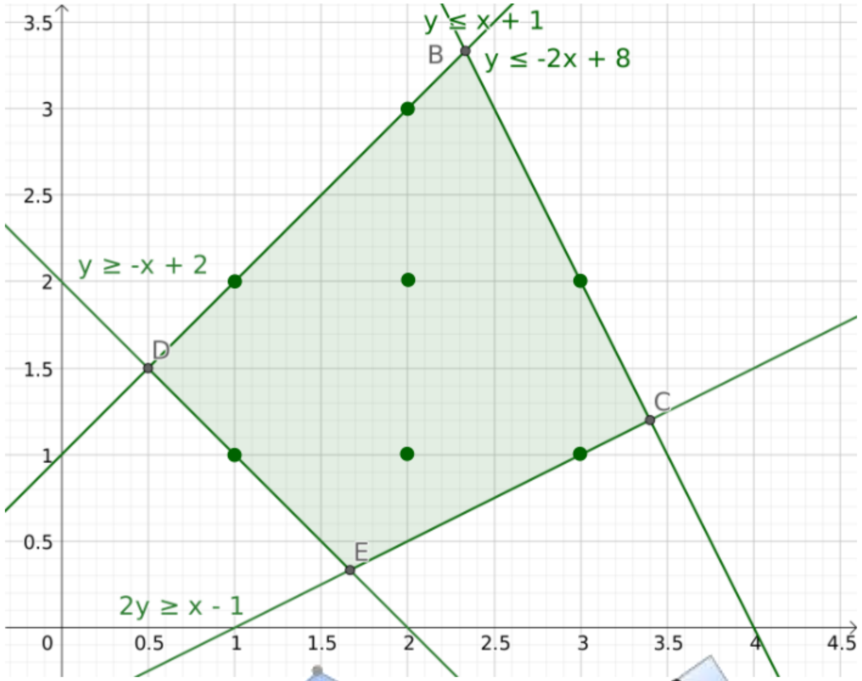
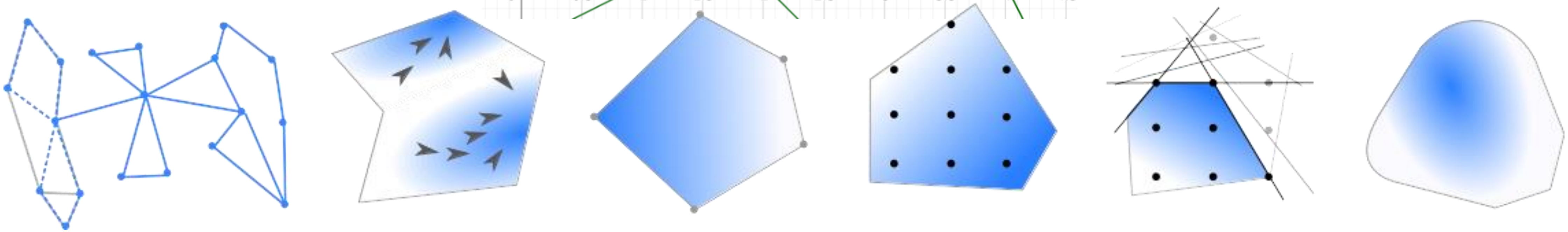
Network Flows

COMP4691/8691





Outline

- Network Flows
  - Max Flow in combinatoric optimization
  - Max Flow as LPs
  - Min Cut
- Planning as Network Flows
- Next Week
  - Tue: review lectures – post questions and topics on Ed
  - Fri: Guest Lecture (A/Prof Charles Gretton)

Network Flows

Imagine a network of pipes

- Different widths
- Different lengths
- Different capacities
- Complex connectivity

Image credits: indiamart - https://www.indiamart.com/

Question:

- What is the maximum flow we can push through the network?

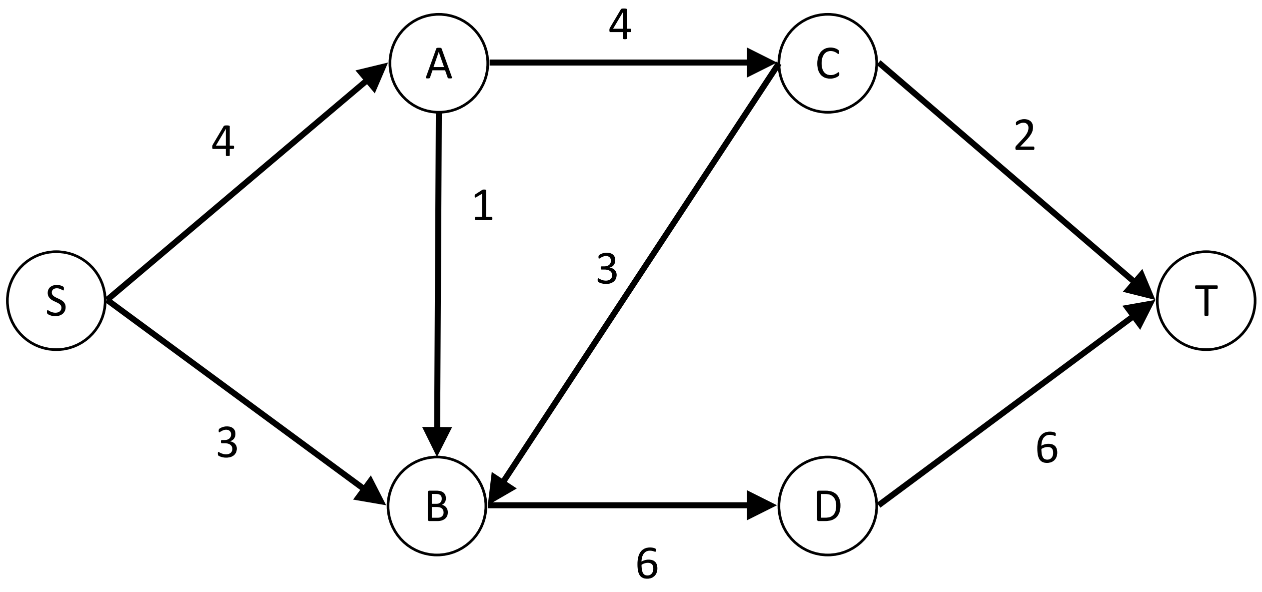
Network Flows

- We have a graph *G(V,E)* with nodes in *V* and edges in *E*.
- We want to find the maximum flow from a source *s* to a destination *t; s, t* $\in$ *V*
- Capacity on edge (*u,v*) is c_{uv}
- Flow from *u* to *v* is *f(u,v)*
- 2 constraints:
  – Capacity: The flow on an arc is less or equal its capacity
  – Flow Conservation: The flow *out of* node *u* is the same as the flow *in* (except at source and destination)

$$f(u,v) \leq c_{u,v} \quad \forall (u,v) \in E$$

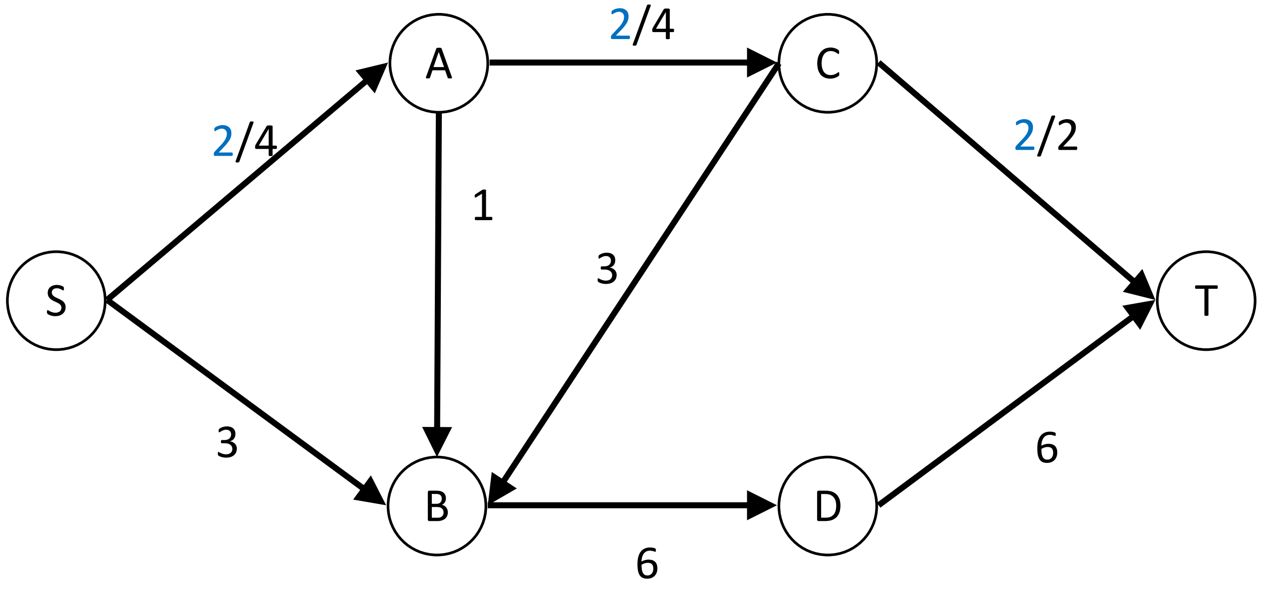
$$\sum_v f(u,v) = \sum_v f(v,u) \quad \forall u \in V \setminus \{s,t\}$$

Network Flows



Nodes and links, links labelled with capacity

Network Flows



Nodes and links, links labelled with flow / capacity

Feasible, but not optimal

Network Flows

- Network flow problems pop up often in practice
- Many supply chain optimisation problems are network flow problems:
  - A natural disaster several locations in NSW
    - I have a food store in Sydney
    - I have a network of truck and rail routes that link cities in NSW
  - **How quickly can I distribute the food I have?**
- Capacities/flows can be
  - Litres/hour
  - Amperes
  - Cars
  - Pallets

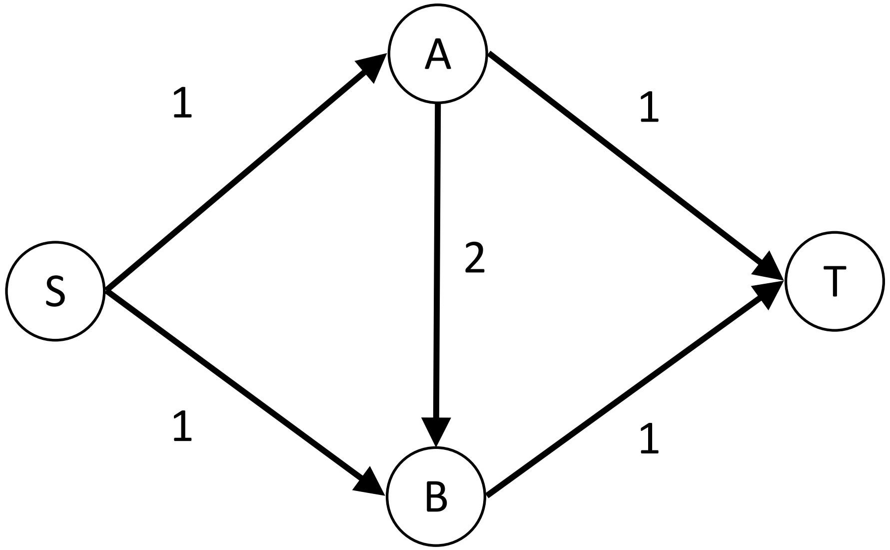
Solving Network Flow problems

- Linear Programming

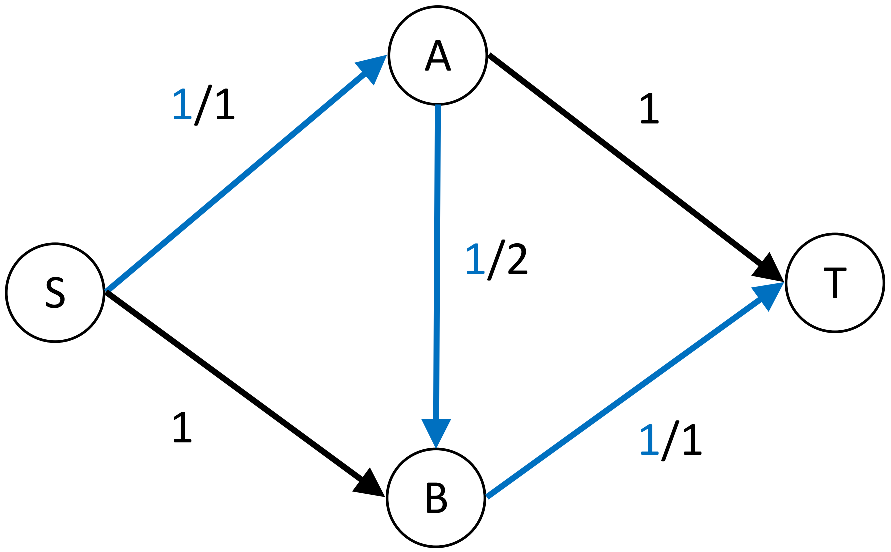
- "Augmenting Flow" algorithms
  - Ford-Fulkerson algorithm[1] — $O(M\ |E|)$, where M is the max flow
  - Edmonds-Karp — $O(|V||E|^2)$ = F-F with tweaks
  - Dinic's — $O(|V|^2\ |E|)$ = E-K with more tweaks

1: Ford, L. R.; Fulkerson, D. R. (1956). "*Maximal flow through a network*". Canadian Journal of Mathematics(**8**): 399–404. doi:10.4153/CJM-1956-045-5.

Ford Fulkerson Algorithm



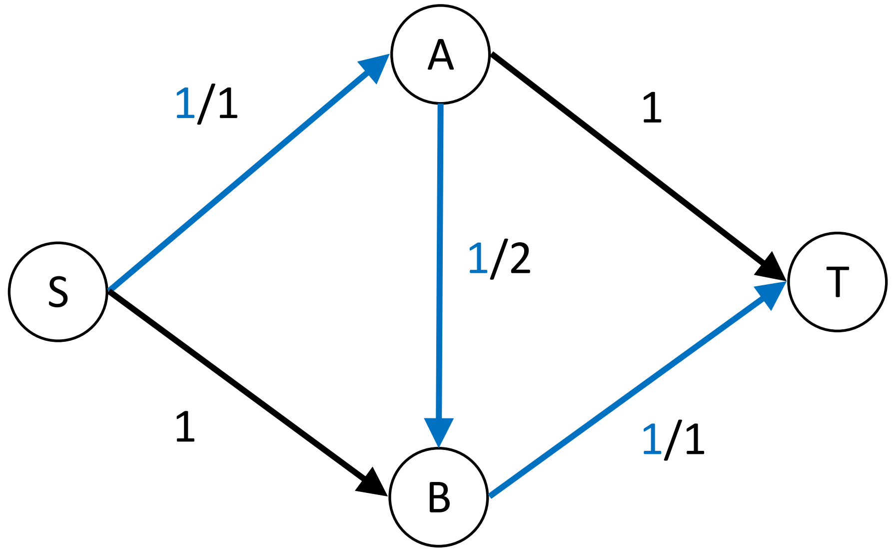
Ford Fulkerson Algorithm



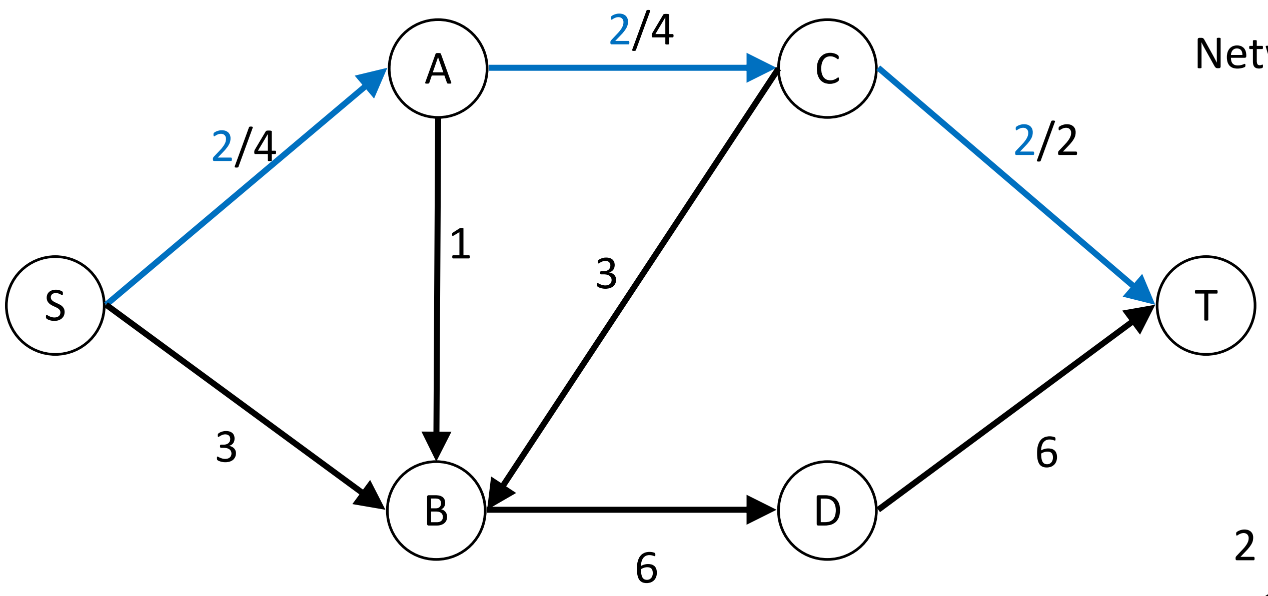
Ford Fulkerson Algorithm

- Key concept: The ***residual network***
- The residual network is the network of "spare capacity"

- It has two types of arcs
  - Forward arcs: $c^f_{uv} = c_{uv} - f(u,v)$
  - Backward arcs: $c^r_{uv} = f(v,u)$

