

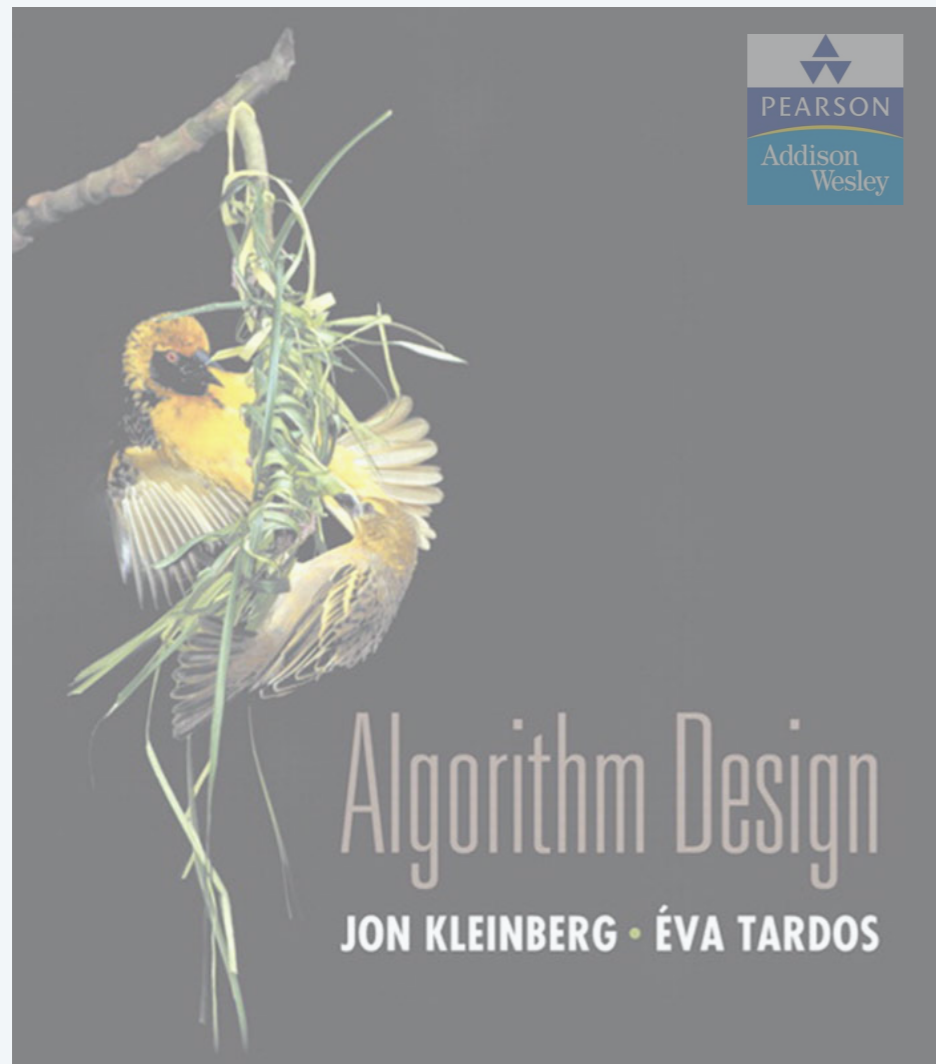
7. NETWORK FLOW I

- ▶ *max-flow and min-cut problems*
- ▶ *Ford–Fulkerson algorithm*
- ▶ *max-flow min-cut theorem*
- ▶ *capacity-scaling algorithm*
- ▶ *shortest augmenting paths*

Lecture slides by Kevin Wayne

Copyright © 2005 Pearson–Addison Wesley

<http://www.cs.princeton.edu/~wayne/kleinberg-tardos>



SECTION 7.1

7. NETWORK FLOW I

- ▶ *max-flow and min-cut problems*
- ▶ *Ford–Fulkerson algorithm*
- ▶ *max-flow min-cut theorem*
- ▶ *capacity-scaling algorithm*
- ▶ *shortest augmenting paths*

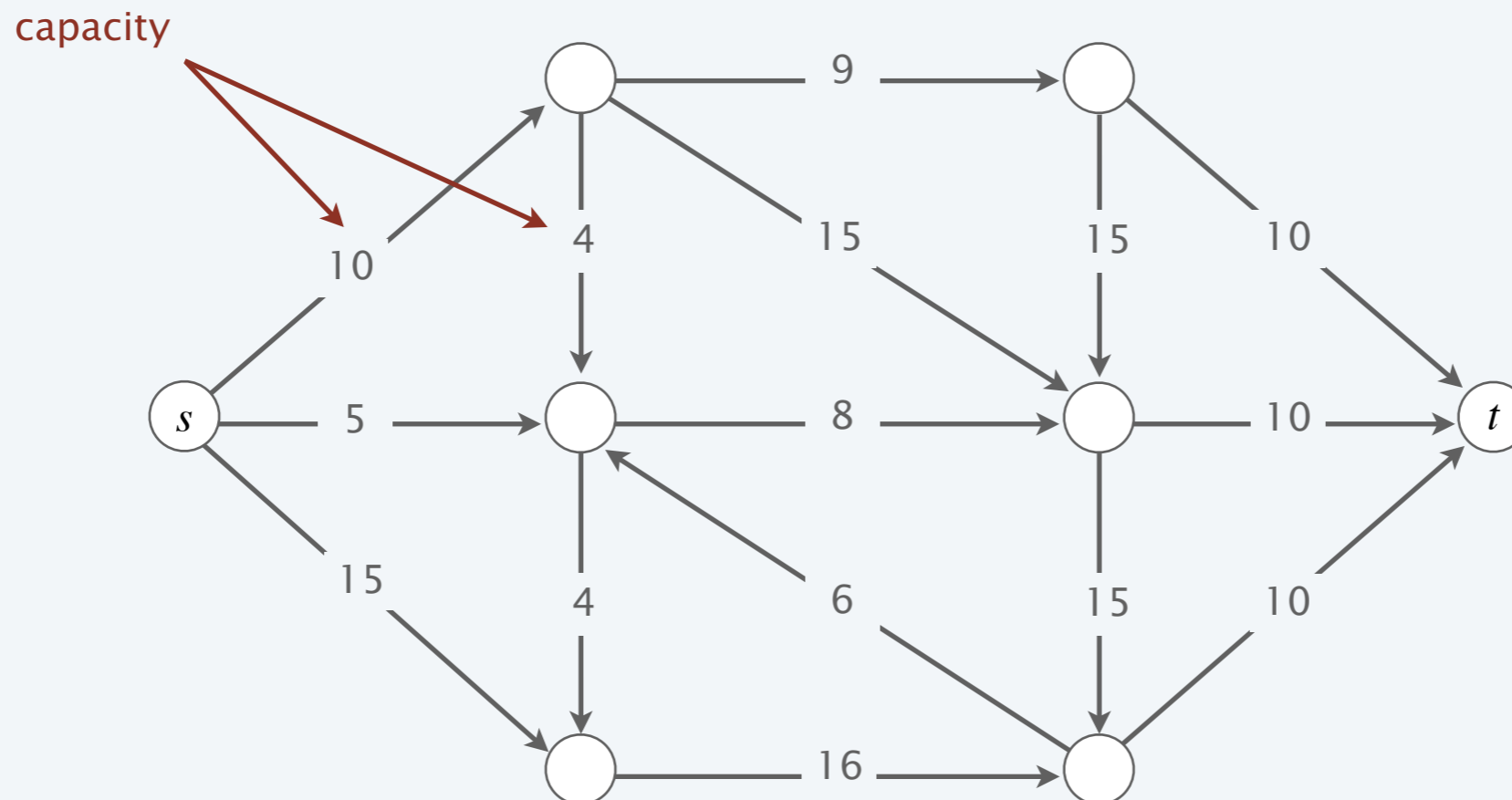
Flow network

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 $e \in E$.

