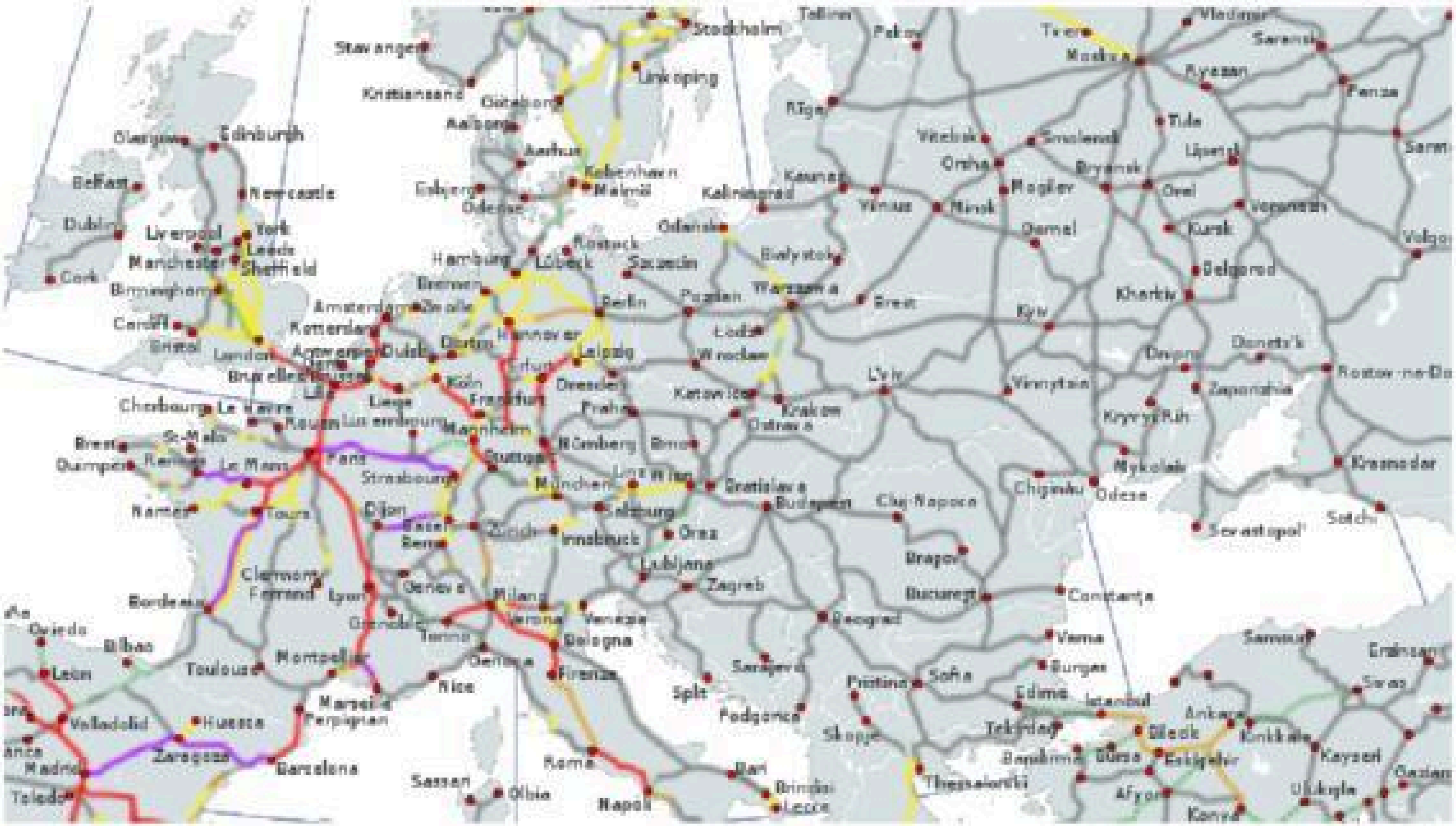
Algorithm II

7. Network Flow I

WU Xiaokun 吴晓堃

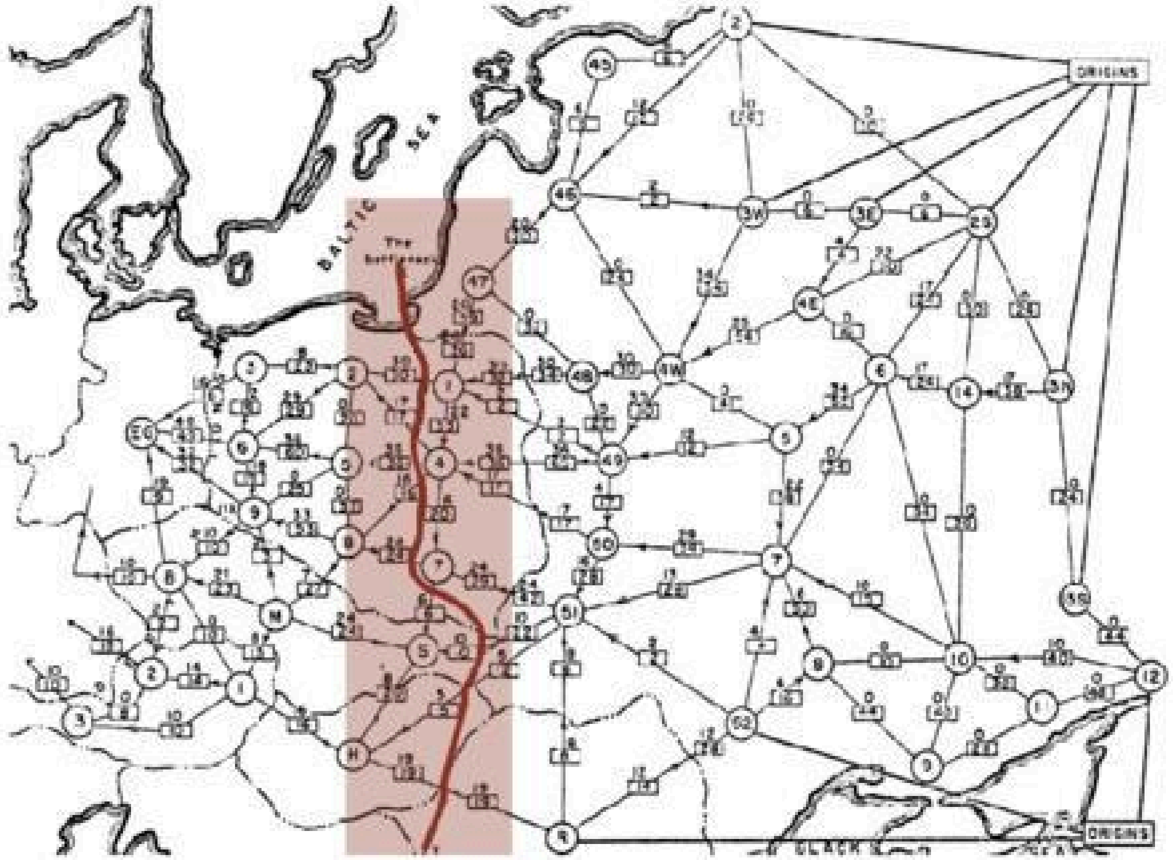
xkun.wu [at] gmail

Flow and Cut



Stockholm
Stavanger
Kristiansand
Linköping
Riga
Edinburgh
Glasgow
Belfast
Newcastle
Dublin
Liverpool
York
Leeds
Manchester
Sheffield
Cork
Cardiff
Bristol
London
Hamburg
Lübeck
Rostock
Szczecin
Berlin
Poznan
Bremen
Amsterdam
Rotterdam
Hannover
Leipzig
Dresden
Köln
Frankfurt
Praha
Katowice
Krakow
Ostrava
Cherbourg
Le Havre
Rouen
Brest
Rennes
Le Mans
Paris
Strasbourg
Stuttgart
Mannheim
München
Salzburg
Innsbruck
Wien
Bratislava
Budapest
Graz
Ljubljana
Zagreb
Nantes
Tours
Dijon
Basel
Bern
Zürich
Genève
Lyon
Grenoble
Milano
Verona
Venezia
Bologna
Torino
Genova
Firenze
Bordeaux
Toulouse
Montpellier
Marseille
Perpignan
Nice
Oviedo
Bilbao
Leon
Valladolid
Huesca
Zaragoza
Barcelona
Madrid
Toledo
Roma
Napoli
Bari
Brindisi
Lecce
Sassari
Olbia
Split
Sarajevo
Podgorica
Skopje
Beograd
Pristina
Sofia
Thessaloniki
Bucuresti
Brasov
Cluj-Napoca
Constanta
Varna
Burgas
Edirne
Istanbul
Bursa
Ankara
Konya
Afyon
Kayseri
Sivas
Samsun
Kharkiv
Kyiv
L'viv
Vinnytsia
Chişinău
Odesa
Mykolaiv
Sevastopol'
Krasnodar
Sochi
Rostov-na-Donu
Donets'k
Zaporizhia
Moskva
Tver
Smolensk
Bryansk
Tula
Ryazan
Kursk
Belgorod
Orel
Voronezh
Penza
Volgograd
Minsk
Vilnius
Kaunas
Kaliningrad
Gdansk
Bialystok
Warszawa
Lodz
Brest
Tallinn
Pskov
Vitebsk
Orsha
Mogilev
Gomel

Flow and Cut - supply



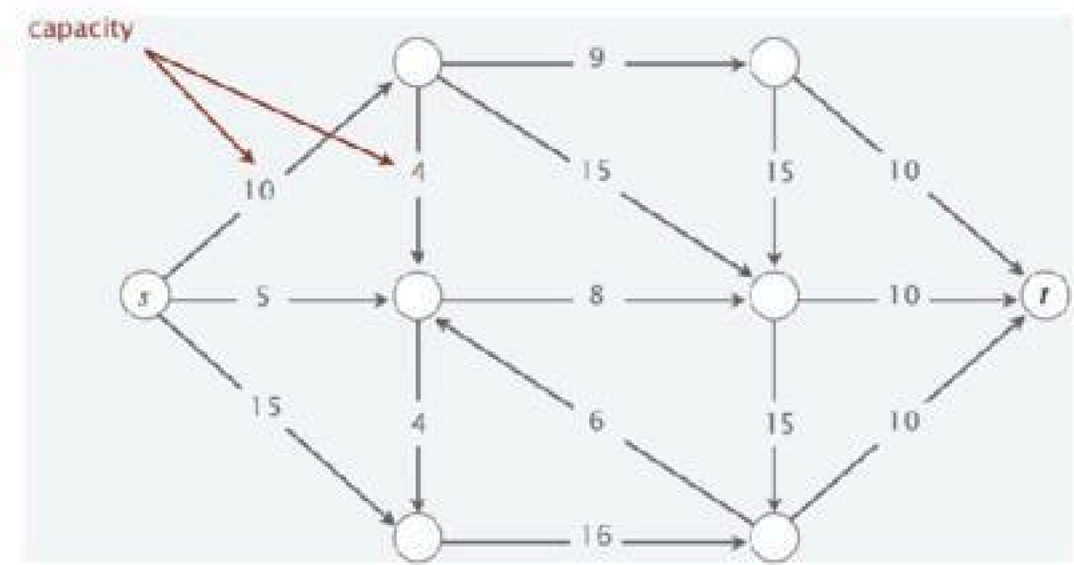
Max-flow and min-cut problems

Flow network

A **flow network** is a tuple $G = (V, E, s, t, c)$.

- Digraph (V, E) with source $s \in V$ and sink $t \in V$.
- Capacity $c(e) \geq 0$ for each $e \in E$.

Intuition. Material flowing through a transportation network; material originates at source and is sent to sink.

