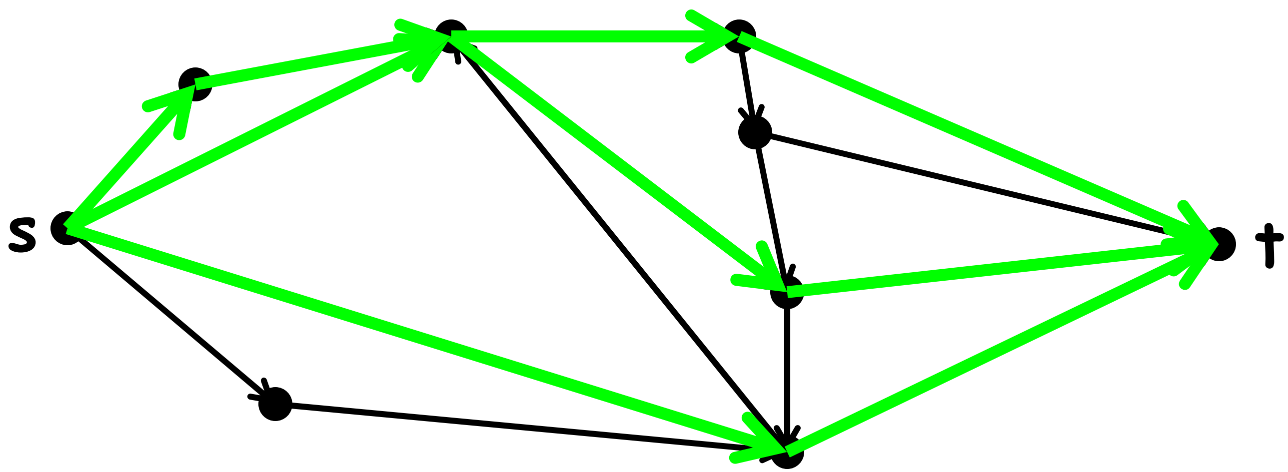
(forward-cuts = upper set / maximal antichains in the poset)



Ball & Provan's reduction

Given an unweighted (multi-)graph:and vertices s,t:

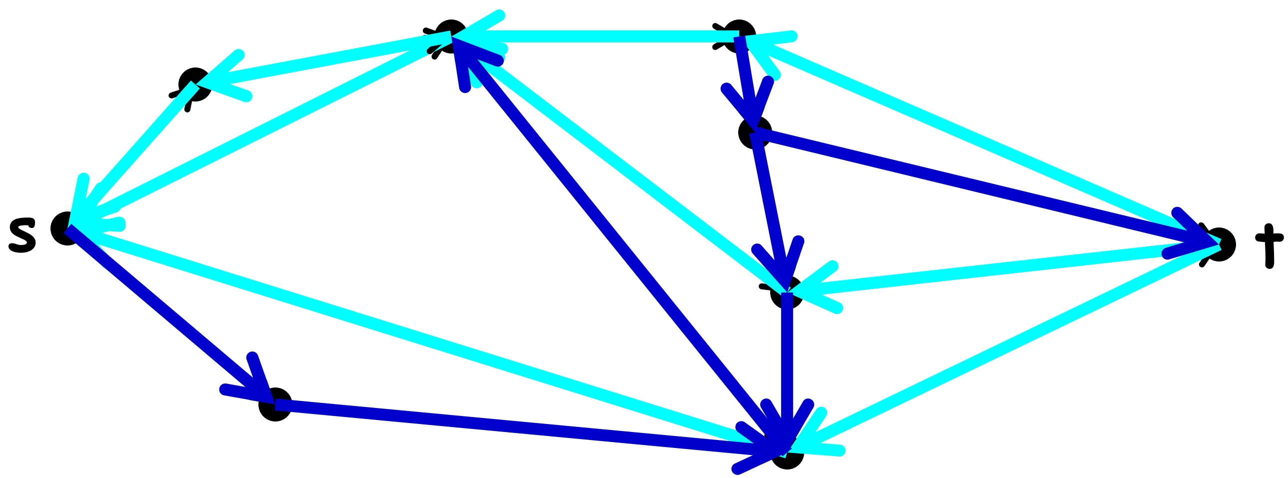
1. Find a max flow and construct the residual graph
2. Contract strongly connected components
3. Compute # "forward-cuts" in the DAG



Ball & Provan's reduction

Given an unweighted (multi-)graph and vertices s,t:

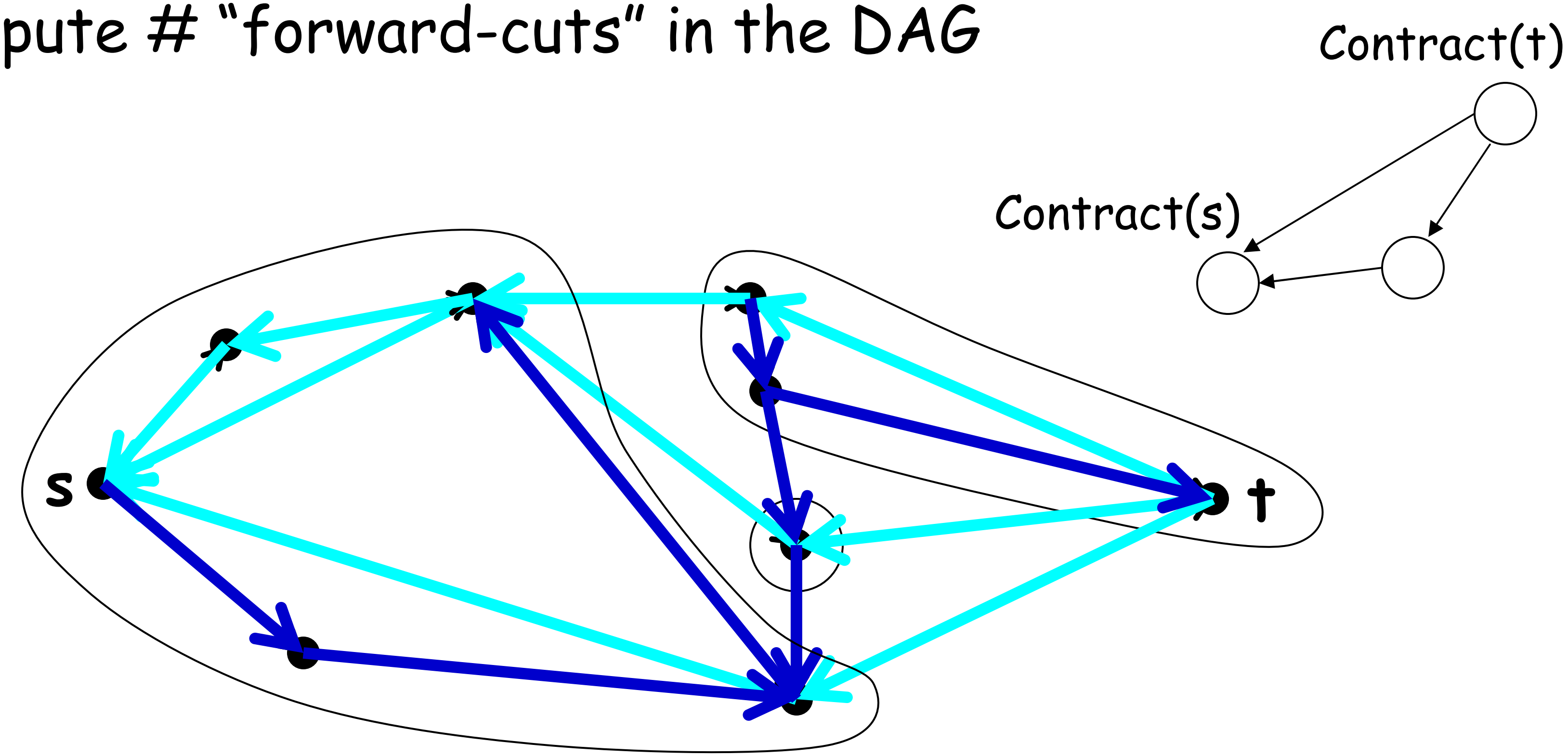
1. Find a max flow and construct the residual graph
2. Contract strongly connected components
3. Compute # "forward-cuts" in the DAG



Ball & Provan's reduction

Given an unweighted (multi-)graph and vertices s,t:

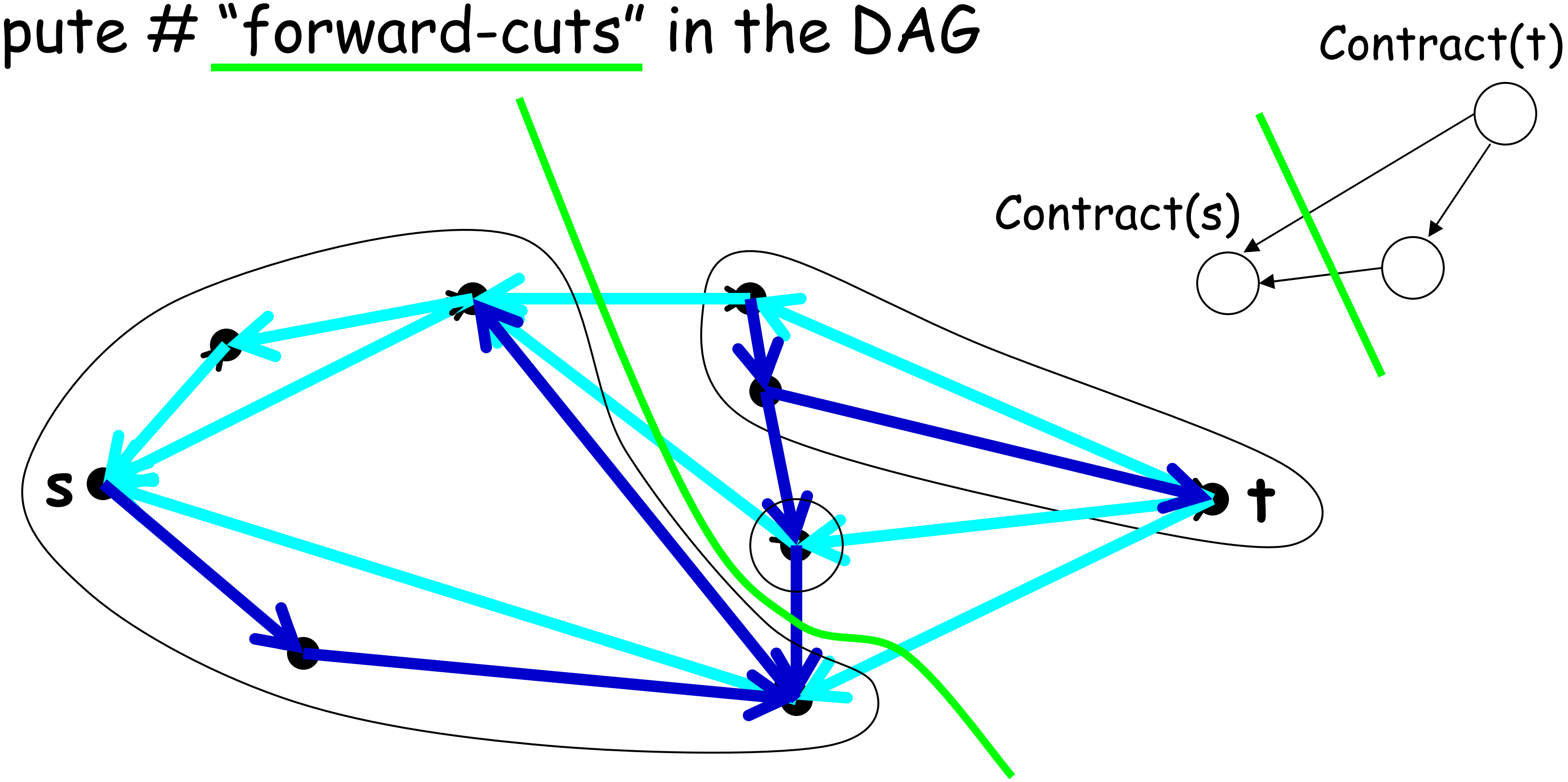
1. Find a max flow and construct the residual graph
2. Contract strongly connected components
3. Compute # "forward-cuts" in the DAG



Ball & Provan's reduction

Given an unweighted (multi-)graph and vertices s,t:

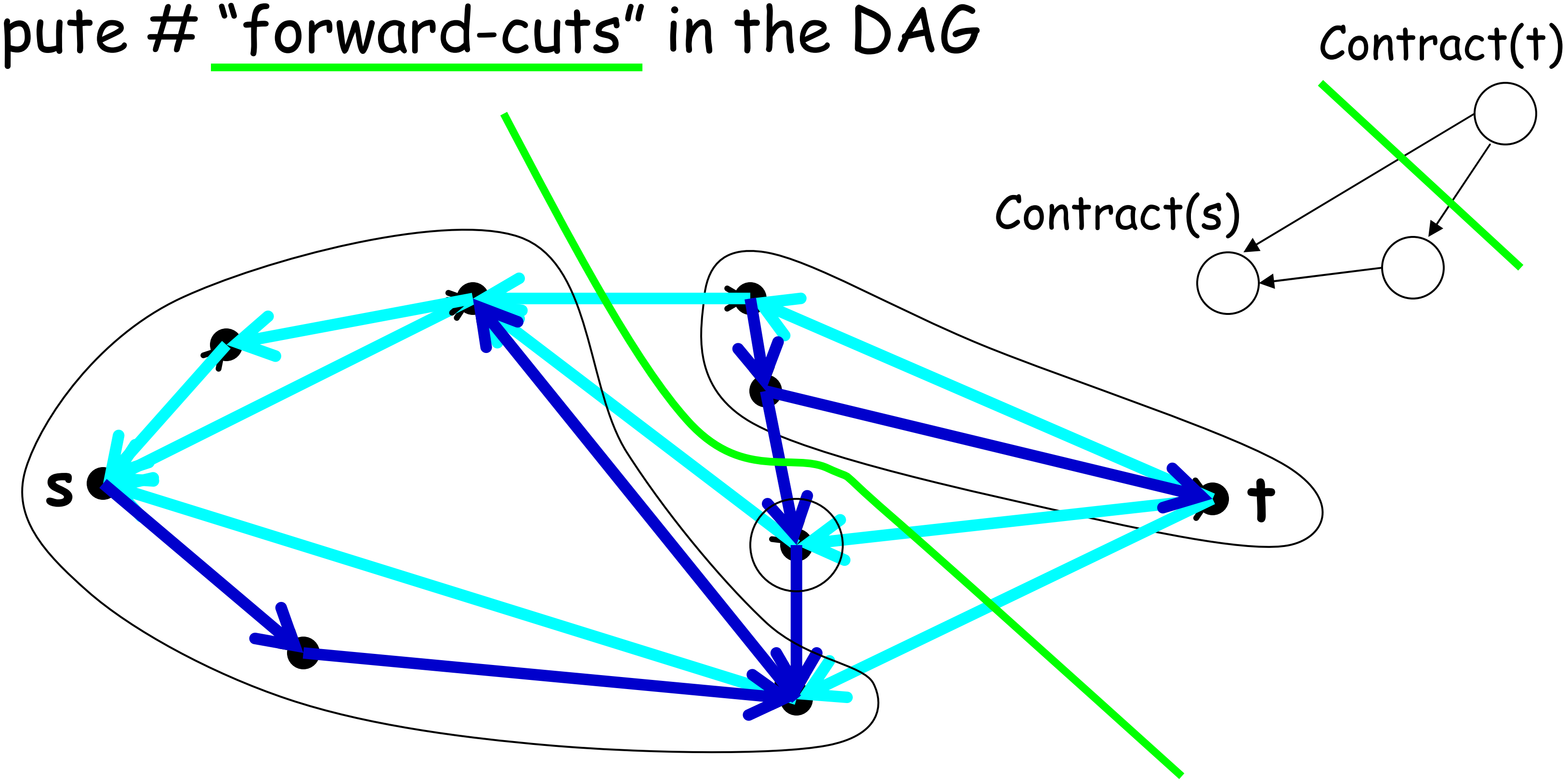
1. Find a max flow and construct the residual graph
2. Contract strongly connected components
3. Compute # "forward-cuts" in the DAG



Ball & Provan's reduction

Given an unweighted (multi-)graph and vertices s,t:

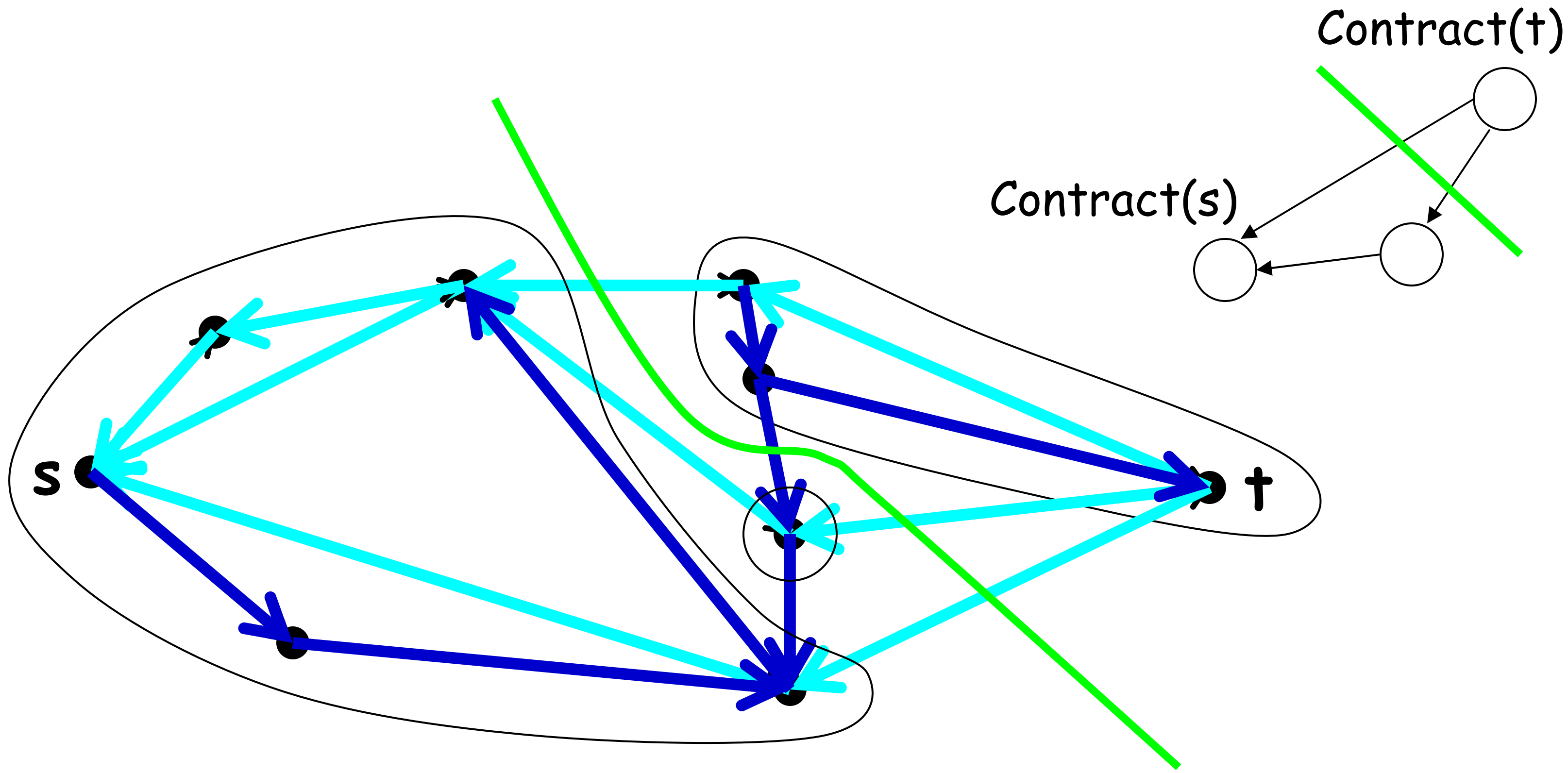
1. Find a max flow and construct the residual graph
2. Contract strongly connected components
3. Compute # "forward-cuts" in the DAG



Ball & Provan's reduction

"Forward-cut:" a set of vertices S such that:

- contains Contract(s) and not Contract(t), and
- for every vertex in S, all successors also in S

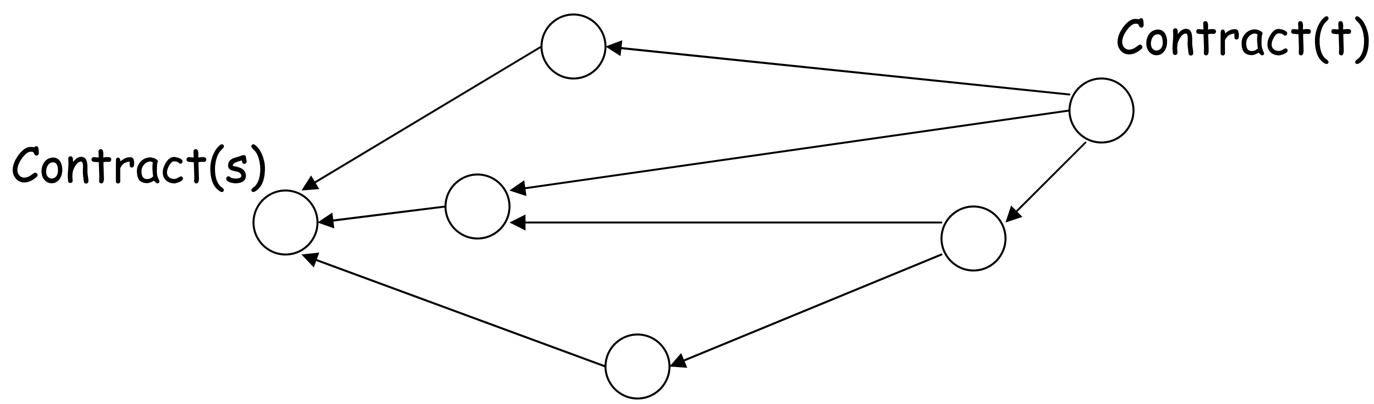


Ball & Provan's reduction

"Forward-cut:" a set of vertices S such that:

- contains Contract(s) and not Contract(t), and
- for every vertex in S, all successors also in S

Another DAG example:

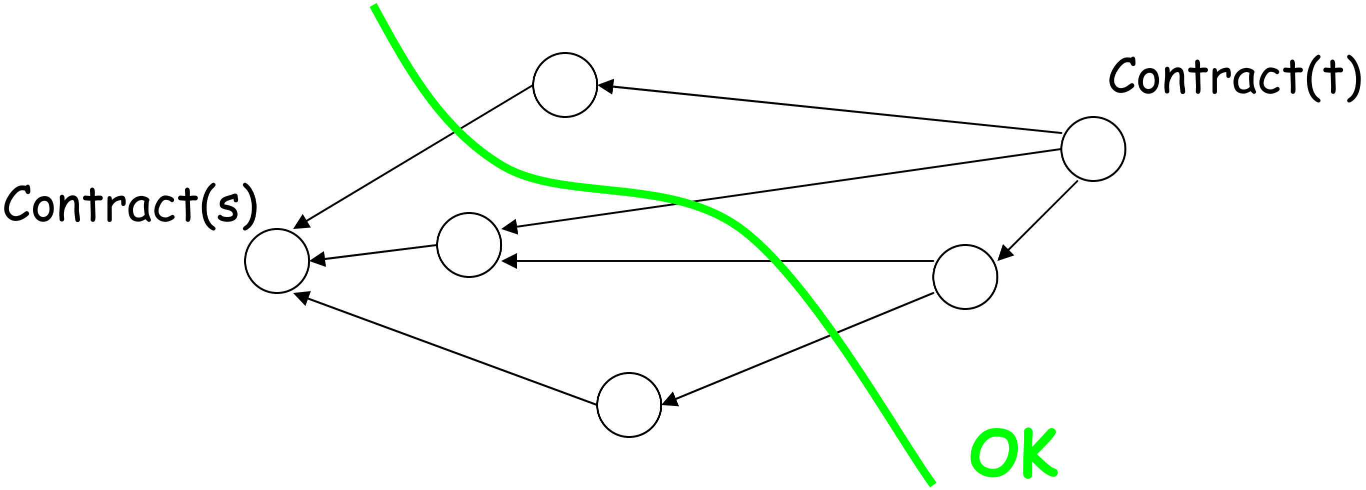


Ball & Provan's reduction

"Forward-cut:" a set of vertices S such that:

- contains Contract(s) and not Contract(t), and
- for every vertex in S, all successors also in S

Another DAG example:

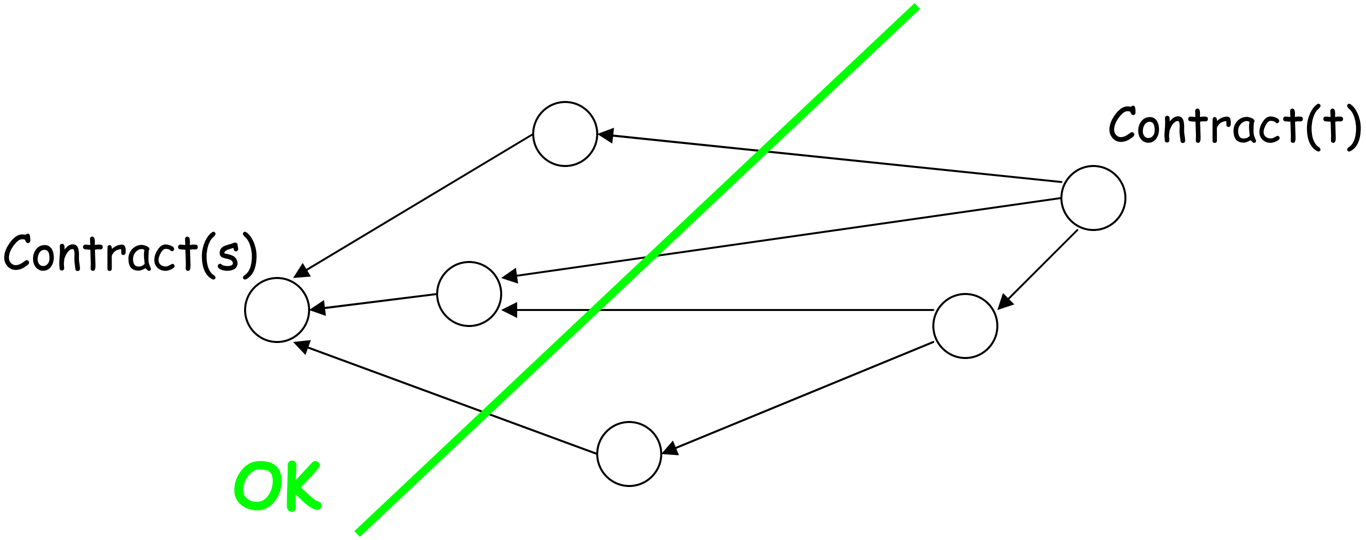


Ball & Provan's reduction

"Forward-cut:" a set of vertices S such that:

- contains Contract(s) and not Contract(t), and
- for every vertex in S, all successors also in S

Another DAG example:

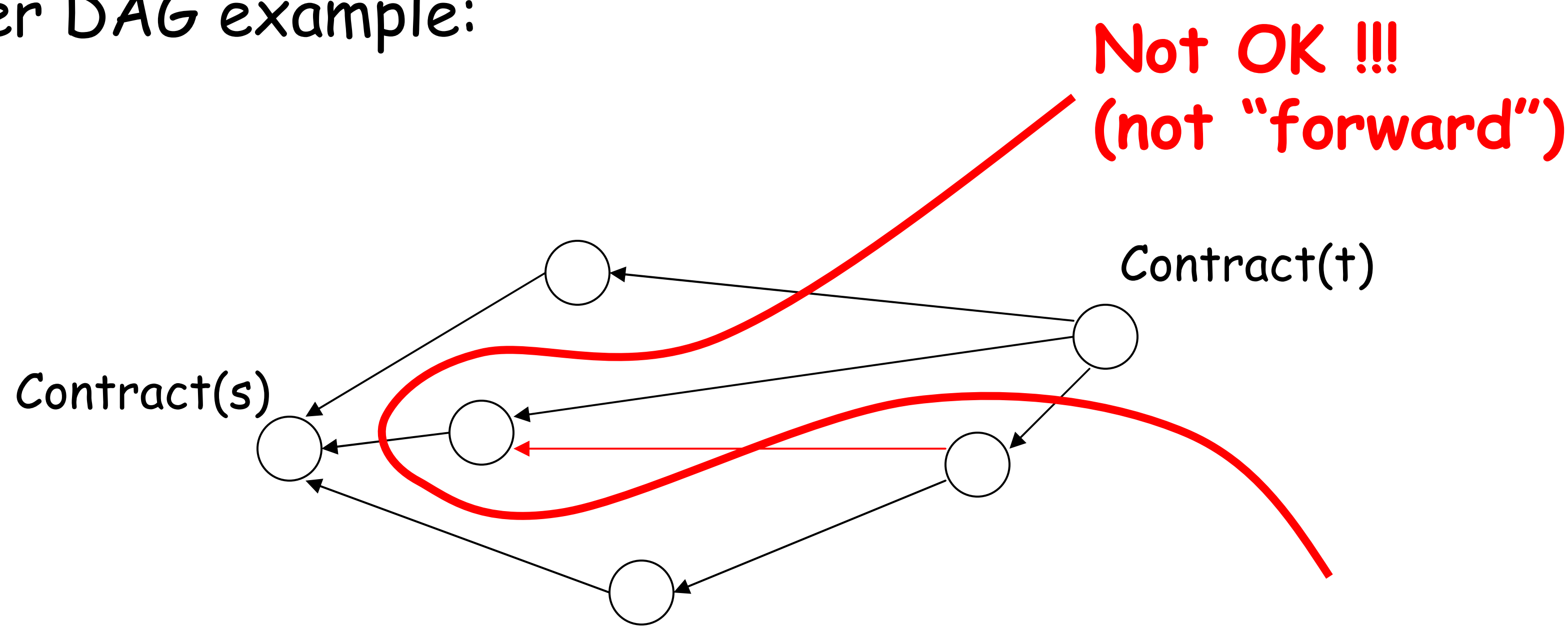


Ball & Provan's reduction

"Forward-cut:" a set of vertices S such that:

- contains Contract(s) and not Contract(t), and
- for every vertex in S, all successors also in S

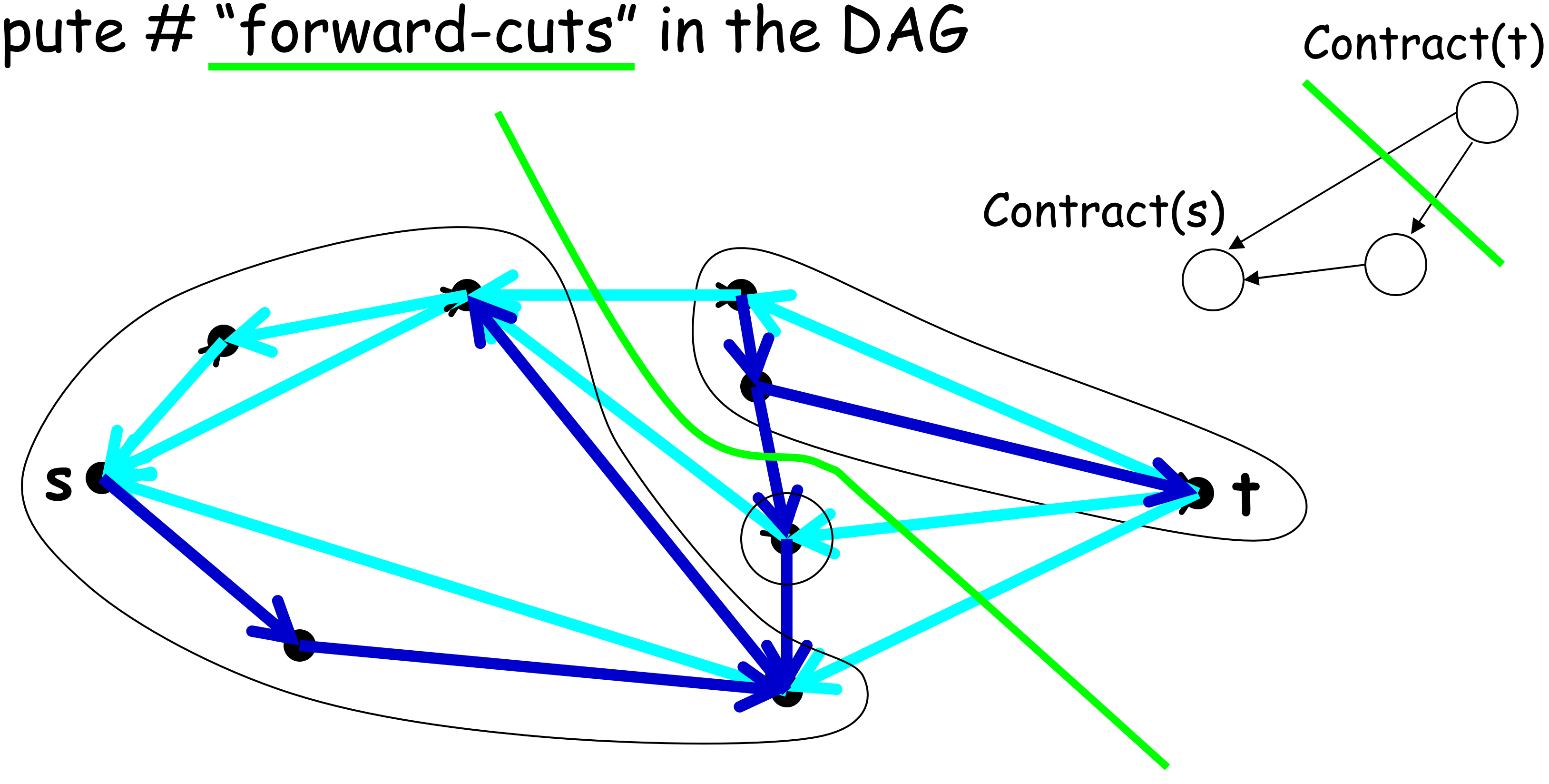
Another DAG example:



Ball & Provan's reduction

Given an unweighted (multi-)graph and vertices s,t:

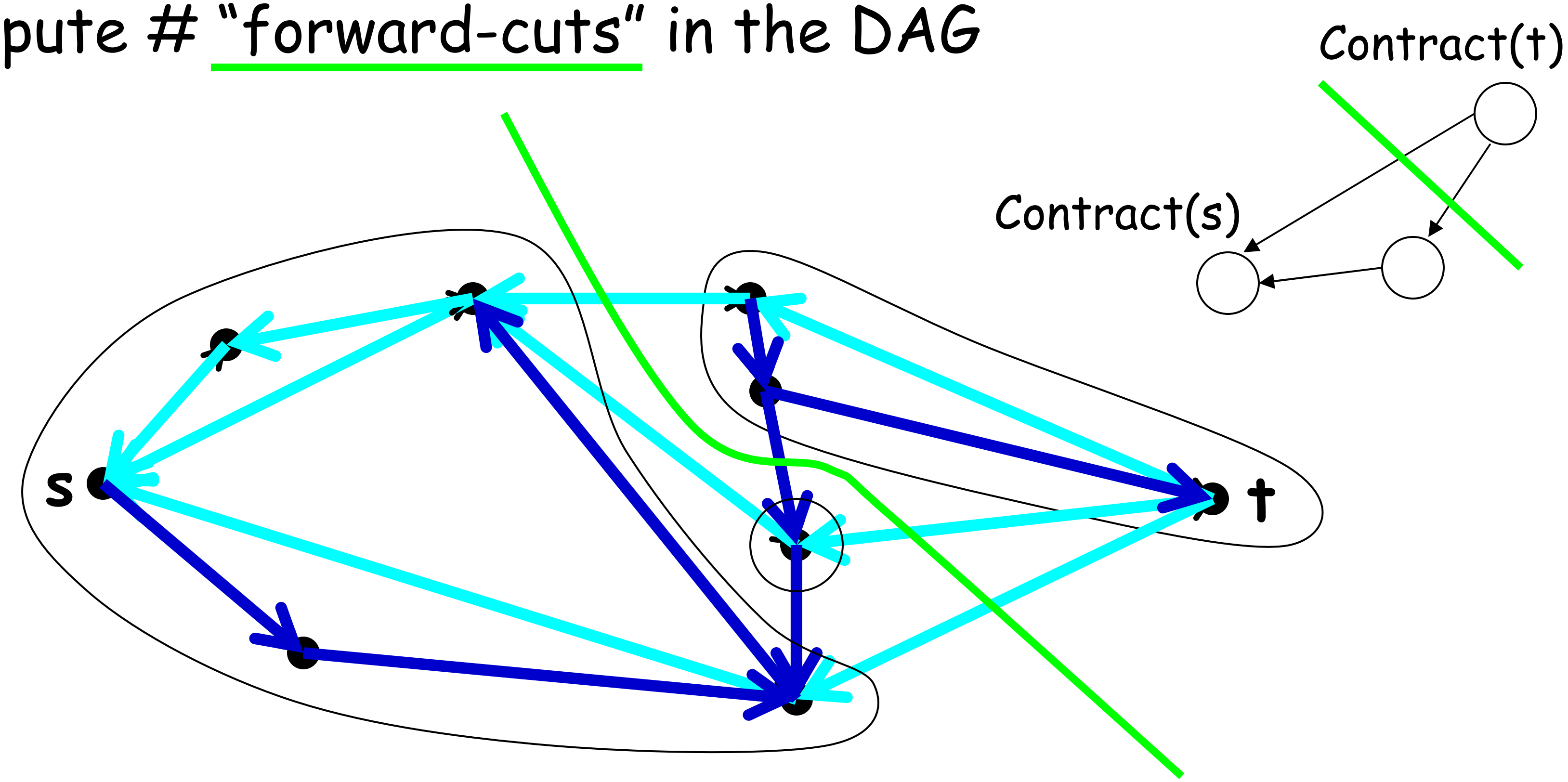
1. Find a max flow and construct the residual graph
2. Contract strongly connected components
3. Compute # "forward-cuts" in the DAG



Ball & Provan's reduction

Given an ~~unweighted~~ positively weighted (multi-)graph and vertices s,t:

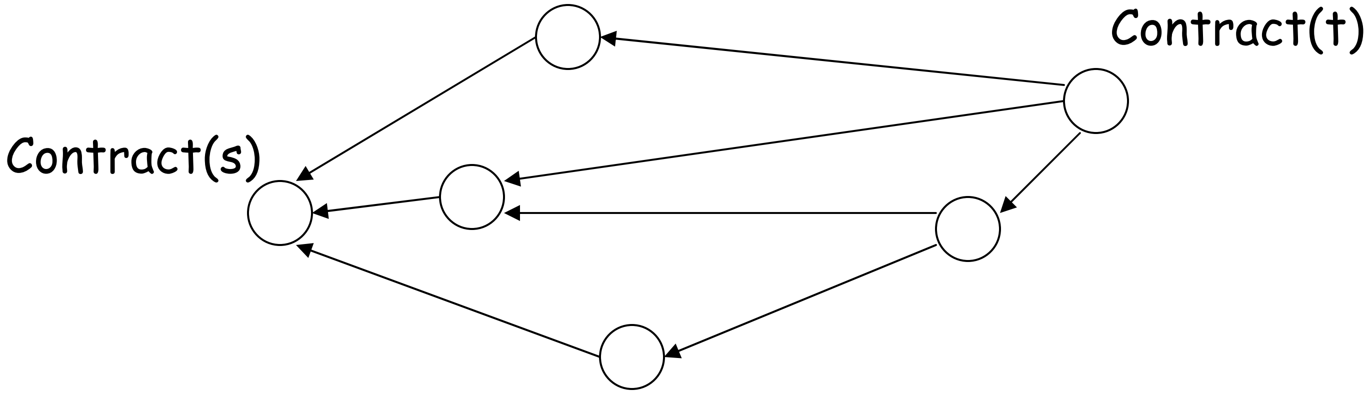
1. Find a max flow and construct the residual graph
2. Contract strongly connected components
3. Compute # "forward-cuts" in the DAG



"Forward-cuts" in planar DAGs

Observe: Planar input graph -> planar DAG

Goal: count "forward-cuts" (or maximal antichains)



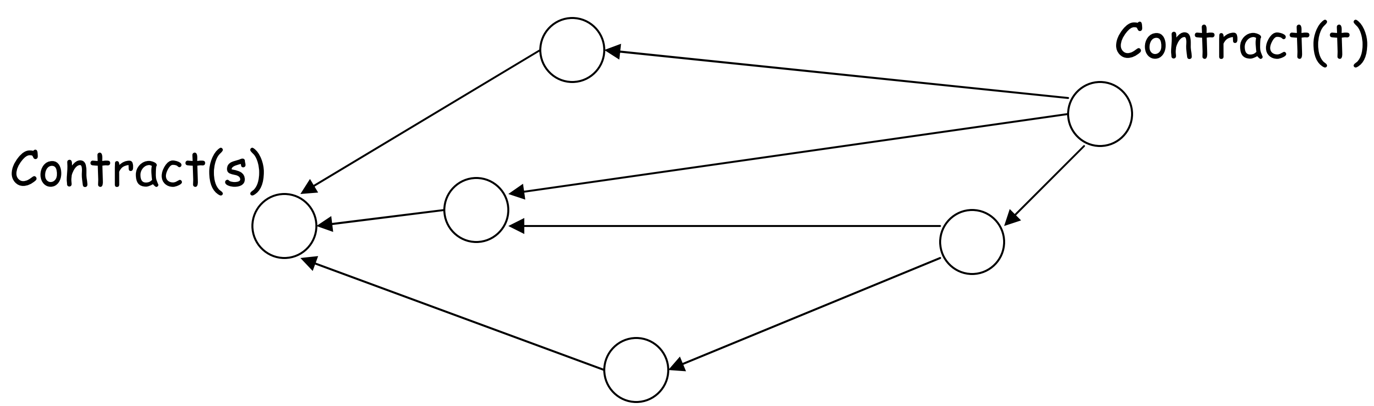
"Forward-cuts" in planar DAGs

Observe: Planar input graph -> planar DAG

Goal: count "forward-cuts" (or maximal antichains)

Ball & Provan's algorithm for both s,t on the outer face:

- split the outer face into the "top" and the "bottom" face
- count all "top"-"bottom" paths in the dual graph



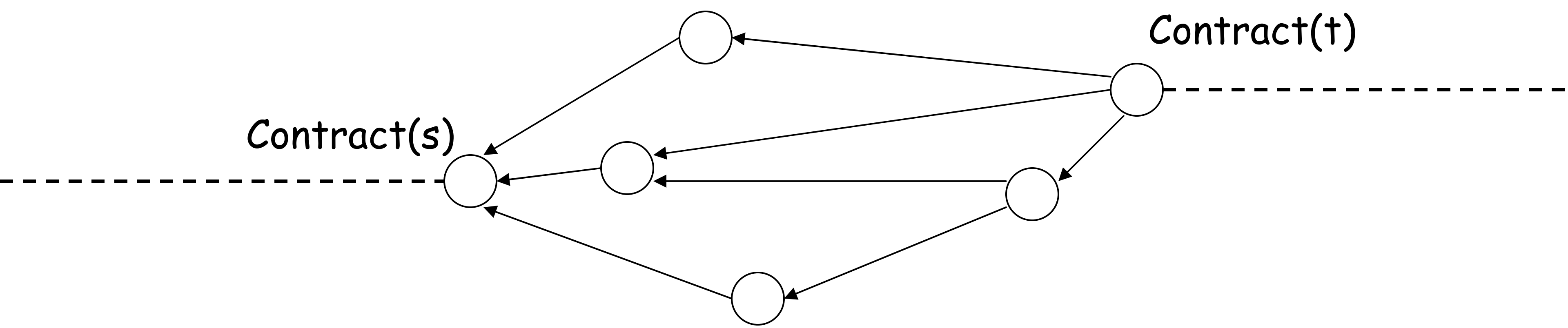
"Forward-cuts" in planar DAGs

Observe: Planar input graph -> planar DAG

Goal: count "forward-cuts" (or maximal antichains)

Ball & Provan's algorithm for both s,t on the outer face:

- split the outer face into the "top" and the "bottom" face
- count all "top"-"bottom" paths in the dual graph



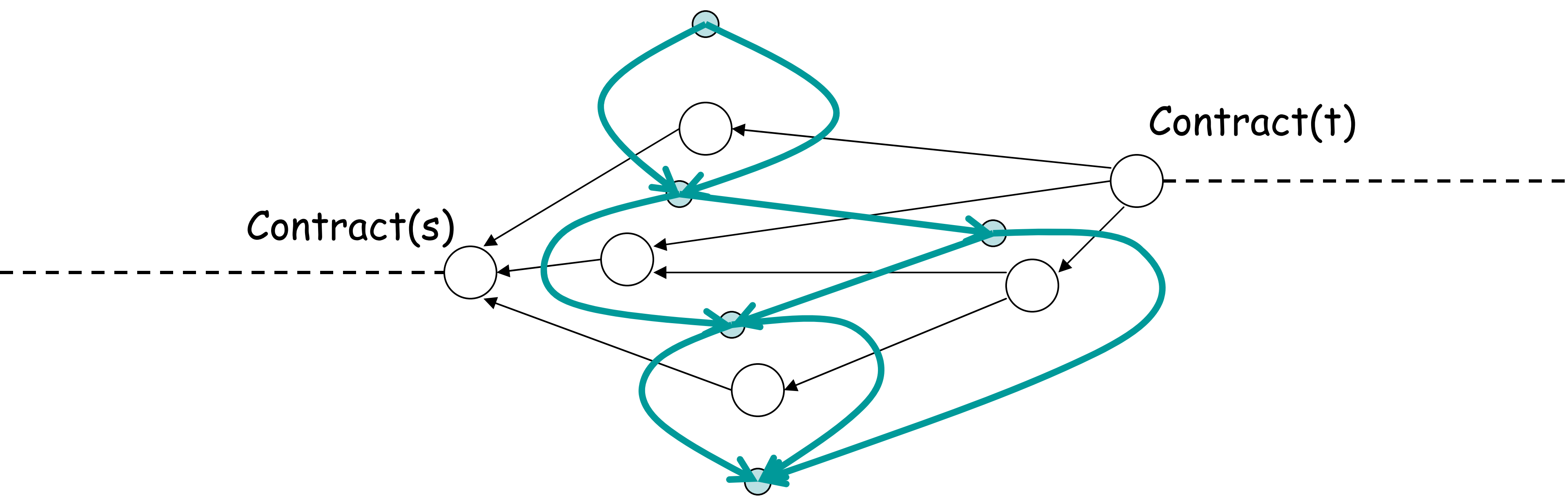
"Forward-cuts" in planar DAGs

Observe: Planar input graph -> planar DAG

Goal: count "forward-cuts" (or maximal antichains)

Ball & Provan's algorithm for both s,t on the outer face:

- split the outer face into the "top" and the "bottom" face
- count all "top"-"bottom" paths in the dual graph



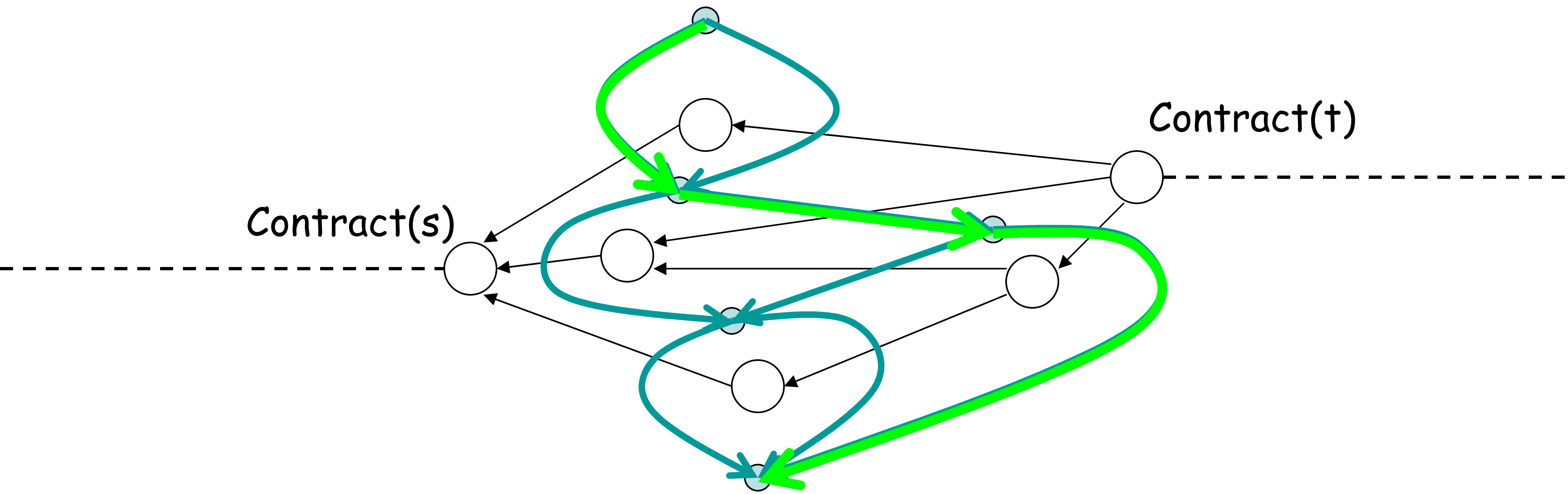
"Forward-cuts" in planar DAGs

Observe: Planar input graph -> planar DAG

Goal: count "forward-cuts" (or maximal antichains)