assume all nodes are reachable from s

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

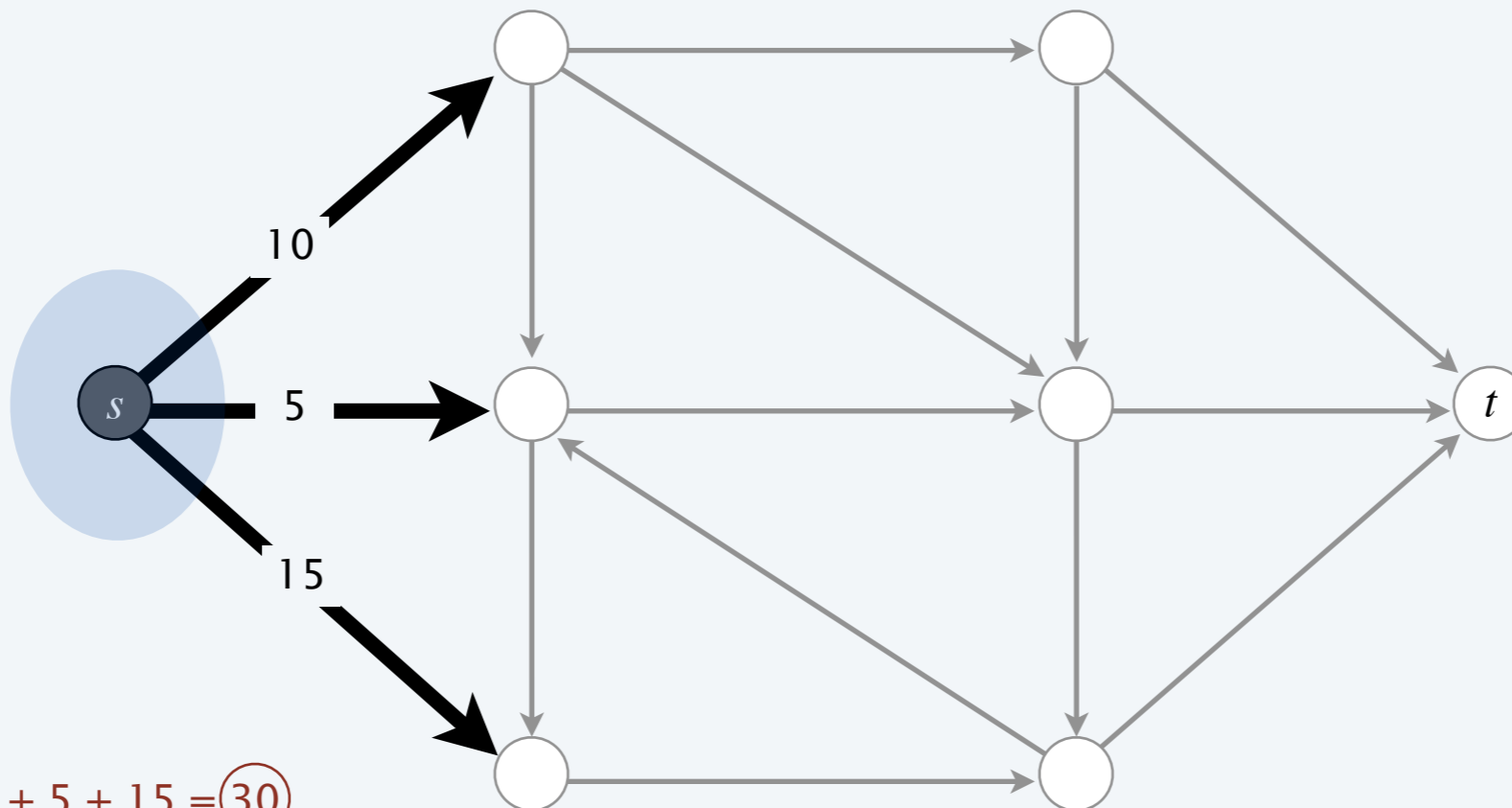


Minimum-cut problem

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 .

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



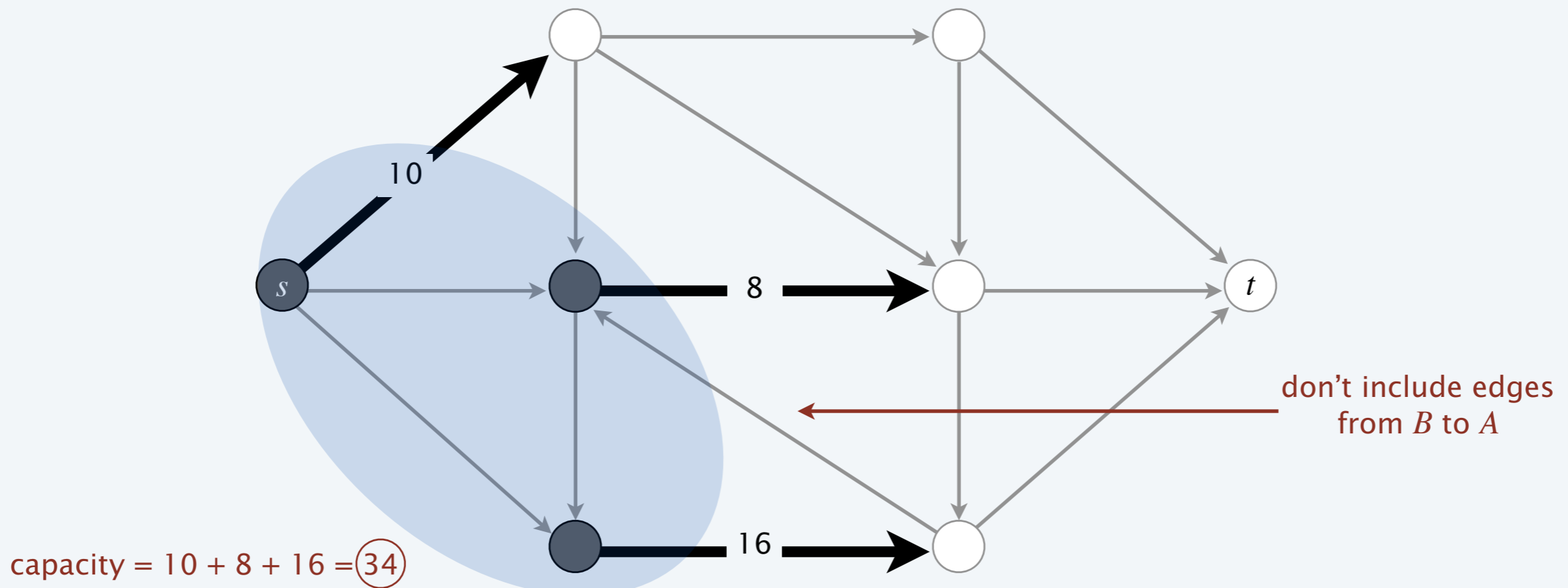
capacity = $10 + 5 + 15 = 30$

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 .