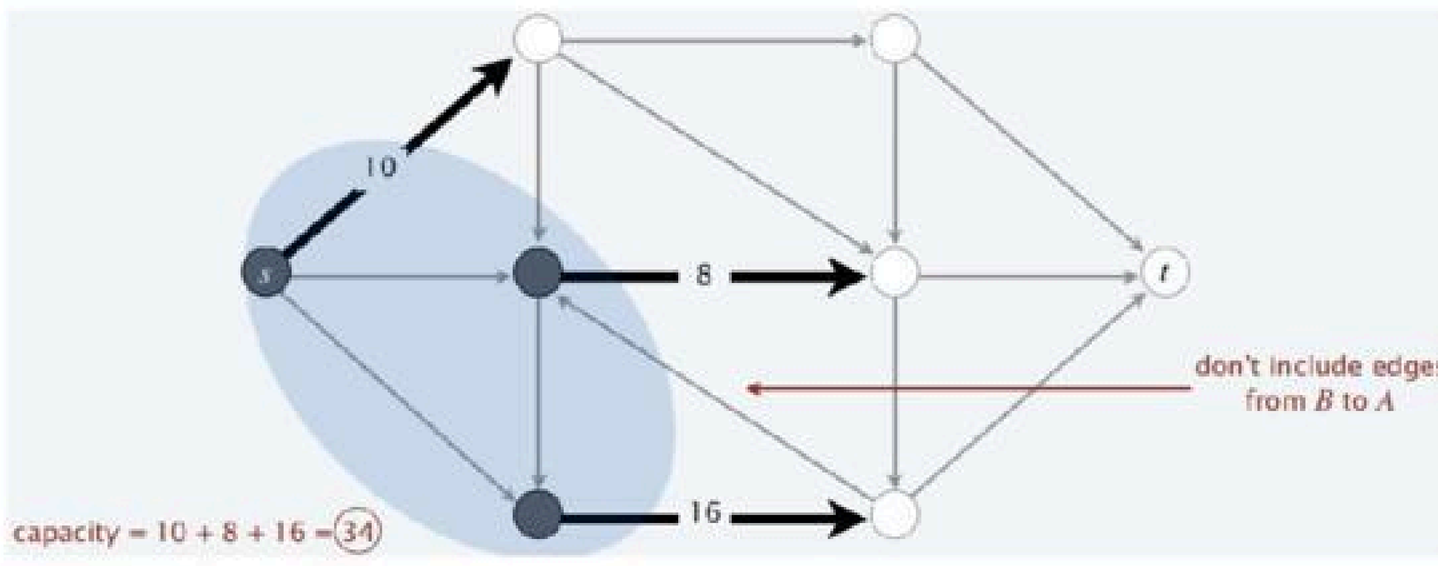


Minimum-cut problem

Def. An **st-cut (cut)** is a partition (A, B) of the nodes with $s \in A$ and $t \in B$.

Def. Its capacity is the sum of the capacities of the edges from A to B.

- $cap(A, B) = \sum_{e \text{ out } A} c(e)$

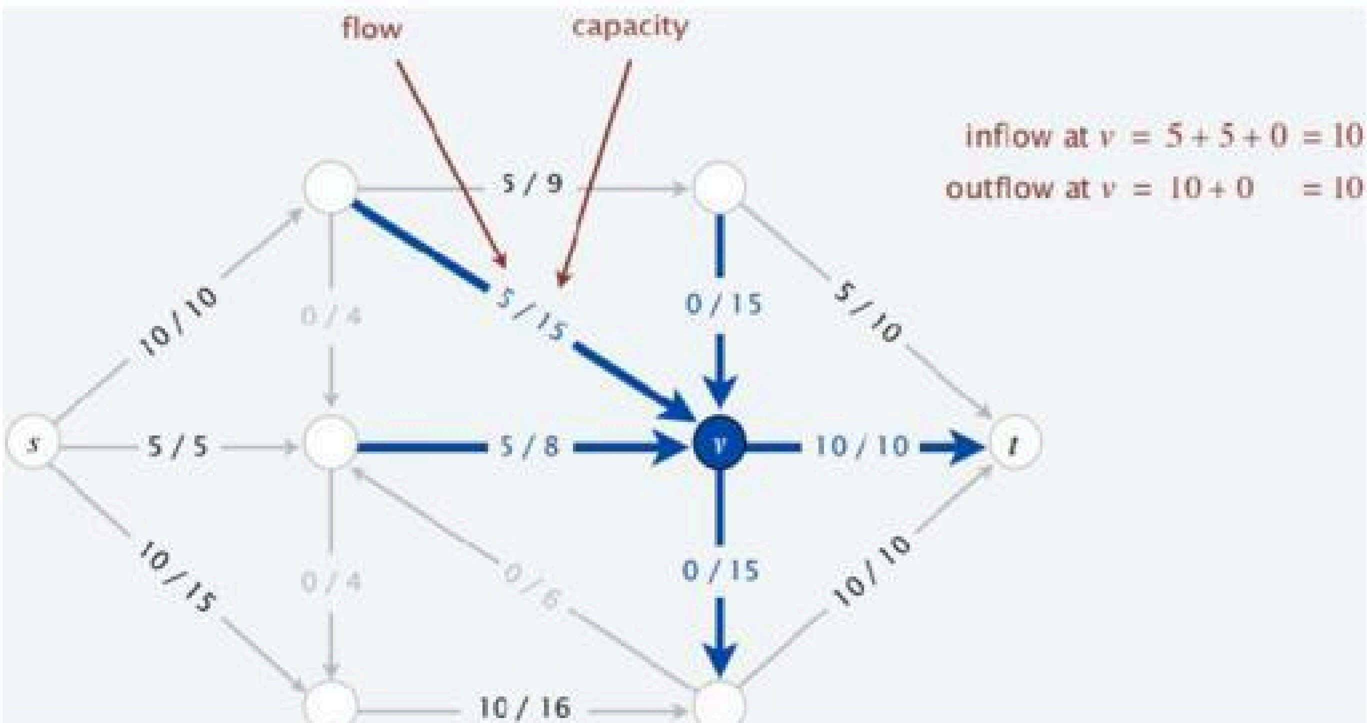


Min-cut problem. Find a cut of minimum capacity.

Maximum-flow problem

Def. An **st-flow (flow)** f is a function that satisfies:

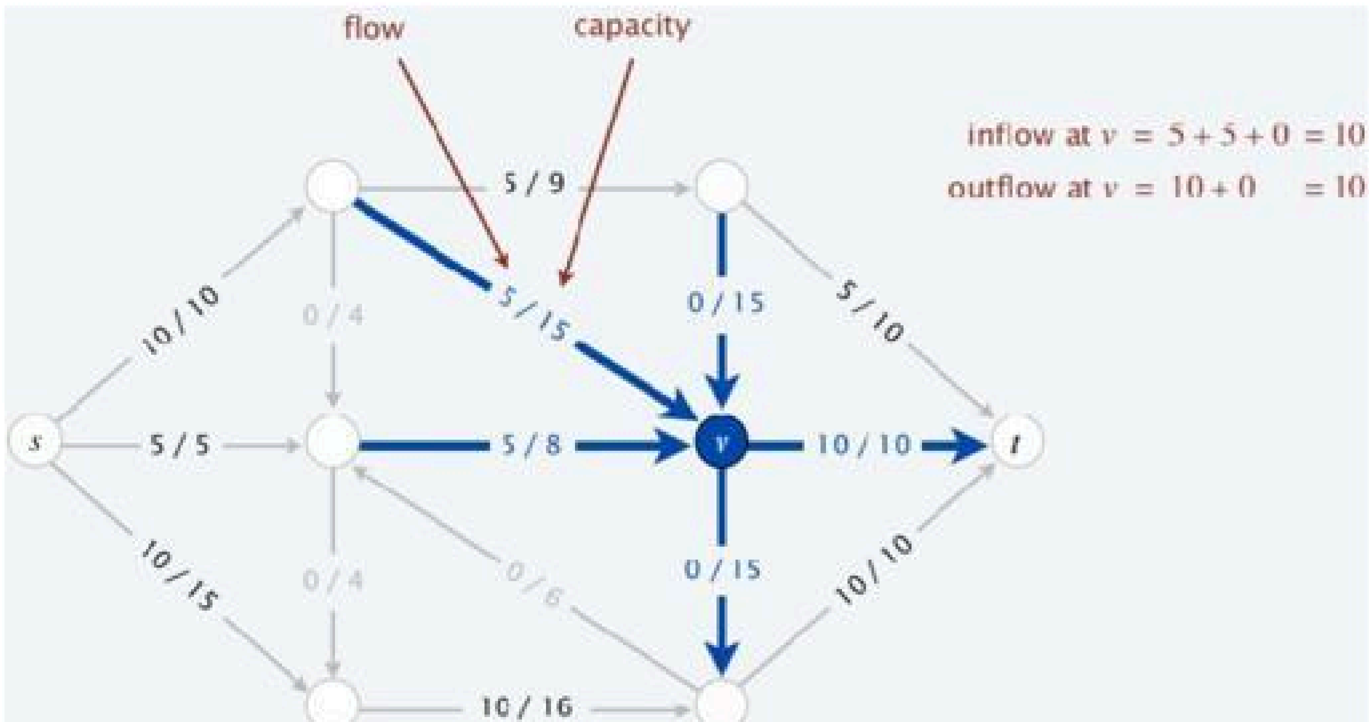
- [**capacity**] For each $e \in E$: $0 \leq f(e) \leq c(e)$
- [**flow conservation**] For each $v \in V - \{s, t\}$: $\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out } v} f(e)$



Maximum-flow problem (cont.)

Def. The **value** of a flow f is: $val(f) = \sum_{e \text{ out } s} f(e) - \sum_{e \text{ into } s} f(e)$

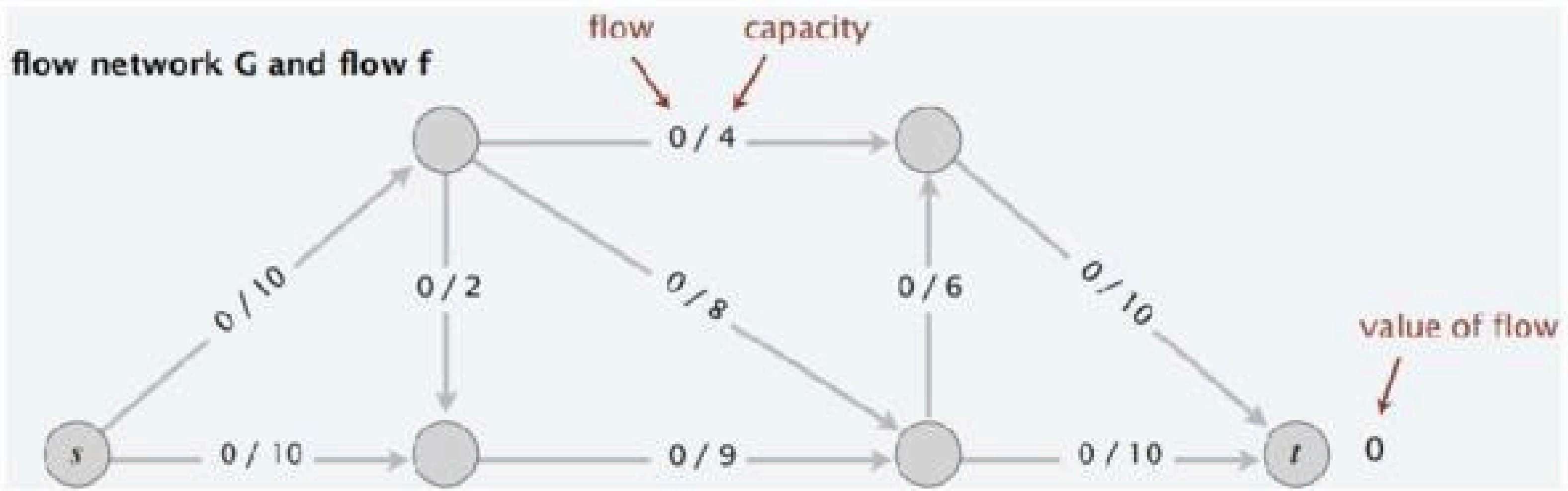
Max-flow problem. Find a flow of maximum value.