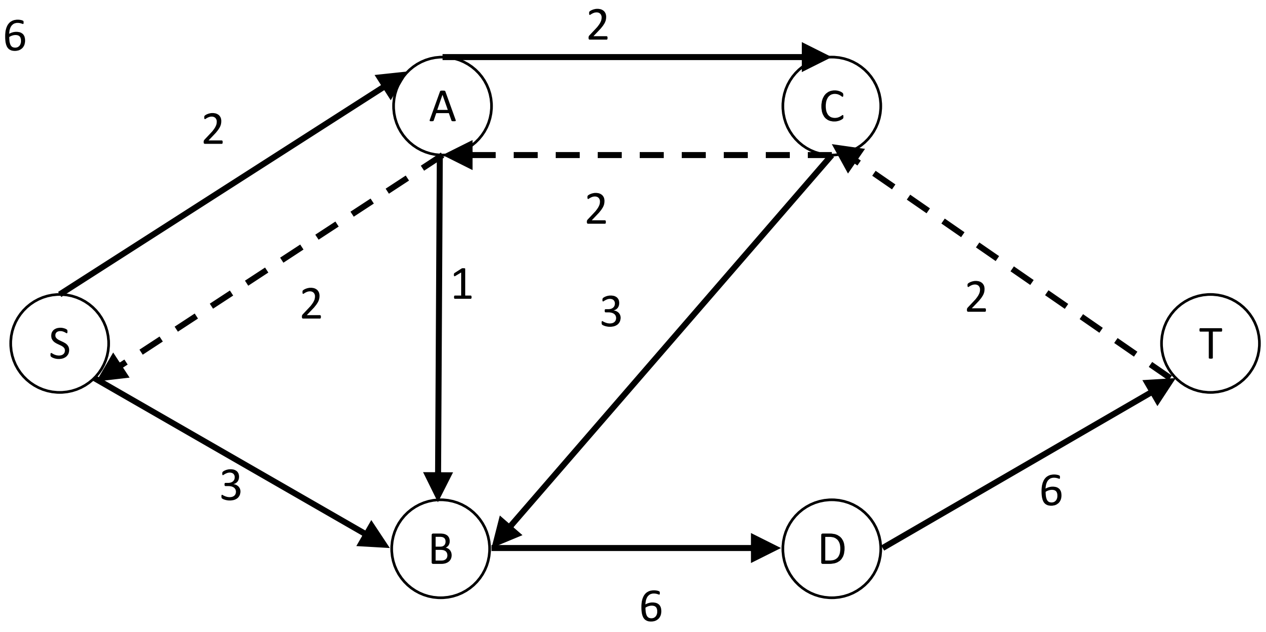


Residual Network

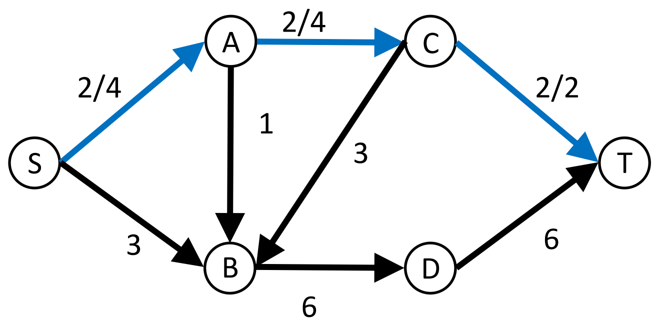


Network with Flow

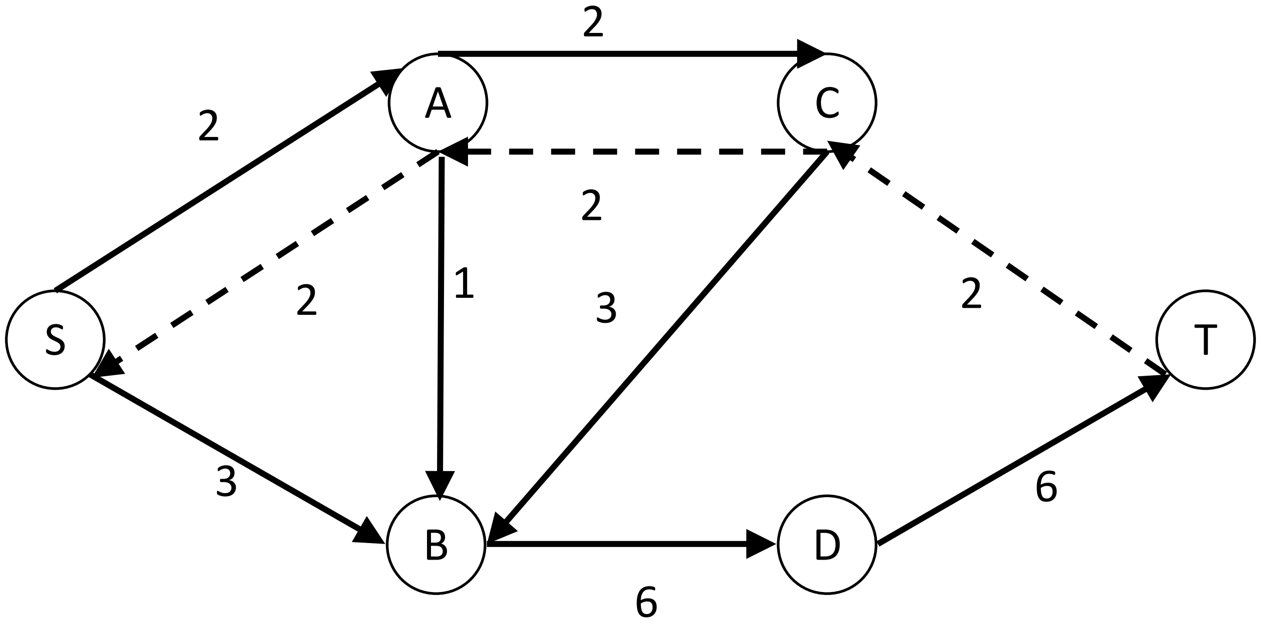
Residual Network



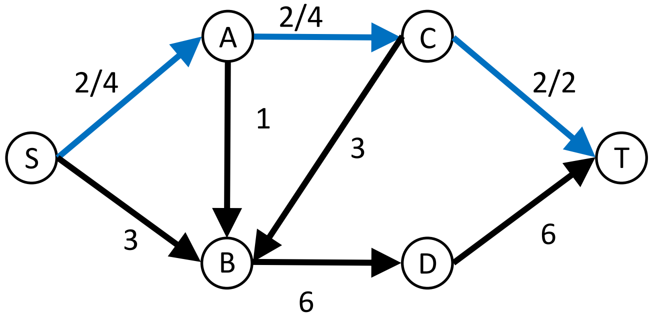
Residual network



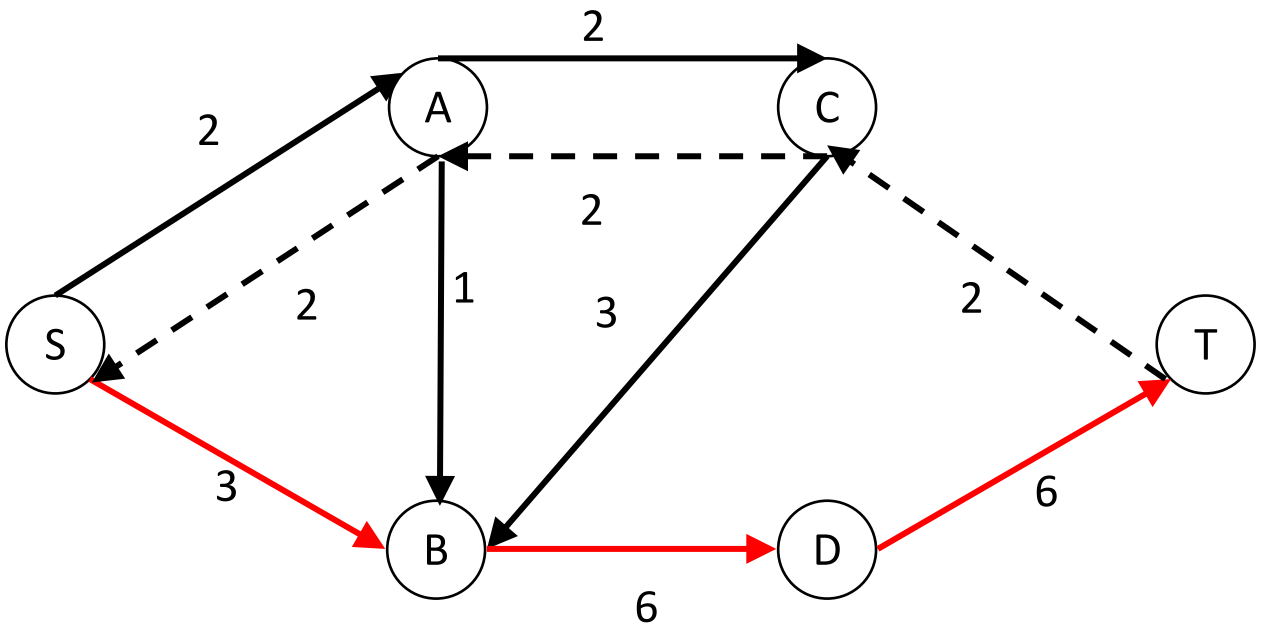
- A flow through the residual network is called an *augmenting flow*
- It represents a potential feasible flow