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



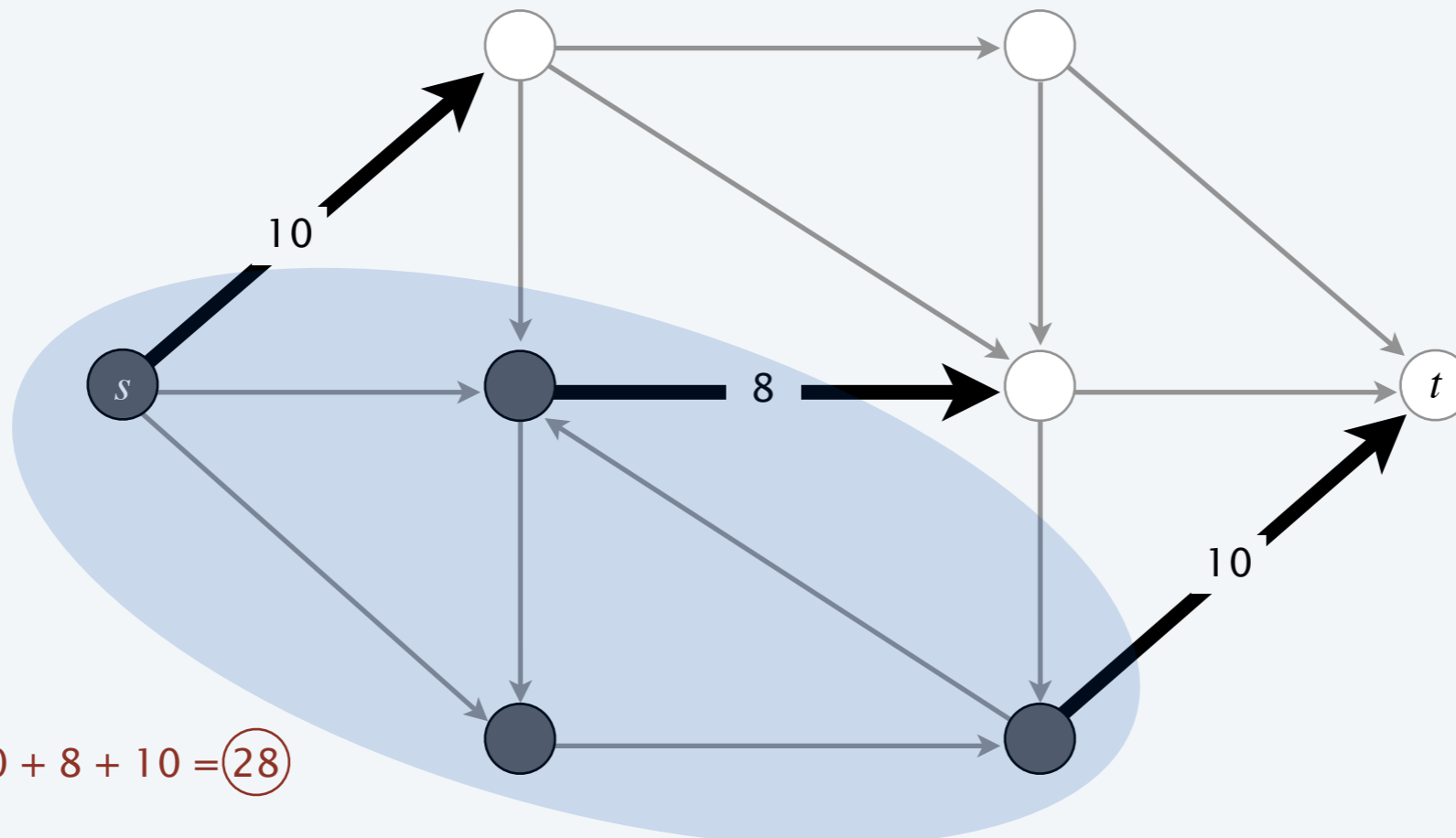
Minimum-cut problem

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 .

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

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

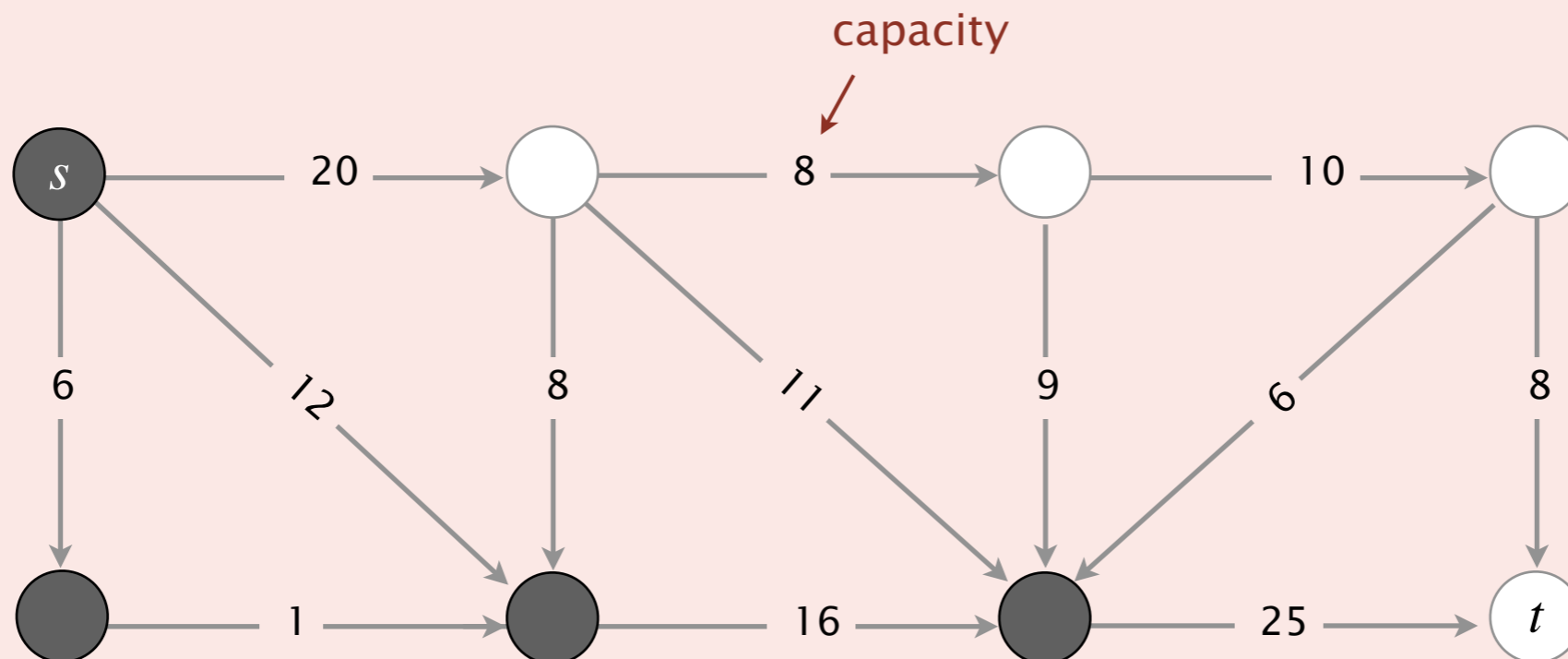


capacity = $10 + 8 + 10 = 28$



Which is the capacity of the given st -cut?

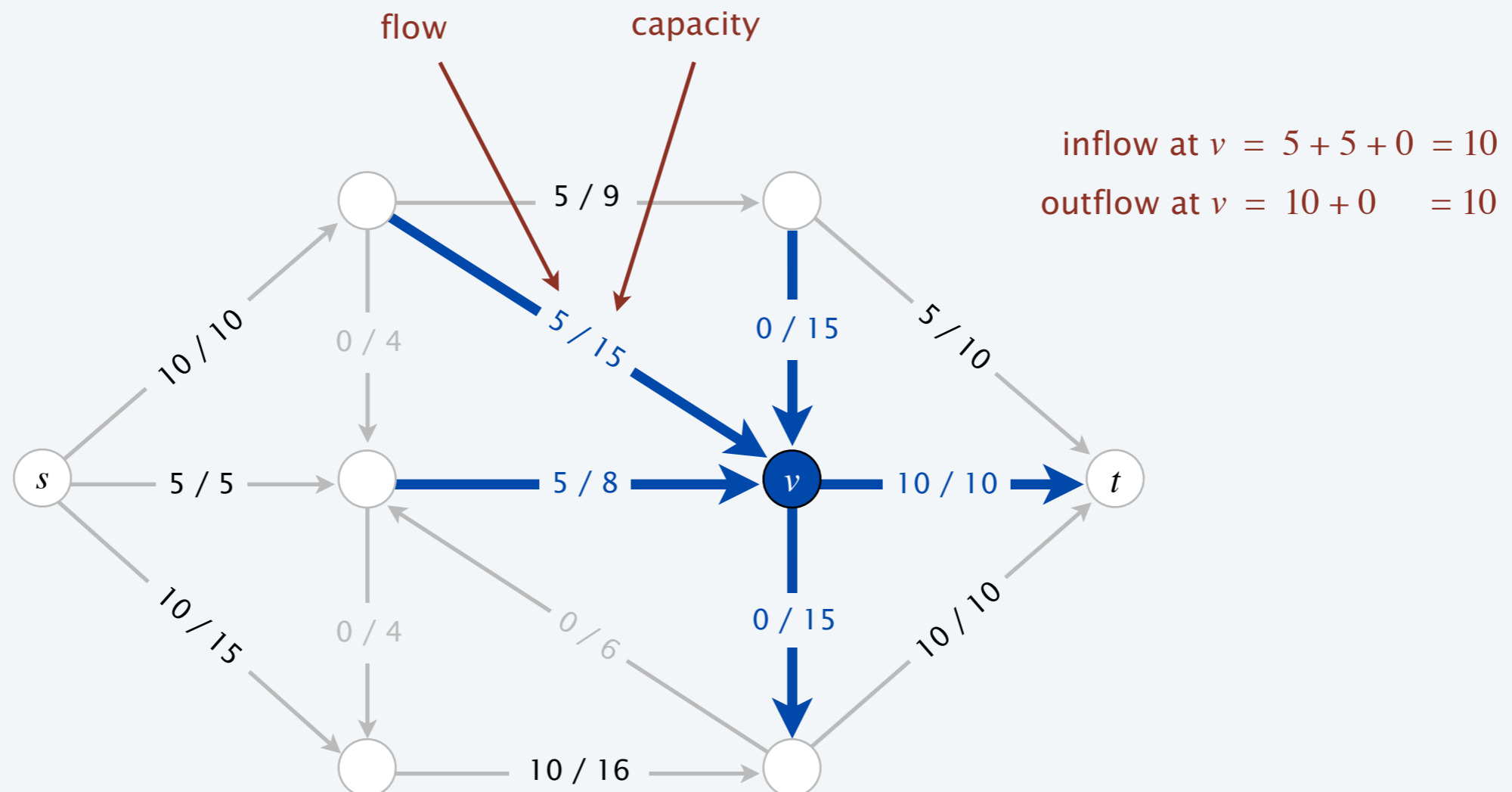
- A. 11 ($20 + 25 - 8 - 11 - 9 - 6$)
- B. 34 ($8 + 11 + 9 + 6$)
- C. 45 ($20 + 25$)
- D. 79 ($20 + 25 + 8 + 11 + 9 + 6$)