Ford-Fulkerson algorithm

Toward a max-flow algorithm

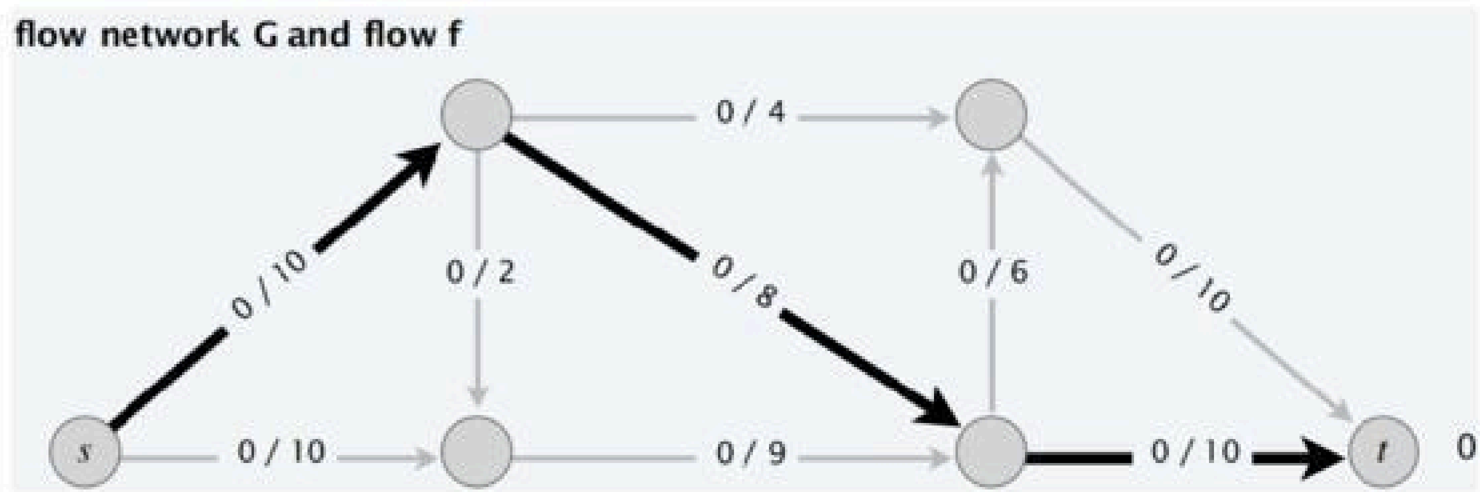
Greedy algorithm.



- Start with $f(e) = 0$ for each edge $e \in E$.

Toward a max-flow algorithm

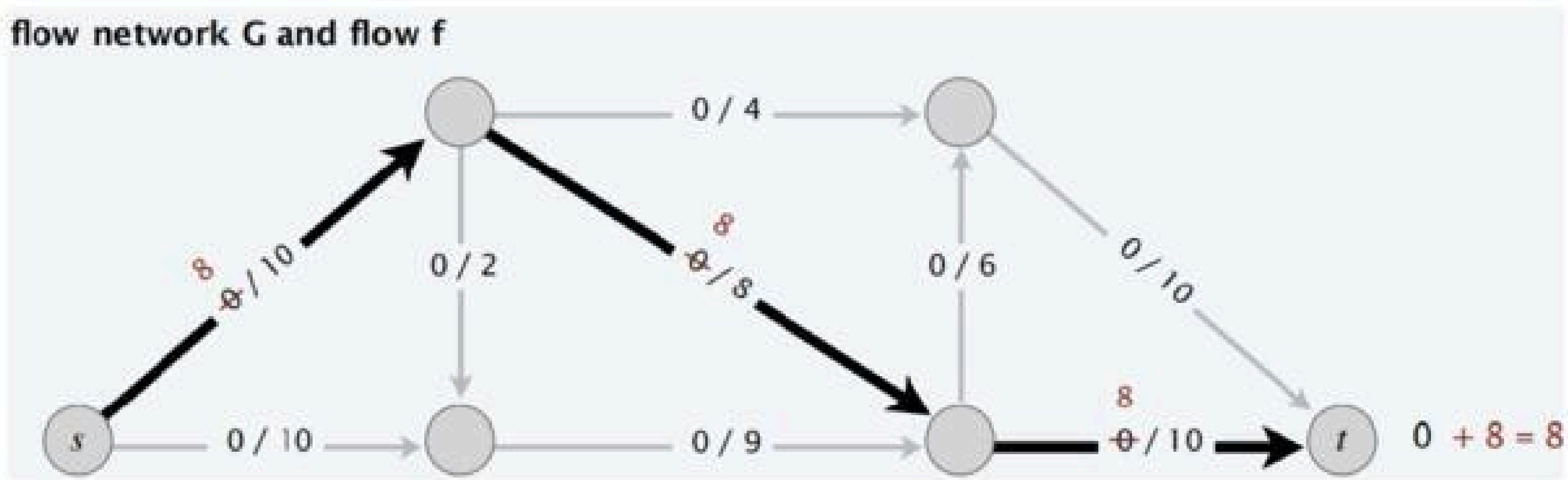
Greedy algorithm.



- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$.

Toward a max-flow algorithm

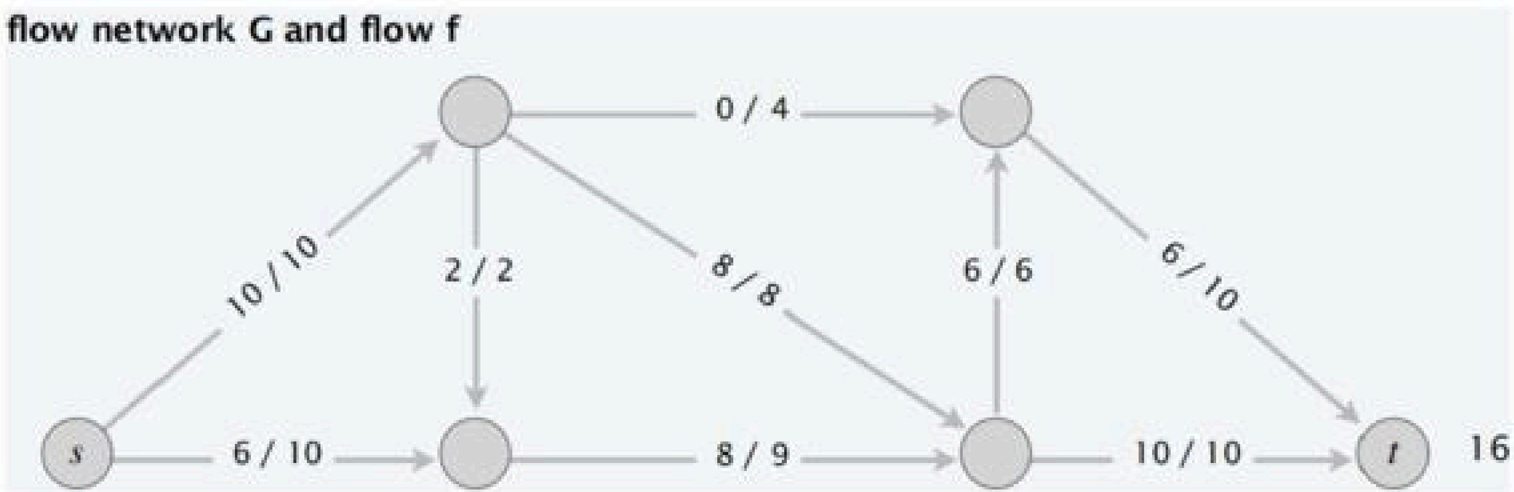
Greedy algorithm.



- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.

Toward a max-flow algorithm

Greedy algorithm.

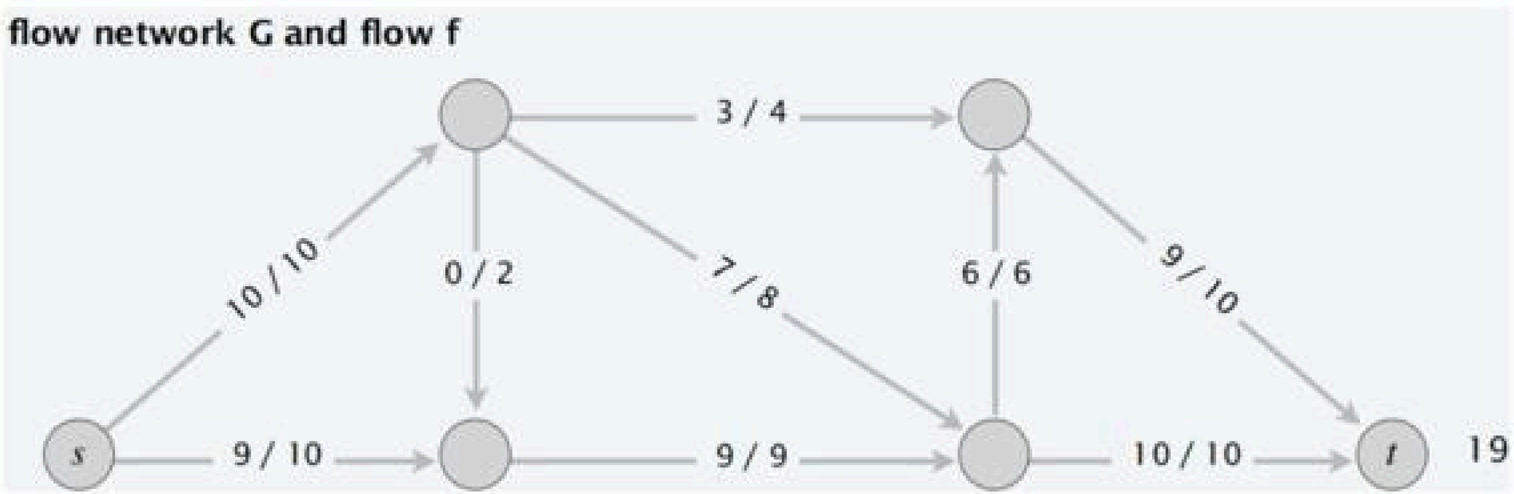


- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.

Ending flow value = 16

Toward a max-flow algorithm

Greedy algorithm.



- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.

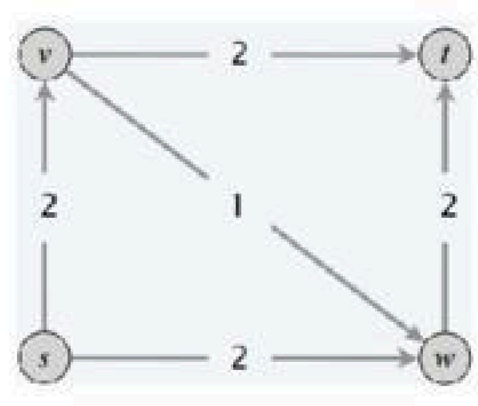
But max-flow value = 19

Why greedy algorithm fails

Q. Why does the greedy algorithm fail?
A. Once greedy algorithm increases flow on an edge, it never decreases it.

Ex. Consider flow network G.

- Max flow f^* has $f^*(v, w) = 0$.
- Greedy algorithm could choose $s \to v \to w \to t$ as first path.



Bottom line. Need some mechanism to “undo” a bad decision.

Residual network

Original edge. $e = (u, v) \in E$.

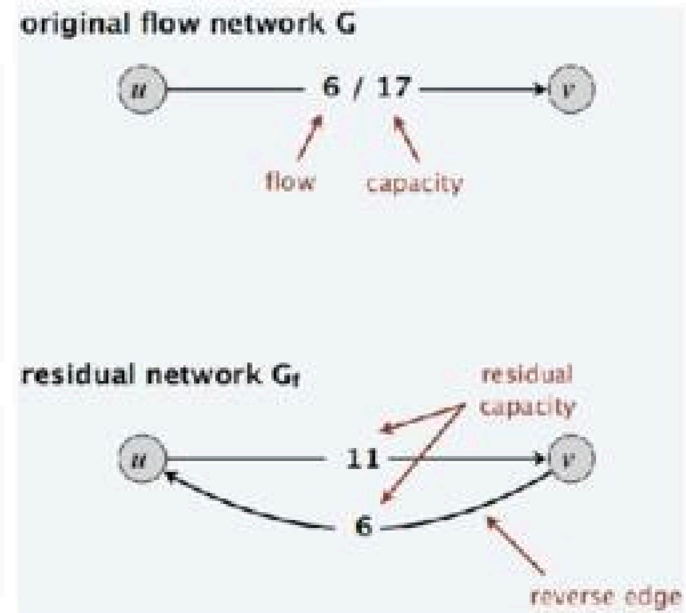
- Flow $f(e)$; Capacity $c(e)$.