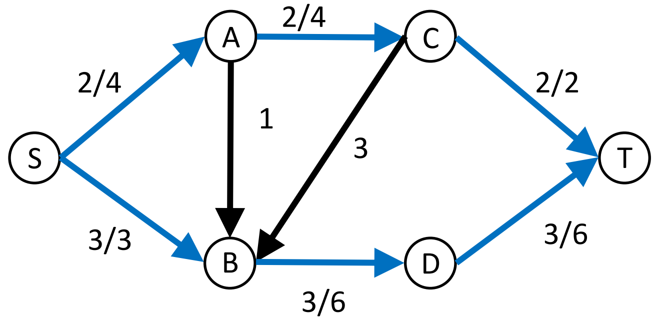
Residual network



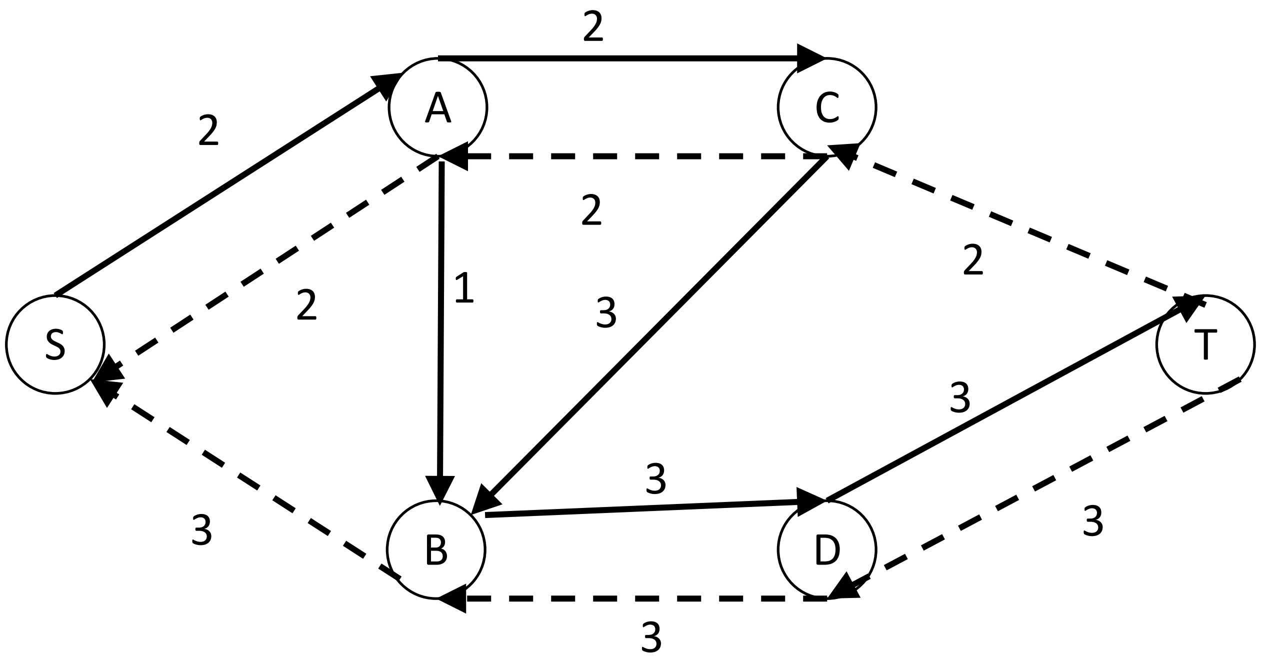
- A flow in the residual network



Residual network



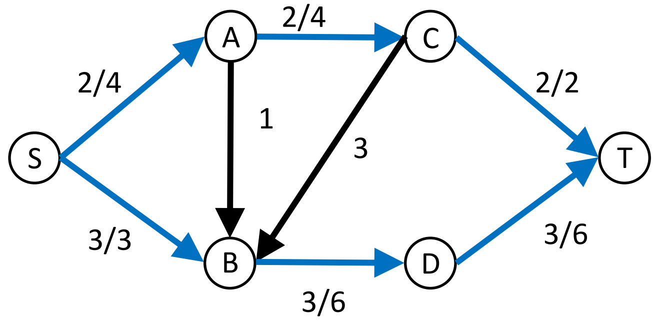
- Updated residual network



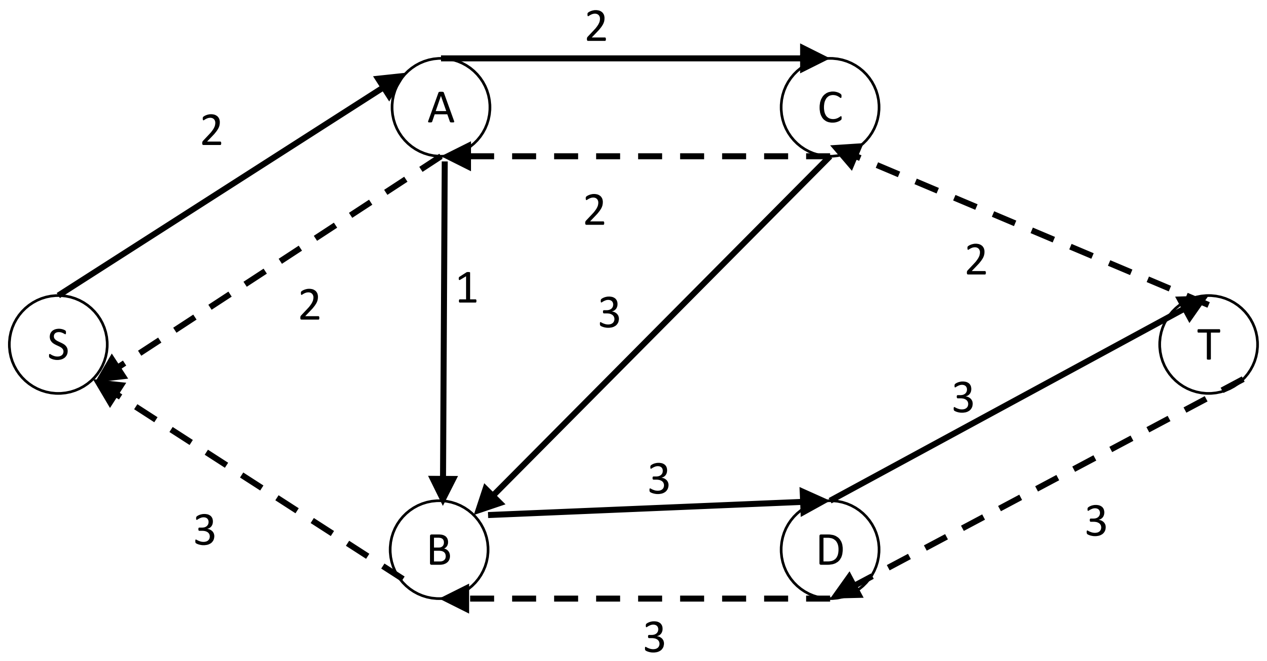
Ford Fulkerson Algorithm

1. While an augmenting path exists
2.   Push the maximum flow through the path
3.   Update residual graph

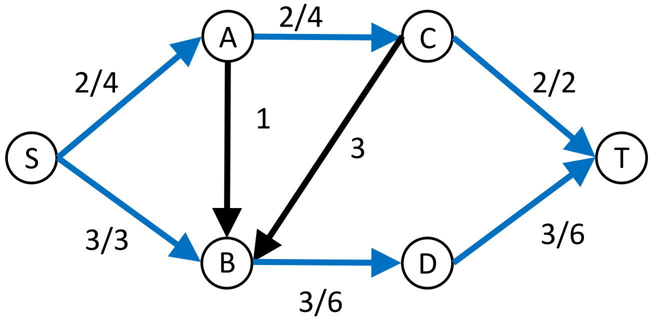
Residual network



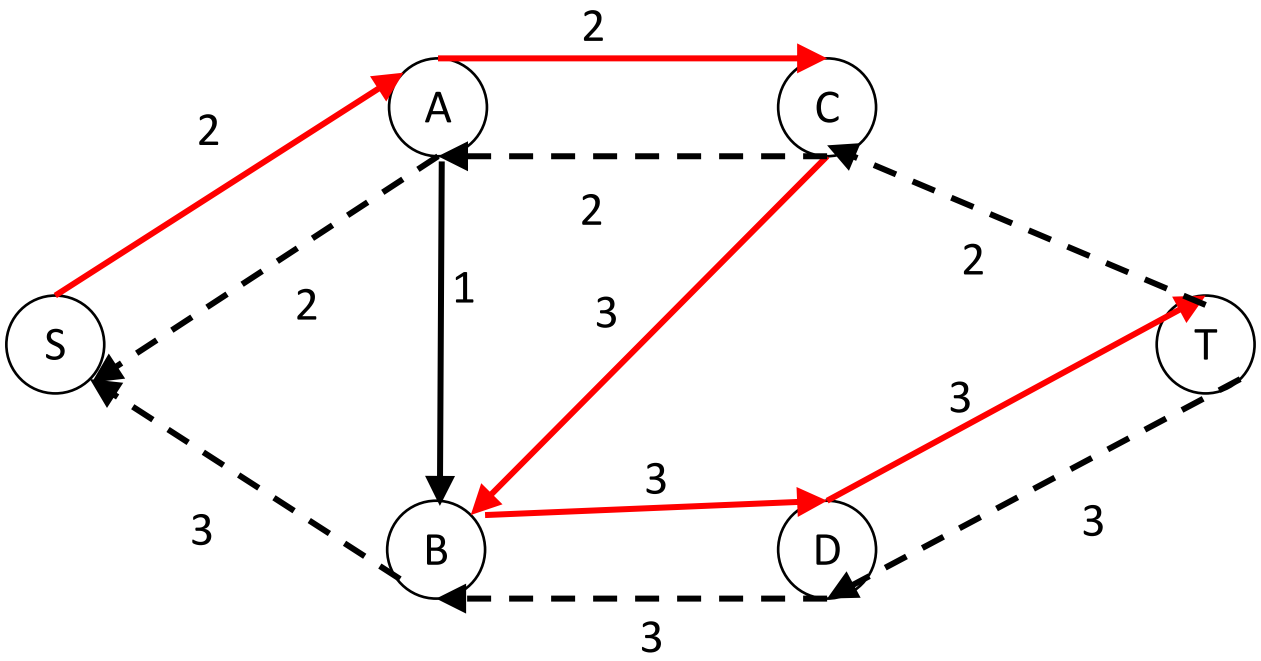
- Flow in the residual network



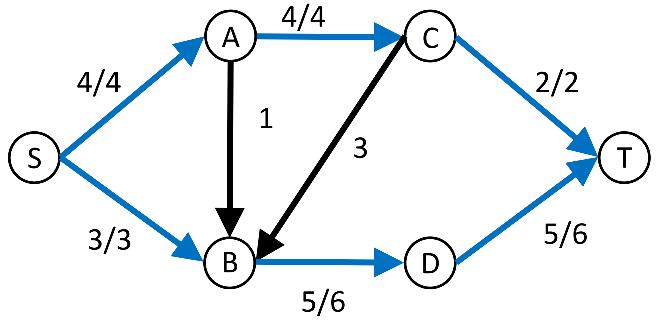
Residual network



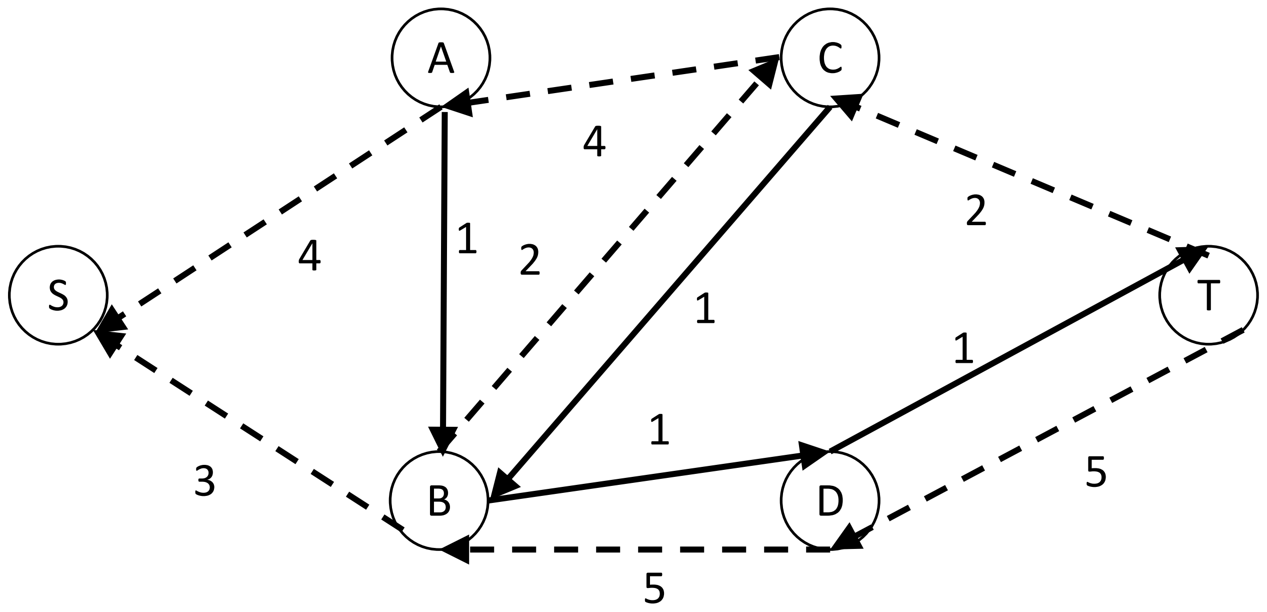
- Flow in the residual network



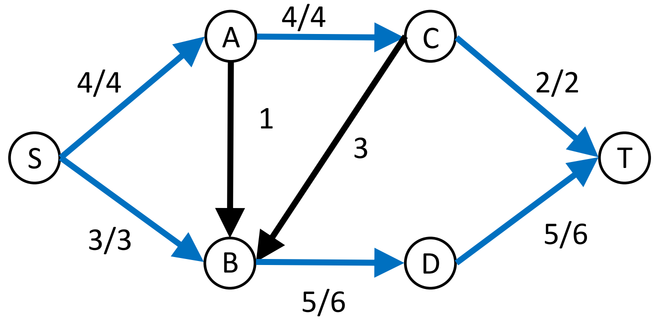
Residual network



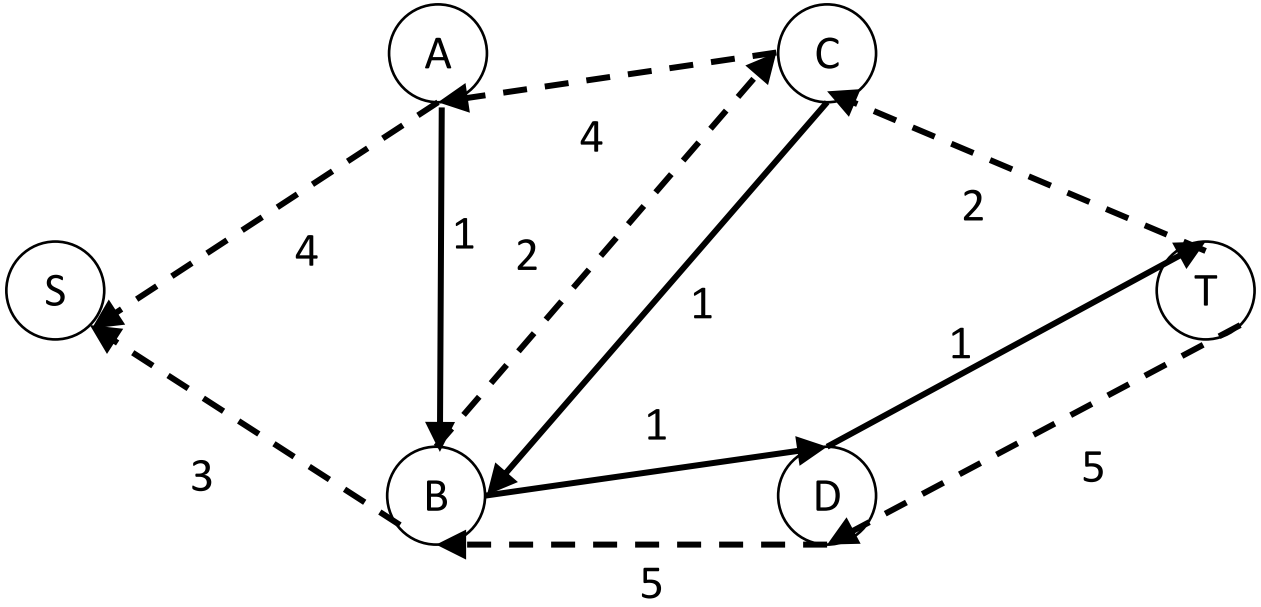
- Updated residual network



Residual network

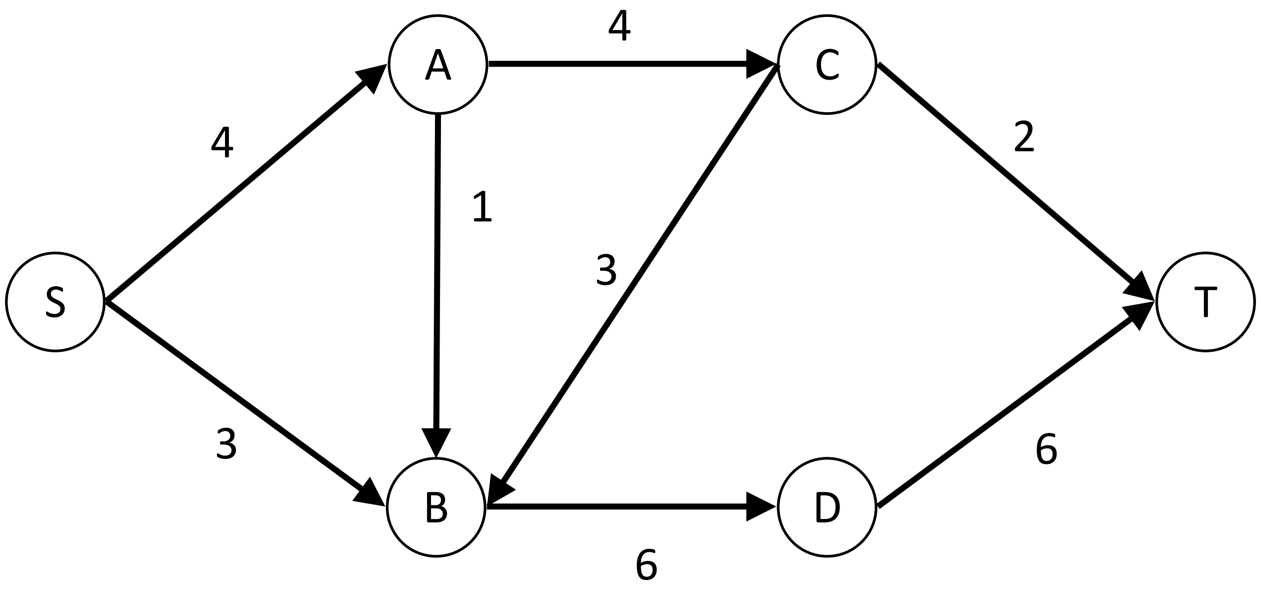


- No more augmenting paths exist: we are done.