Maximum-flow problem

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

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

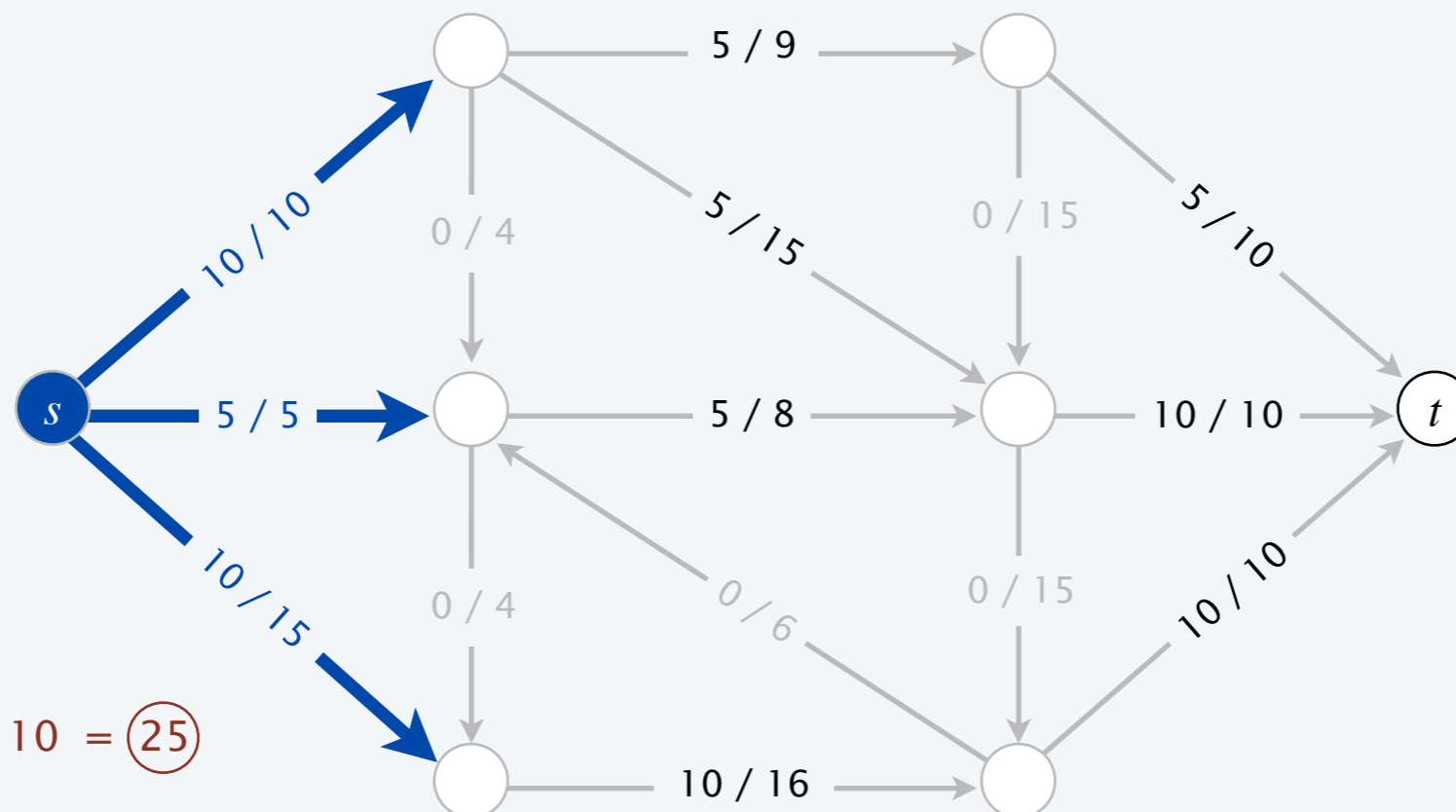


Maximum-flow problem

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

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

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



value = $5 + 10 + 10 = 25$

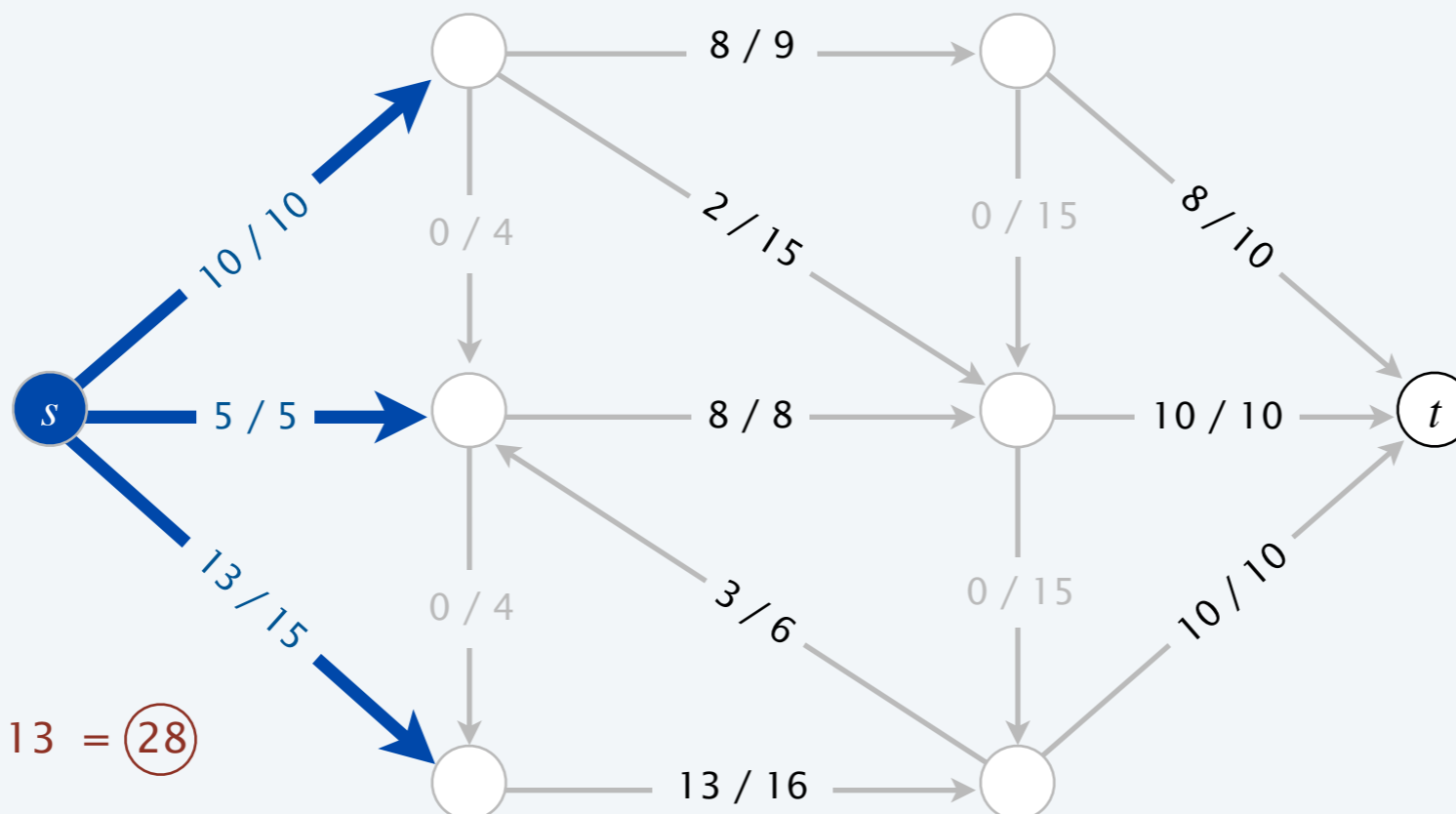
Maximum-flow problem

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

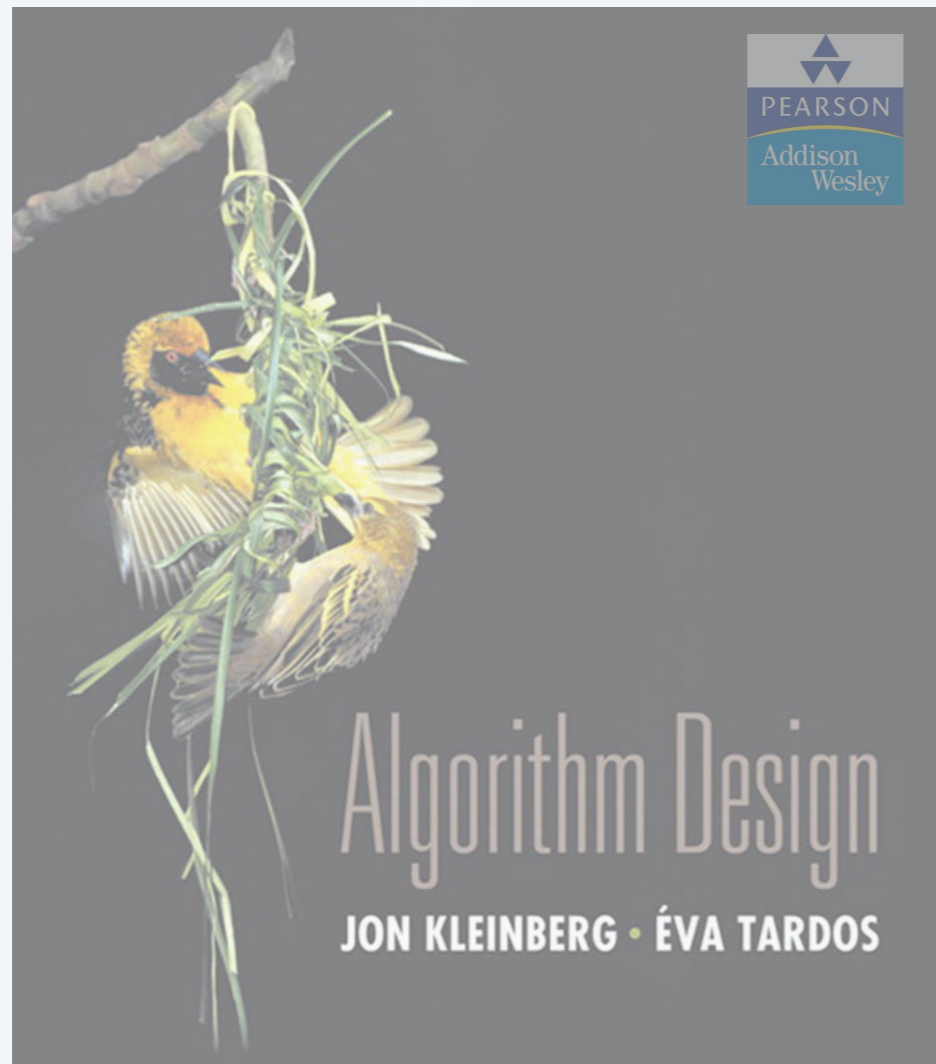