Reverse edge. $e^{-1} = (v, u)$.

- "Undo" flow sent.

Residual capacity.

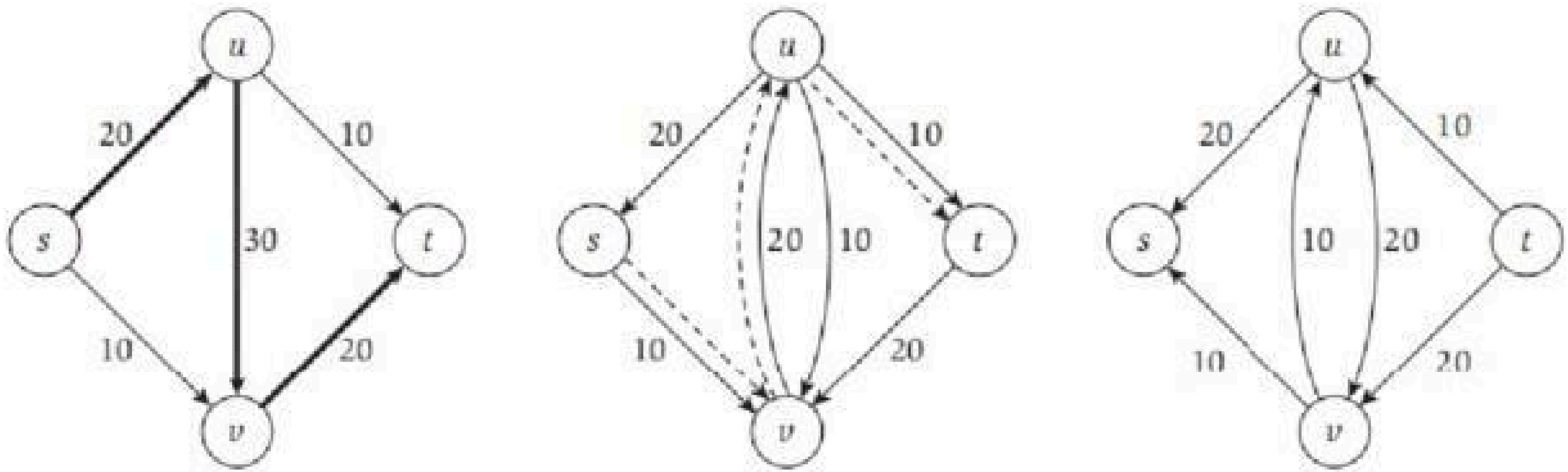
$$c_f(e) = \begin{cases} c(e) - f(e) & \text{if} \quad e \in E \\ f(e) & \text{if} \quad e^{-1} \notin E \end{cases}$$



Residual network (cont.)

Residual network. $G_f = (V, E_f, s, t, c_f)$.

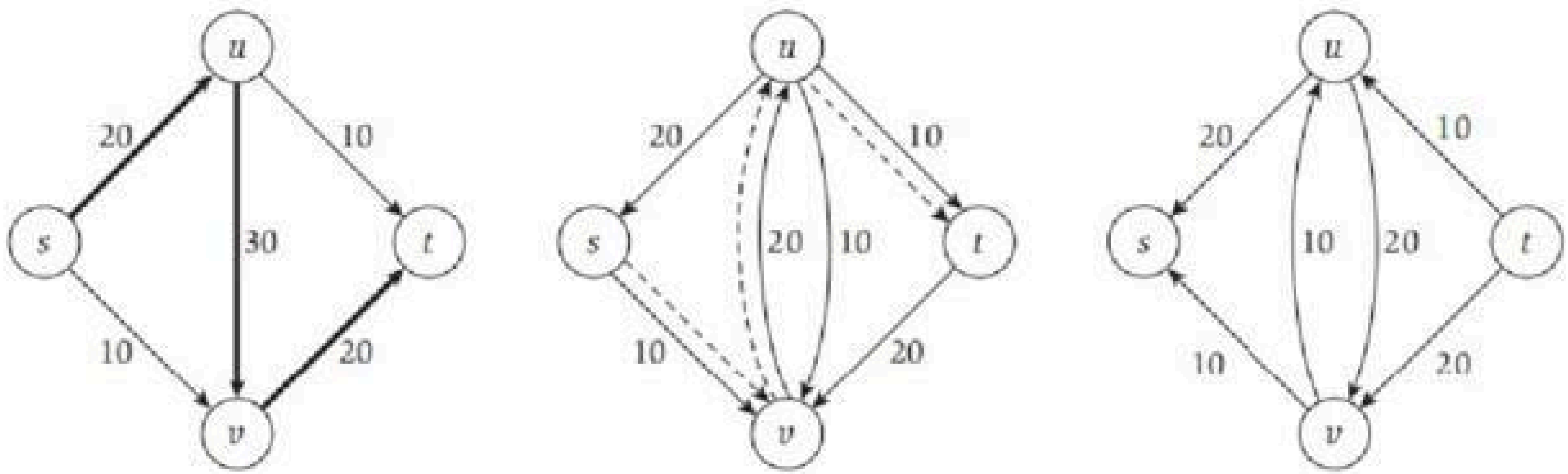
- $E_f = \{e : f(e) < c(e)\} \cup \{e^{-1} : f(e) > 0\}$.
- **Key property**: f' is a flow in G_f iff $f + f'$ is a flow in G.



Augmenting path

Def. An **augmenting path** is a simple $s \rightsquigarrow t$ path in the residual network G_f.

Def. The **bottleneck capacity** of an augmenting path P is the *minimum residual capacity* of any edge in P.



Key property. Let f be a flow and let P be an augmenting path in G_f. Then, after calling f' = AUGMENT(f, c, P), the resulting f' is a flow and $val(f') = val(f) + bottleneck(G_f, P)$.

Augmenting path: algorithm

Key property. Let f be a flow and let P be an augmenting path in G_f. Then, after calling f' = $\texttt{AUGMENT}(f, c, P)$, the resulting f' is a flow and $val(f') = val(f) + bottleneck(G_f, P)$.

$\texttt{AUGMENT}(f, c, P)$

1. Δ = bottleneck capacity of augmenting path P.
2. FOREACH edge $e \in P$:
   1. IF $(e \in E)$ $f(e) = f(e) + \Delta$;
   2. ELSE $f(e^{-1}) = f(e^{-1}) - \Delta$;
3. RETURN f;

Ford-Fulkerson algorithm

Ford-Fulkerson augmenting path algorithm.

- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \leadsto t$ path P in the residual network G_f .
- Augment flow along path P.
- Repeat until you get stuck.

FORD-FULKERSON(G)

1. FOREACH edge $e \in E$: $f(e) = 0$;
2. G_f = residual network of G with respect to flow f;
3. WHILE (there exists an $s \leadsto t$ path P in G_f)
   1. f = AUGMENT(f, c, P);
   2. Update G_f;
4. RETURN f;

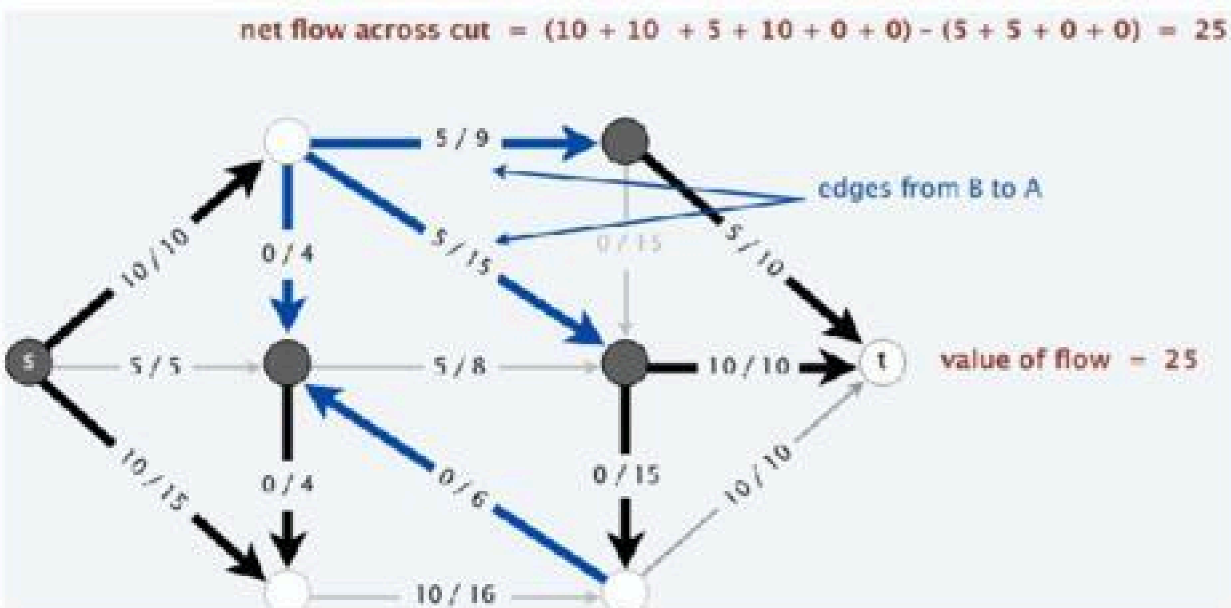
Demo: Ford-Fulkerson

Max-Flow Min-Cut Theorem

Flows and cuts: relationship

Flow value lemma. Let f be any flow and let (A, B) be any cut. Then, the value of the flow f equals the net flow across the cut (A, B).

$$val(f) = \sum_{e \text{ out } A} f(e) - \sum_{e \text{ into } A} f(e)$$



Flows and cuts: relationship (cont.)

Flow value lemma. Let f be any flow and let (A, B) be any cut. Then, the value of the flow f equals the net flow across the cut (A, B).

$$val(f) = \sum_{e \text{ out } A} f(e) - \sum_{e \text{ into } A} f(e)$$

Pf.

$$\begin{aligned} val(f) &= \sum_{e \text{ out } s} f(e) - \sum_{e \text{ into } s} f(e) \\ &= \sum_{v \in A} \Big(\sum_{e \text{ out } v} f(e) - \sum_{e \text{ into } v} f(e) \Big) \\ &= \sum_{e \text{ out } A} f(e) - \sum_{e \text{ into } A} f(e) \end{aligned}$$

Flows and cuts: duality

Weak duality. Let f be any flow and (A, B) be any cut. Then, $val(f) \leq cap(A, B)$.

Pf.

$$\begin{aligned} val(f) &= \sum_{e \text{ out } A} f(e) - \sum_{e \text{ into } A} f(e) \\ &\leq \sum_{e \text{ out } A} f(e) \\ &\leq \sum_{e \text{ out } A} c(e) \\ &= cap(A, B) \end{aligned}$$

Certificate of optimality

Corollary. Let f be a flow and let (A, B) be any cut. If $val(f) = cap(A, B)$, then f is a max flow and (A, B) is a min cut.

Pf.

- For any flow f': $val(f') \leq cap(A, B) = val(f)$.
- For any cut (A', B'): $cap(A', B') \geq val(f) = cap(A, B)$.

Max-flow min-cut theorem I

Max-flow min-cut theorem. [strong duality] Value of a max flow = capacity of a min cut.

Augmenting path theorem. A flow f is a max flow iff no augmenting paths.

Pf. The following three conditions are equivalent for any flow f:

- A. There exists a cut (A, B) such that $cap(A, B) = val(f)$.
- B. f is a max flow.
- C. There is no augmenting path with respect to f.
  - Or, if Ford-Fulkerson terminates, then f is max flow.

[$A \Rightarrow B$]

- Weak duality corollary.