Ball & Provan's algorithm for both s,t on the outer face:

- split the outer face into the "top" and the "bottom" face
- count all "top"-"bottom" paths in the dual graph



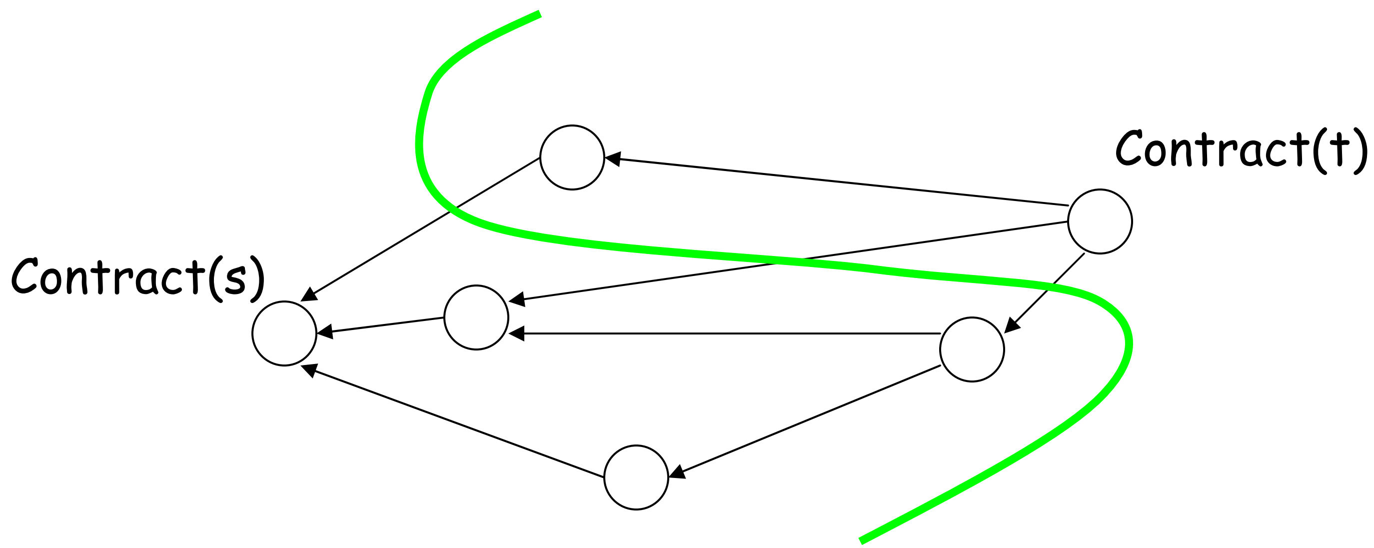
“Forward-cuts” in planar DAGs

Observe: Planar input graph -> planar DAG

Goal: count “forward-cuts” (or maximal antichains)

Ball & Provan’s algorithm for both s,t on the outer face:

- split the outer face into the “top” and the “bottom” face
- count all “top”-“bottom” paths in the dual graph



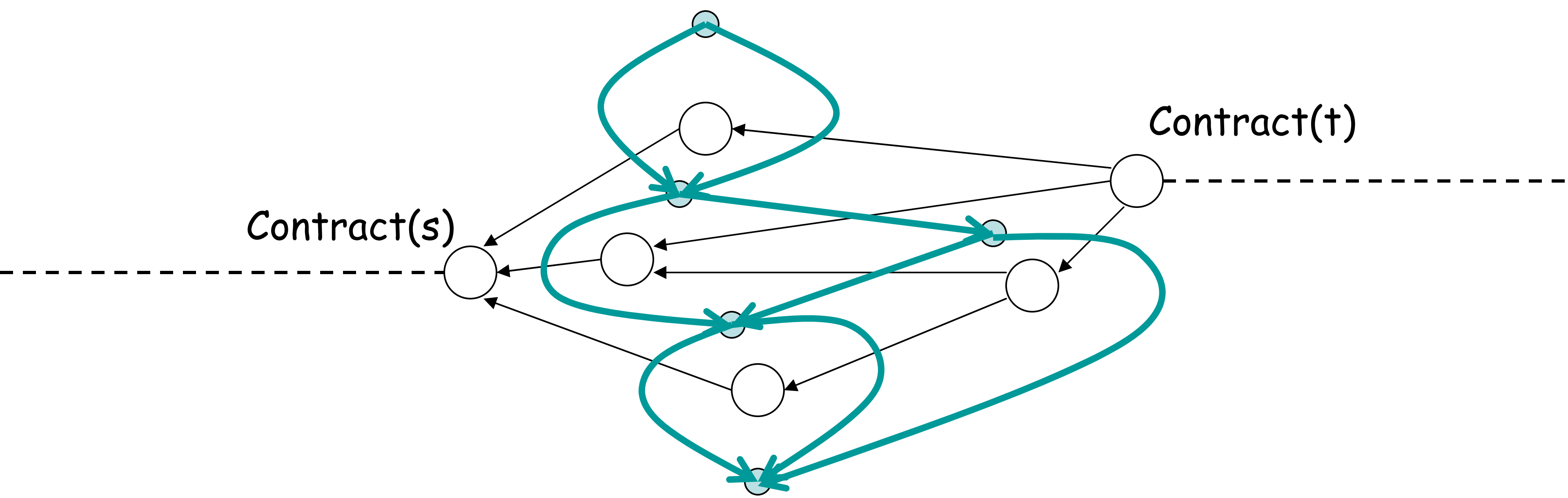
“Forward-cuts” in planar DAGs

Observe: Planar input graph -> planar DAG

Goal: count “forward-cuts” (or maximal antichains)

Ball & Provan’s algorithm for both s,t on the outer face:

- split the outer face into the “top” and the “bottom” face
- count all “top”-“bottom” paths in the dual graph



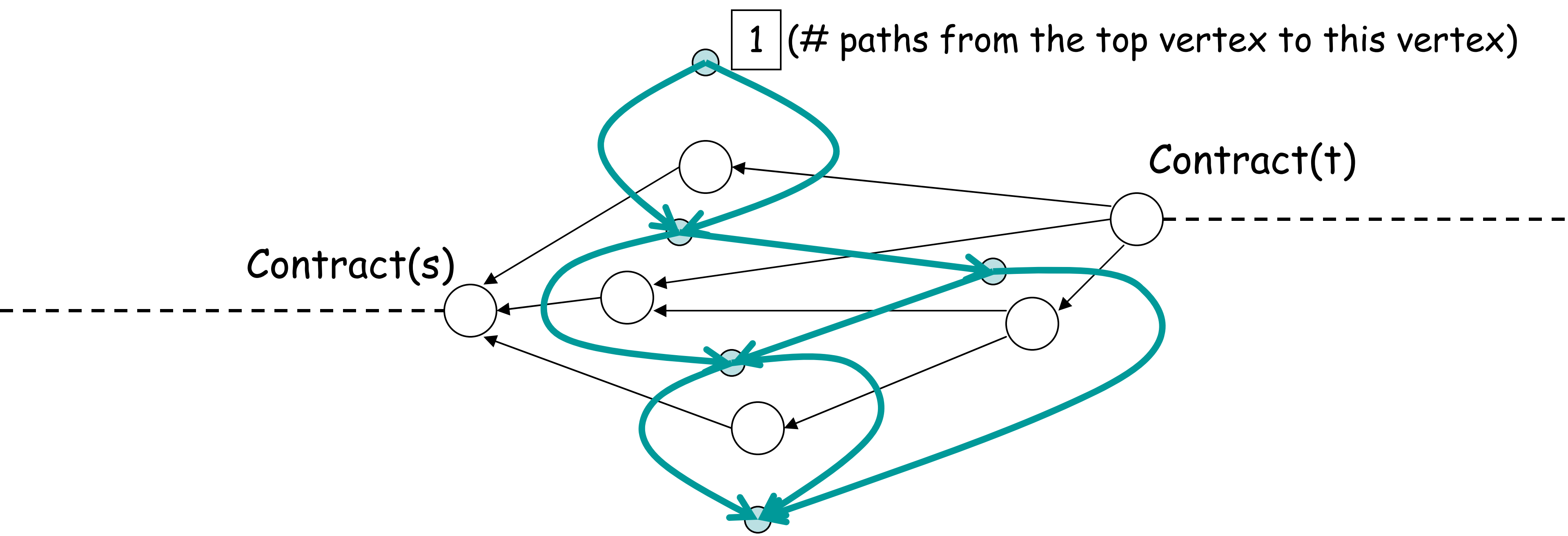
"Forward-cuts" in planar DAGs

Observe: Planar input graph -> planar DAG

Goal: count "forward-cuts" (or maximal antichains)

Ball & Provan's algorithm for both s,t on the outer face:

- split the outer face into the "top" and the "bottom" face
- count all "top"-"bottom" paths in the dual graph



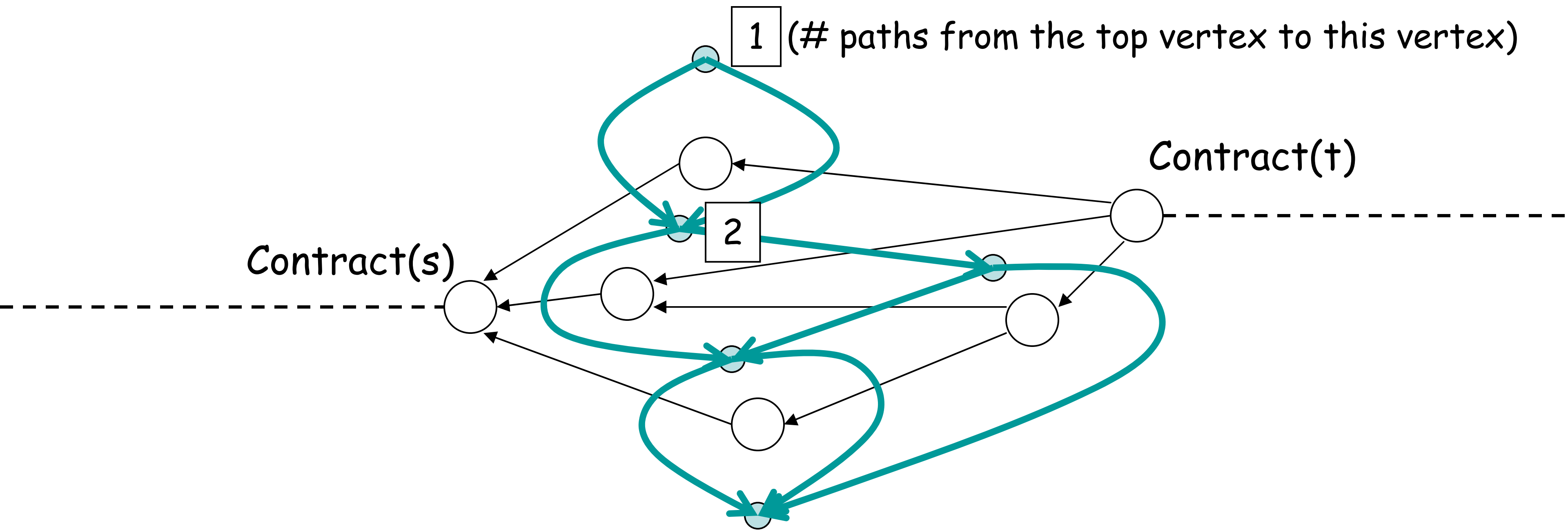
“Forward-cuts” in planar DAGs

Observe: Planar input graph -> planar DAG

Goal: count “forward-cuts” (or maximal antichains)

Ball & Provan’s algorithm for both s,t on the outer face:

- split the outer face into the “top” and the “bottom” face
- count all “top”-“bottom” paths in the dual graph



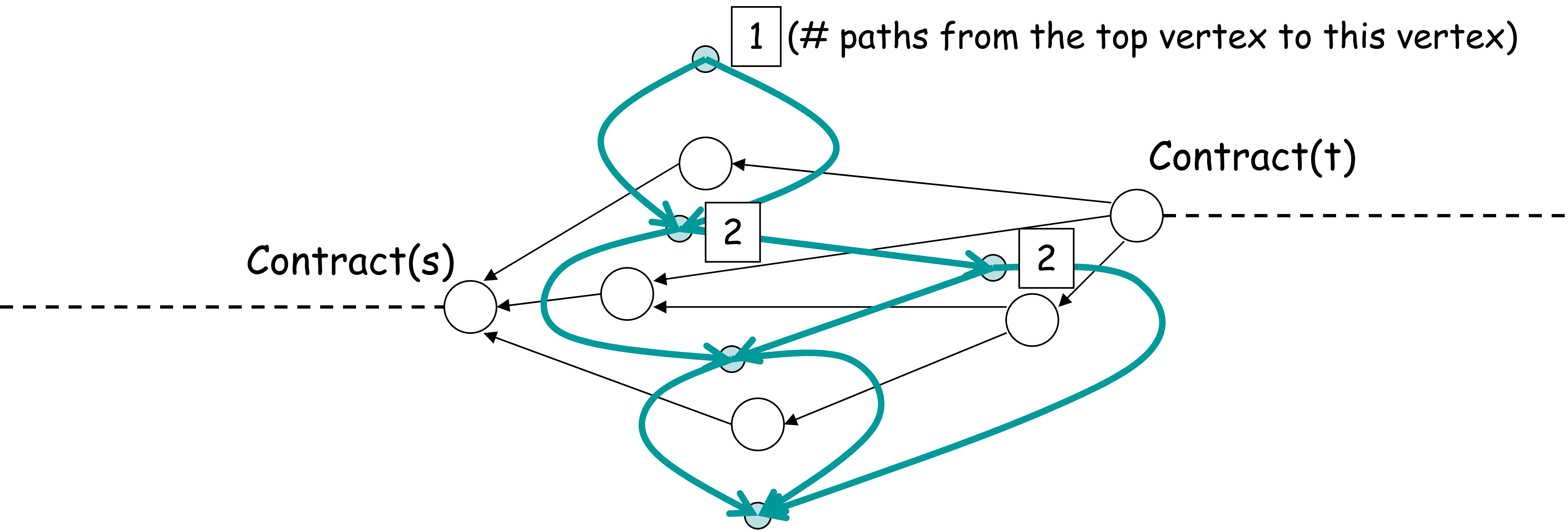
"Forward-cuts" in planar DAGs

Observe: Planar input graph -> planar DAG

Goal: count "forward-cuts" (or maximal antichains)

Ball & Provan's algorithm for both s,t on the outer face:

- split the outer face into the "top" and the "bottom" face
- count all "top"-"bottom" paths in the dual graph