- For each $e \in E$: $0 \leq f(e) \leq c(e)$ [capacity]
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [flow conservation]

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

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



value = 10 + 5 + 13 = 28



SECTION 7.1

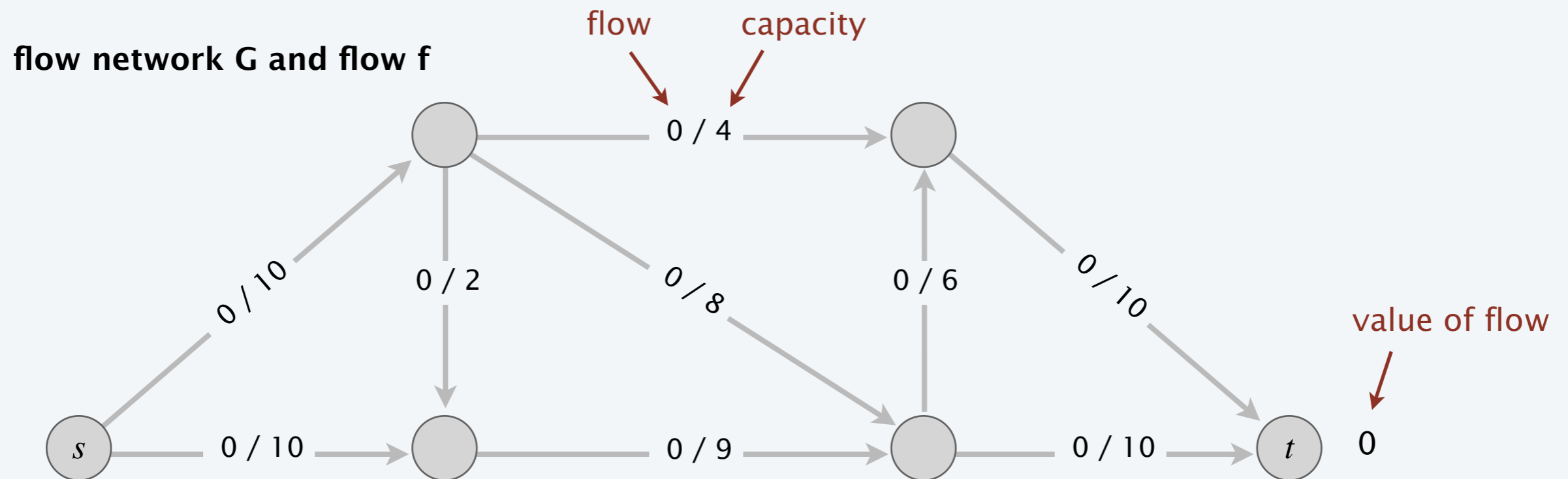
7. NETWORK FLOW I

- ▶ *max-flow and min-cut problems*
- ▶ ***Ford–Fulkerson algorithm***
- ▶ *max-flow min-cut theorem*
- ▶ *capacity-scaling algorithm*
- ▶ *shortest augmenting paths*

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.