Max-flow min-cut theorem II

Max-flow min-cut theorem. [strong duality] Value of a max flow = capacity of a min cut.

Augmenting path theorem. A flow f is a max flow iff no augmenting paths.

Pf. The following three conditions are equivalent for any flow f:

- A. There exists a cut (A, B) such that $cap(A, B) = val(f)$.
- B. f is a max flow.
- C. There is no augmenting path with respect to f.

[$B \Rightarrow C$] We prove contrapositive: $\neg C \Rightarrow \neg B$.

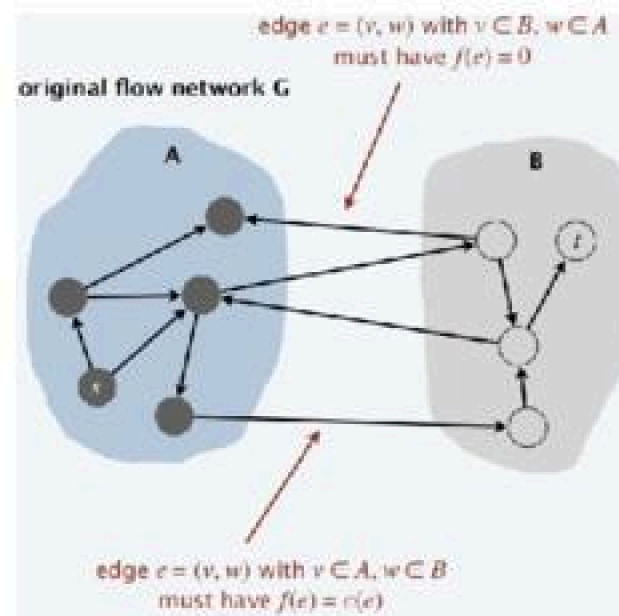
- Suppose that there is an augmenting path with respect to f.
- Can improve flow f by sending flow along this path.
- Thus, f is not a max flow, contradiction.

Max-flow min-cut theorem III

$[\, C \Rightarrow A \,]$

- Let f be a flow with no augmenting paths.
- Let A = set of nodes reachable from s in residual network G_f.
- By definition of A: $s \in A$.
- By definition of flow f: $t \notin A$.

$$
\begin{aligned}
val(f) &= \sum_{e \text{ out } A} f(e) - \sum_{e \text{ into } A} f(e) \\
&= \sum_{e \text{ out } A} c(e) - 0 \\
&= cap(A, B)
\end{aligned}
$$



Computing a minimum cut

Theorem. Given any max flow f, can compute a min cut (A, B) in $O(m)$ time.

Pf. Let A = set of nodes reachable from s in residual network G_f.

- argument from previous slide implies that capacity of (A, B) = value of flow f

Capacity-scaling algorithm

Ford-Fulkerson: analysis

Assumption. Every edge capacity $c(e)$ is an integer between 1 and C.

Integrality invariant. Throughout Ford-Fulkerson, every edge flow $f(e)$ and residual capacity $c_f(e)$ is an integer.
Pf. By induction on the number of augmenting paths.

Ford-Fulkerson: analysis

Assumption. Every edge capacity $c(e)$ is an integer between 1 and C.

Integrality invariant. Throughout Ford-Fulkerson, every edge flow $f(e)$ and residual capacity $c_f(e)$ is an integer.
Pf. By induction on the number of augmenting paths.

Theorem. Ford-Fulkerson terminates after at most $val(f^*) \leq nC$ augmenting paths, where f^* is a max flow.
Pf. Each augmentation increases the value of the flow by at least 1.

Corollary. The running time of Ford-Fulkerson is $O(mnC)$.
Pf. Can use either BFS or DFS to find an augmenting path in $O(m)$ time.

Ford-Fulkerson: analysis

Assumption. Every edge capacity $c(e)$ is an integer between 1 and C.

Integrality invariant. Throughout Ford-Fulkerson, every edge flow $f(e)$ and residual capacity $c_f(e)$ is an integer.
Pf. By induction on the number of augmenting paths.

Theorem. Ford-Fulkerson terminates after at most $val(f^*) \leq nC$ augmenting paths, where f^* is a max flow.
Pf. Each augmentation increases the value of the flow by at least 1.

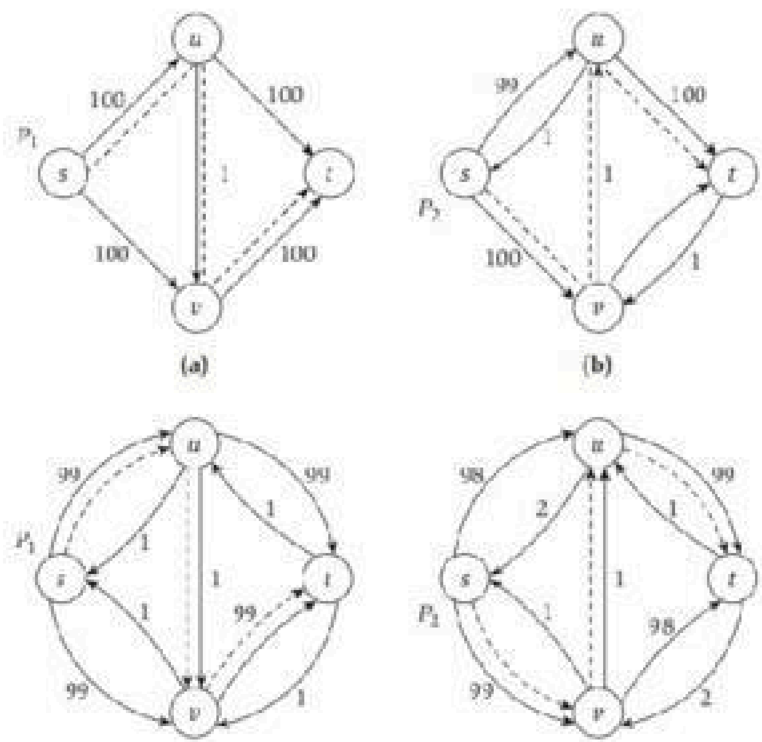
Corollary. The running time of Ford-Fulkerson is $O(mnC)$.
Pf. Can use either BFS or DFS to find an augmenting path in $O(m)$ time.

Integrality theorem. There exists an integral max flow f^*.
Pf. Since Ford-Fulkerson terminates, theorem follows from integrality invariant (and augmenting path theorem).

Ford-Fulkerson: exponential example

Q. Is generic Ford-Fulkerson algorithm poly-time in input size $(m, n, \log C)$?

A. No. If max capacity is C, then algorithm can take $\geq C$ iterations.



- See Demo.

Quiz: Ford-Fulkerson

The Ford-Fulkerson algorithm is guaranteed to terminate if the edge capacities are …

A. Rational numbers.
B. Real numbers.
C. Both A and B.
D. Neither A nor B.

Quiz: Ford-Fulkerson

The Ford-Fulkerson algorithm is guaranteed to terminate if the edge capacities are …

A. Rational numbers.
B. Real numbers.
C. Both A and B.
D. Neither A nor B.

Rational = integer / integer

Choosing good augmenting paths

Use care when selecting augmenting paths.

- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.

Pathology. When edge capacities can be irrational, no guarantee that Ford-Fulkerson terminates (or converges to a maximum flow)!

- See Demo.

Choosing good augmenting paths

Use care when selecting augmenting paths.

- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.

Pathology. When edge capacities can be irrational, no guarantee that Ford-Fulkerson terminates (or converges to a maximum flow)!

- See Demo.

Goal. Choose augmenting paths so that:

- Can find augmenting paths efficiently.
- Few iterations.

Choosing good augmenting paths (cont.)

Goal. Choose augmenting paths so that:

- Can find augmenting paths efficiently.
- Few iterations.

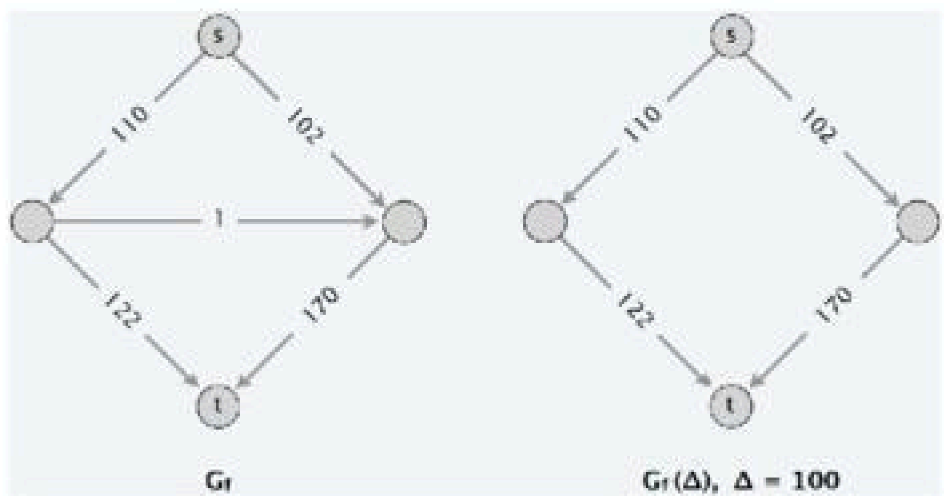
Choose augmenting paths with:

- Max bottleneck capacity ("fattest").
  - How to find?
  - [Next] Sufficiently large bottleneck capacity.
- [Ahead] Fewest edges.

Capacity-scaling

Overview. Choosing augmenting paths with "large" bottleneck capacity.

- Maintain scaling parameter Δ.
- Let $G_f(\Delta)$ be the part of the residual network containing only those edges with capacity $\geq \Delta$.
- Any augmenting path in $G_f(\Delta)$ has bottleneck capacity $\geq \Delta$.



Capacity-scaling: algorithm

CAPACITY-SCALING(G)

1. FOREACH edge $e \in E$: $f(e) = 0$;
2. Δ = largest power of $2 \leq C$;
3. WHILE ($\Delta \geq 1$)
   1. $G_f(\Delta)$ = Δ-residual network of G with respect to flow f;
   2. WHILE (there exists an $s \rightsquigarrow t$ path P in $G_f(\Delta)$)
      1. f = AUGMENT(f, c, P);
      2. Update $G_f(\Delta)$;
   3. $\Delta = \Delta / 2$;
4. RETURN f;

Capacity-scaling: correctness

Assumption. All edge capacities are integers between 1 and C.

Invariant. The scaling parameter Δ is a power of 2.
Pf. Initially a power of 2; each phase divides Δ by exactly 2.

Integrality invariant. Throughout the algorithm, every edge flow $f(e)$ and residual capacity $c_f(e)$ is an integer.
Pf. Same as for generic Ford-Fulkerson.

Capacity-scaling: correctness

Assumption. All edge capacities are integers between 1 and C.

Invariant. The scaling parameter Δ is a power of 2.
Pf. Initially a power of 2; each phase divides Δ by exactly 2.

Integrality invariant. Throughout the algorithm, every edge flow $f(e)$ and residual capacity $c_f(e)$ is an integer.
Pf. Same as for generic Ford-Fulkerson.

Theorem. If capacity-scaling algorithm terminates, then f is a max flow.
Pf.

- By integrality invariant, when $\Delta = 1 \Rightarrow G_f(\Delta) = G_f$.
- Upon termination of $\Delta = 1$ phase, there are no augmenting paths.
- Result follows augmenting path theorem.

Capacity-scaling: analysis

Lemma 1. There are $1 + \lfloor \log_2 C \rfloor$ scaling phases.
Pf. Initially $C/2 < \Delta \leq C$; Δ decreases by a factor of 2 in each iteration.

Lemma 2. Let f be the flow at the end of a Δ-scaling phase. Then, the max-flow value $\leq val(f) + m\Delta$.
Pf. Next slide.

Lemma 3. There are $\leq 2m$ augmentations per scaling phase.
Pf.

- Let f be the flow at the beginning of a Δ-scaling phase.
- Lemma 2 $\Rightarrow$ max-flow value $\leq val(f) + m(2\Delta)$.
- Each augmentation in a Δ-phase increases $val(f)$ by at least Δ.