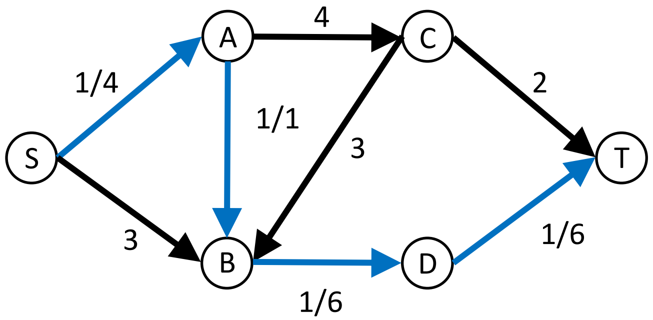


Residual network

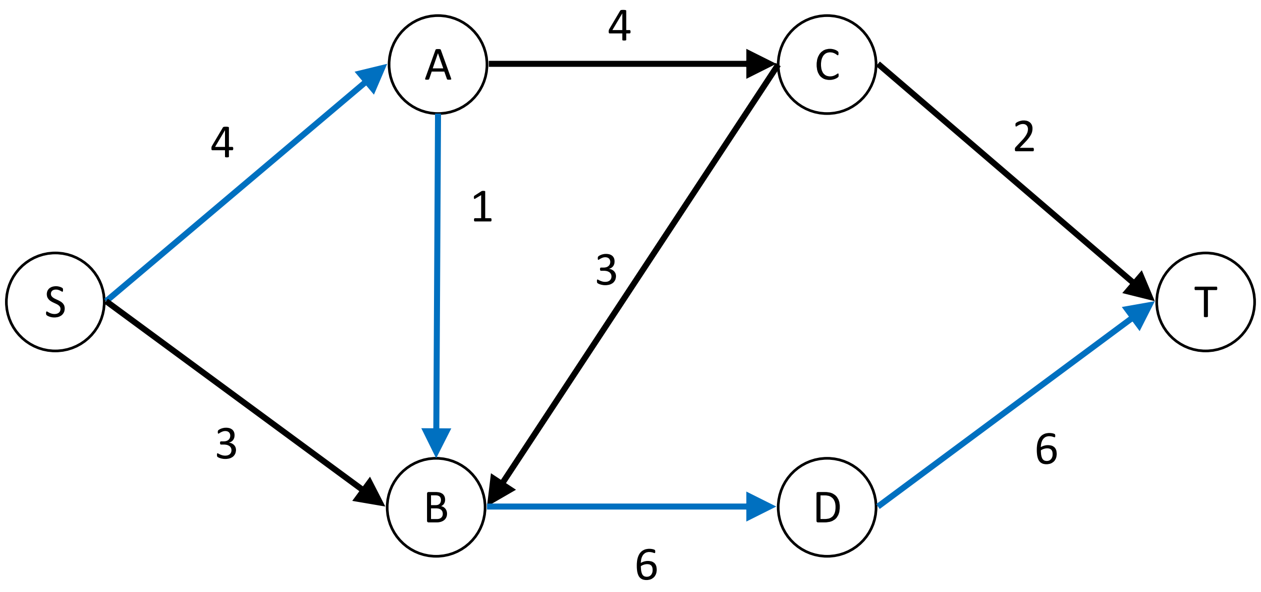
- What if we started out differently



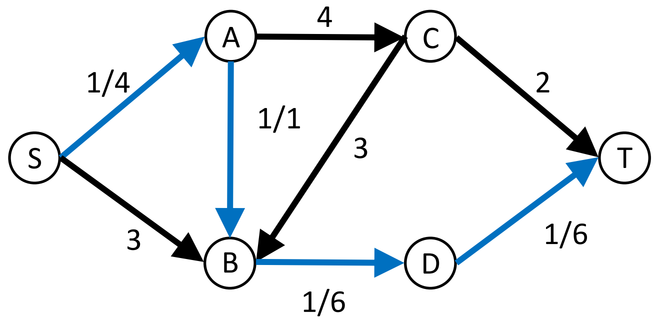
Residual network



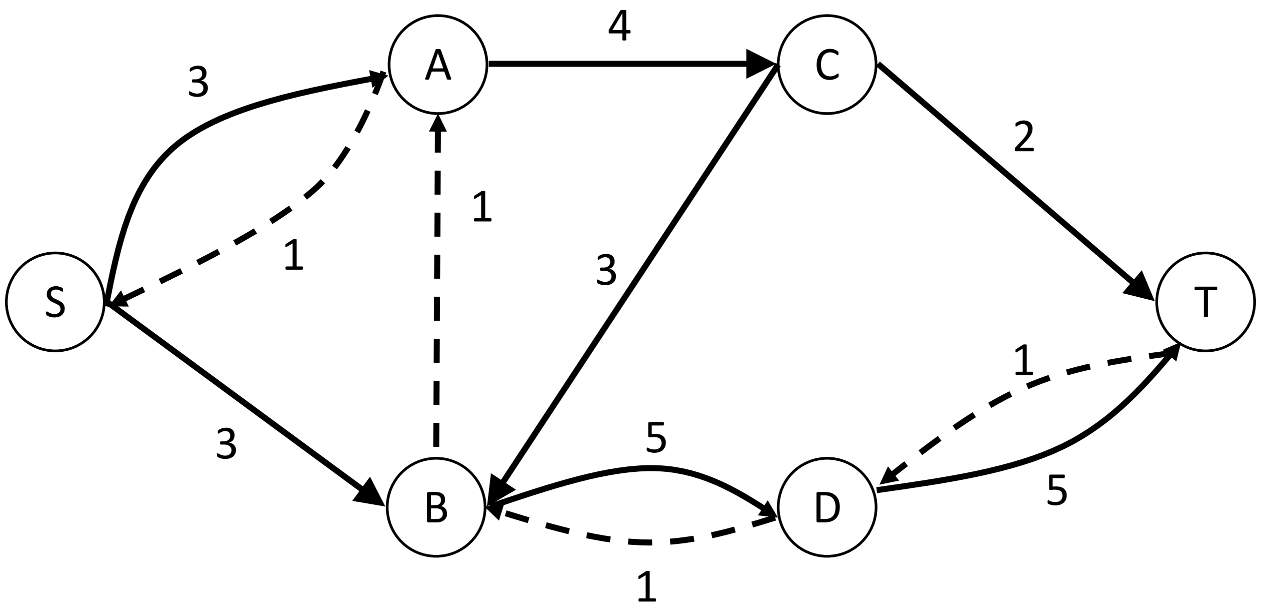
- What if we started out differently



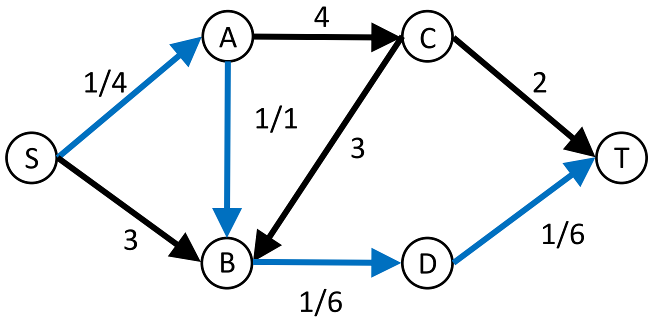
Residual network



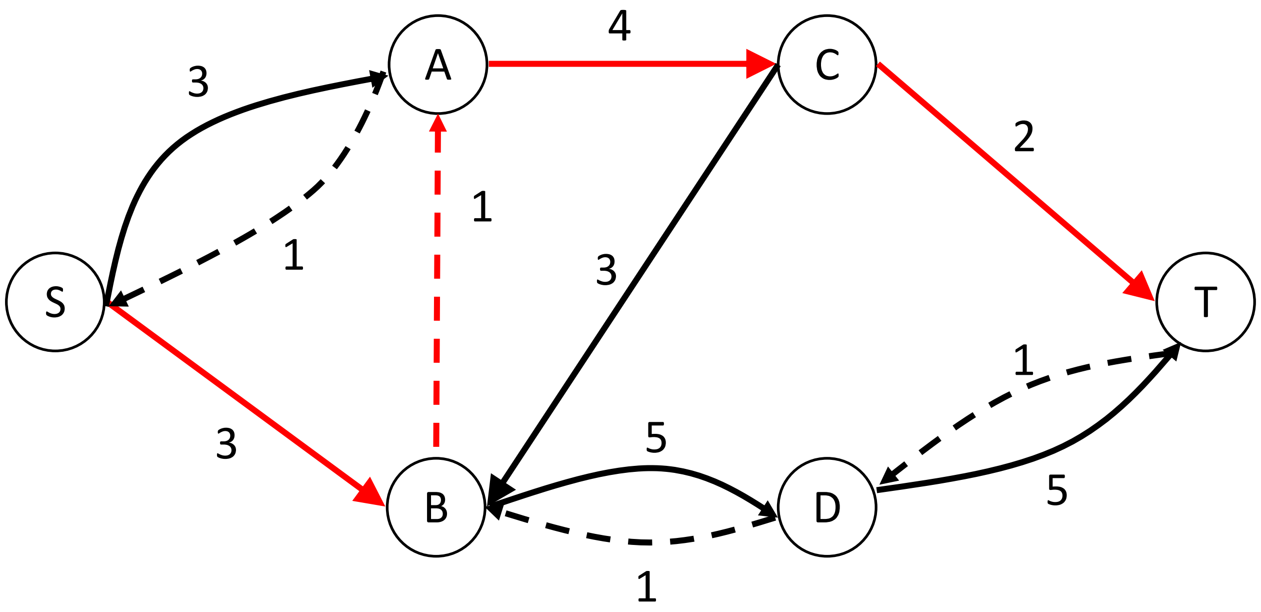
- Residual network



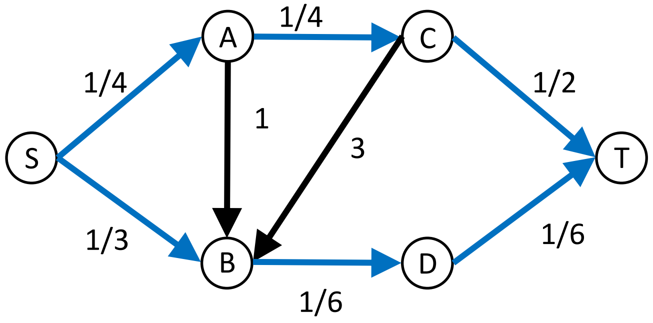
Residual network



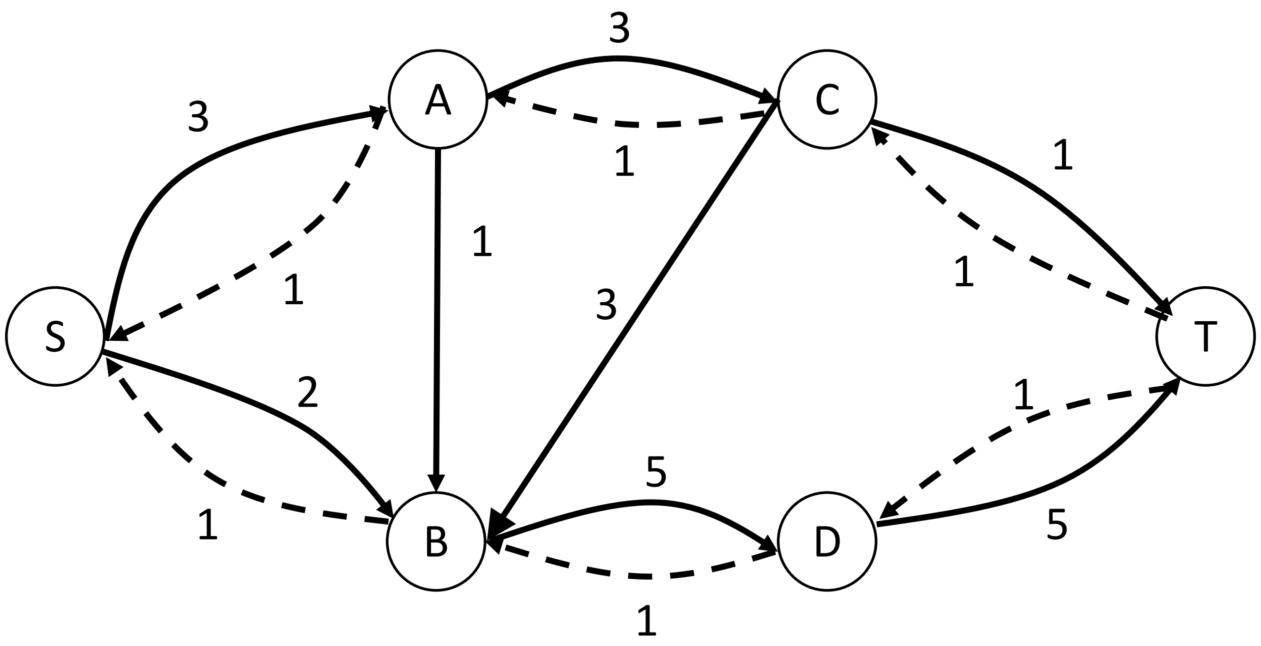
- Augmenting path using a reverse arc



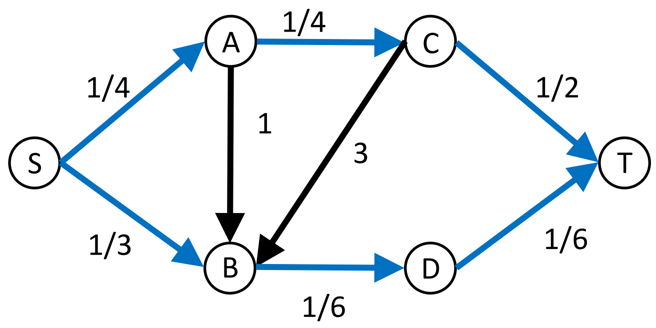
Residual network



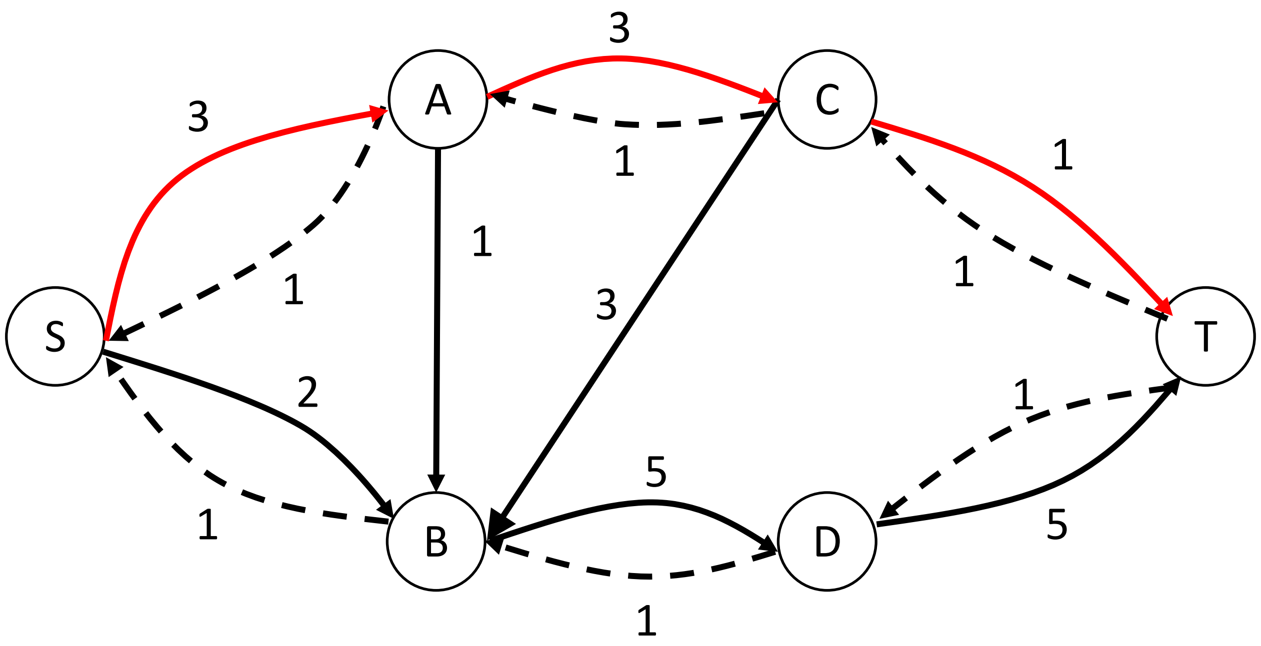
- Residual network



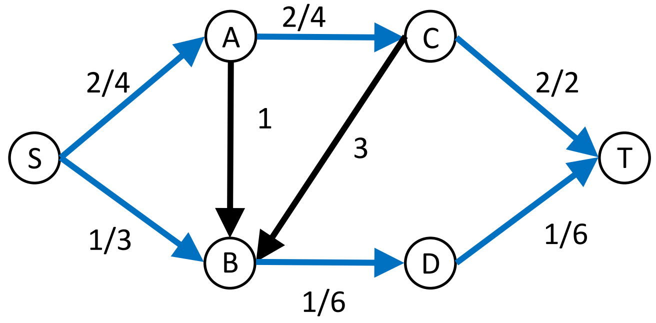
Residual network



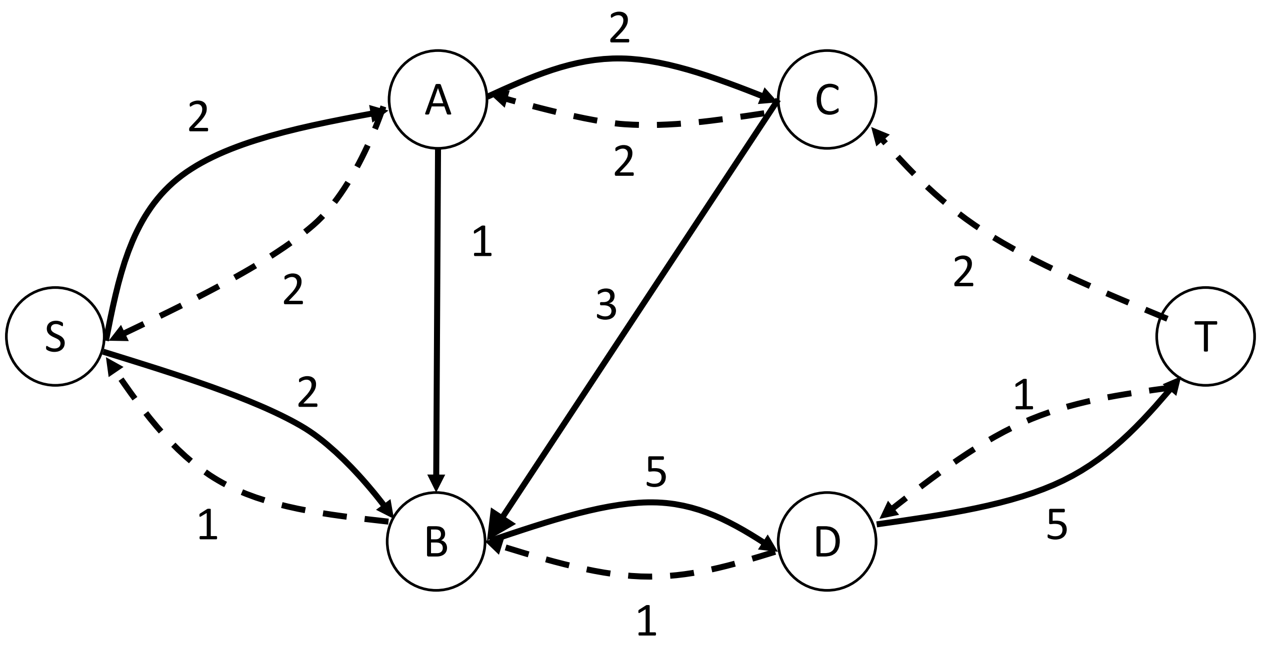
- Augmenting path



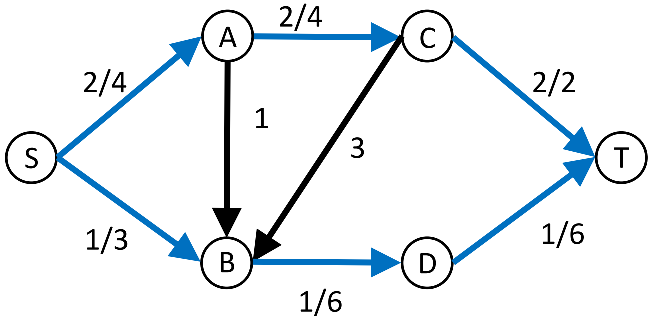
Residual network



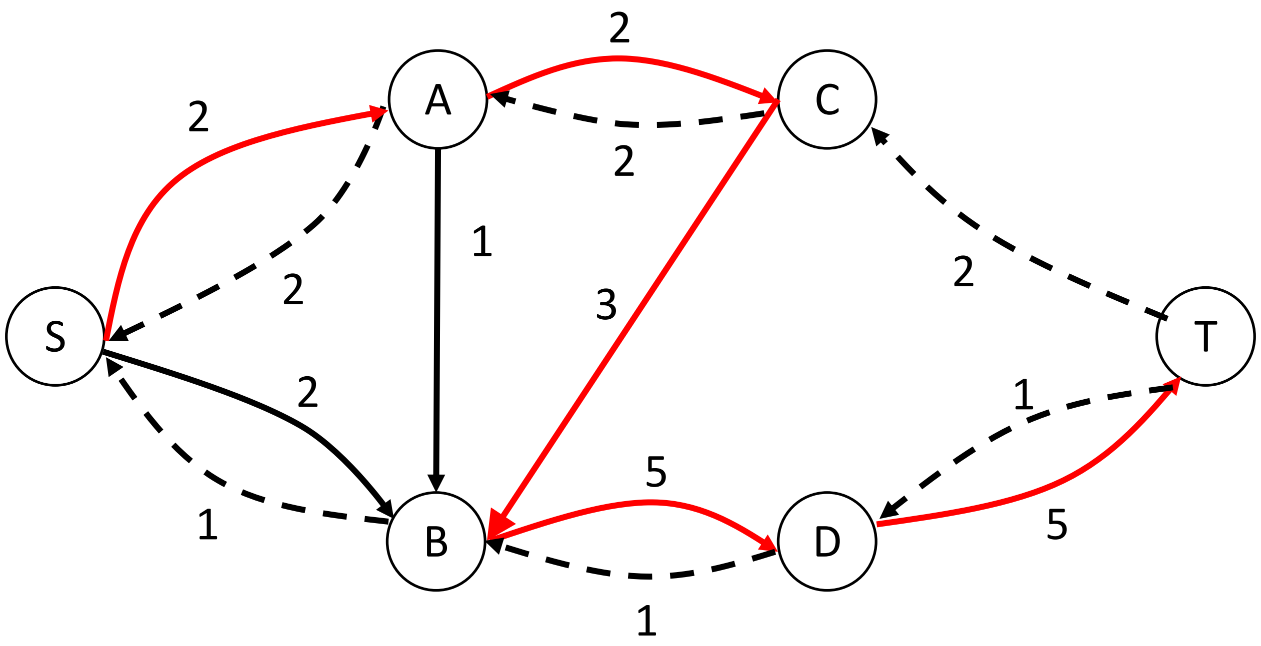
- Residual network



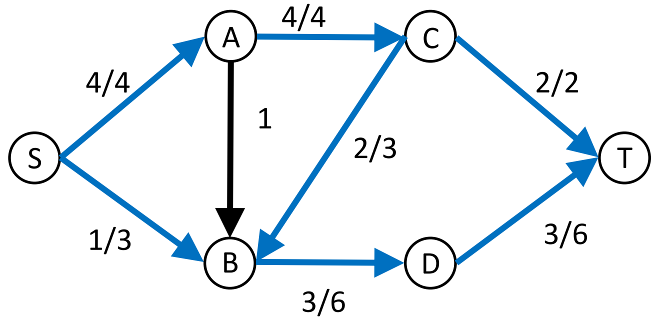
Residual network



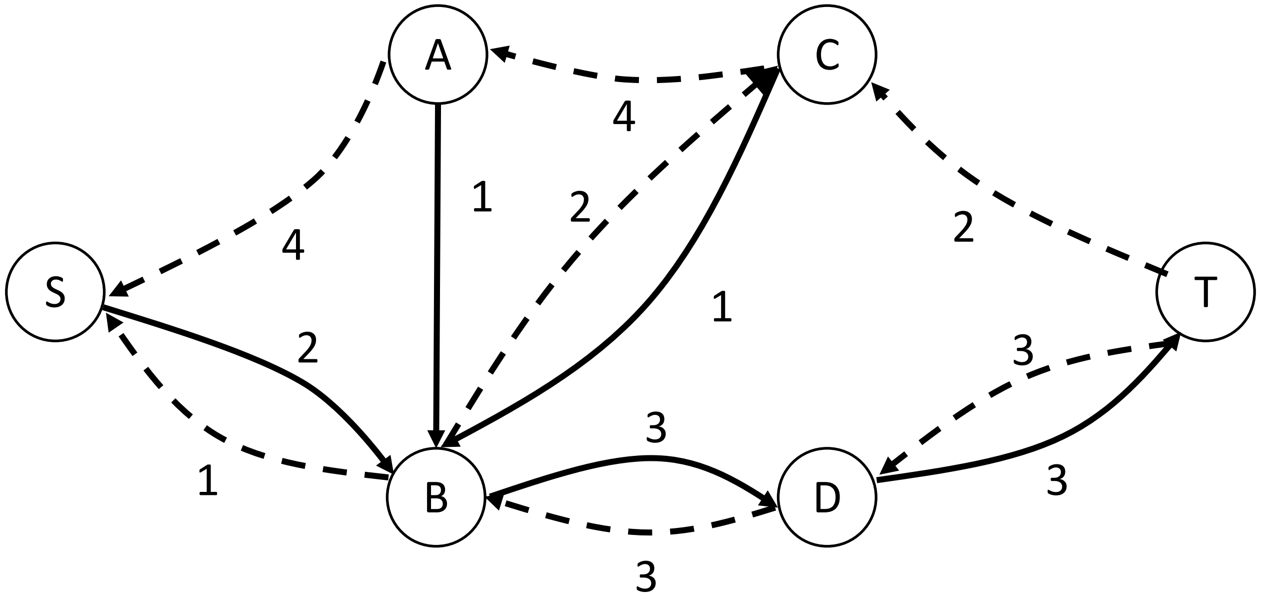
- Augmenting path



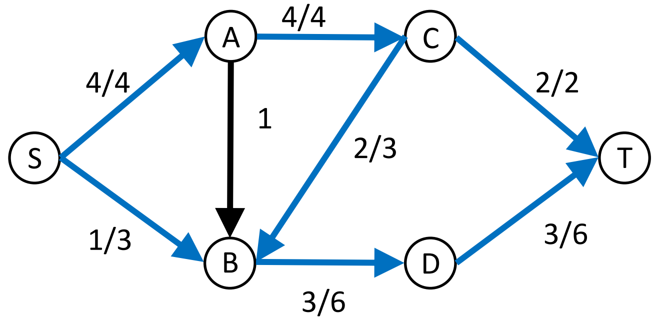
Residual network



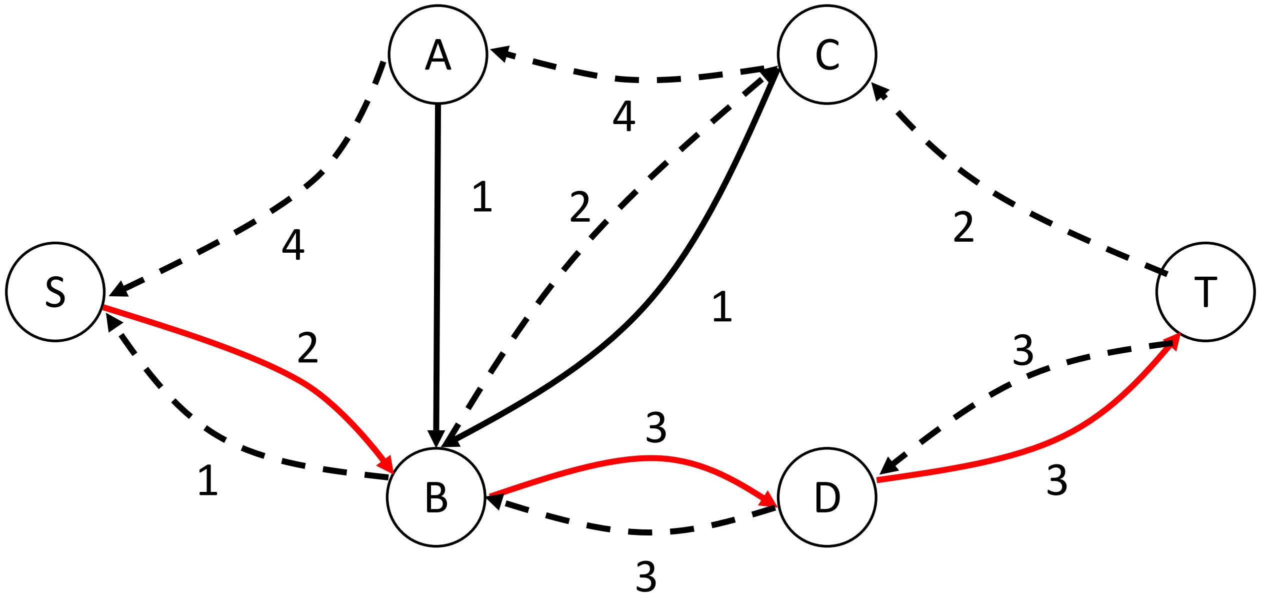
- Residual network



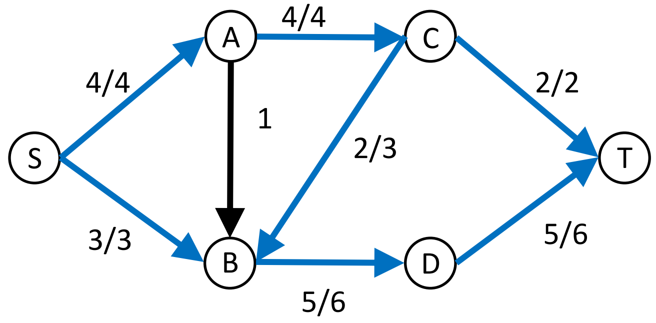
Residual network



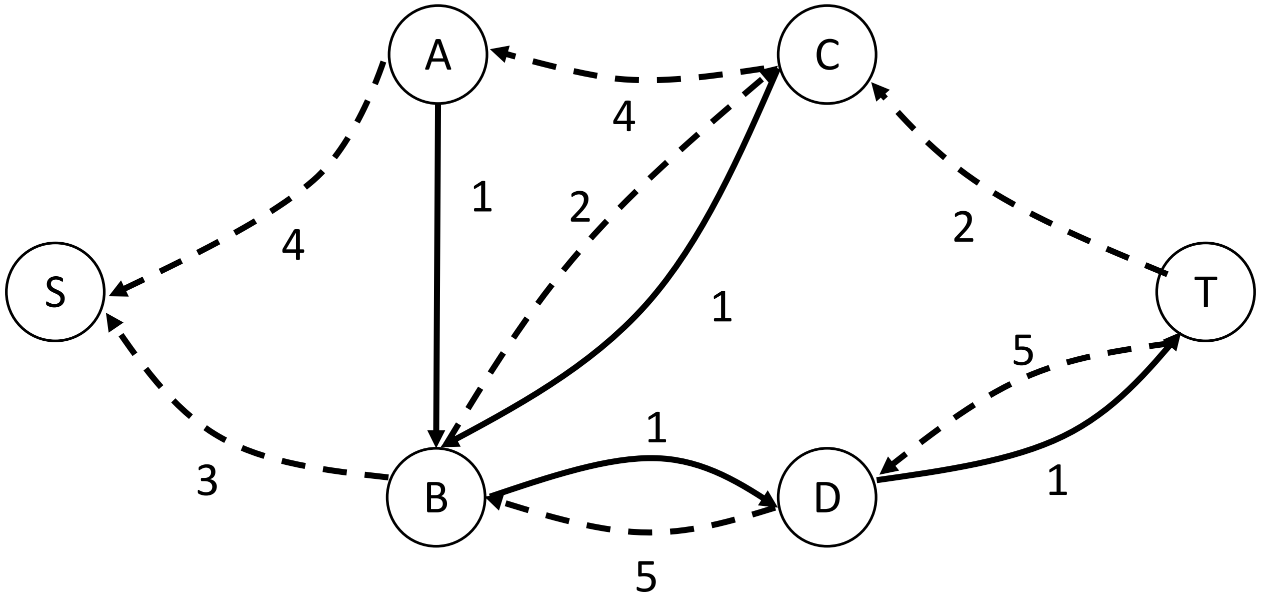
- Augmenting path



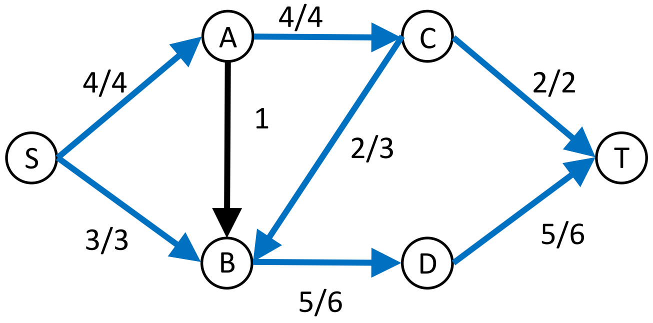
Residual network



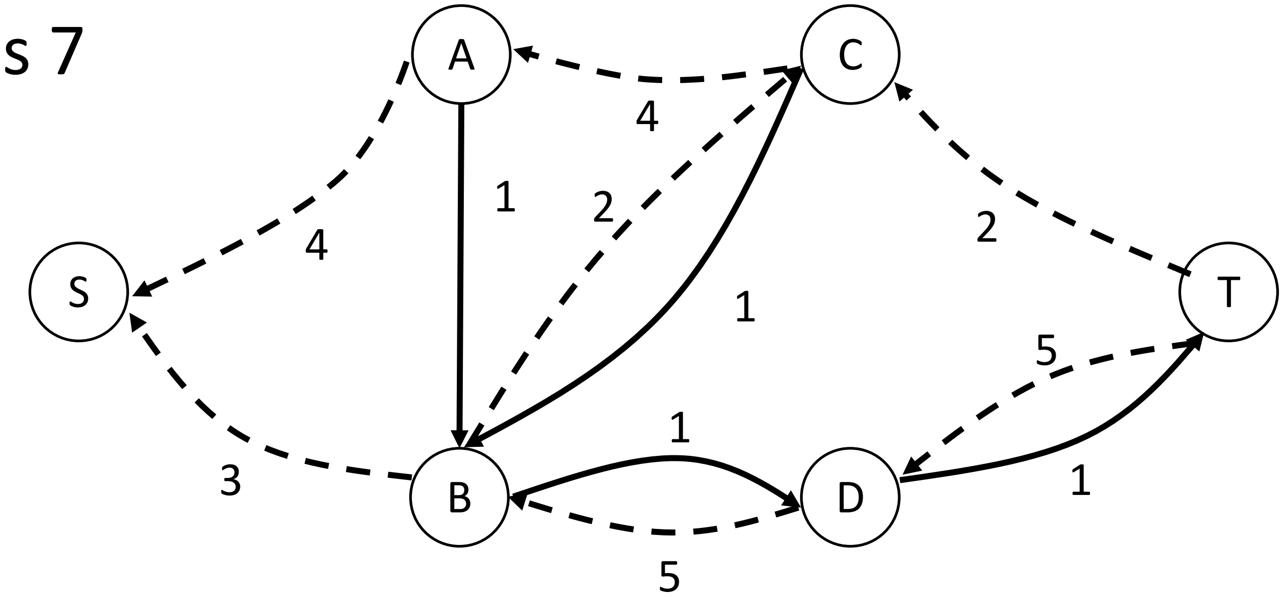
- Residual network



Residual network



- Residual network
- No augmenting path – Done!
- Our max flow is 7



Ford Fulkerson Algorithm

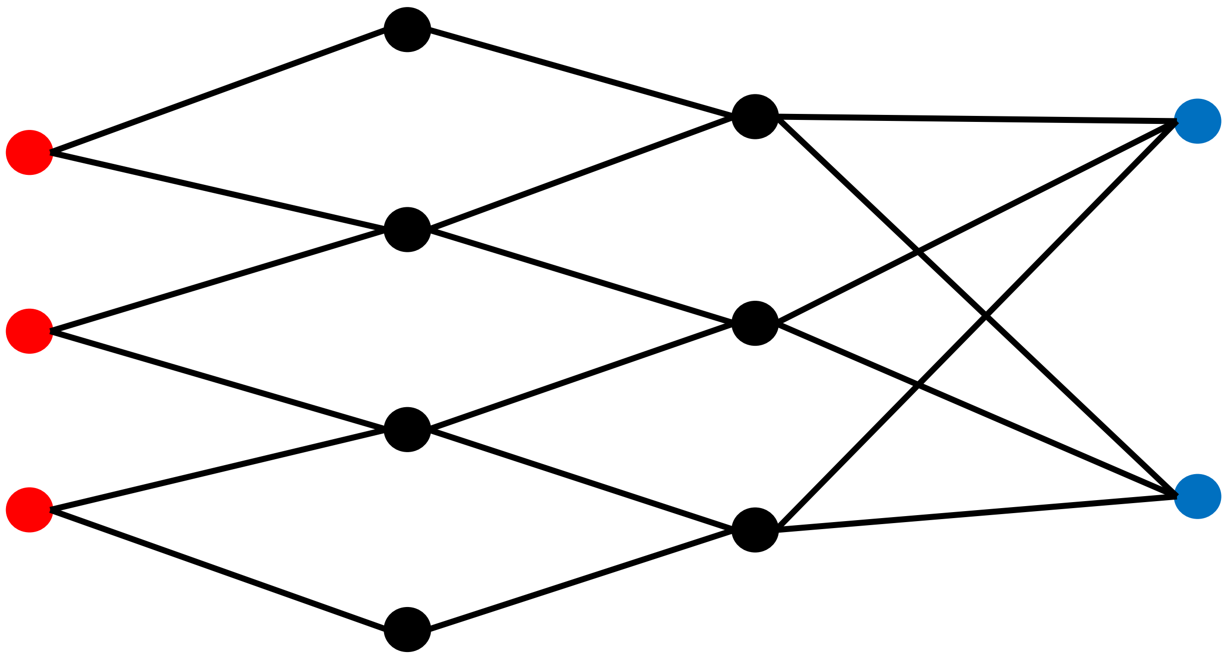
Relies on the theorem

A flow is maximum iff there is no augmenting path.

- "Only If" part is obvious: If there *is* an augmenting path, then we can add more flow and we are not optimal
- "If" part needs more work

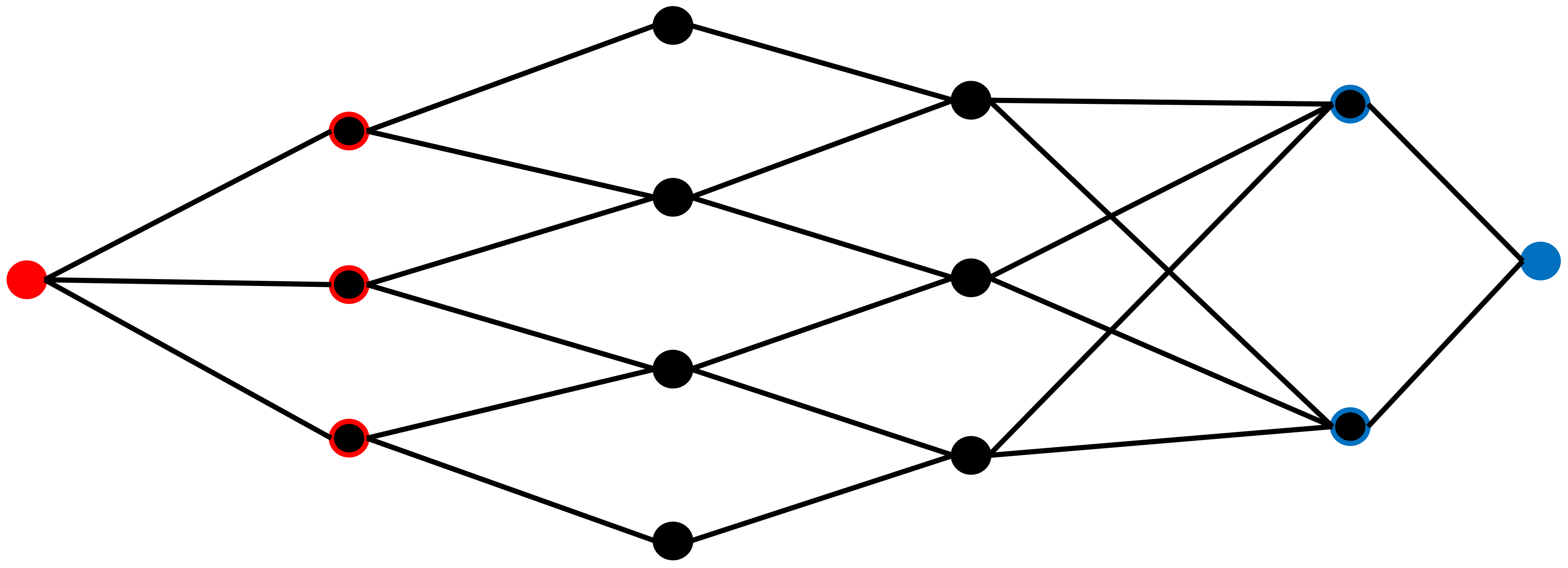
Network Flow

- What about multiple sources / sinks?



Network Flow

- What about multiple sources / sinks?
- Add super-source and super-sink, with infinite capacity



Network Flow as LP or MILP?

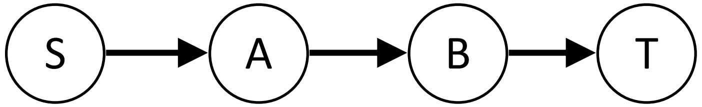
- In lecture 3, we saw that we can compute max flows using this LP:

$$
\begin{aligned}
\max_x \quad & \sum_{(s,j)\in E} x_{s,j} \\
\text{s.t.} \quad & \sum_{(i,k)\in E} x_{i,k} - \sum_{(k,j)\in E} x_{k,j} = 0 \quad \forall k \in V \setminus \{s, n\} \\
& x_{i,j} \le c_{i,j} \quad \forall (i,j) \in E \\
& x_{i,j} \in \mathbb{R}_{\ge 0}
\end{aligned}
$$

- All the combinatorial algorithms find integer solutions. Can this LP do the same?

Total Unimodularity

- A matrix **A** is totally unimodular if all its subdeterminants are +1, 0 or -1
  - subdeterminat: the determinant of a squared submatrix of A

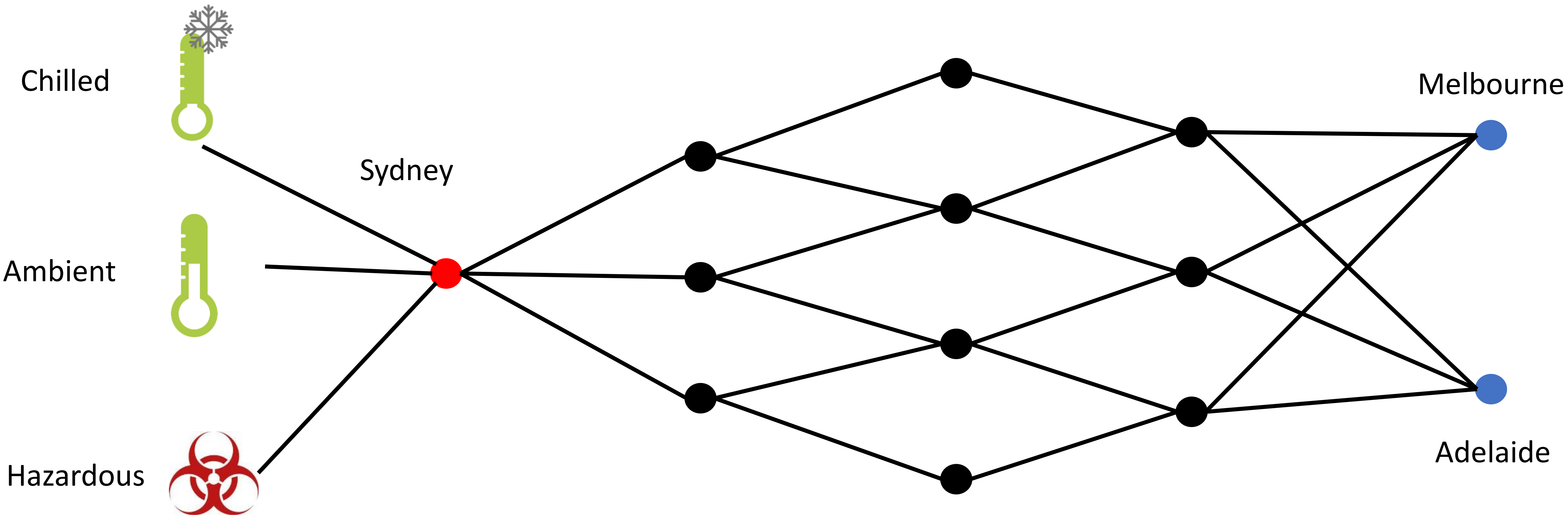


$$\begin{array}{cc} & \begin{array}{cccc} S & A & B & T \end{array} \\ \begin{array}{c} A \\ B \end{array} & \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \end{array}$$

- An **LP with totally unimodular constraint matrix A** and integer vector b (RHS of the constraint) **has an integer optimal solution x***
  - Thm: the polyhedron defined by Ax ≤ b is integral, i.e., all its vertices are in $\mathbb{Z}^n$
- Max-flow has this property:
  - arc-node incidence matrix is totally unimodular (flow-preservation constraints)
  - Identity matrix for capacity constraints is also totally unimodular

Multi-commodity Network flow (1)

- What about multiple commodities?



Multi-commodity Network flow (2)

- Bad News: Multi-commodity flow does not have a totally unimodular constraint matrix
  - i.e., it does not have “integer relaxation” property
- Hence: Multi-commodity flow is NP-hard
  - Unlikely to have an efficient solution

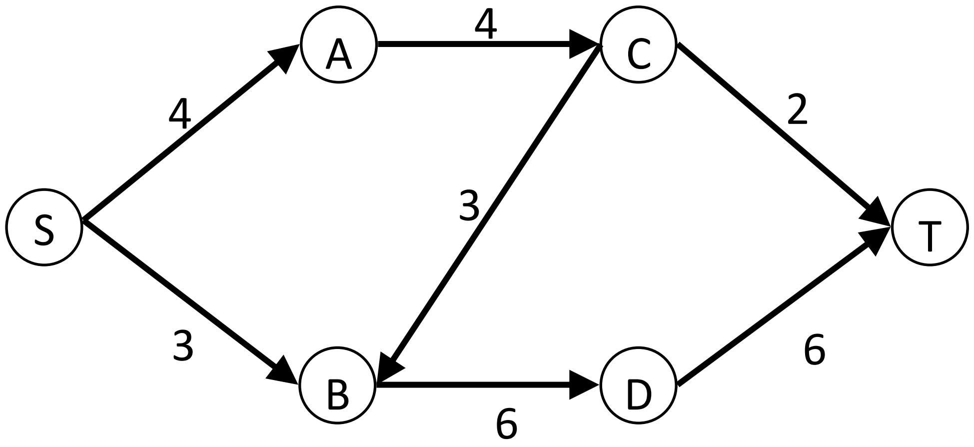
Cuts

Another interesting question, for example in a communications network:

I want to disable communication from s to t.
What is the smallest capacity I need to cut to do it?

Equivalently

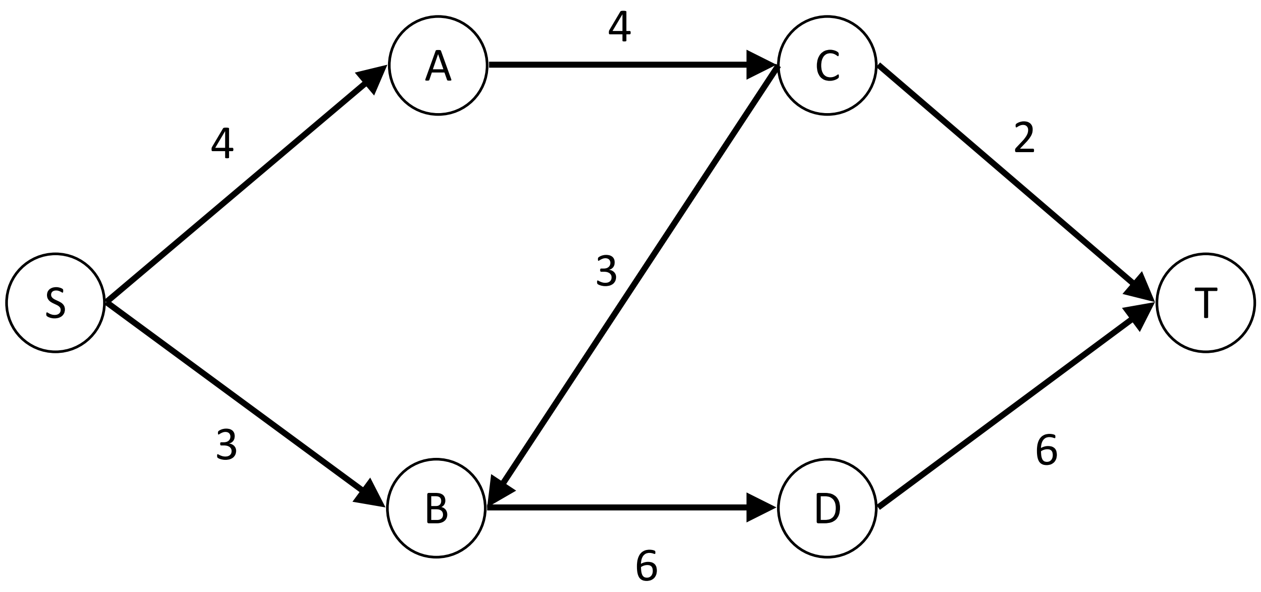
Find sets S and T which partition N, and for which
the capacity of edges crossing the partition is least.



Cuts

Another interesting question, for example in a communications network:

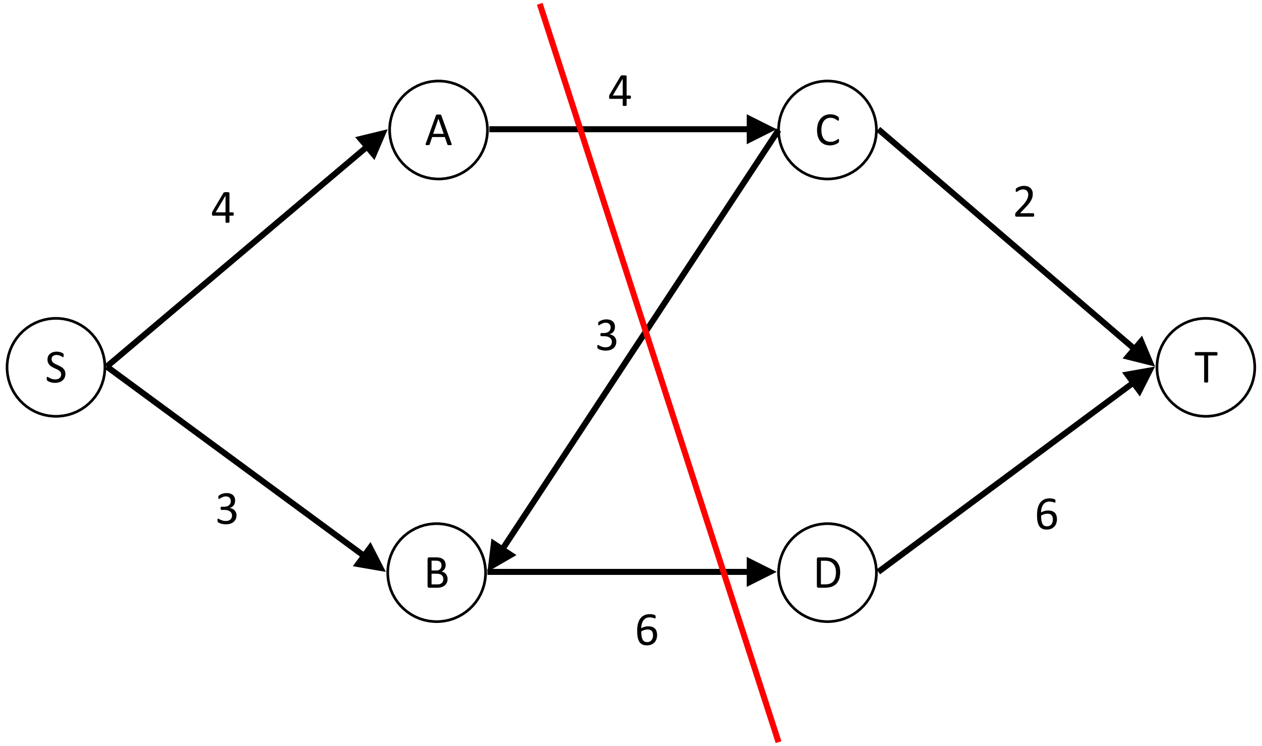
I want to disable communication from s to t.
What is the smallest capacity I need to cut to do it?