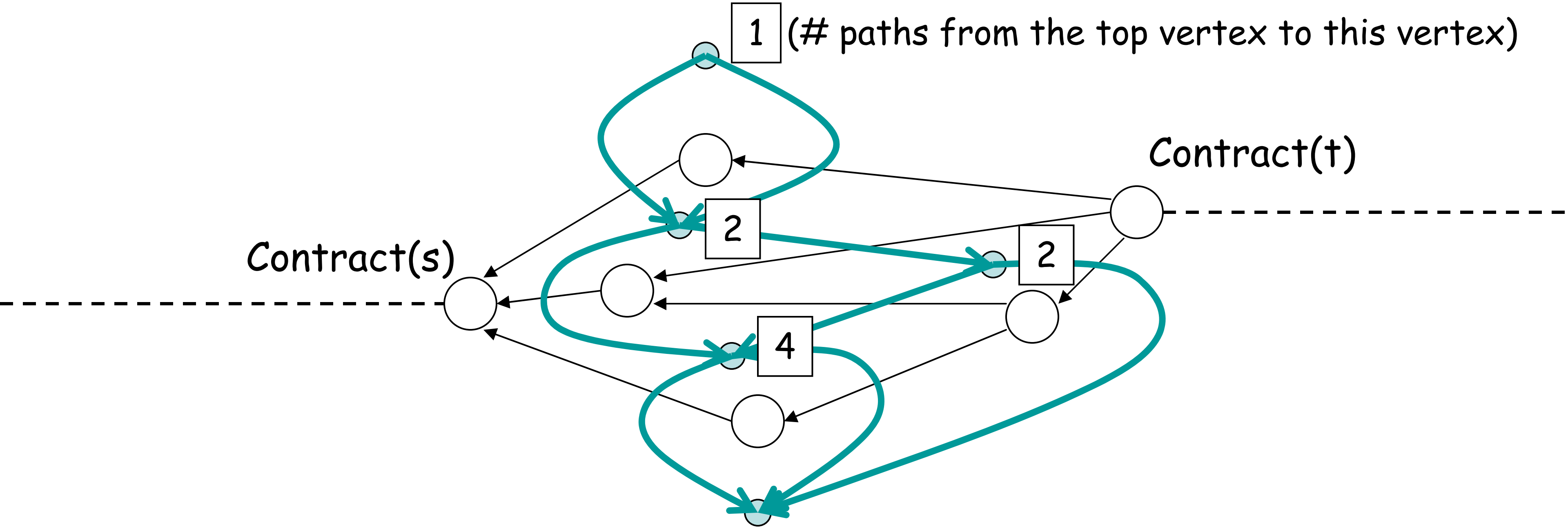
"Forward-cuts" in planar DAGs

Observe: Planar input graph -> planar DAG

Goal: count "forward-cuts" (or maximal antichains)

Ball & Provan's algorithm for both s,t on the outer face:

- split the outer face into the "top" and the "bottom" face
- count all "top"-"bottom" paths in the dual graph



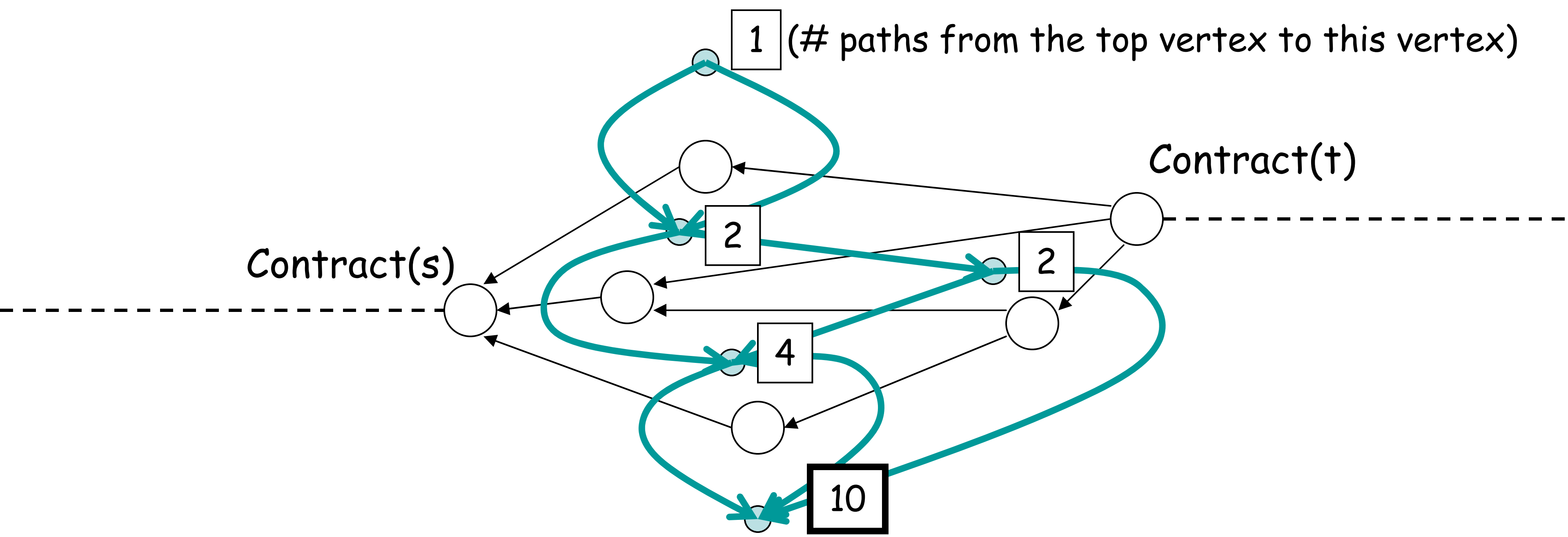
"Forward-cuts" in planar DAGs

Observe: Planar input graph -> planar DAG

Goal: count "forward-cuts" (or maximal antichains)

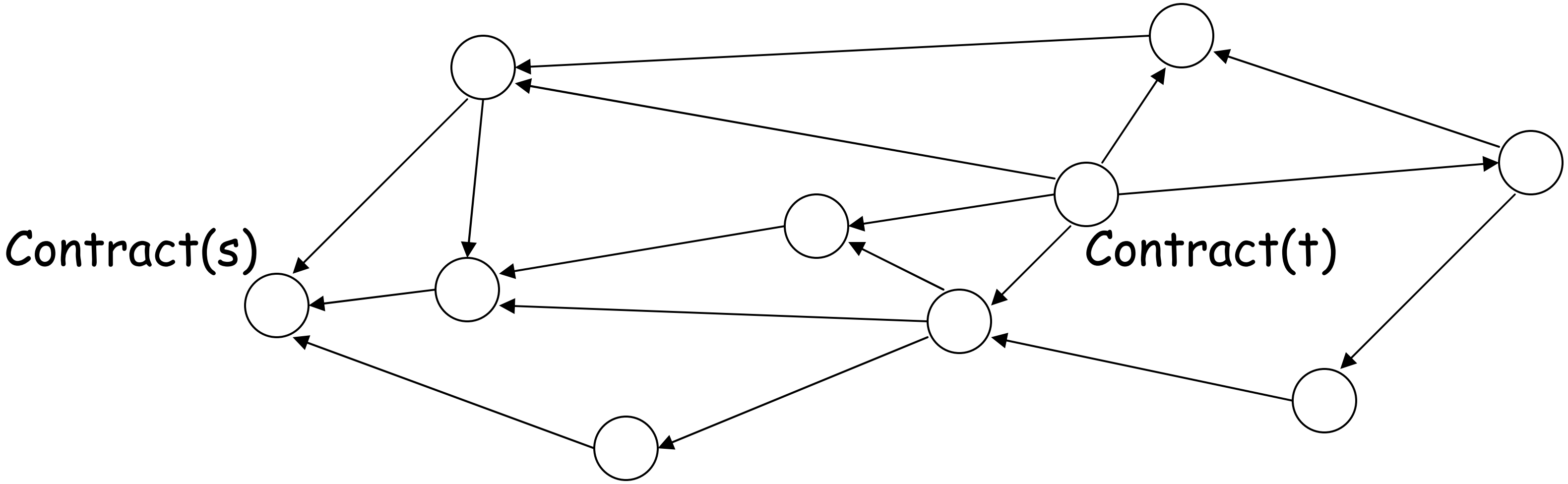
Ball & Provan's algorithm for both s,t on the outer face:

- split the outer face into the "top" and the "bottom" face
- count all "top"-"bottom" paths in the dual graph



“Forward-cuts” and different faces

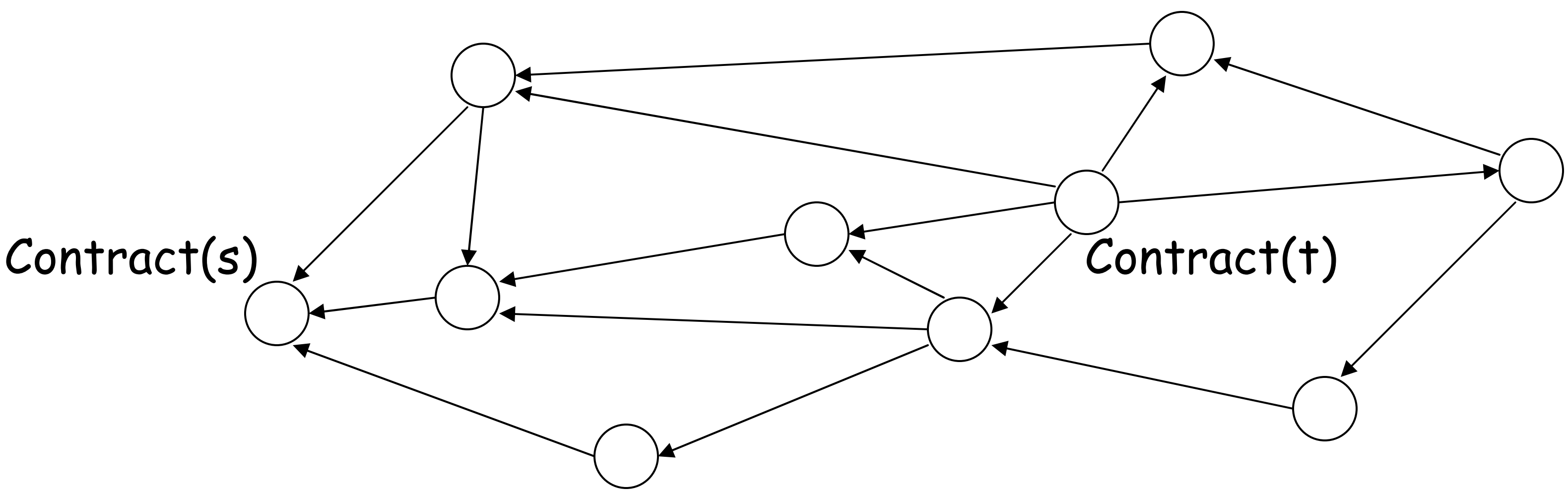
What if s,t are on different faces ?



“Forward-cuts” and different faces

What if s,t are on different faces ?

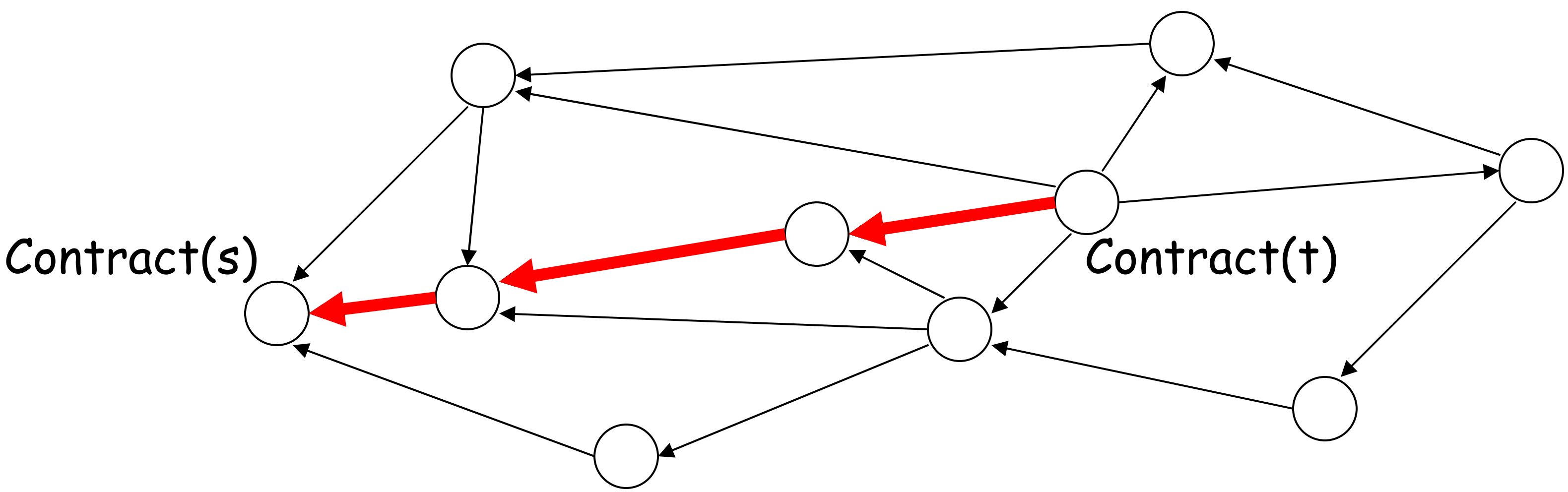
- find a contract(t)-contract(s) path
- construct the dual, except no edges cross the path
- sum # paths between faces sharing an edge on the path



"Forward-cuts" and different faces

What if s,t are on different faces ?

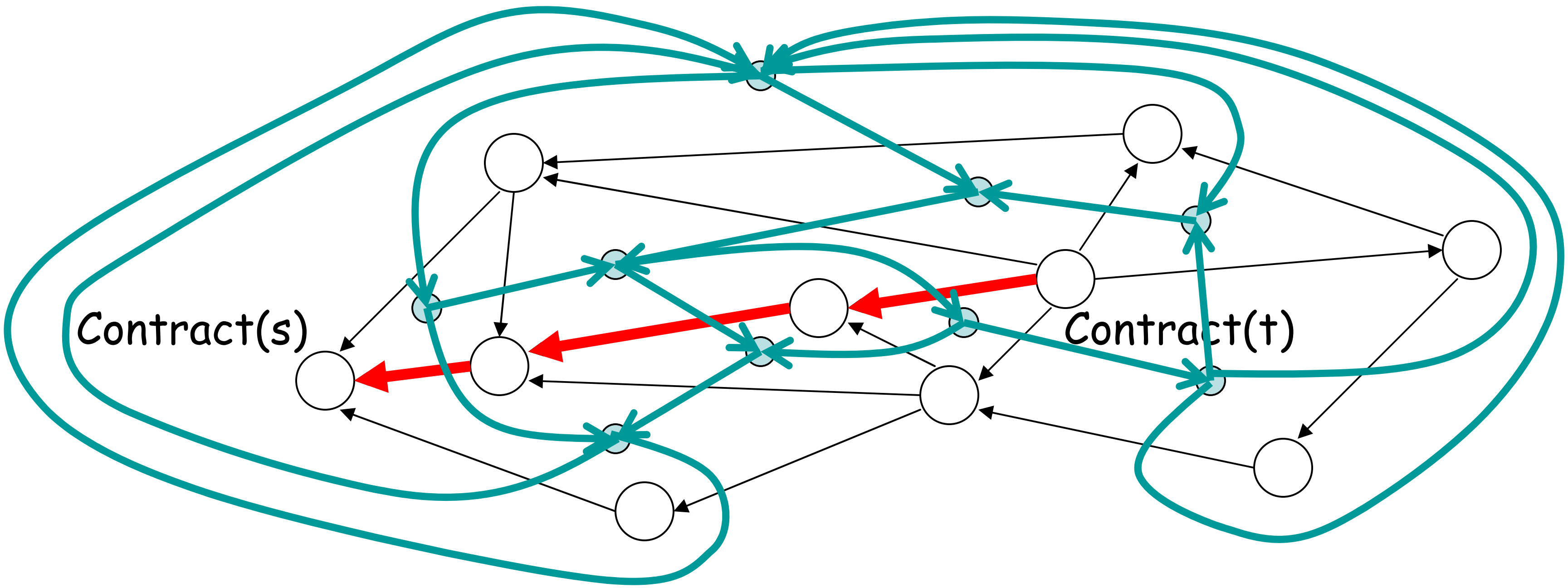
- find a contract(t)-contract(s) path
- construct the dual, except no edges cross the path
- sum # paths between faces sharing an edge on the path



“Forward-cuts” and different faces

What if s,t are on different faces ?