≡

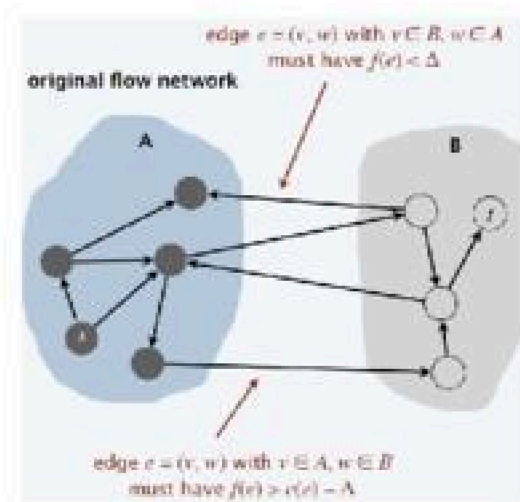
Capacity-scaling: analysis (cont.)

Lemma 2. Let f be the flow at the end of a Δ-scaling phase. Then, the max-flow value $\leq val(f) + m\Delta$.

Pf.

- We show there exists a cut (A, B) such that $cap(A, B) \leq val(f) + m\Delta$.
- Choose A to be the set of nodes reachable from s in $G_f(\Delta)$.
- By definition of A: $s \in A$; By definition of flow f: $t \notin A$.

$$
\begin{aligned}
val(f) &= \sum_{e \text{ out } A} f(e) - \sum_{e \text{ into } A} f(e) \\
&\geq \sum_{e \text{ out } A} (c(e) - \Delta) - \sum_{e \text{ into } A} \Delta \\
&\geq \sum_{e \text{ out } A} c(e) - \sum_{e \text{ out } A} \Delta - \sum_{e \text{ into } A} \Delta \\
&\geq cap(A, B) m\Delta
\end{aligned}
$$



Capacity-scaling: running time

Lemma 1. There are $1 + \lfloor \log_2 C \rfloor$ scaling phases.

Lemma 2. Let f be the flow at the end of a Δ-scaling phase. Then, the max-flow value $\leq val(f) + m\Delta$.

Lemma 3. There are $\leq 2m$ augmentations per scaling phase.

Theorem. The capacity-scaling algorithm takes $O(m^2 \log C)$ time.
Pf.

- Lemma 1 + Lemma 3 $\Rightarrow O(mlogC)$ augmentations.
- Finding an augmenting path takes $O(m)$ time.

Shortest augmenting paths

Shortest augmenting

Q. How to choose next augmenting path in Ford-Fulkerson?
A. Pick one that uses the fewest edges (*via BFS*).

SHORTEST-AUGMENTING-PATH(G)

1. FOREACH $e \in E$: $f(e) = 0$;
2. G_f = residual network of G with respect to flow f;
3. WHILE (there exists an $s \rightsquigarrow t$ path in G_f)
   1. P = BREADTH-FIRST-SEARCH(G_f);
   2. f = AUGMENT(f, c, P);
   3. Update G_f;
4. RETURN f;

Shortest augmenting: analysis overview

Lemma 1. The length (number of edges) of a shortest augmenting path never decreases.
Pf. Ahead.

Lemma 2. After at most m shortest-path augmentations, the length of a shortest augmenting path strictly increases.
Pf. Ahead.

Shortest augmenting: analysis overview

Lemma 1. The length (number of edges) of a shortest augmenting path never decreases.
Pf. Ahead.

Lemma 2. After at most m shortest-path augmentations, the length of a shortest augmenting path strictly increases.
Pf. Ahead.

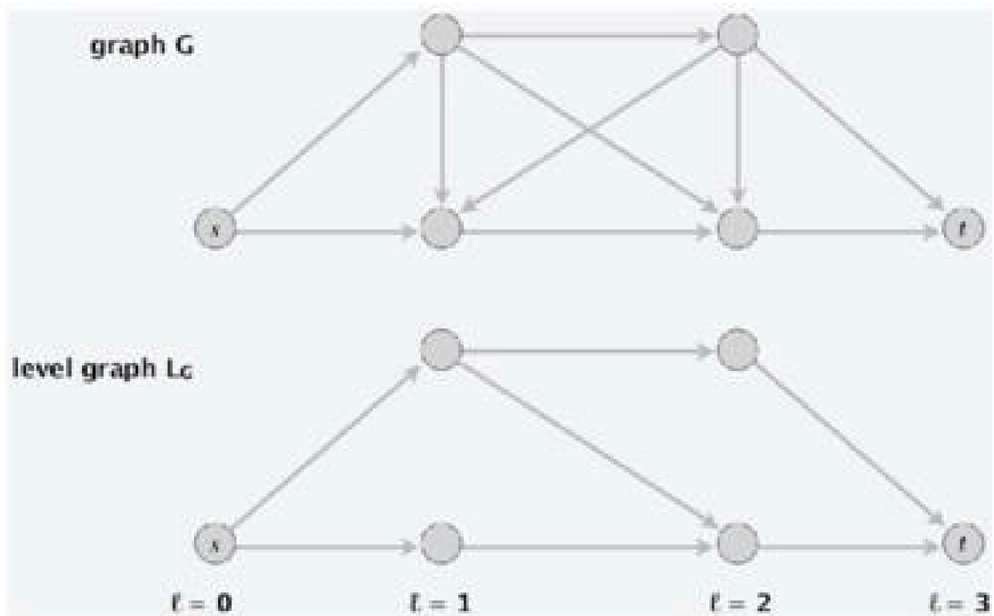
Theorem. The shortest-augmenting-path algorithm takes $O(m^2 n)$ time.
Pf.

- $O(m)$ time to find a shortest augmenting path via BFS.
- There are $\leq mn$ augmentations.
  - [from Lemmas] at most m augmenting paths of length k
  - [simple path] at most $n - 1$ different lengths

Level graph

Def. Given a digraph $G = (V, E)$ with source s, its **level graph** is defined by:

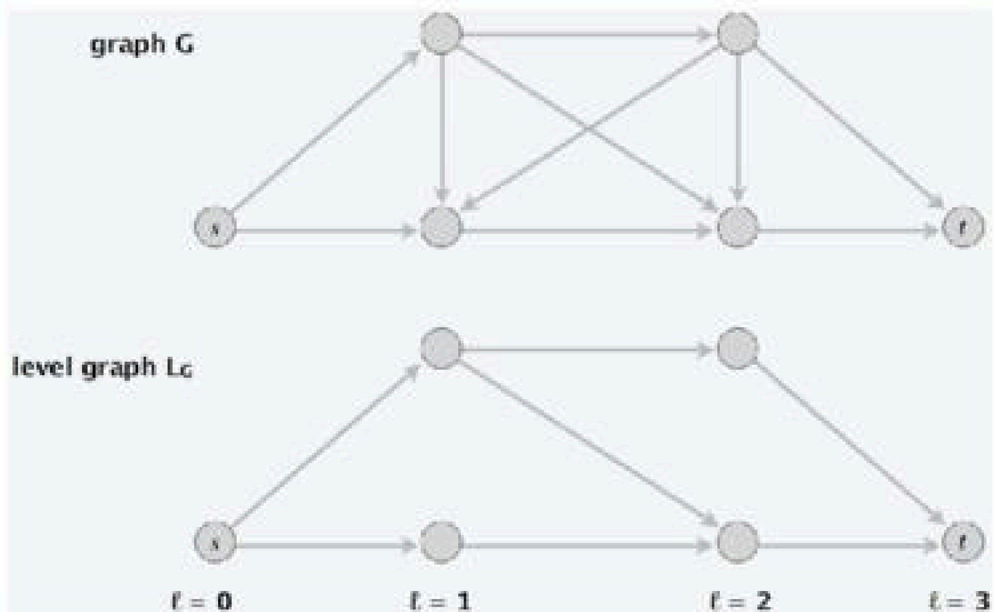
- l(v) = number of edges in shortest $s \rightsquigarrow v$ path.
- $L_G = (V, E_G)$ is the subgraph of G that contains only those edges $(v, w) \in E$ with $l(w) = l(v) + 1$.



Level graph (cont.)

Key property. P is a shortest $s \rightsquigarrow v$ path in G iff P is an $s \rightsquigarrow v$ path in L_G.

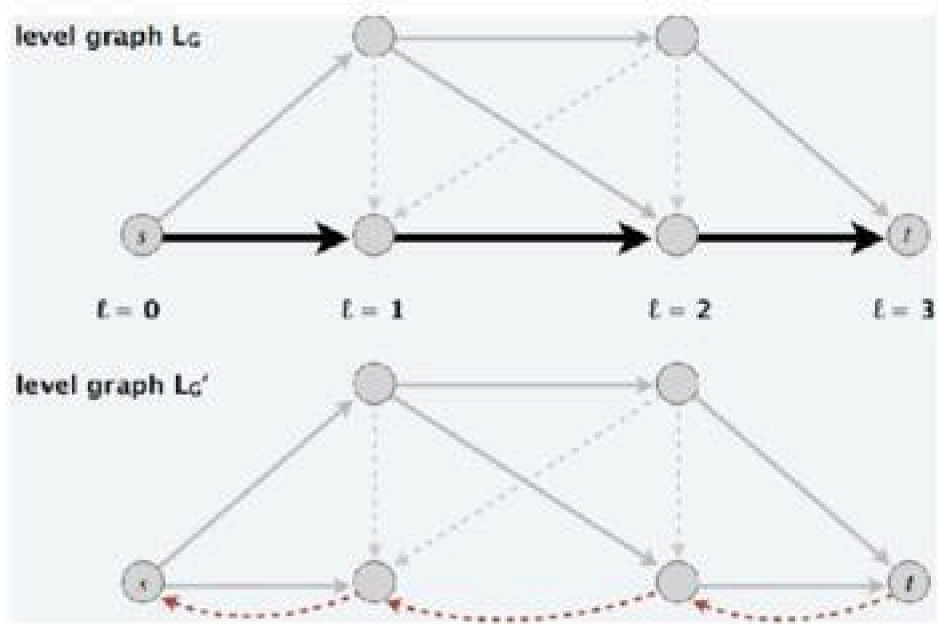
- nodes are ordered the same with BFS
- "back-edges" are removed



Shortest augmenting: Lemma 1

Lemma 1. The length of a shortest augmenting path never decreases.

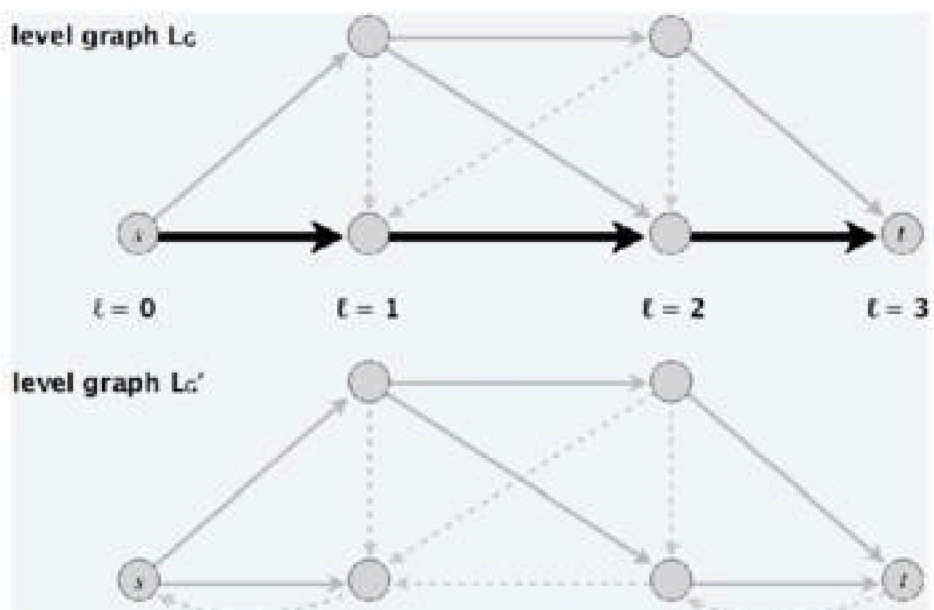
- Let f and f' be flow before and after a shortest-path augmentation.
- Let L_G and $L_G{}'$ be level graphs of G_f and G'_f.
- Only back edges added to G'_f; bottleneck broken.
- any $s \rightsquigarrow t$ path uses back edge is longer than previous length.