Cuts

Another interesting question, for example in a communications network:

I want to disable communication from s to t.
What is the smallest capacity I need to cut to do it?

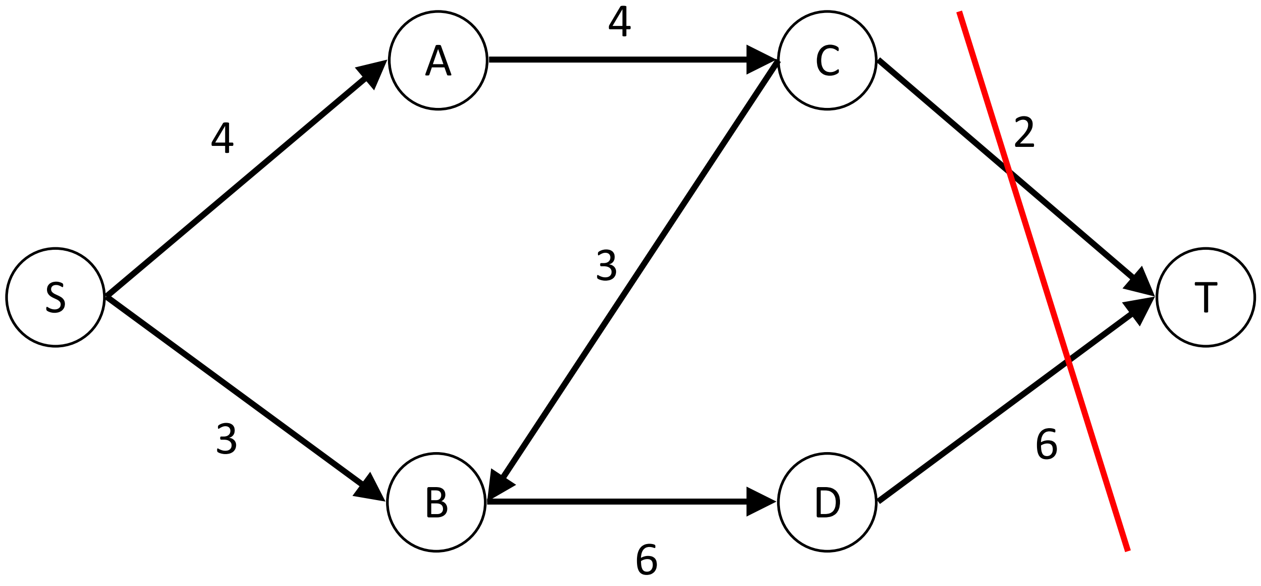


S = {S, A, B}
T = {C, D, T}
Weight = 13

Cuts

Another interesting question, for example in a communications network:

I want to disable communication from s to t.
What is the smallest capacity I need to cut to do it?

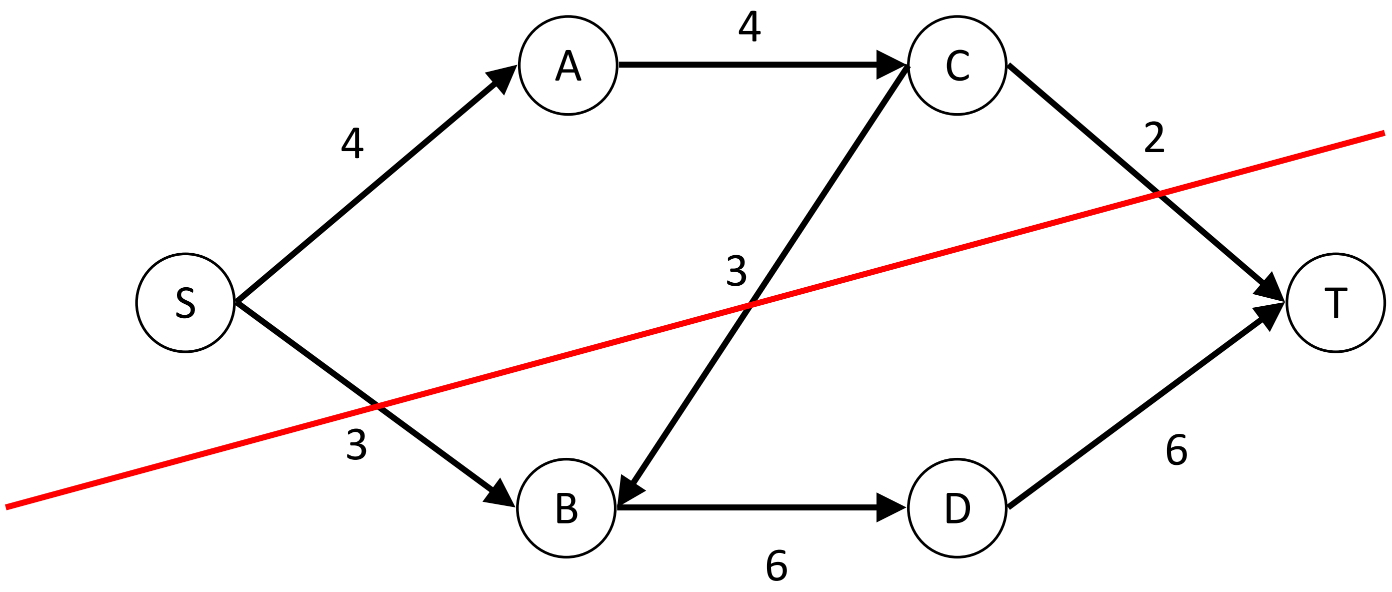


S = {S, A, B , C, D}
T = {T}
Weight = 8

Cuts

Another interesting question, for example in a communications network:

I want to disable communication from s to t.
What is the smallest capacity I need to cut to do it?

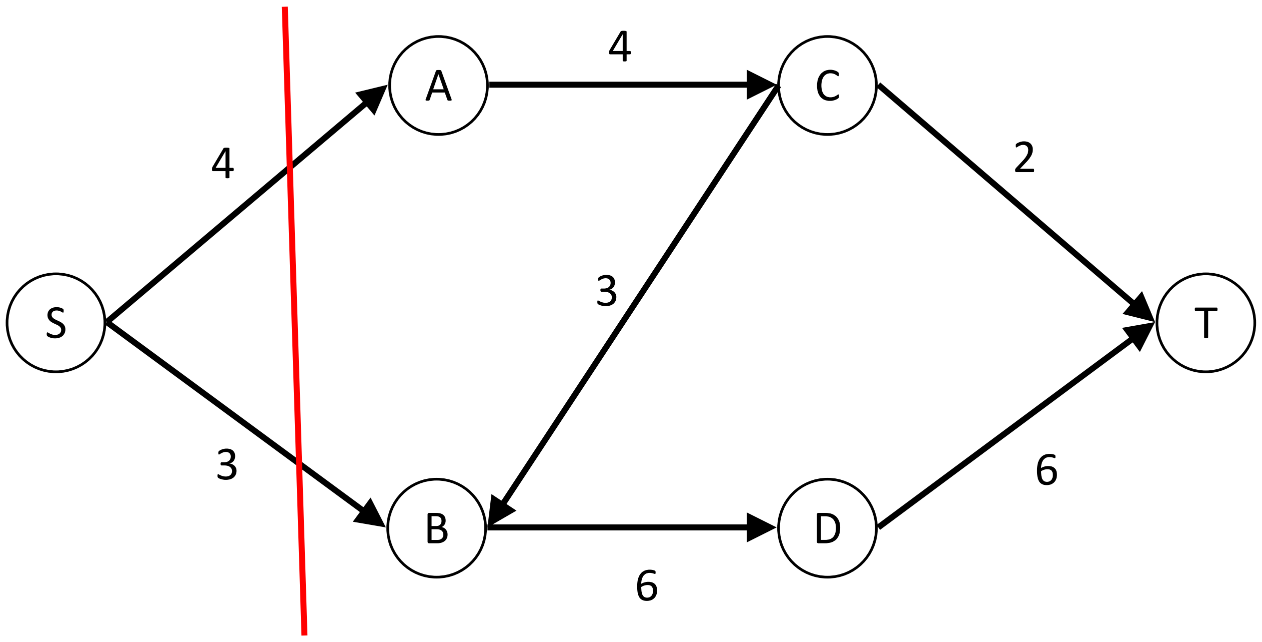


S = {S, A , C}
T = {B, D, T}
Weight = 8

Cuts

Another interesting question, for example in a communications network:

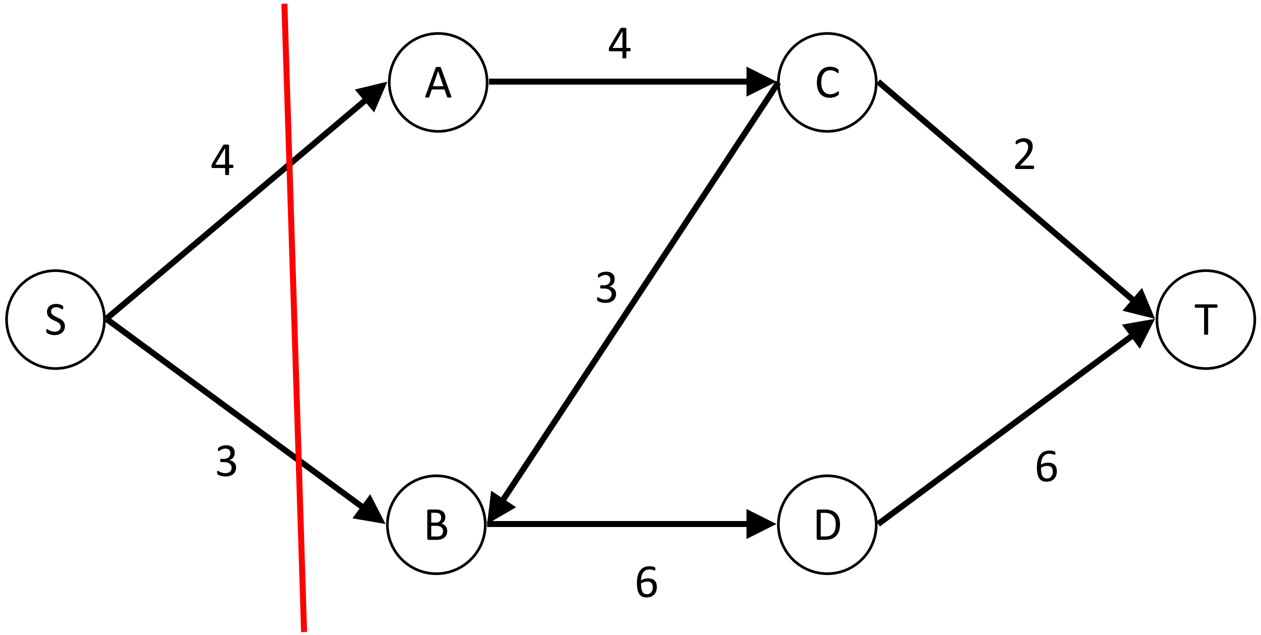
I want to disable communication from s to t.
What is the smallest capacity I need to cut to do it?



S = {S}
T = {A, B, C, D, T}
Weight = 7

Cuts

- 7 is the capacity of the min cut
- Also the capacity of the max flow
- Coincidence?



S = {S}
T = {A, B, C, D, T}
Weight = 7

Max Flow Min Cut Theorem

- Coincidence? **Well, no**.

Max Flow Min Cut Theorem

The value of the maximum flow is equal to the value of the minimum cut

- Omit proof
- See lecture 5 about duality: dual of the Max Flow LP is the Min Cut LP

Max Flow Min Cut Theorem

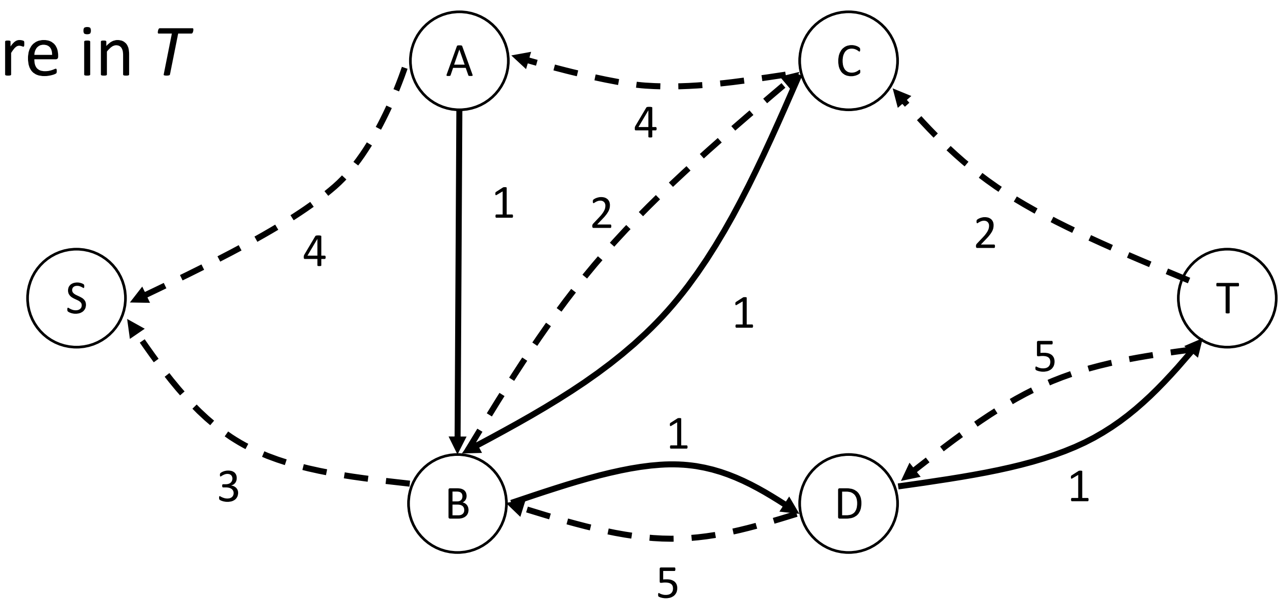
- We know the *value* of the min cut, but what nodes are involved

- Use the residual graph of the maximum flow.
- Recursively label any node that can be reached from *s*
- Any labelled node is in *S,* the rest are in *T*

Max Flow Min Cut Theorem

- We know the ***value*** of the min cut, but what nodes are involved

- Use the residual graph of the maximum flow.
- Recursively label any node that can be reached from *s*
- Any labelled node is in *S, the rest are in *T*

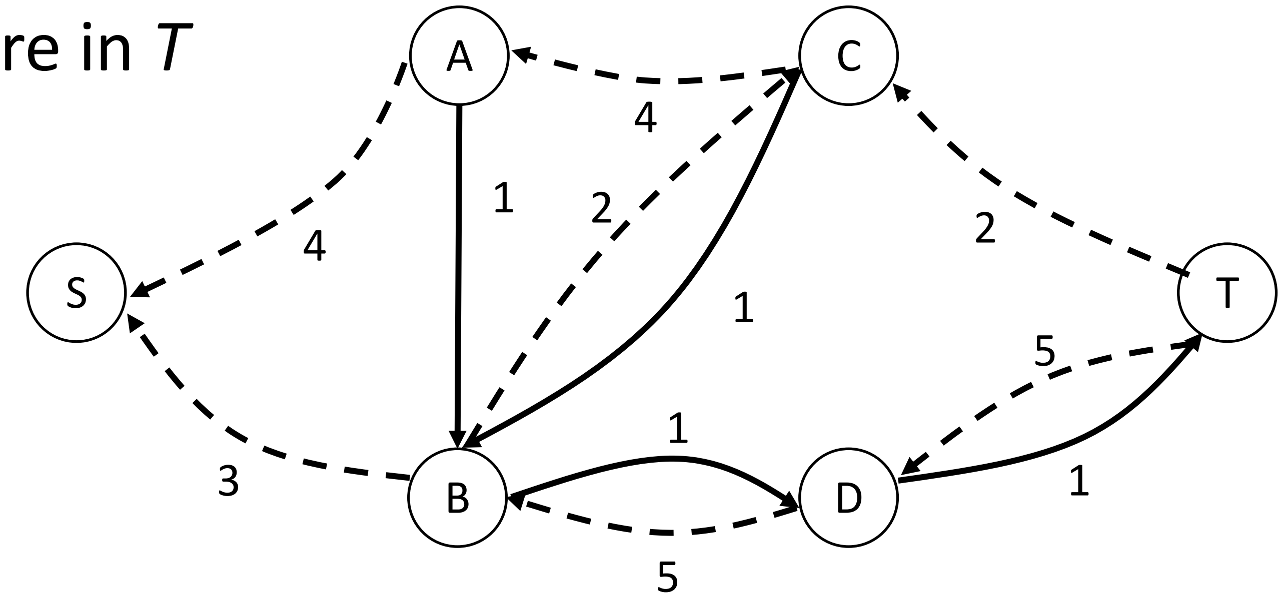


Max Flow Min Cut Theorem

- We know the *value* of the min cut, but what nodes are involved

- Use the residual graph of the maximum flow.
- Recursively label any node that can be reached from *s*
- Any labelled node is in *S,* the rest are in *T*

- No labels
- S = {S}
- T = {A, B,C, D, T}

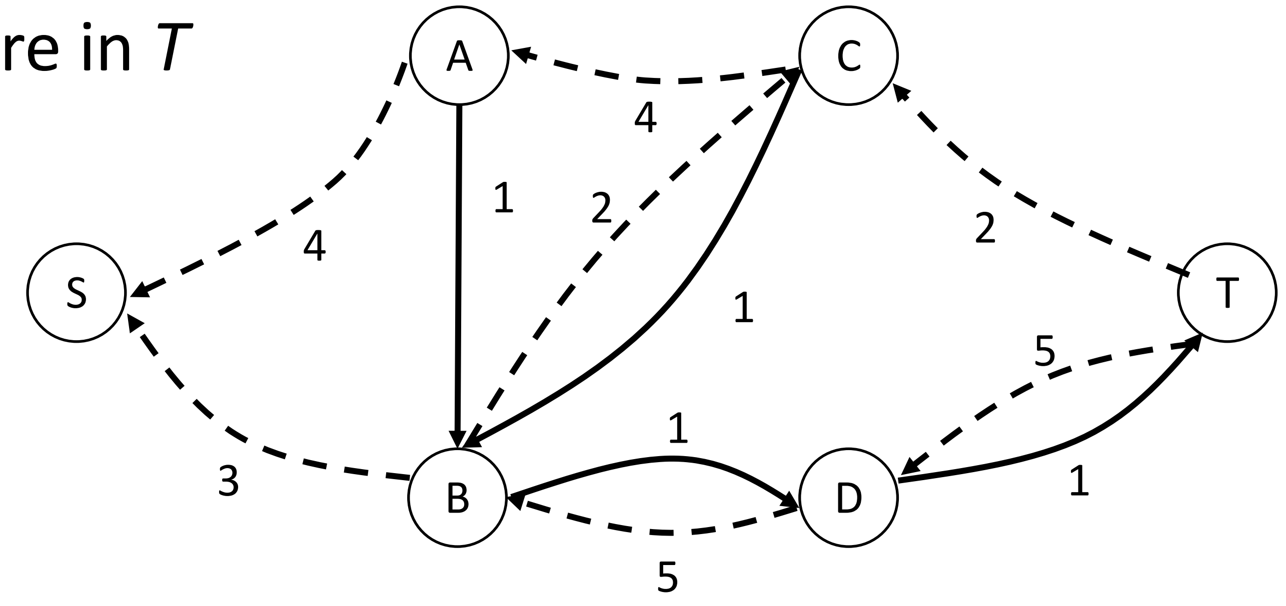


Max Flow Min Cut Theorem

- We know the *value* of the min cut, but what nodes are involved

- Use the residual graph of the maximum flow.
- Recursively label any node that can be reached from *s*
- Any labelled node is in *S,* the rest are in *T*