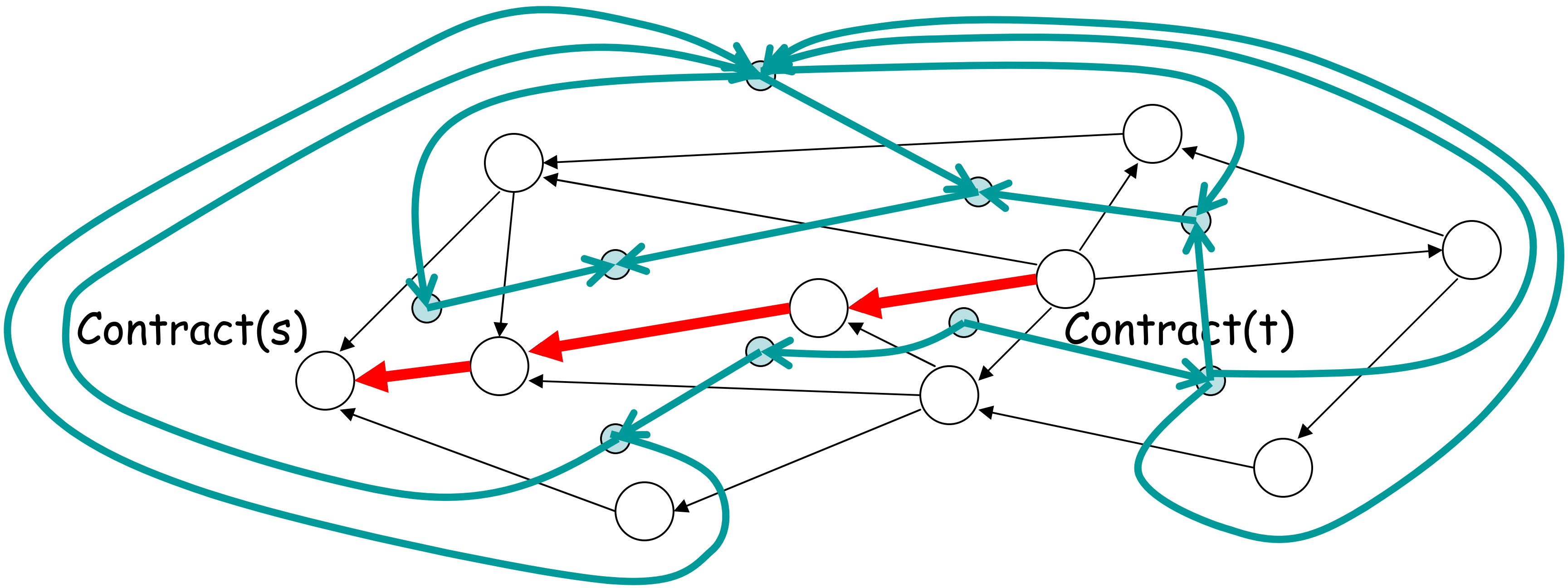
- find a contract(t)-contract(s) path
- construct the dual, except no edges cross the path
- sum # paths between faces sharing an edge on the path



“Forward-cuts” and different faces

What if s,t are on different faces ?

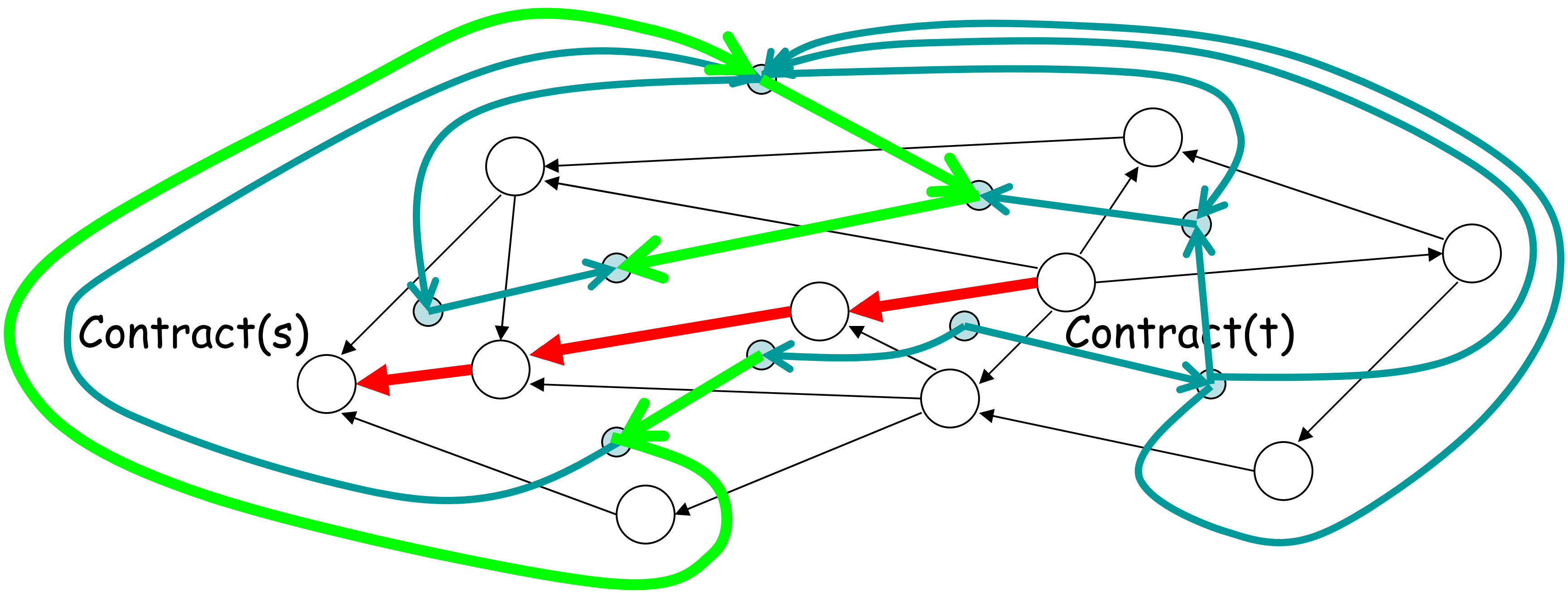
- find a contract(t)-contract(s) path
- construct the dual, except no edges cross the path -> DAG
- sum # paths between faces sharing an edge on the path



"Forward-cuts" and different faces

What if s,t are on different faces ?

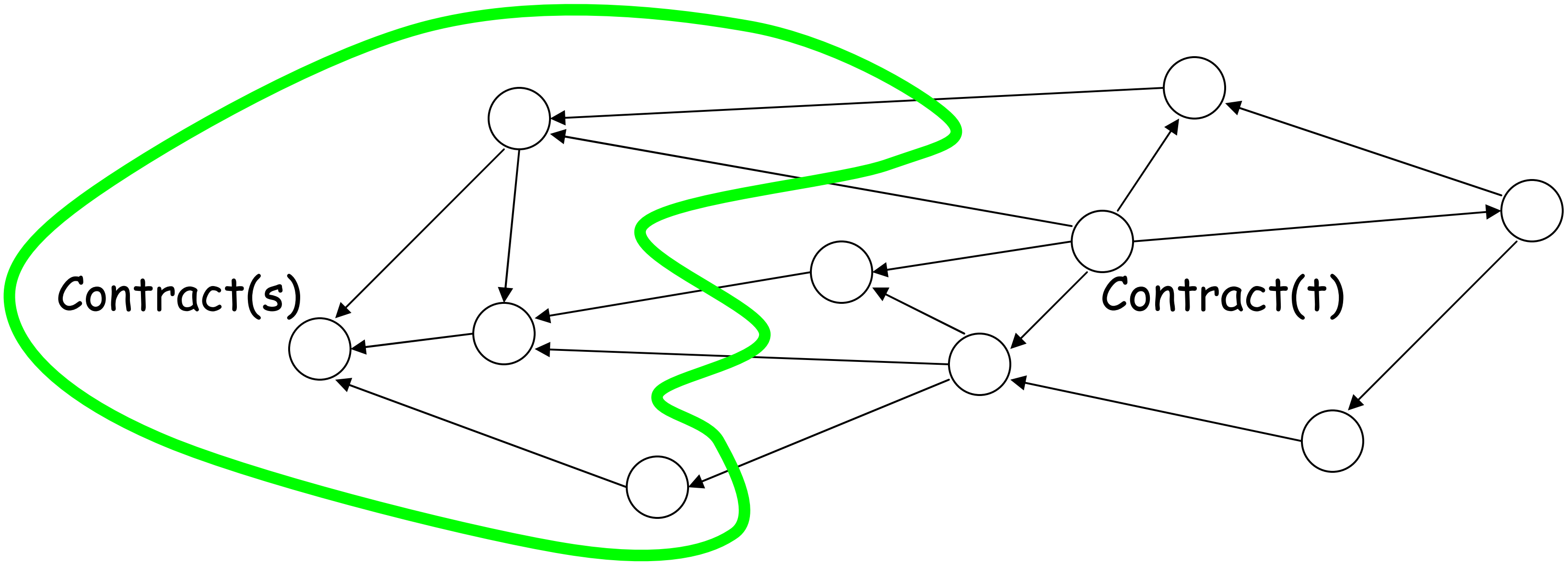
- find a contract(t)-contract(s) path
- construct the dual, except no edges cross the path -> DAG
- sum # paths between faces sharing an edge on the path



“Forward-cuts” and different faces

What if s,t are on different faces ?

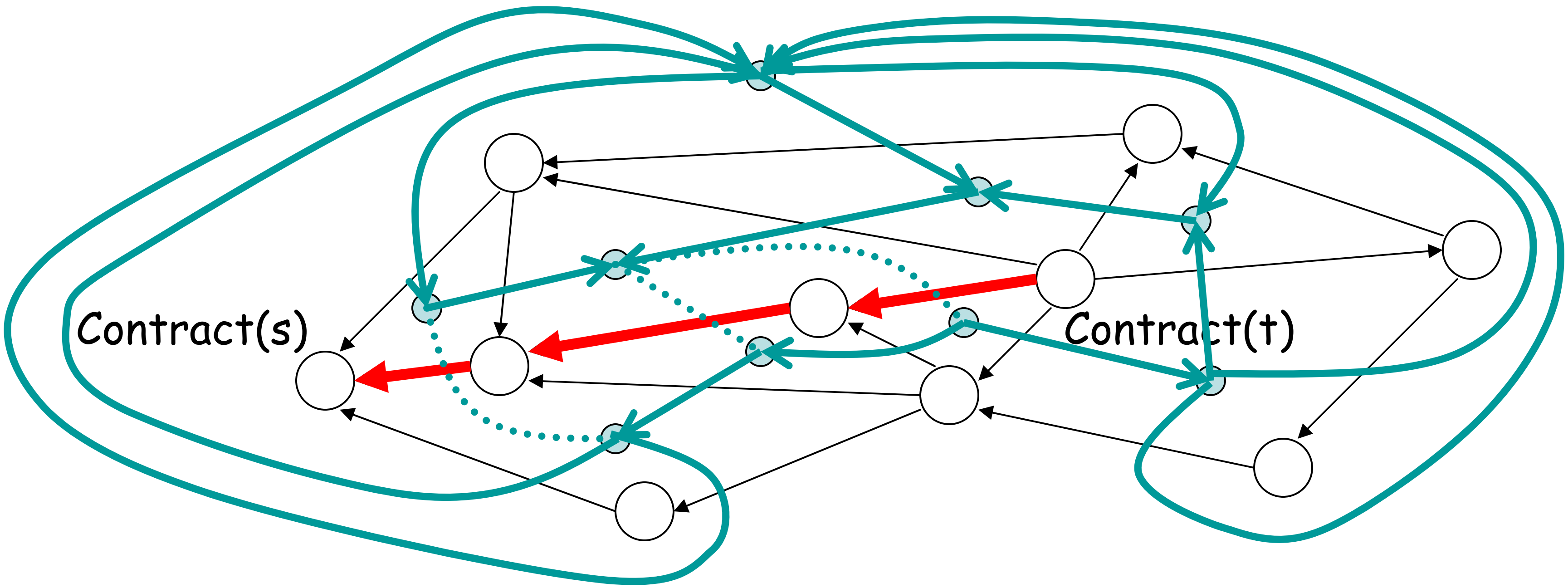
- find a contract(t)-contract(s) path
- construct the dual, except no edges cross the path -> DAG
- sum # paths between faces sharing an edge on the path



“Forward-cuts” and different faces

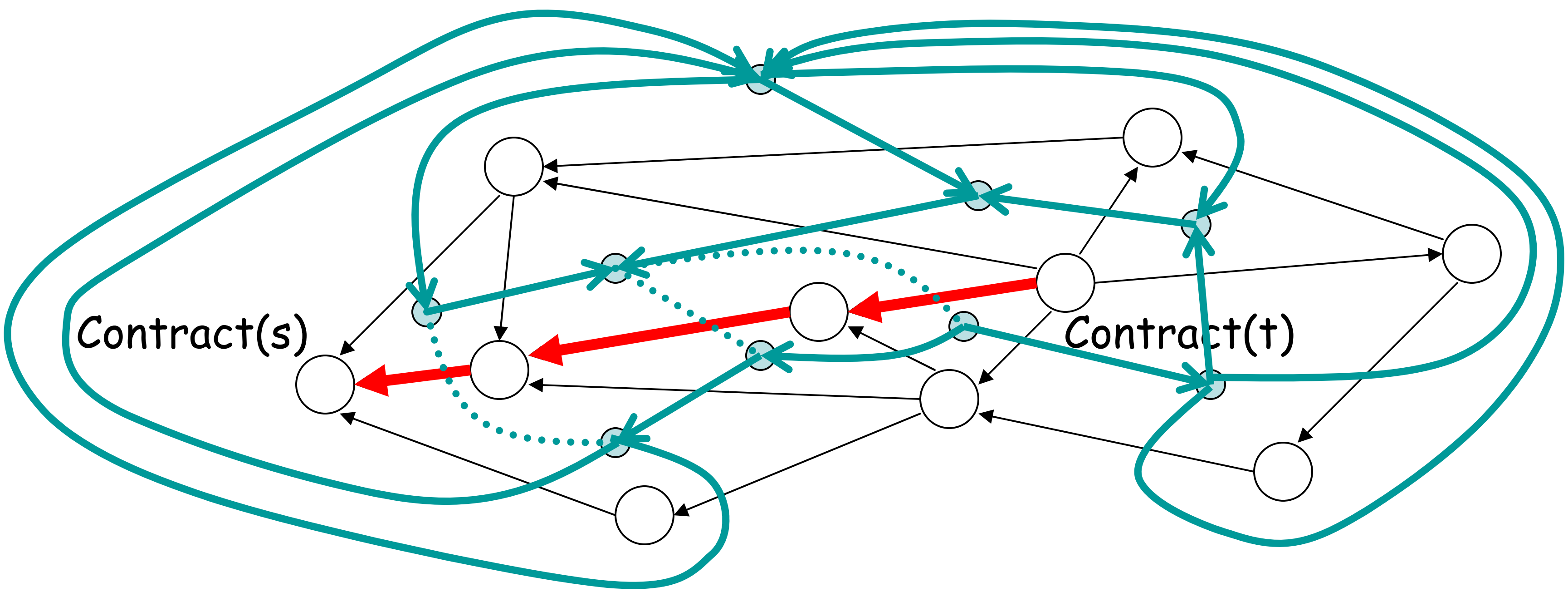
What if s,t are on different faces ?