Shortest augmenting: Lemma 2

Lemma 2. After at most m shortest-path augmentations, the length of a shortest augmenting path strictly increases.

- At least one (bottleneck) edge is deleted from L_G per augmentation.
- No new edge added to L_G until shortest path length strictly increases.



Shortest augmenting: analysis

Lemma 1. The length (number of edges) of a shortest augmenting path never decreases.

Lemma 2. After at most m shortest-path augmentations, the length of a shortest augmenting path strictly increases.

Theorem. The shortest-augmenting-path algorithm takes $O(m^2 n)$ time.

Shortest augmenting: analysis

Lemma 1. The length (number of edges) of a shortest augmenting path never decreases.

Lemma 2. After at most m shortest-path augmentations, the length of a shortest augmenting path strictly increases.

Theorem. The shortest-augmenting-path algorithm takes $O(m^2 n)$ time.

Note. $\Theta(mn)$ augmentations necessary for some flow networks.

- Try to decrease time per augmentation instead.
  - Simple idea $\Rightarrow O(mn^2)$ [Dinitz 1970]
  - Dynamic trees $\Rightarrow O(mn \log n)$ [Sleator-Tarjan 1983]

Dinitz' algorithm

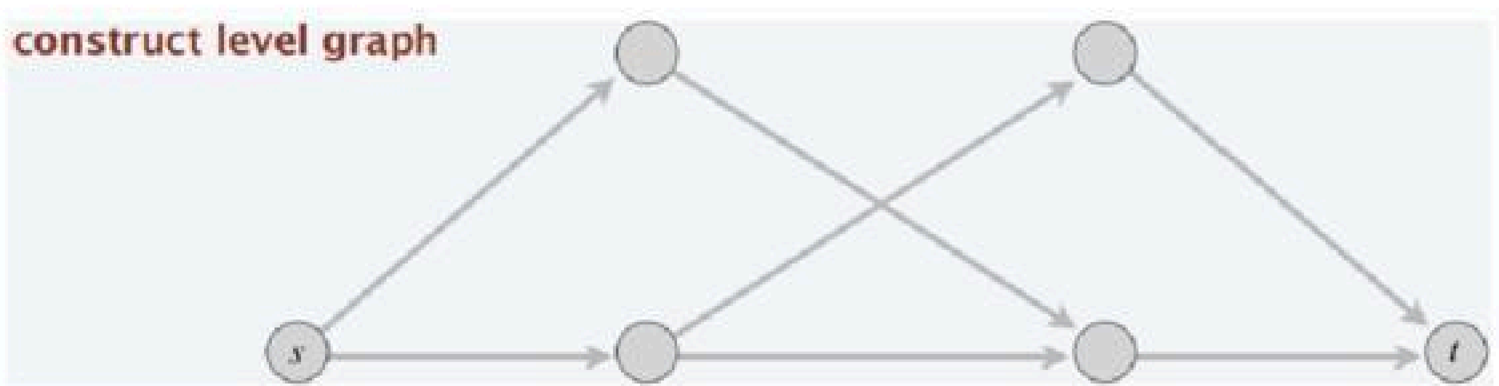
Dinitz' algorithm

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Construct level graph L_G.
- Start at s, advance along an edge in L_G until reach t or get stuck.
  - If reach t, augment flow; update L_G; and restart from s.
  - If get stuck, delete node from L_G and retreat to previous node.



Demo: Dinitz' algorithm

Dinitz' algorithm (refined)

INITIALIZE(G, f)

1. L_G = level-graph of G_f;
2. $P = \emptyset$;
3. GOTO ADVANCE(s);

RETREAT(v)

1. IF $(v = s)$ STOP;
2. ELSE
   1. Delete v (and all incident edges) from L_G;
   2. Remove last edge (u, v) from P ;
   3. GOTO ADVANCE(u);

ADVANCE(v)

1. IF $(v = t)$ AUGMENT(P);
   1. Remove saturated edges from L_G;
   2. $P = \emptyset$;
   3. GOTO ADVANCE(s);
2. IF (there exists edge $(v, w) \in L_G$)
   1. Add edge (v, w) to P;
   2. GOTO ADVANCE(w);
3. ELSE
   1. GOTO RETREAT(v);

Quiz: level graph

How to compute the level graph L_G efficiently?

A. Depth-first search.
B. Breadth-first search.
C. Both A and B.
D. Neither A nor B.

Dinitz' algorithm: analysis

Lemma. A phase can be implemented to run in $O(mn)$ time.

Pf.

- Initialization happens once per phase. $O(m)$ using BFS.
- At most m augmentations per phase. $O(mn)$ per phase.
  - (because an augmentation deletes at least one edge from L_G)
- At most n retreats per phase. $O(m + n)$ per phase
  - (because a retreat deletes one node from L_G)
- At most mn advances per phase. O(mn) per phase
  - (because at most n advances before retreat or augmentation)

Dinitz' algorithm: analysis

Lemma. A phase can be implemented to run in $O(mn)$ time.

Pf.

- Initialization happens once per phase. $O(m)$ using BFS.
- At most m augmentations per phase. $O(mn)$ per phase.
  - (because an augmentation deletes at least one edge from L_G)
- At most n retreats per phase. $O(m+n)$ per phase
  - (because a retreat deletes one node from L_G)
- At most mn advances per phase. O(mn) per phase
  - (because at most n advances before retreat or augmentation)

Theorem. [Dinitz 1970] Dinitz' algorithm runs in $O(mn^2)$ time.

Pf.

- By Lemma, $O(mn)$ time per phase.
- At most $n-1$ phases (as in shortest-augmenting-path analysis).

Maximum-flow algorithms: practice

Push-relabel algorithm. [Goldberg-Tarjan 1988] Increases flow one edge at a time instead of one augmenting path at a time.

- (SECTION 7.4, Algorithm Design.)

Maximum-flow algorithms: practice

Push-relabel algorithm. [Goldberg-Tarjan 1988] Increases flow one edge at a time instead of one augmenting path at a time.

- (SECTION 7.4, Algorithm Design.)

Caveat. Worst-case running time is generally not useful for predicting or comparing max-flow algorithm performance in practice.

Best in practice. Push-relabel method with gap relabeling: $O(m^{3/2})$ in practice.

Maximum-flow algorithms: CV

Computer vision. Different algorithms work better for some dense problems that arise in applications to computer vision.

- [Boykov and Kolmogorov 2004] An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision.

Simple unit-capacity networks

Quiz: bipartite matching

Which max-flow algorithm to use for bipartite matching?

A. Ford-Fulkerson: $O(mnC)$.
B. Capacity scaling: $O(m^2 logC)$.
C. Shortest augmenting path: $O(m^2 n)$.
D. Dinitz' algorithm: $O(mn^2)$.

Quiz: bipartite matching

Which max-flow algorithm to use for bipartite matching?

A. Ford-Fulkerson: $O(mnC)$.
B. Capacity scaling: $O(m^2 logC)$.
C. Shortest augmenting path: $O(m^2 n)$.
D. Dinitz' algorithm: $O(mn^2)$.

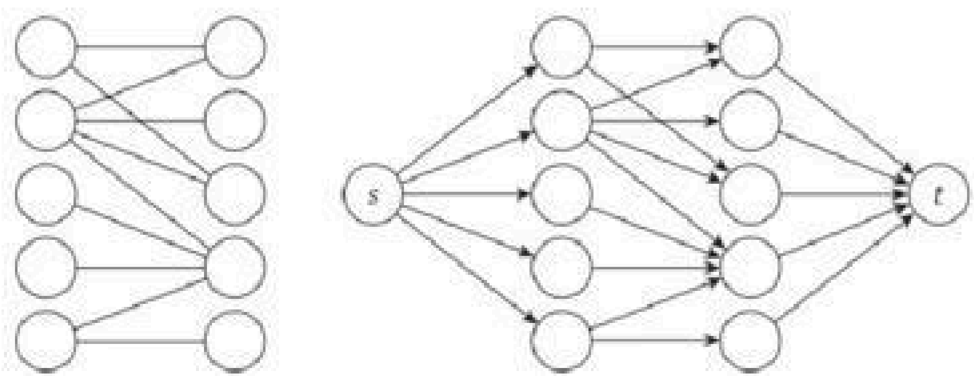
D. the graph may be dense.

Simple unit-capacity networks

Def. A flow network is a **simple unit-capacity network** if:

- Every edge has capacity 1.
- Every node (other than s or t) has exactly one entering edge, or exactly one leaving edge, or both.

Ex. Bipartite matching.

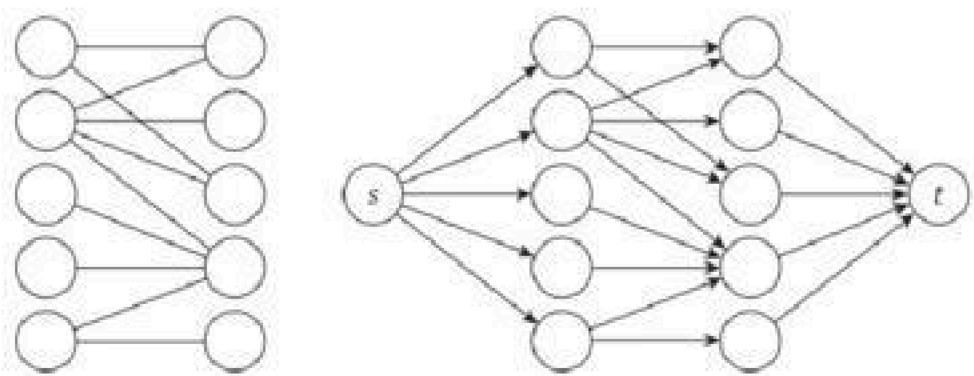


Simple unit-capacity networks

Def. A flow network is a **simple unit-capacity network** if:

- Every edge has capacity 1.
- Every node (other than s or t) has exactly one entering edge, or exactly one leaving edge, or both.

Ex. Bipartite matching.



Property. Let G be a simple unit-capacity network and let f be a 0-1 flow. Then, residual network G_f is also a simple unit-capacity network.

Unit-capacity: algorithm overview

Shortest-augmenting-path algorithm.

- Normal augmentation: length of shortest path does not change.
- Special augmentation: length of shortest path strictly increases.

Unit-capacity: algorithm overview

Shortest-augmenting-path algorithm.

- Normal augmentation: length of shortest path does not change.
- Special augmentation: length of shortest path strictly increases.

Theorem. [Even-Tarjan 1975] In simple unit-capacity networks, Dinitz' algorithm computes a maximum flow in $O(mn^{1/2})$ time.

Pf.

- Lemma 1. Each phase of normal augmentations takes $O(m)$ time.
- Lemma 2. After $n^{1/2}$ phases, $val(f) \geq val(f^*) - n^{1/2}$.
- Lemma 3. After $\leq n^{1/2}$ additional augmentations, flow is optimal.

Unit-capacity: algorithm overview

Shortest-augmenting-path algorithm.

- Normal augmentation: length of shortest path does not change.
- Special augmentation: length of shortest path strictly increases.

Theorem. [Even-Tarjan 1975] In simple unit-capacity networks, Dinitz' algorithm computes a maximum flow in $O(mn^{1/2})$ time.
Pf.

- Lemma 1. Each phase of normal augmentations takes $O(m)$ time.
- Lemma 2. After $n^{1/2}$ phases, $val(f) \geq val(f^*) - n^{1/2}$.
- Lemma 3. After $\leq n^{1/2}$ additional augmentations, flow is optimal.