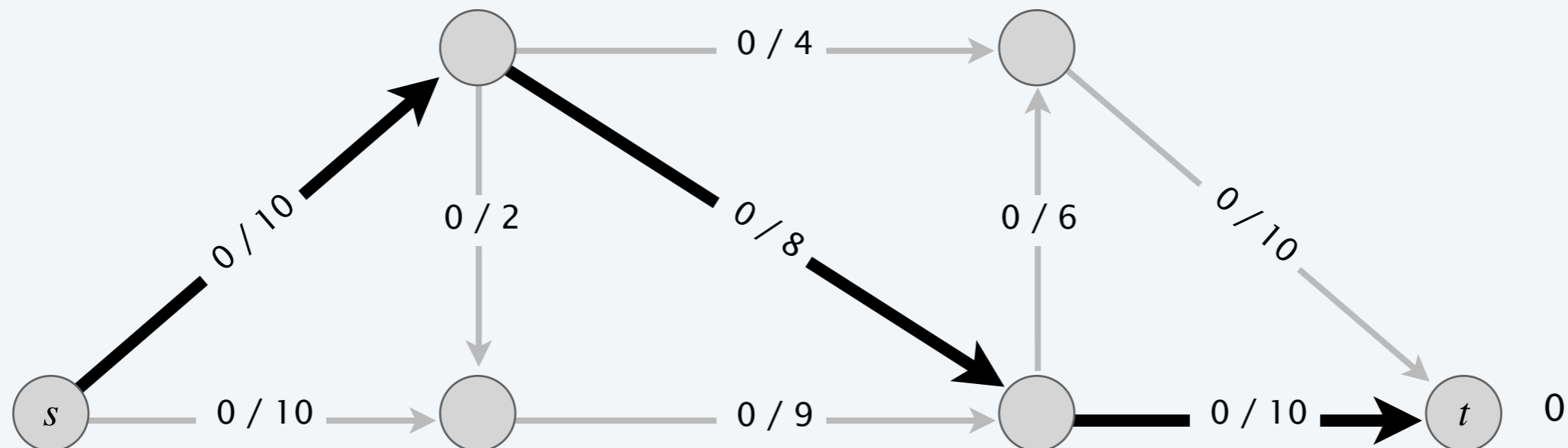


Toward a max-flow algorithm

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.

flow network G and flow f

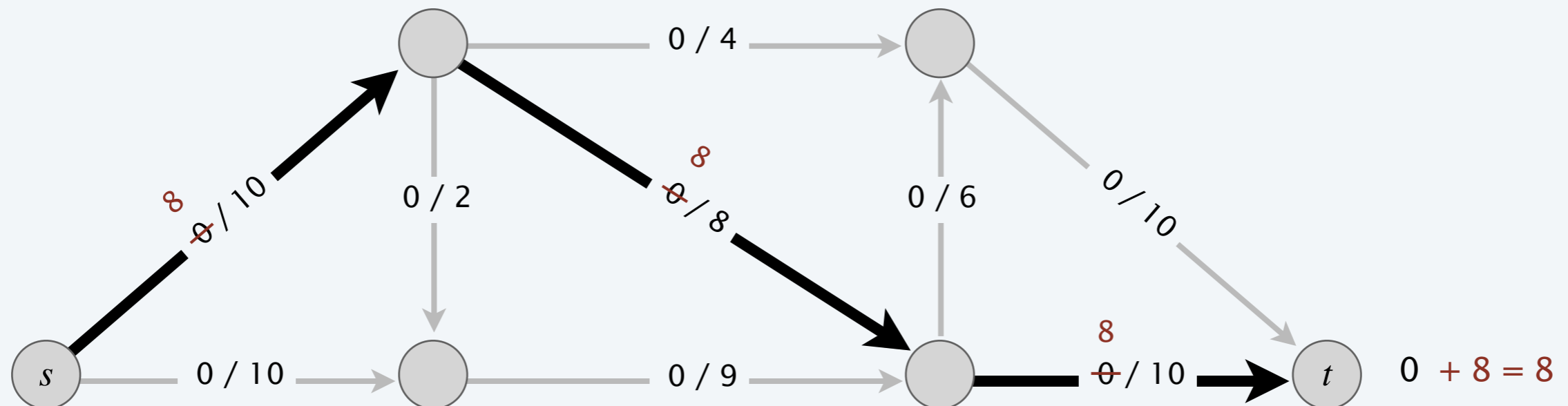


Toward a max-flow algorithm

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.

flow network G and flow f

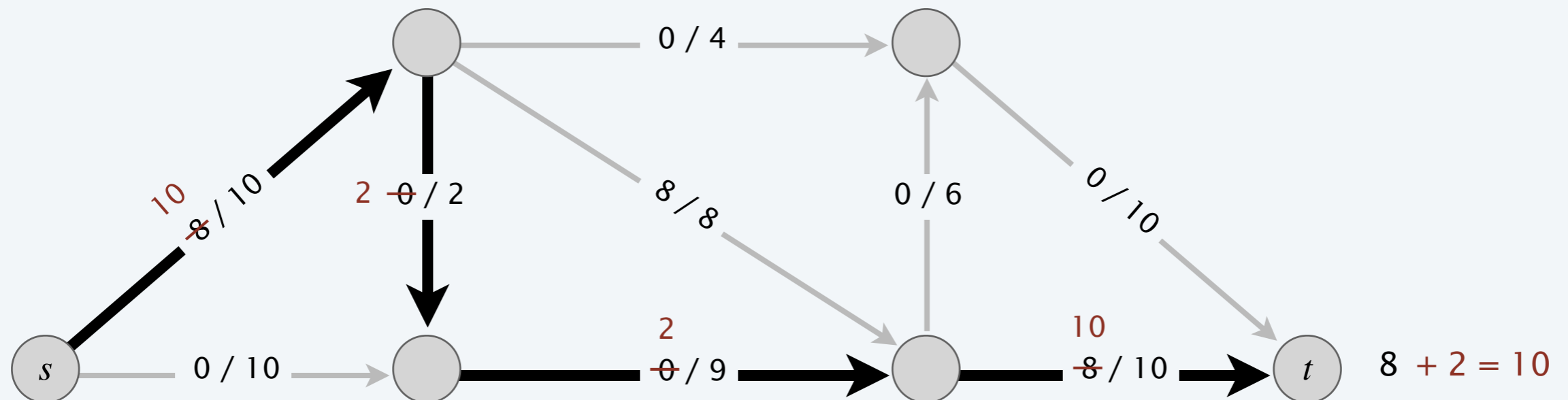


Toward a max-flow algorithm

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.

flow network G and flow f

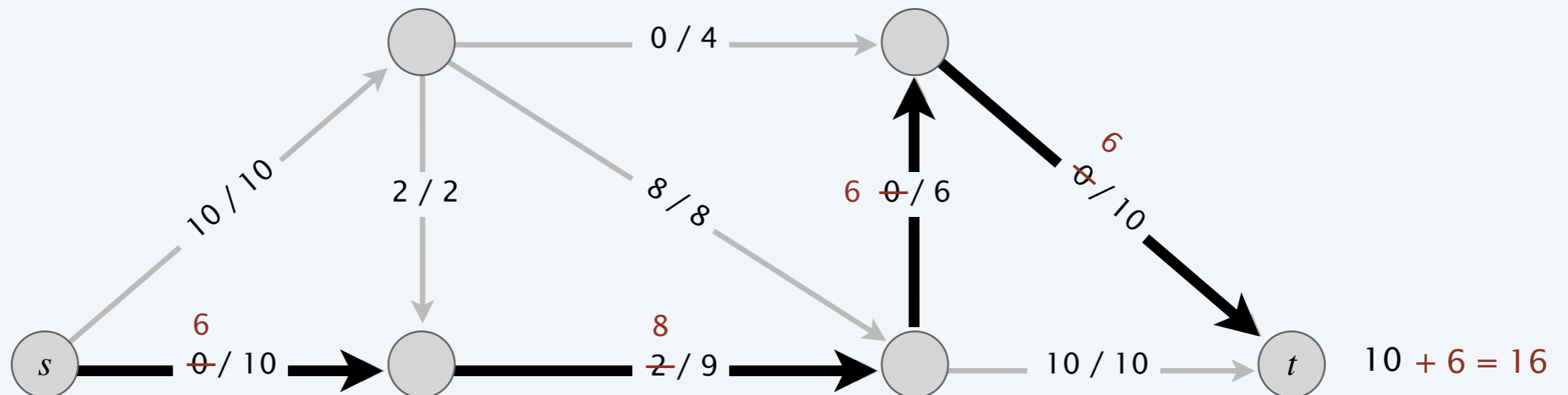


Toward a max-flow algorithm

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.