- find a contract(t)-contract(s) path
- construct the dual, except no edges cross the path -> DAG
- sum # paths between faces sharing an edge on the path



Sampling minimum (s,t)-cuts

- Choose a red edge proportionally to the corresponding path count.
- Starting at the corresponding **end** vertex, choose a predecessor vertex proportionally to the stored value
- Continue until get to the start vertex



Running time

Reduction to forward cuts:

- $O(n \log n)$ to find a (acyclic) max-flow in planar graphs [Borradaile-Klein '09]
- $O(n)$ to find and contract the strongly connected components

Counting forward cuts:

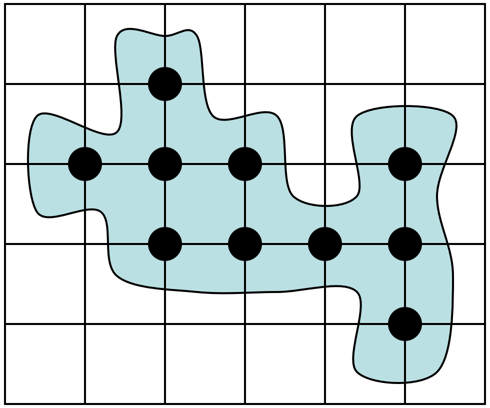
- $O(n)$ find the path, construct the dual graph
- $O(n)$ compute #paths between two end-points in the dual
- $O(dn)$ overall computation of paths, at most d end-point pairs where d = length of the s-t path

TOTAL: $O(dn + n \log n)$

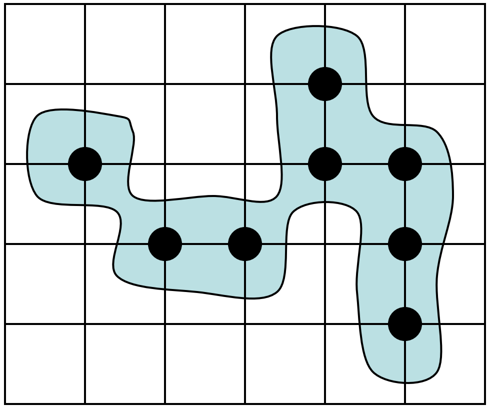
Contiguous cuts

Contiguous set of vertices: can be separated from the other vertices by a simply connected planar region that intersects every edge at most once

Examples:



contiguous

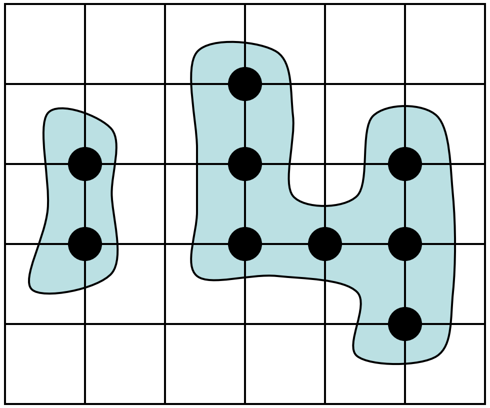


contiguous

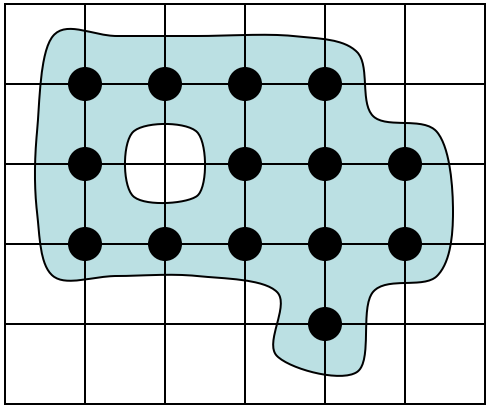
Contiguous cuts

Contiguous set of vertices: can be separated from the other vertices by a simply connected planar region that intersects every edge at most once

Examples:



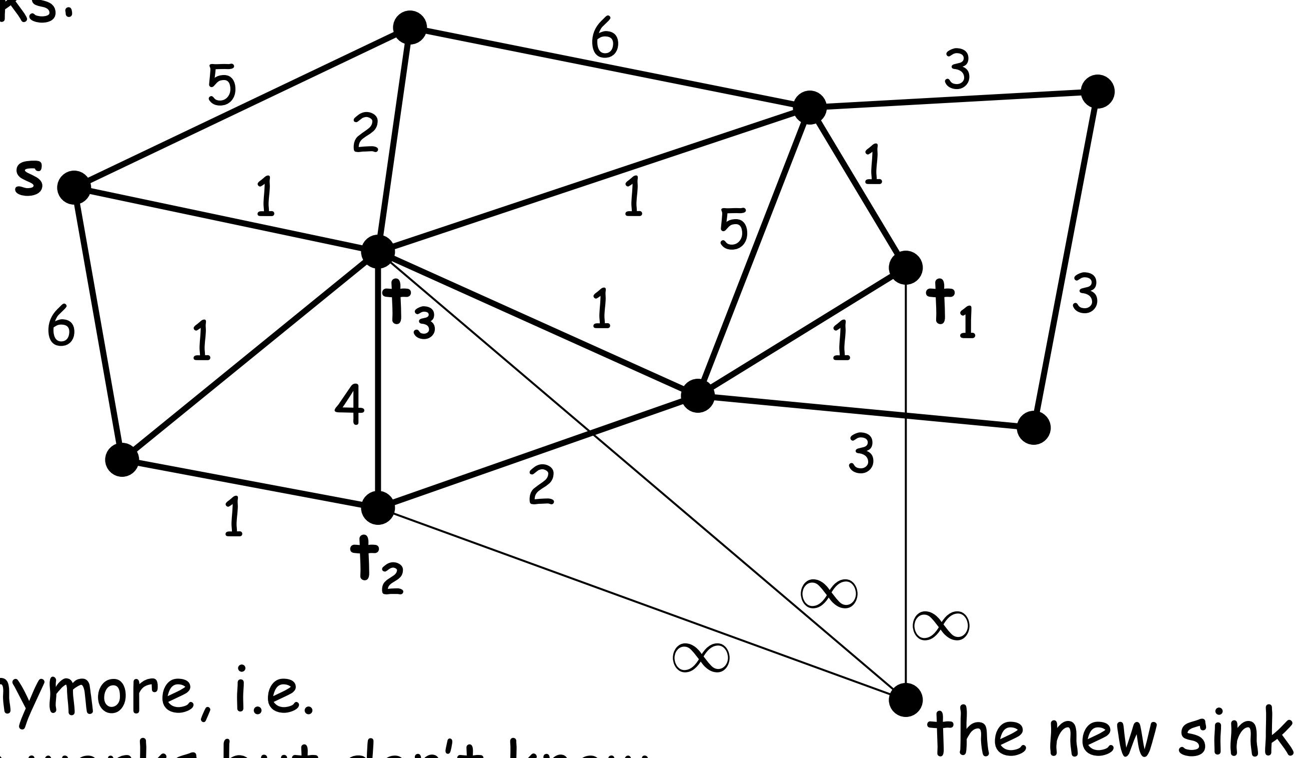
not contiguous



not contiguous

Back to the Ball & Provan's reduction

- Every minimum cut separating s and t is contiguous
- For multiple sinks:

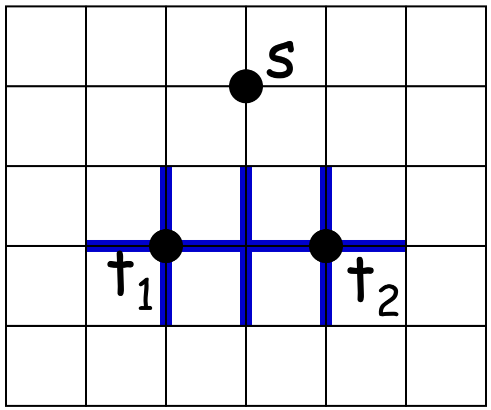


  - not planar anymore, i.e.
    BP reduction works but don't know
    how to count forward cuts
  - does the reduction "preserve" contiguous cuts ?

Back to the Ball & Provan's reduction

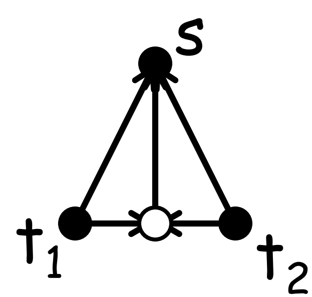
Does the reduction preserve contiguous cuts ?

NO...



All edges weight ∞, except for blue edges: weight 1.

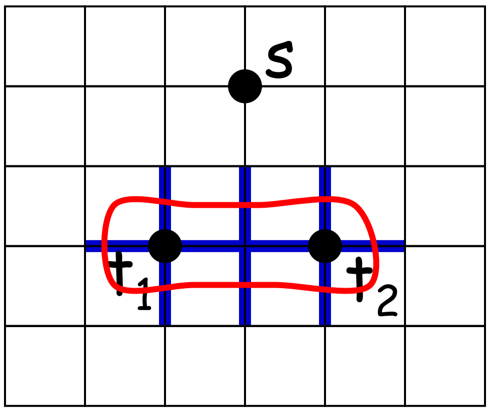
Blue edges used up to their capacity, other edges not ->
Only blue edges survive in the contracted residual graph:

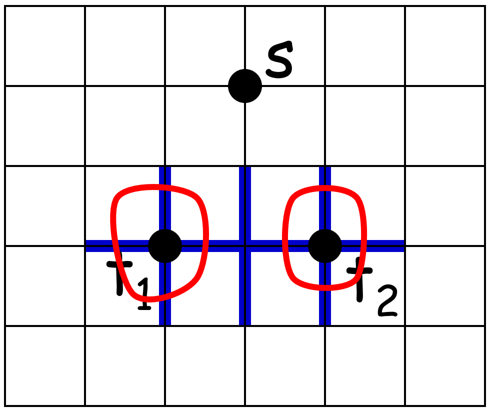


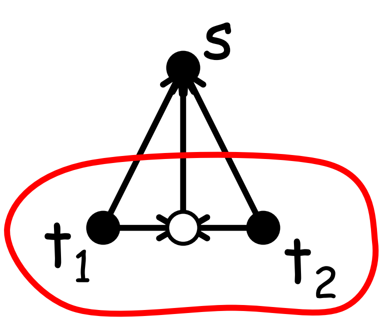
Back to the Ball & Provan's reduction

Does the reduction preserve contiguous cuts ?

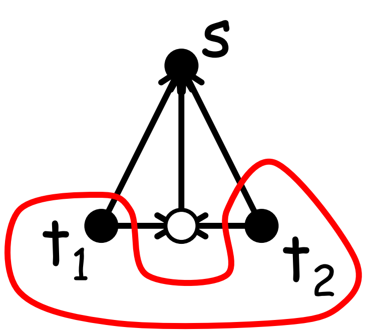
NO…







both contiguous



only bottom contiguous

The new reduction and counting

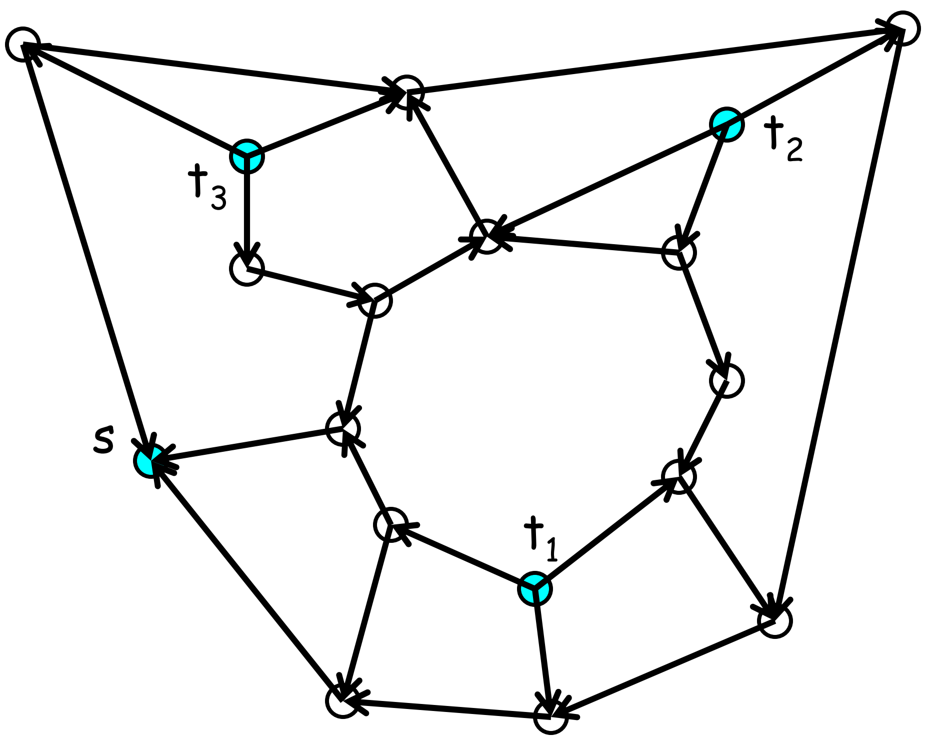
- A new reduction: creates a different DAG where contiguous forward (T',s')-cuts are in bijection with contiguous (s,T)-cuts in G

- Computing the number of contiguous forward (T',s')-cuts: dynamic programming across a tree-structure connecting T' with s' in the dual graph

Tours in the dual graph

Cut is not a simple cycle !

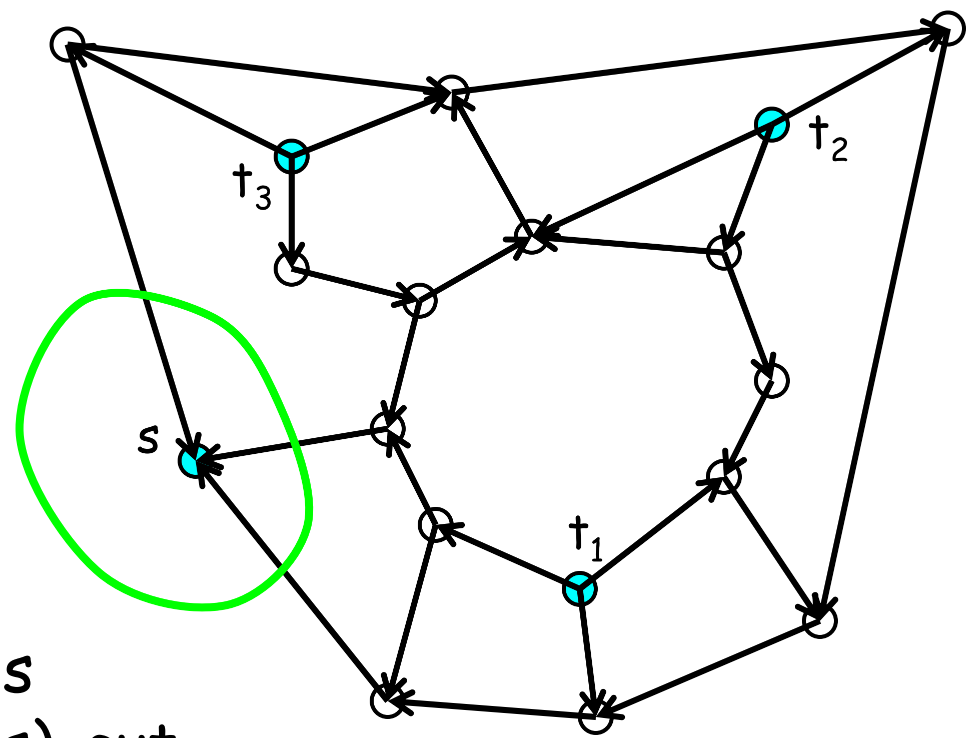
-> a tour in the dual graph.



Tours in the dual graph

Cut is not a simple cycle !

-> a tour in the dual graph.

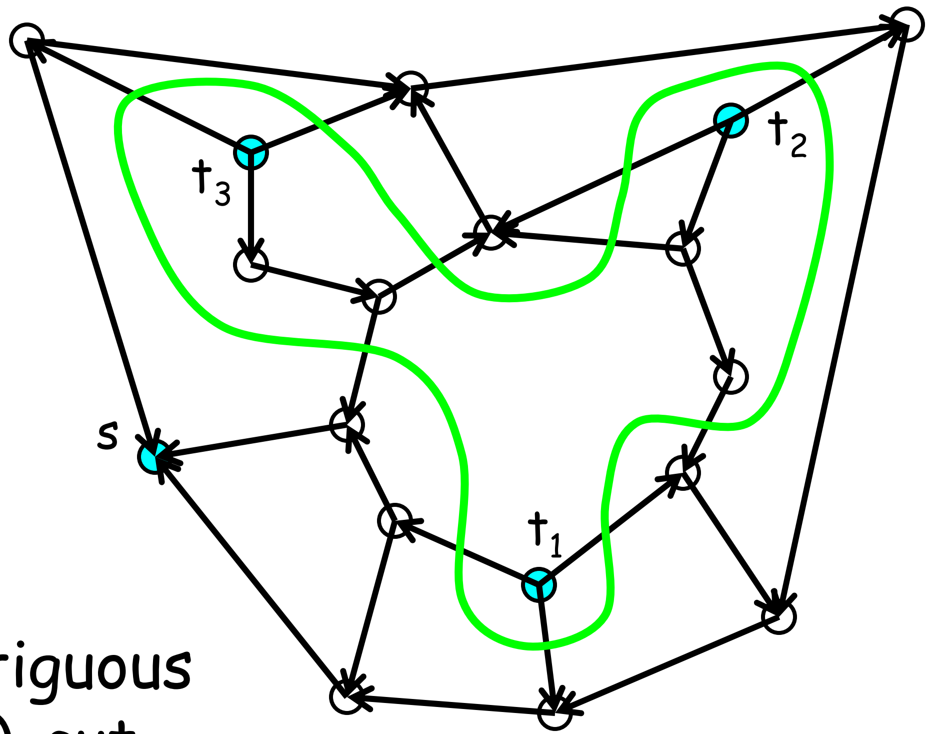


A contiguous forward (T,s)-cut

Tours in the dual graph

Cut is not a simple cycle !

-> a tour in the dual graph.