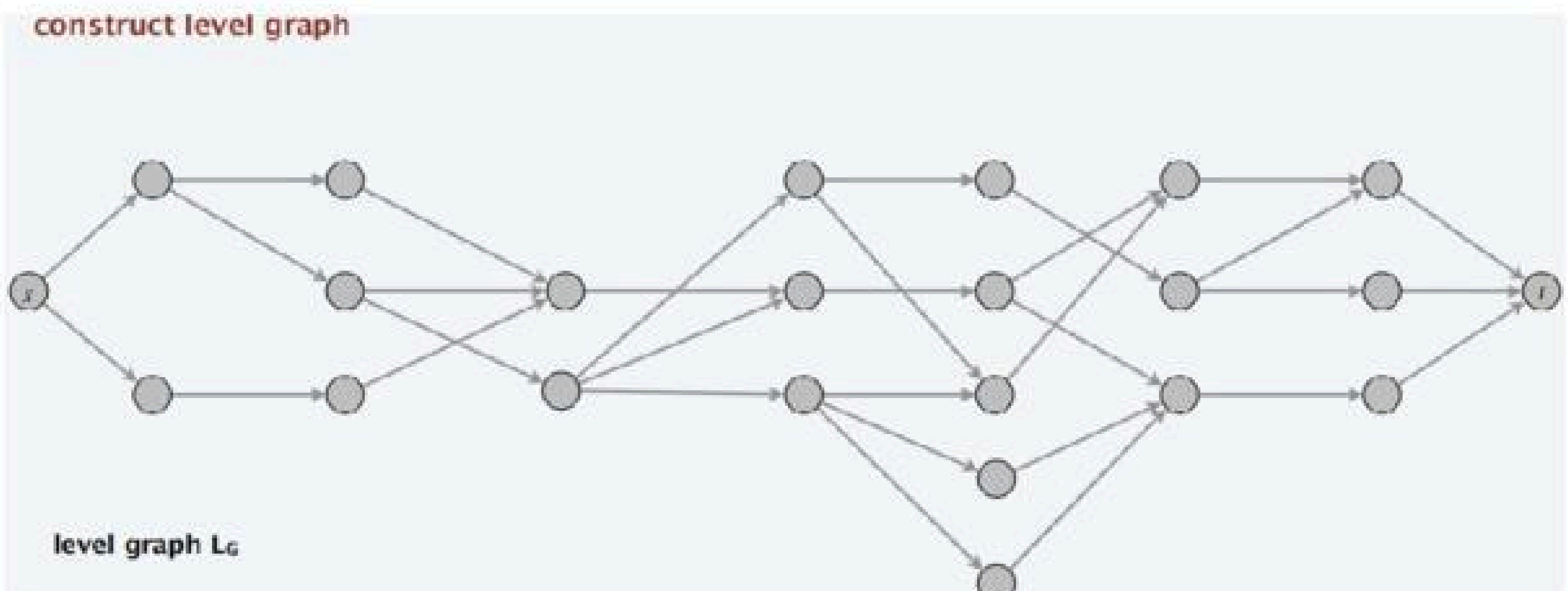
Lemma 3. After $\leq n^{1/2}$ additional augmentations, flow is optimal.
Pf. Each augmentation increases flow value by at least 1.

Lemma 1 and Lemma 2. Ahead.

Unit-capacity: Dinitz'

Phase of normal augmentations.

- Construct level graph L_G.
- Start at s, advance along an edge in L_G until reach t or get stuck.
- If reach t, augment flow; update L_G; and restart from s.
- If get stuck, delete node from L_G and go to previous node.



Demo: Dinitz' for Unit-capacity

Unit-capacity: Lemma 1

Phase of normal augmentations.

- Construct level graph L_G.
- Start at s, advance along an edge in L_G until reach t or get stuck.
- If reach t, augment flow; update L_G; and restart from s.
- If get stuck, delete node from L_G and go to previous node.

Lemma 1. A phase of normal augmentations takes $O(m)$ time.
Pf.

- $O(m)$ to create level graph L_G.
- $O(1)$ per edge (each edge involved in at most one advance, retreat, and augmentation).
- $O(1)$ per node (each node deleted at most once).

Quiz: non-unit-capacity

Consider running advance-retreat algorithm in a unit-capacity network (but not necessarily a simple one). What is running time?

A. $O(m)$.
B. $O(m^{3/2})$.
C. $O(mn)$.
D. May not terminate.

Hint: both indegree and outdegree of a node can be larger than 1.

Quiz: non-unit-capacity

Consider running advance-retreat algorithm in a unit-capacity network (but not necessarily a simple one). What is running time?

A. $O(m)$.
B. $O(m^{3/2})$.
C. $O(mn)$.
D. May not terminate.

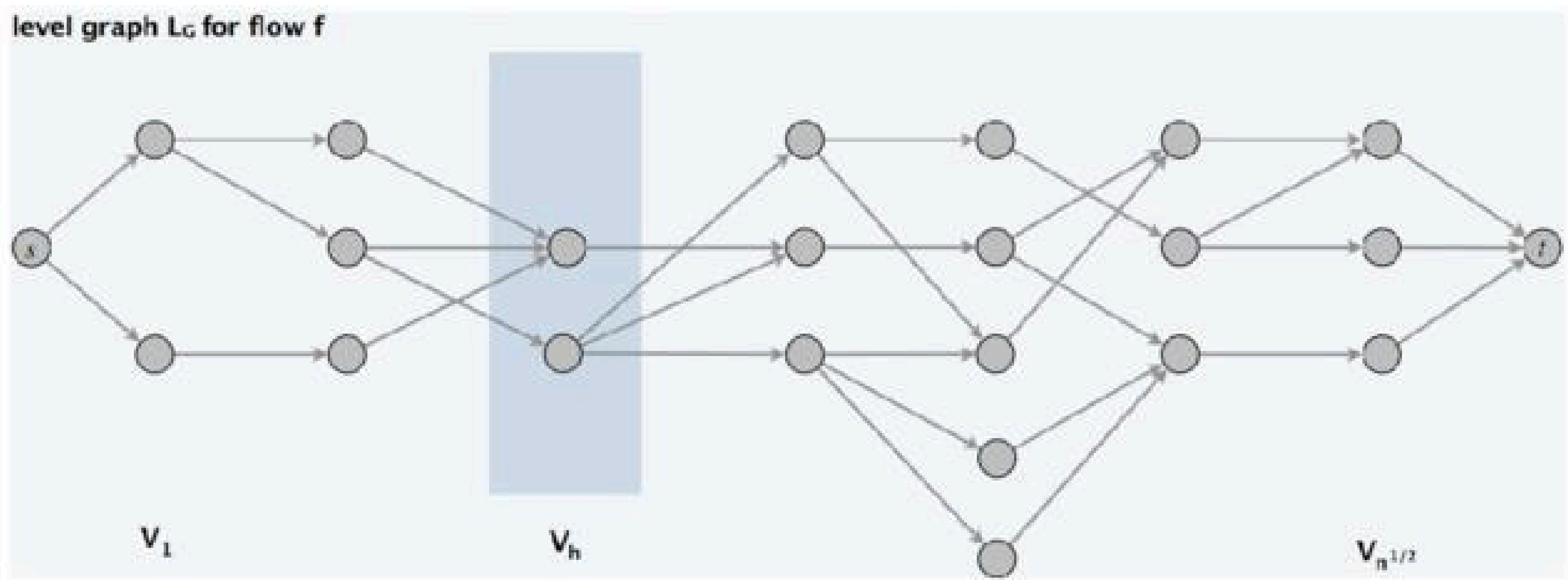
Hint: both indegree and outdegree of a node can be larger than 1.

A. may take m operations per node

Unit-capacity: Lemma 2

Lemma 2. After $n^{1/2}$ phases, $val(f) \geq val(f^*) - n^{1/2}$.

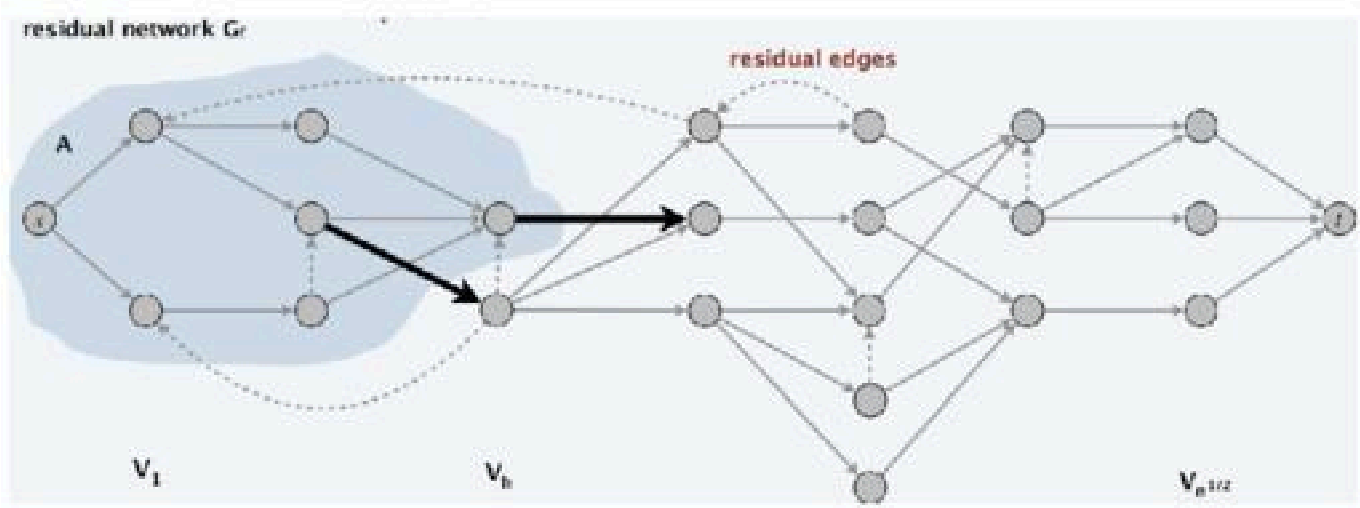
- After $n^{1/2}$ phases, length of shortest augmenting path is $> n^{1/2}$.
- Thus, level graph has $\geq n^{1/2}$ levels (not including levels for s or t).
- Let $1 \leq h \leq n^{1/2}$ be a level with min number of nodes $\Rightarrow |V_h| \leq n^{1/2}$.



Unit-capacity: Lemma 2 (cont.)

Lemma 2. After $n^{1/2}$ phases, $val(f) \geq val(f^*) - n^{1/2}$.

- After $n^{1/2}$ phases, length of shortest augmenting path is $> n^{1/2}$.
- Thus, level graph has $\geq n^{1/2}$ levels (not including levels for s or t).
- Let $1 \leq h \leq n^{1/2}$ be a level with min number of nodes $\Rightarrow |V_h| \leq n^{1/2}$.
- Let $A = \{v : l(v) < h\} \cup \{v : l(v) = h \text{ and } v \text{ has } \leq 1 \text{ outgoing residual edge}\}$.
- $cap_f(A, B) \leq |V_h| \leq n^{1/2} \Rightarrow val(f) \geq val(f^*) - n^{1/2}$.



Unit-capacity: analysis

Theorem. [Even-Tarjan 1975] In simple unit-capacity networks, Dinitz' algorithm computes a maximum flow in $O(mn^{1/2})$ time.

Pf.

- Lemma 1. Each phase of normal augmentations takes $O(m)$ time.
- Lemma 2. After $n^{1/2}$ phases, $val(f) \geq val(f^*) - n^{1/2}$.
- Lemma 3. After $\leq n^{1/2}$ additional augmentations, flow is optimal.

Unit-capacity: analysis

Theorem. [Even-Tarjan 1975] In simple unit-capacity networks, Dinitz' algorithm computes a maximum flow in $O(mn^{1/2})$ time.

Pf.

- Lemma 1. Each phase of normal augmentations takes $O(m)$ time.
- Lemma 2. After $n^{1/2}$ phases, $val(f) \geq val(f^*) - n^{1/2}$.
- Lemma 3. After $\leq n^{1/2}$ additional augmentations, flow is optimal.

Corollary. Dinitz' algorithm computes max-cardinality bipartite matching in $O(mn^{1/2})$ time.