flow network G and flow f



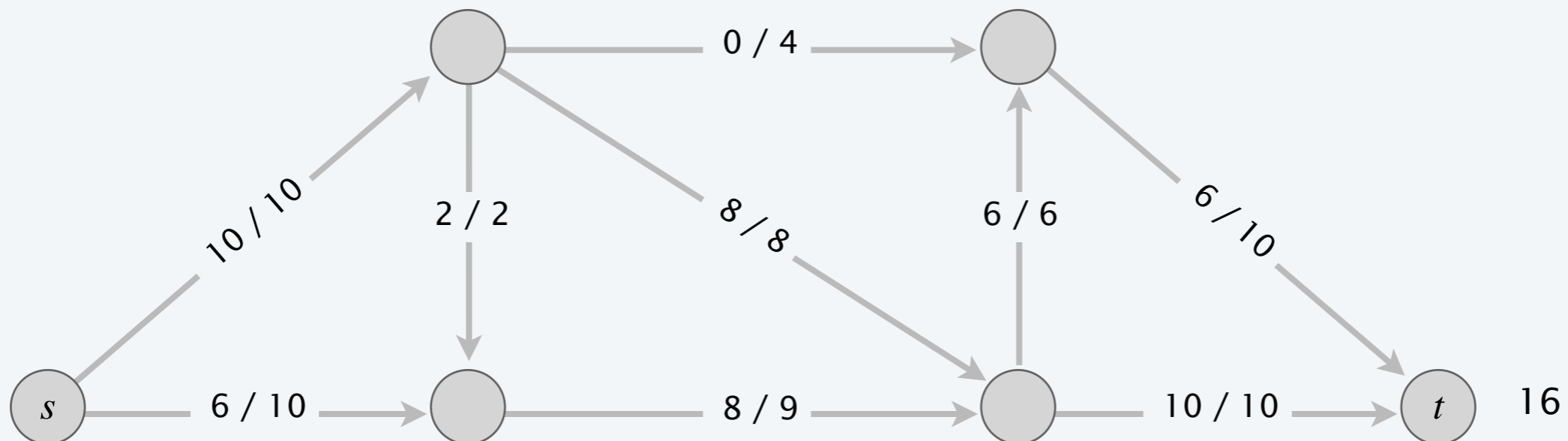
Toward a max-flow algorithm

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

flow network G and flow f



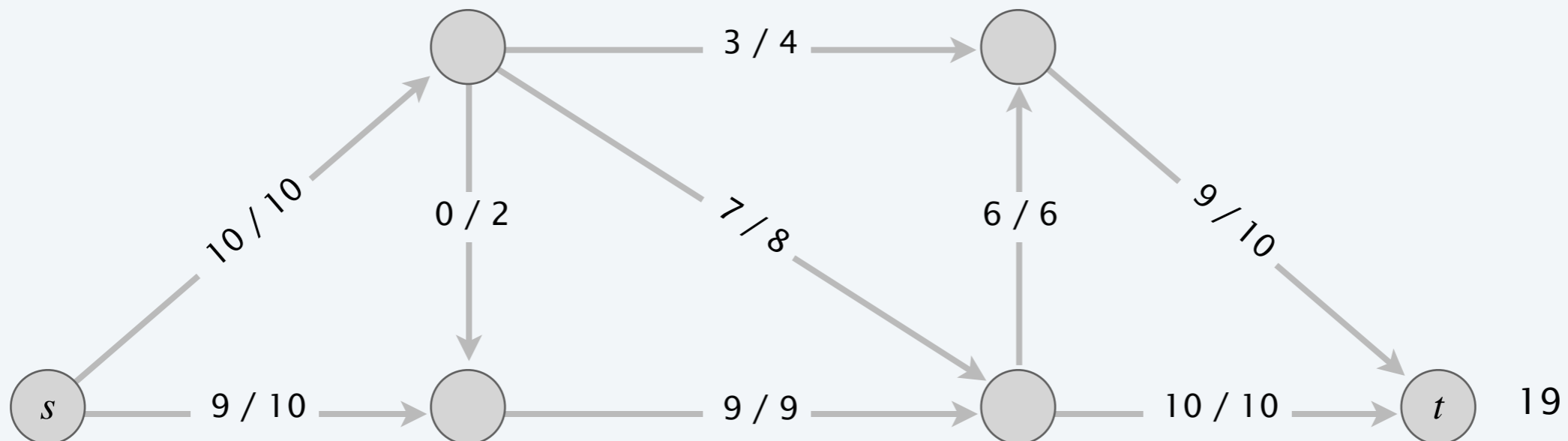
Toward a max-flow algorithm

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

flow network G and flow f



Why the greedy algorithm fails

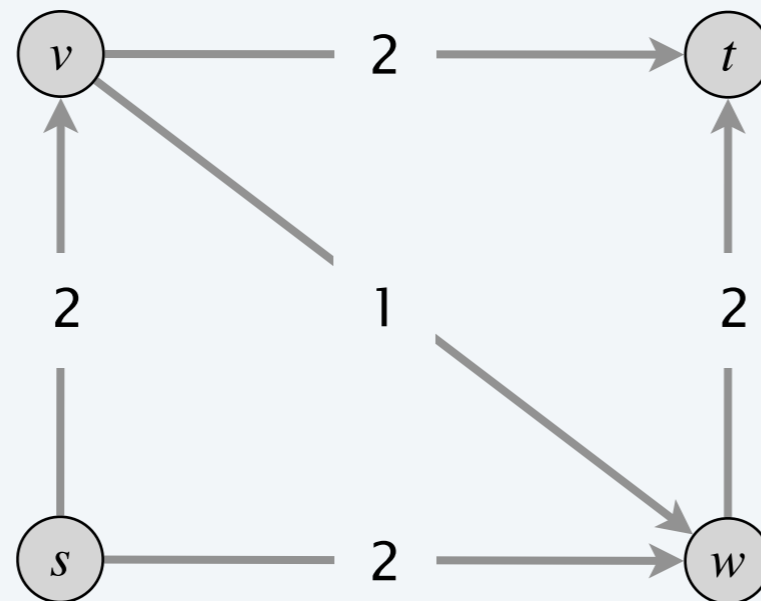
Q. Why does the greedy algorithm fail?

A. Once greedy algorithm increases flow on an edge, it never decreases it.

Ex. Consider flow network G .

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

flow network G



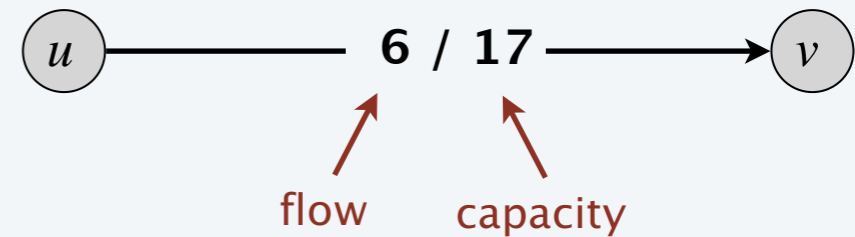
Bottom line. Need some mechanism to “undo” a bad decision.

Residual network

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

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

original flow network G



Reverse edge. $e^{\text{reverse}} = (v, u)$.

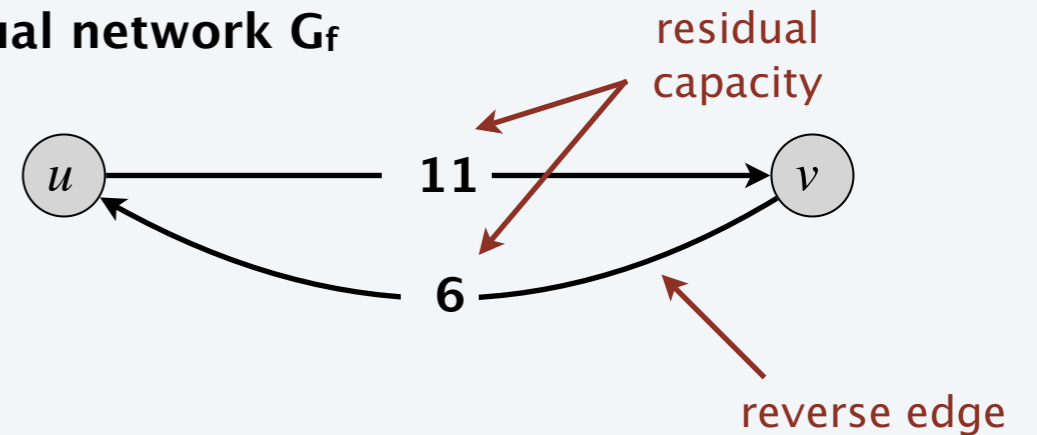
- “Undo” flow sent.