- (Think about labelling from *T,* traversing the arcs in *reverse,* i.e., from head to tail)

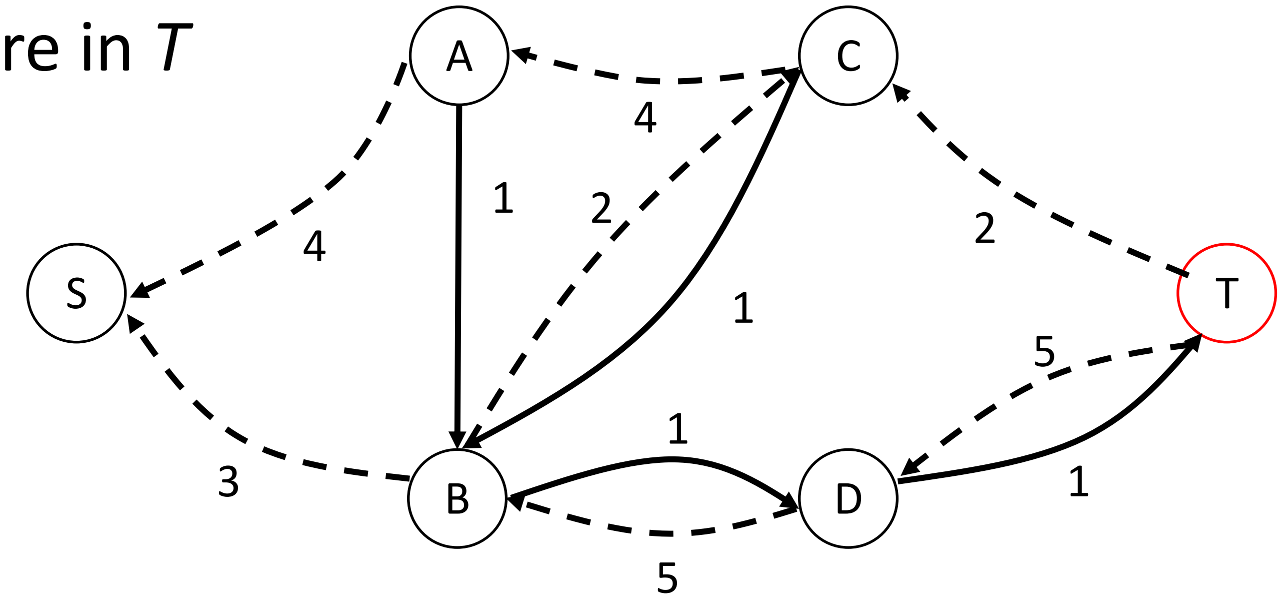


Max Flow Min Cut Theorem

- We know the *value* of the min cut, but what nodes are involved

- Use the residual graph of the maximum flow.
- Recursively label any node that can be reached from *s*
- Any labelled node is in *S,* the rest are in *T*

- (Think about labelling from *T,* traversing the arcs in *reverse,* i.e., from head to tail)

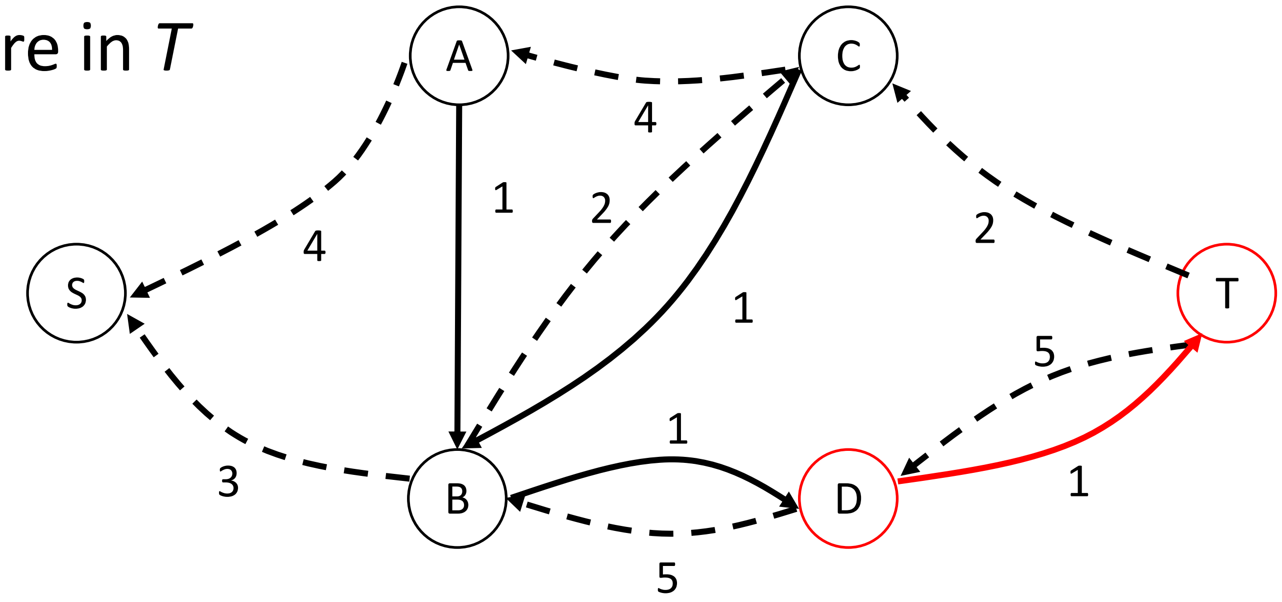


Max Flow Min Cut Theorem

- We know the *value* of the min cut, but what nodes are involved

- Use the residual graph of the maximum flow.
- Recursively label any node that can be reached from *s*
- Any labelled node is in *S,* the rest are in *T*

- (Think about labelling from *T,* traversing the arcs in *reverse,* i.e., from head to tail)

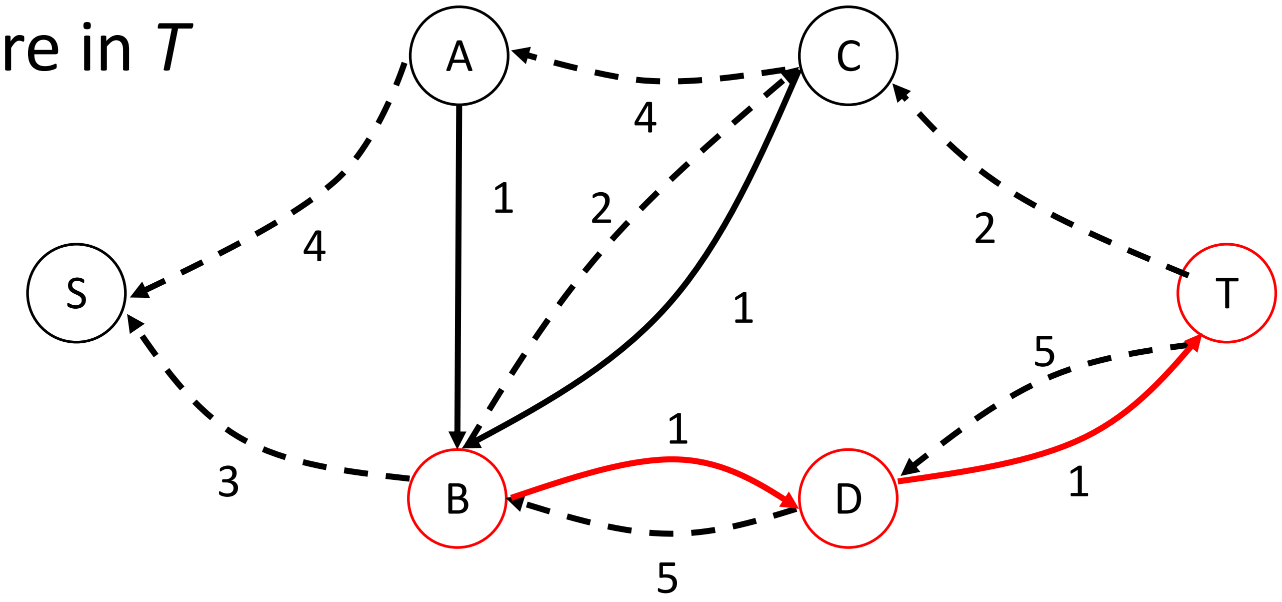


Max Flow Min Cut Theorem

- We know the *value* of the min cut, but what nodes are involved

- Use the residual graph of the maximum flow.
- Recursively label any node that can be reached from *s*
- Any labelled node is in *S,* the rest are in *T*

- (Think about labelling from *T,* traversing the arcs in *reverse,* i.e., from head to tail)

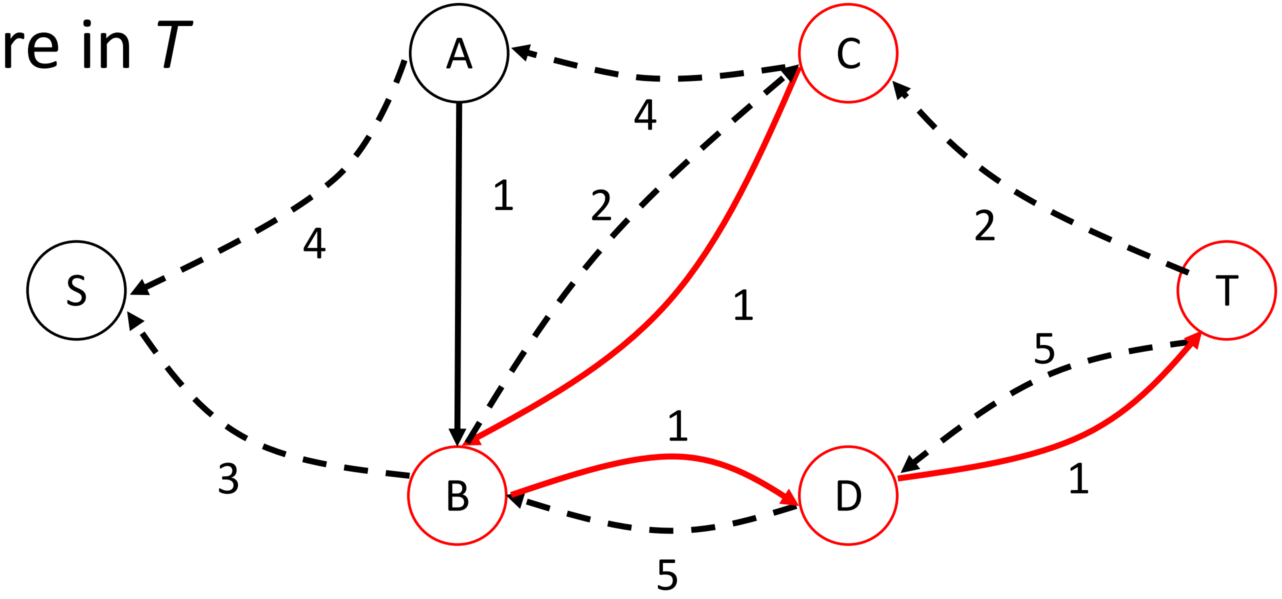


Max Flow Min Cut Theorem

- We know the *value* of the min cut, but what nodes are involved

- Use the residual graph of the maximum flow.
- Recursively label any node that can be reached from *s*
- Any labelled node is in *S,* the rest are in *T*

- (Think about labelling from *T,* traversing the arcs in *reverse,* i.e., from head to tail)

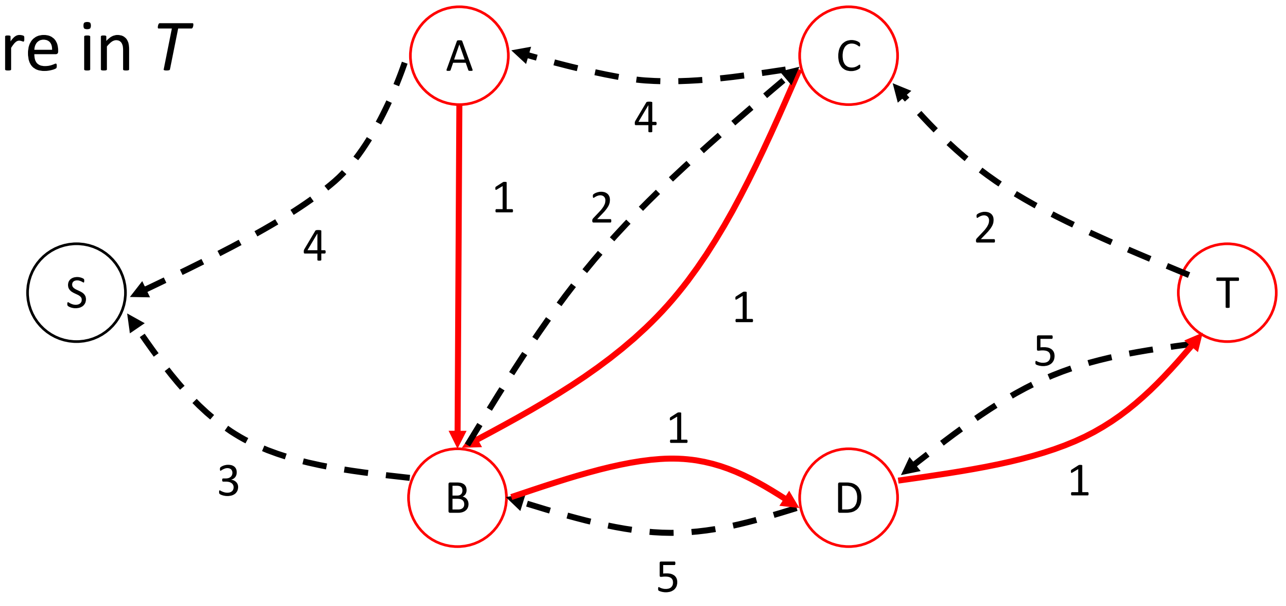


Max Flow Min Cut Theorem

- We know the *value* of the min cut, but what nodes are involved

- Use the residual graph of the maximum flow.
- Recursively label any node that can be reached from *s*
- Any labelled node is in *S, the rest are in T*

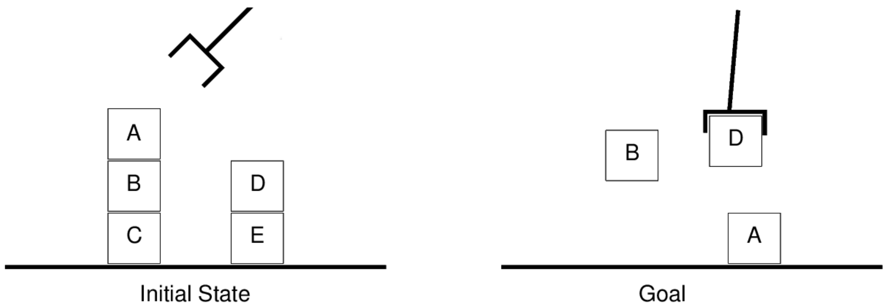
- (Think about labelling from *T*, traversing the arcs in *reverse*, i.e., from head to tail)



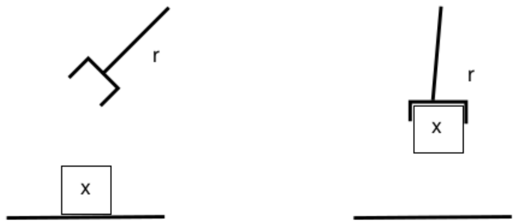
AI Planning as Network Flow

Recap: AI Planning

- AI Planning:
  - generalization of path planning to arbitrary graph
  - search + knowledge representation and reasoning (KRR)
- Example: Blocks World



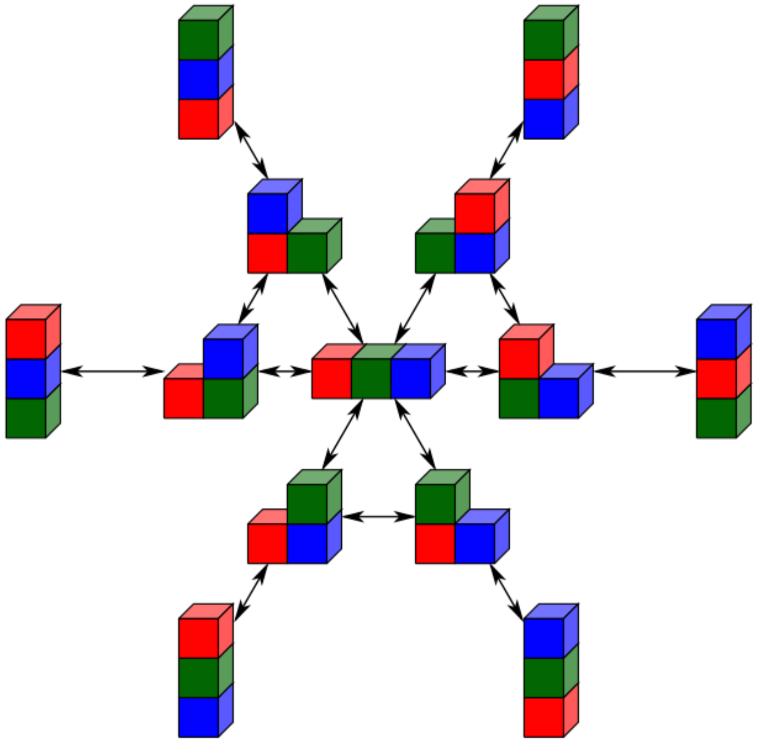

```
(define (domain blocksworld)
  (:requirements :strips :typing)
  (:types robot block)

(:predicates (clear ?x - block)
             (on-table ?x - block )
             (handempty ?r - robot)
             (holding ?r - robot ?x - block)
             (on ?x ?y - block))

(:action pickup
  :parameters (?r - robot ?x - block)
  :precondition (and (clear ?x) (on-table ?x) (handempty ?r))
  :effect (and (holding ?r ?x) (not (clear ?x)) (not (on-table ?x))
               (not (handempty ?r))))

(:action putdown
  :parameters  (?r - robot ?x - block)
  :precondition (holding ?r ?x)
  :effect (and (clear ?x) (handempty ?r) (on-table ?x)
               (not (holding ?x))))
```



State Space Search for Planning

- Search on the graph of representing the states and actions
- **Search algorithms**: A* or Greedy Best First Search
- **Heuristics**: automatically derived from problem description
  - Goal counting: number of propositions in the goal that are currently false
  - in the example: goal = (on blue red) (on red green)