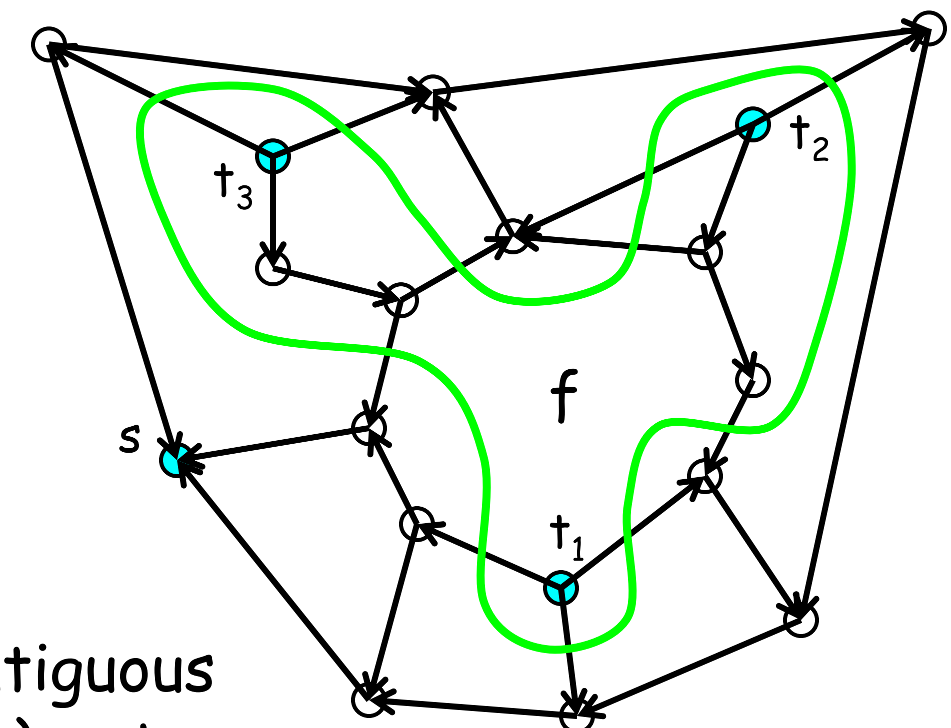


Another contiguous forward (T,s)-cut

Tours in the dual graph

A non-crossing tour:

For every face visited by the tour, its edges must be "cut" in the clockwise order when traversing the tour.

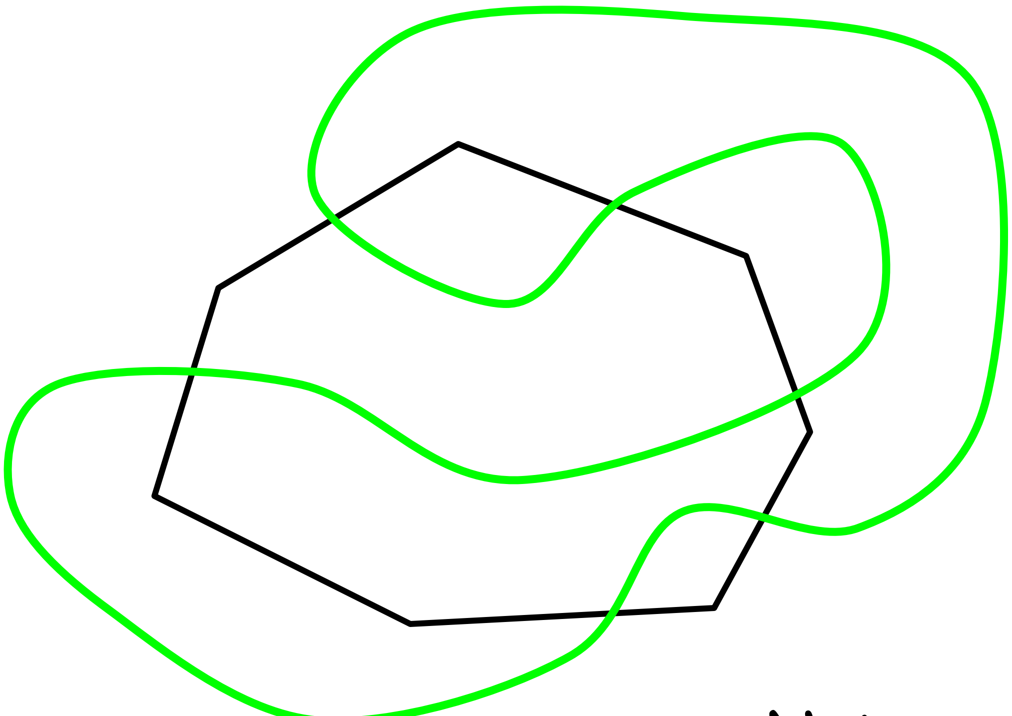


Another contiguous forward (T,s)-cut

Tours in the dual graph

A non-crossing tour:

For every face visited by the tour, its edges must be "cut" in the clockwise order when traversing the tour.

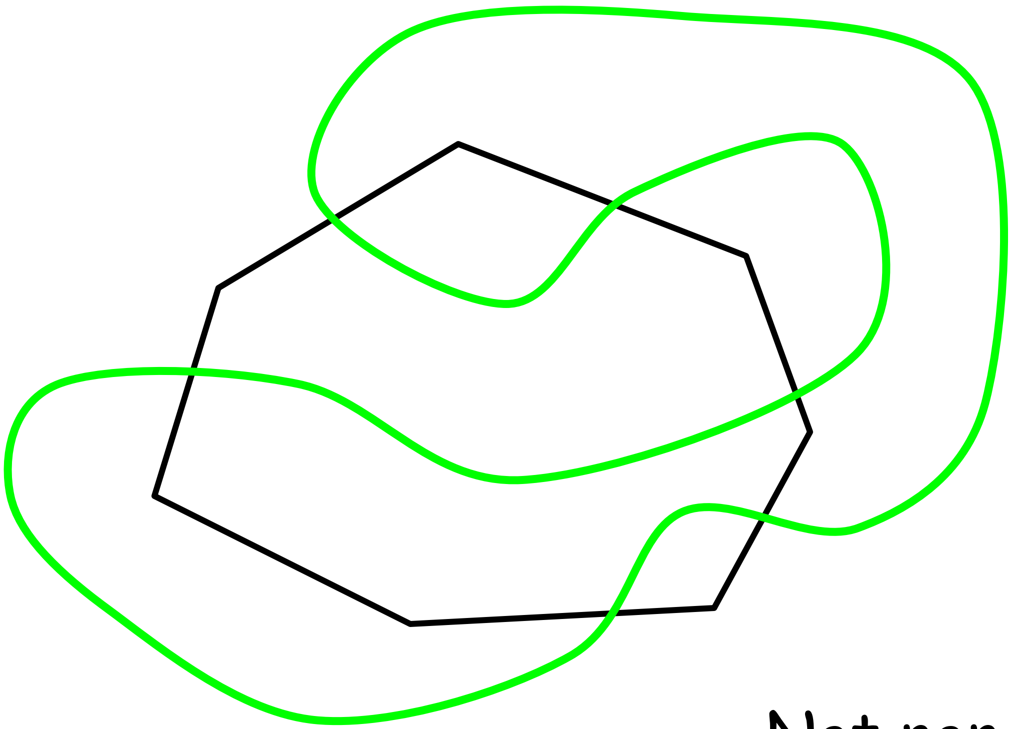


Not non-crossing !

Tours in the dual graph

Lemma:

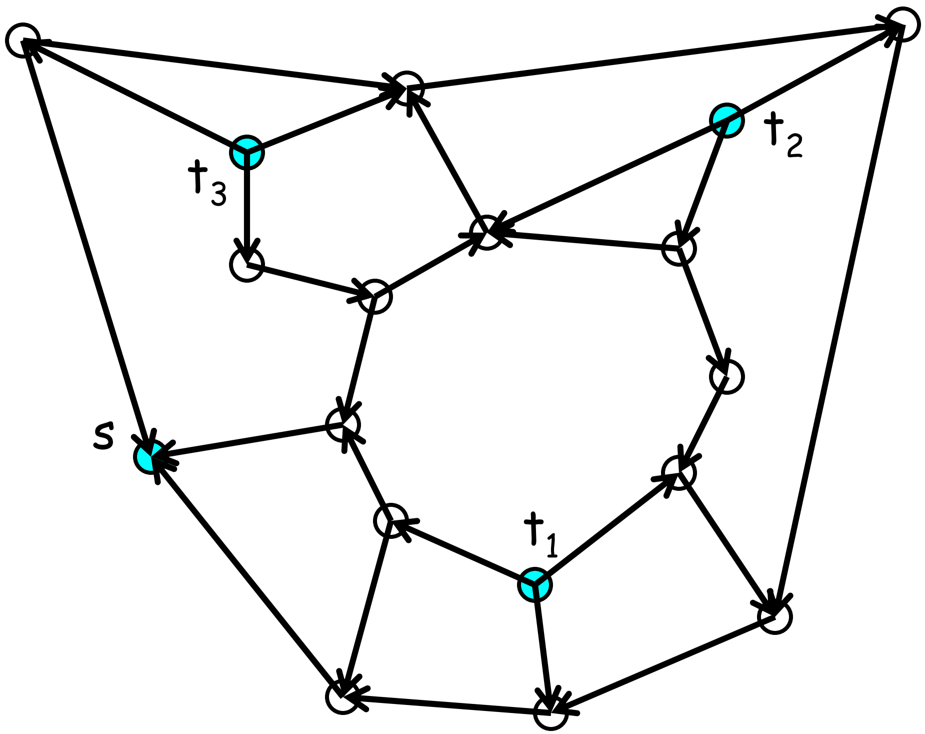
Contiguous forward (T,s)-cuts are in 1-1 correspondence with non-crossing tours in the dual of the graph obtained from the new reduction.



Not non-crossing !

Counting via dynamic programming

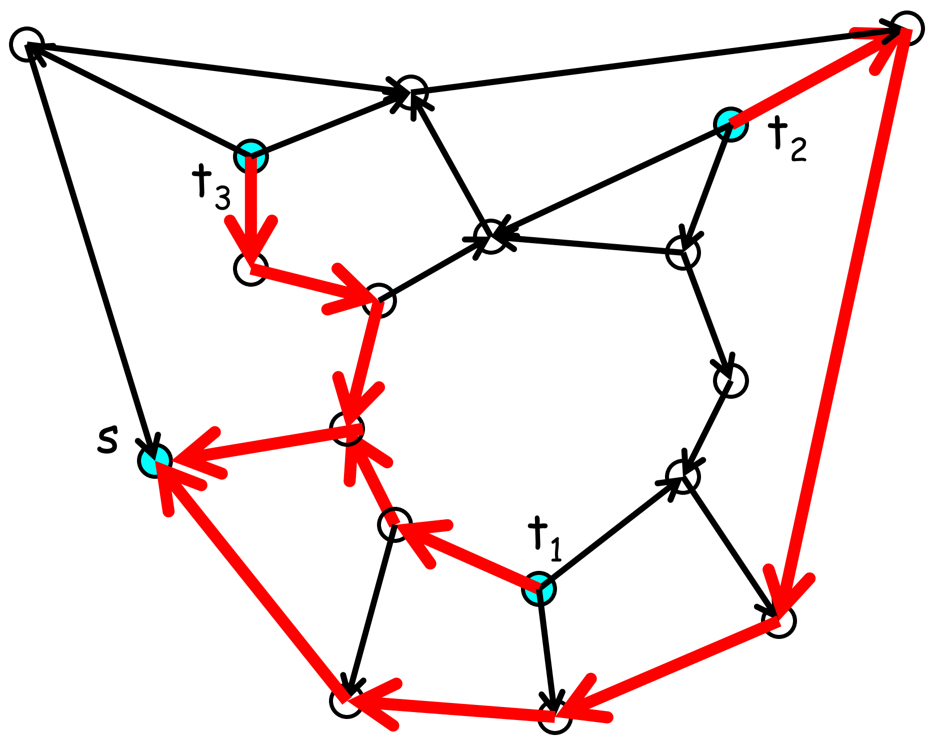
Goal: count non-crossing tours



Counting via dynamic programming

Goal: count non-crossing tours

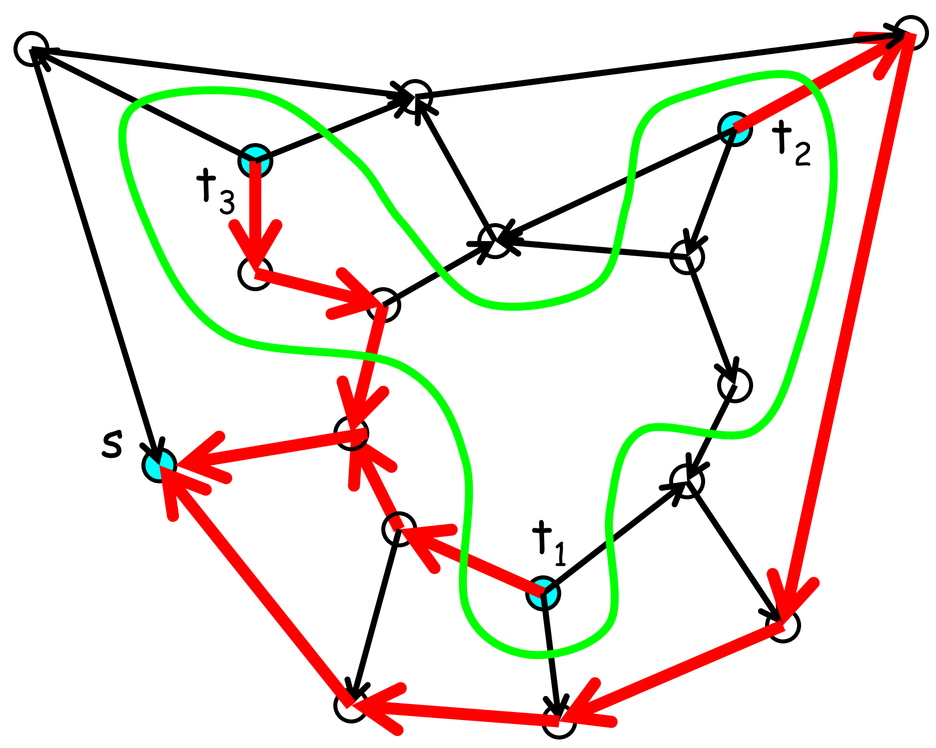
Idea: build a tree from T to s:



Counting via dynamic programming

Goal: count non-crossing tours

Idea: build a tree from T to s:

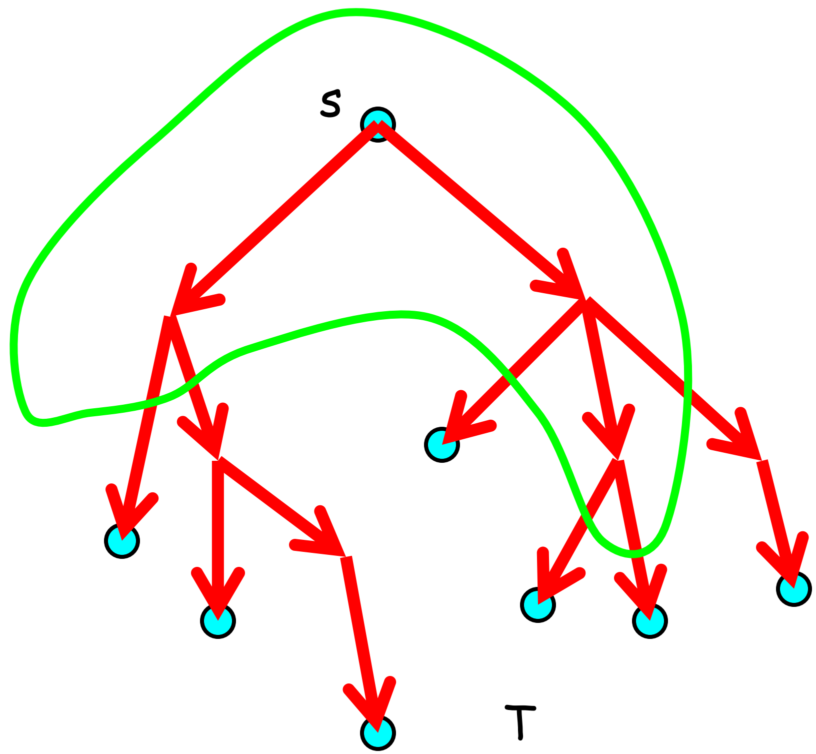


Counting via dynamic programming

Goal: count non-crossing tours

Idea: build a tree from T to s:

Then, compute #paths
between a "left" and
a "right" edge in a "wedge".



Generalize to larger wedge distance (of the left and the right edge) via dynamic programming.

Open Problems

Many open problems:

- multi-source multi-sink min cuts: counting (contig or not)
- multi-source multi-sink contig. min cuts: find
- other notions of contiguity

- graphs arising in computer vision (e.g., high-dimensional grids)
- non-planar graphs ? (unweighted or weighted)
- sampling all cuts proportionally to their weights