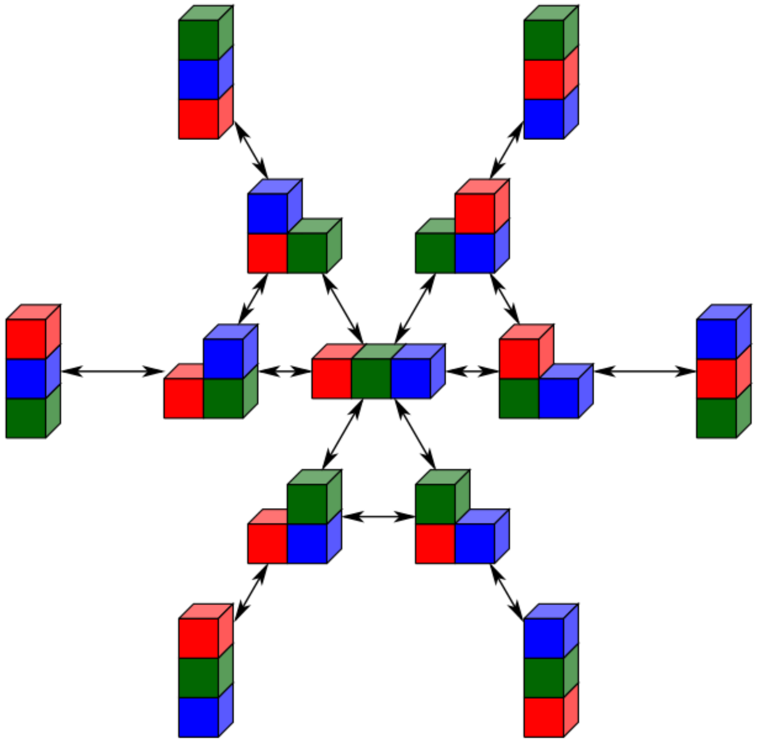
Complete State Space for Blocks World with 3 blocks



Network Flow for Planning: idea

- Initial state: source of the network
  - s0 = (on red blue) (on blue green) (on green table)
- Goal set: multiple sinks of the network
  - G = (on blue red)
- Capacities: 1 for all actions
- Route 1 unit of flow from source to sink
- Minimize the number of actions (pipes) used
  - More generally, each action as a cost C(s,a) and we minimize the total cost

Complete State Space for Blocks World with 3 blocks



Network Flow for Planning

- $x_{s,a}$: action **a** is applied in state **s**

$$\underbrace{\min_x \sum_{\substack{s \in S \\ a \in A(s)}} x_{s,a} C(s,a)}_{\text{Total cost of the solution}}$$

$$\text{s.t.} \quad x_{s,a} = \{0,1\}$$

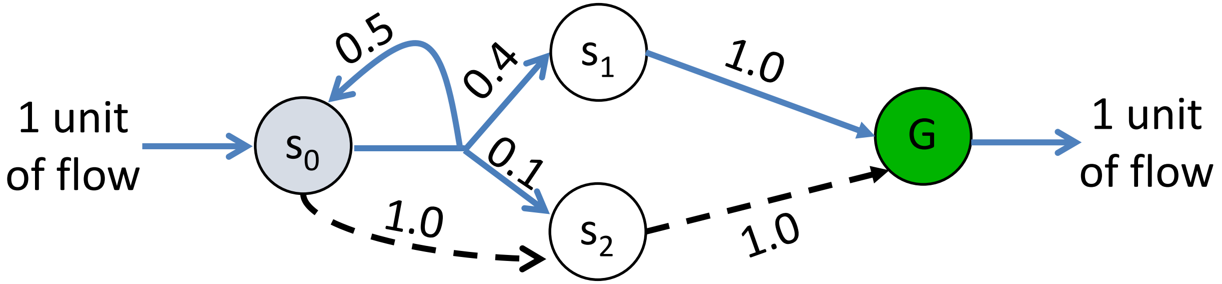
$$\underbrace{\underbrace{\sum_{a \in A(s)} x_{s,a}}_{\text{outflow}} - \underbrace{\sum_{\substack{s' \in S \\ a \in A(s')}} x_{s',a}}_{\text{inflow}} = \begin{cases} 1 & s = s_0 \\ 0 & \forall s \in S \setminus \{s_G, s_0\} \end{cases}}_{\text{Flow conservation for state s}}$$

$$\underbrace{\sum_{\substack{s,a \ s.t. \\ succ(s,a) = s_G}} x_{s,a} = 1}_{\text{Sink}}$$

- Is this a good planning algorithm?

Adding Probabilities

- What if the result of an action is given by a probability distribution?



- Intuition: **"probabilistic" flow** problem
- $x_{s,a}$: **expected** number of times action **a** is applied in state **s**

New Flow conservation for state s

$$\sum_{a \in A(s)} x_{s,a} - \sum_{\substack{s' \in S \\ a \in A(s')}} x_{s',a} P(s|s',a) = \begin{cases} 1 & s = s_0 \\ 0 & \forall s \in S \setminus \{s_G, s_0\} \end{cases}$$

New Sink Constraint

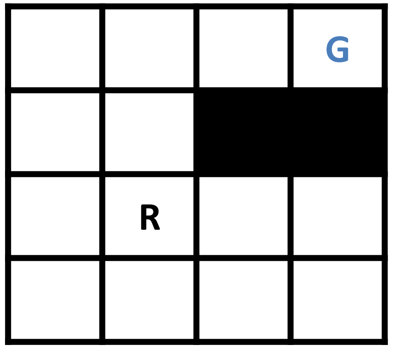
$$\sum_{\substack{s' \in S \\ a \in A(s')}} x_{s',a} P(s_G|s',a) = 1$$

- Still not state of the art: this class of problems (Stochastic Shortest Path problems) can be solved with dynamic programming

Adding Constraints

- The flow formulation let us express secondary constraints
- Example: navigation
  - actions: {North, South, East, West} x {slow, normal, fast}
  - minimize time
  - upper bound (the expected) fuel usage

Action	Pr. North	Pr. Stay	$C_0(s,a)$ Time Cost	$C_1(s,a)$ Fuel Cost
move-north-slow	0.99	0.01	4	2
move-north-normal	0.95	0.05	2	4
move-north-fast	0.90	0.10	1	10



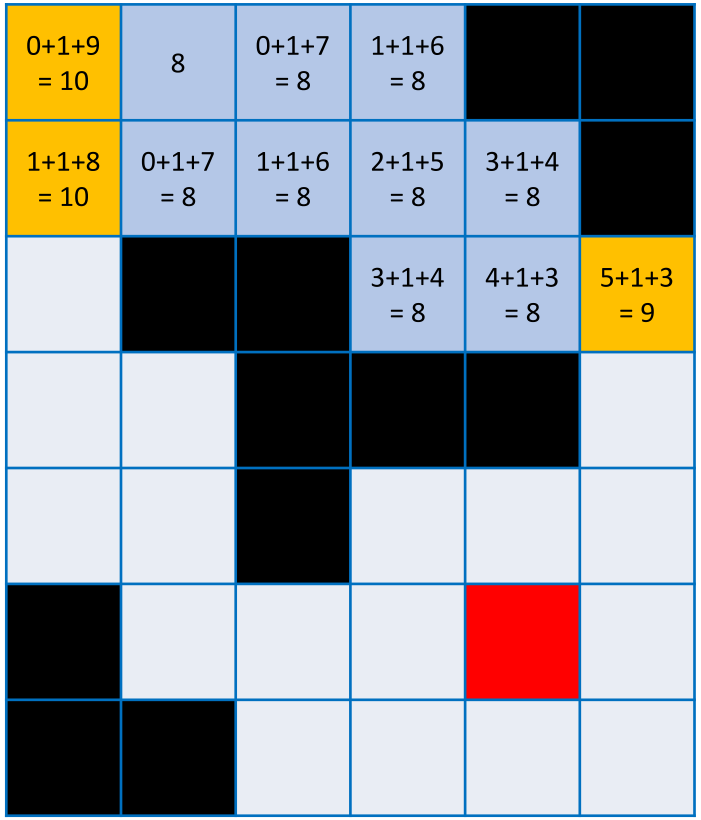
Constraint: Expected fuel used from s_0 to G ≤ 9, i.e., $E[C_1|\pi,s_0] \le 9$

General constraint: $$\sum_{\substack{s \in S \\ a \in A(s)}} x_{s,a} C_i(s,a) \le u_i$$

Can we use A* to solve this class of problems?

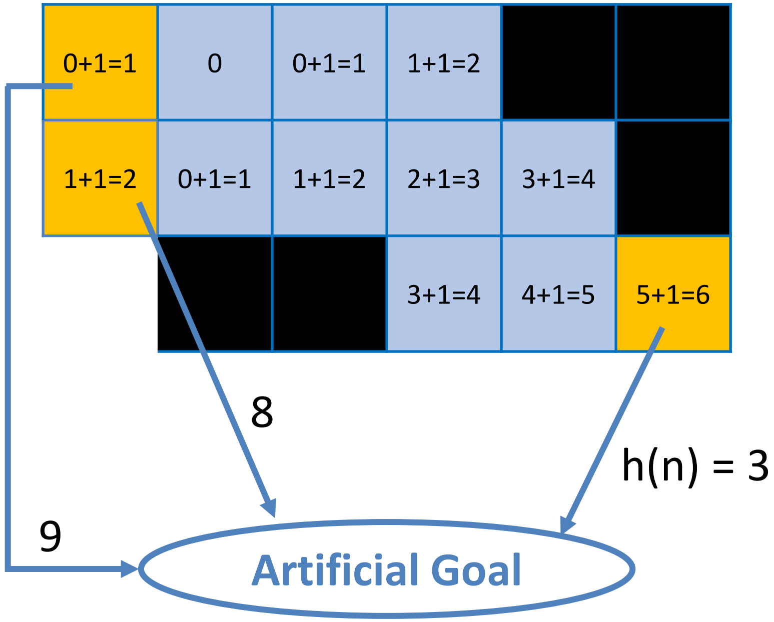
- No because of probabilities and constraint
- However, we can still apply the same principle

Iteration of A*: g(n’) + c(n’,a) + h(n) = f(n)



alternative view of A*

g(n’) + c(n’,a) = g(n)



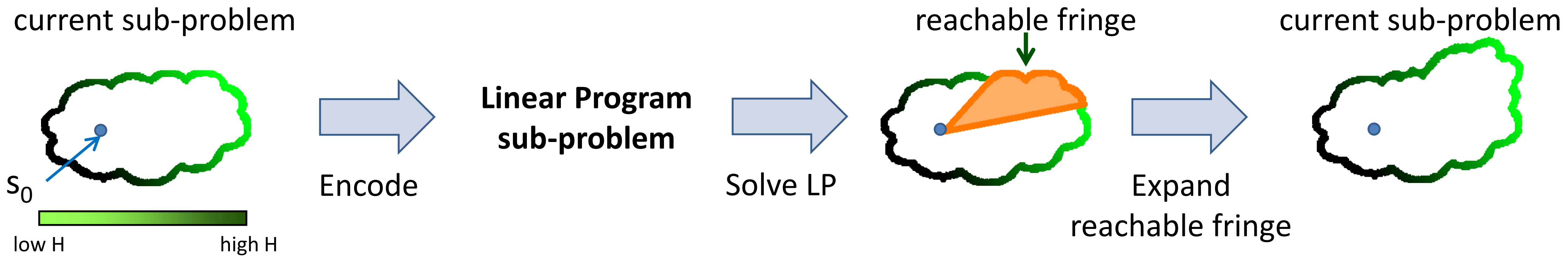
How to generalize?

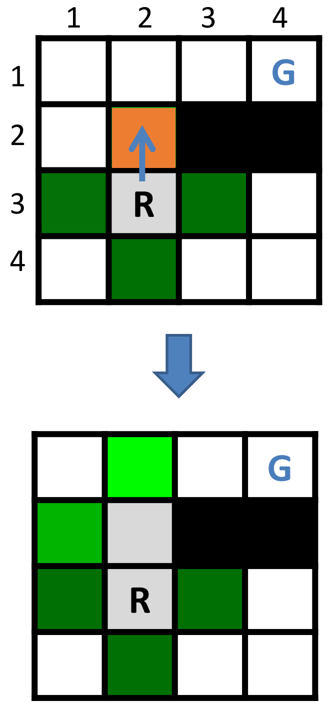
- Solve sub-problem using previous LP
- **Artificial goal** cost:
  - $C_i(s,a) = \underbrace{h_i(s)}$

One heurist for each cost function, e.g., $h_{TIME}(s)$ and $h_{FUEL}(s)$

i-dual

- Generalization of A* for probabilistic problems with constraints from 2016





Heuristic values

	(1,3)	(2,4)	(3,3)	(2,2)
time (H_T)	5	5	5	3
fuel (H_F)	10	10	10	6

Action costs

	time (C_T)	fuel (C_F)
move-up-slow	4	2
move-up-normal	2	4
move-up-fast	1	10

Cost constraints: $E[C_F|\pi,s_0] \leq 9$

1st iteration

- **Derived constraints:**
  - (1,3), (2,4), (3,3) cannot be reached with prob. ≥ 0.9
  - move-up-fast cannot be executed with prob. ≥ 0.9
- **Optimal solution**:
  - $x(s_0, \text{move-up-fast}) \approx 0.84$
  - $x(s_0, \text{move-up-normal}) \approx 0.16$

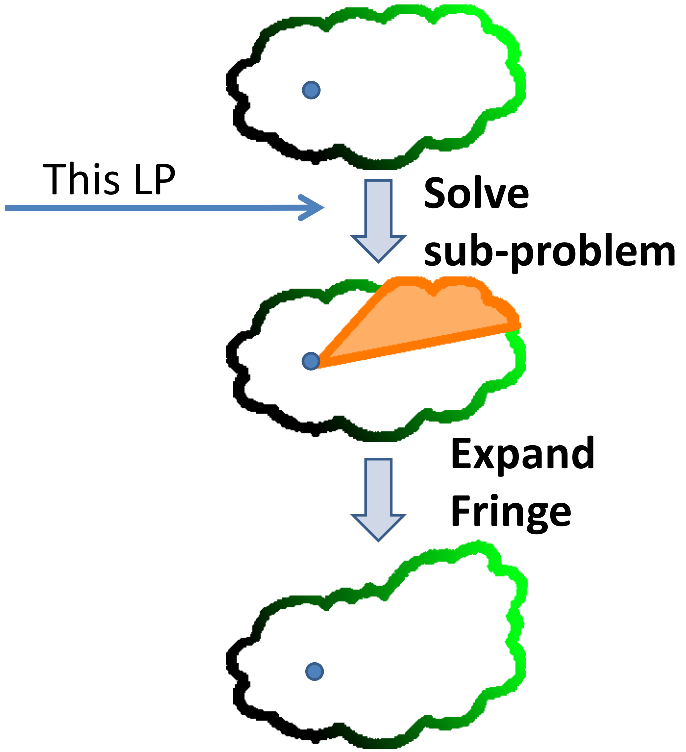
LP solved by i-dual

- At each iteration of i-dual:
  - Ŝ: subset of S explored so far
  - F: fringe of the search

$$
\begin{aligned}
\min_{x} \quad & \sum_{\substack{s \in \hat{S} \\ a \in A(s)}} x_{s,a} C(s,a) + \boxed{\sum_{s \in F} \text{inflow}_s H_0(s)} \\
\text{s.t.} \quad & x_{s,a} \geq 0 \\
& \sum_{a \in A(s)} x_{s,a} - \sum_{\substack{s' \in \hat{S} \setminus F \\ a \in A(s')}} x_{s',a} P(s|s',a) = \begin{cases} 1 & s = s_0 \\ 0 & \forall s \in \hat{S} \setminus (F \cup \{s_G, s_0\}) \end{cases} \\
& \sum_{s \in F \cup s_G} \sum_{\substack{s' \in \hat{S} \setminus F \\ a \in A(s')}} x_{s',a} P(s|s',a) = 1 \\
& \sum_{\substack{s \in \hat{S} \\ a \in A(s)}} x_{s,a} C_i(s,a) + \sum_{s \in F} \text{inflow}_s H_i(s) \leq u_i
\end{aligned}
$$

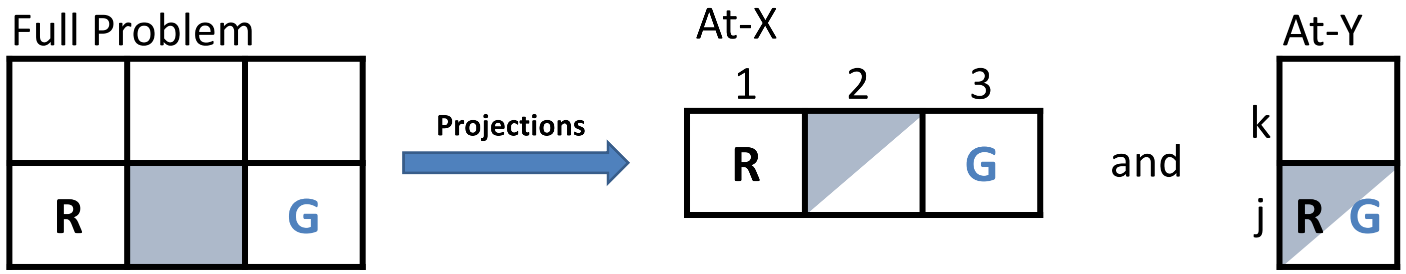
lower bound on expected cost from the fringe

Sink: goal U Fringes

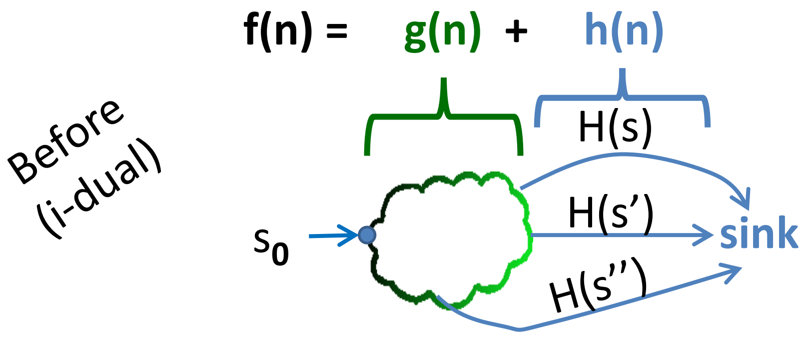


Heuristics as Network Flows

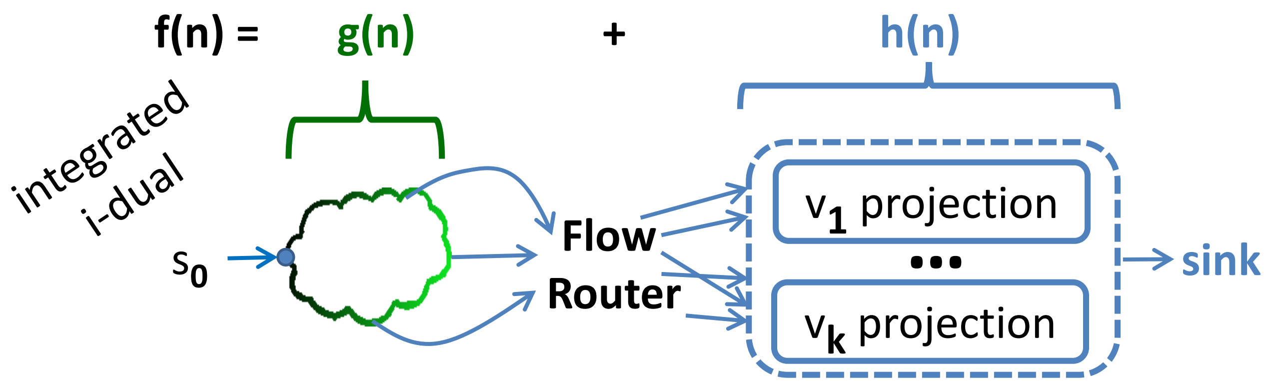
- What if we can represent the heuristic computation as a network flow problem too?
  - For instance, projections:



- Compute both g(n) and h(n) in the same LP while enforcing the constraints



Partial problem is encoded as an LP



Partial problem and heuristic encoded are encoded as a **single LP**

Summary

- Looked at flows in a network
- Defined the residual graph of a flow
- Used this to define the Ford Fulkerson algorithm
- Looked at multi-source/multi-destination network flow
- Related this back to the Assignment Problem
- Looked at multi-commodity flow
- Looked at Min Cut Max Flow theorem
- Looked at how to define the min cut given the max flow
- Looked at AI planning as network flow and how to model probabilities and constraints