Residual capacity.

$$c_f(e) = c(e) - f(e) \text{ for all } e \in E.$$

$$c_f(e^{\text{reverse}}) = f(e) \text{ for all } e \in E.$$

residual network G_f



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

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

edges with positive residual capacity

where flow on a reverse edge negates flow on corresponding forward edge

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 $f' \leftarrow \text{AUGMENT}(f, c, P)$, the resulting f' is a flow and $\text{val}(f') = \text{val}(f) + \text{bottleneck}(G_f, P)$.

```
AUGMENT( $f, c, P$ )
```

```
 $\delta \leftarrow$  bottleneck capacity of augmenting path  $P$ .
```

```
FOREACH edge  $e \in P$  :
```

```
    IF ( $e \in E$ )  $f(e) \leftarrow f(e) + \delta$ .
```

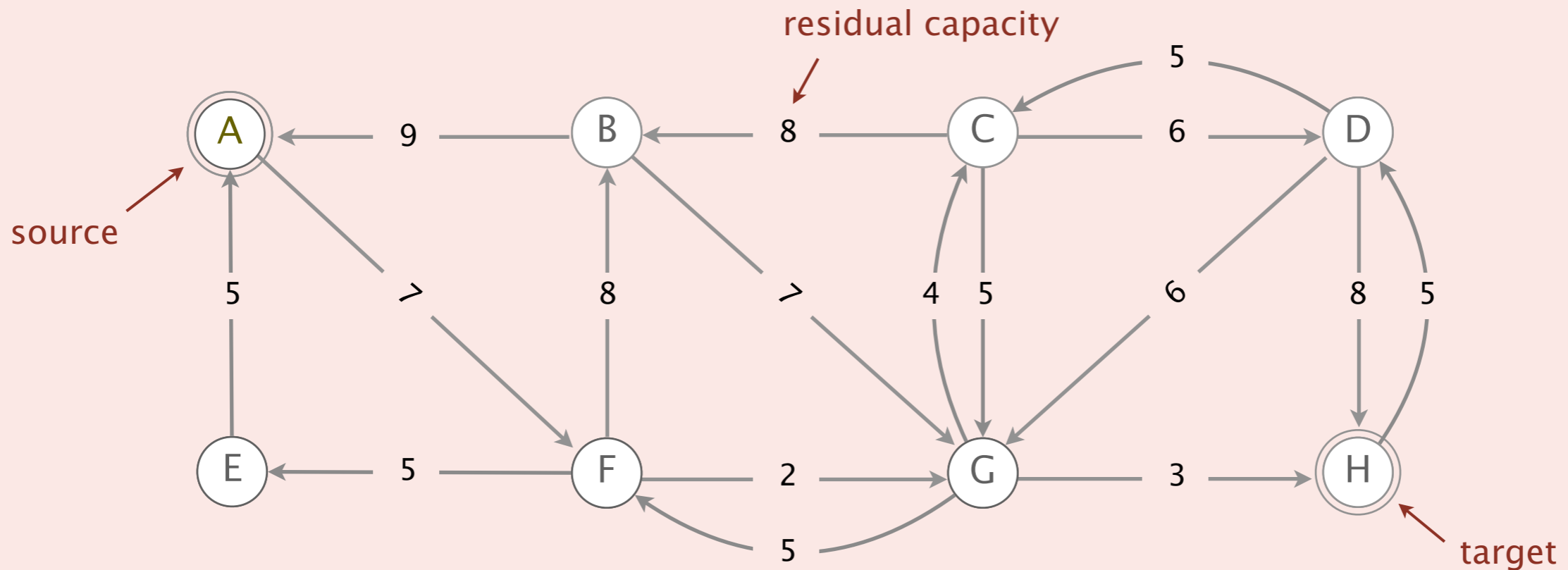
```
    ELSE  $f(e^{\text{reverse}}) \leftarrow f(e^{\text{reverse}}) - \delta$ .
```

```
RETURN  $f$ .
```



Which is the augmenting path of highest bottleneck capacity?

- A. $A \rightarrow F \rightarrow G \rightarrow H$
- B. $A \rightarrow B \rightarrow C \rightarrow D \rightarrow H$
- C. $A \rightarrow F \rightarrow B \rightarrow G \rightarrow H$
- D. $A \rightarrow F \rightarrow B \rightarrow G \rightarrow C \rightarrow D \rightarrow H$



Ford–Fulkerson algorithm

Ford–Fulkerson augmenting path algorithm.



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

FORD–FULKERSON(G)

FOREACH edge $e \in E$: $f(e) \leftarrow 0$.

$G_f \leftarrow$ residual network of G with respect to flow f .

WHILE (there exists an $s \rightsquigarrow t$ path P in G_f)

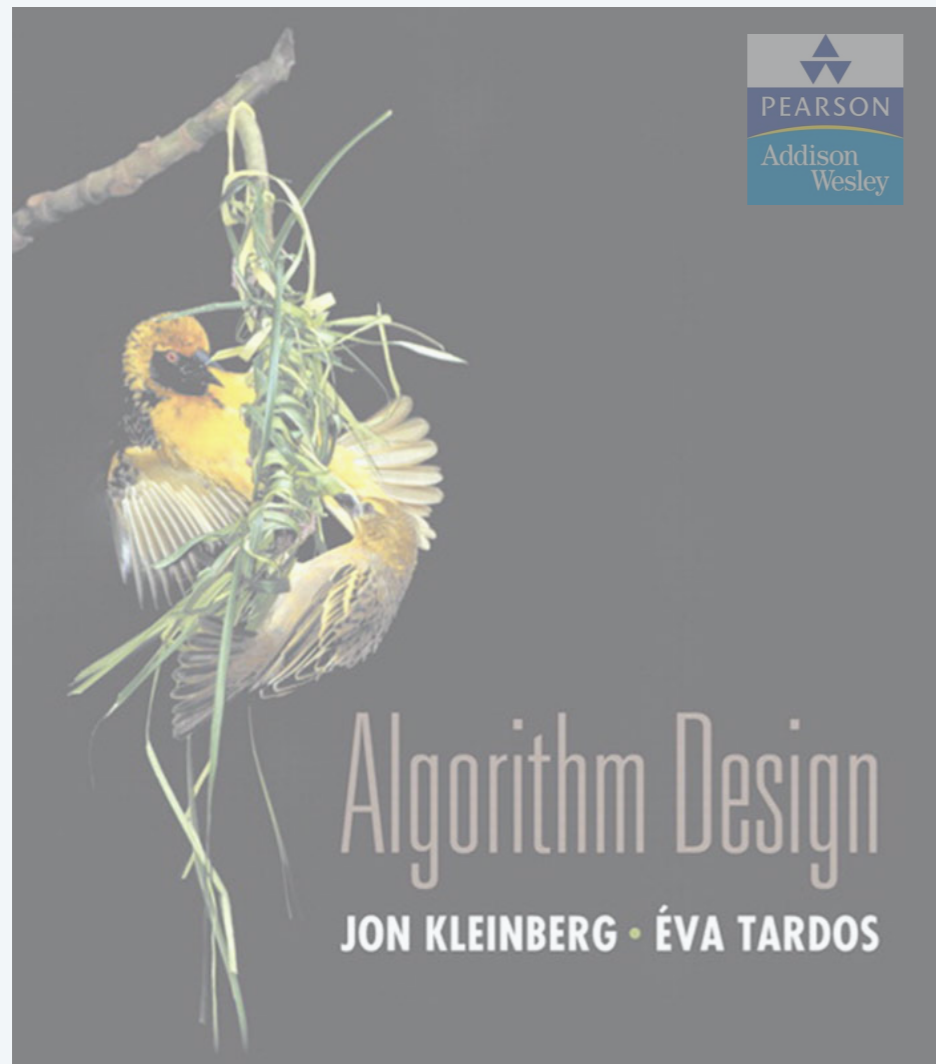
$f \leftarrow$ AUGMENT(f, c, P).

Update G_f .

RETURN f .

augmenting path

A red arrow points from the text 'augmenting path' to the variable 'P' in the 'WHILE' loop condition.



SECTION 7.2

7. NETWORK FLOW I

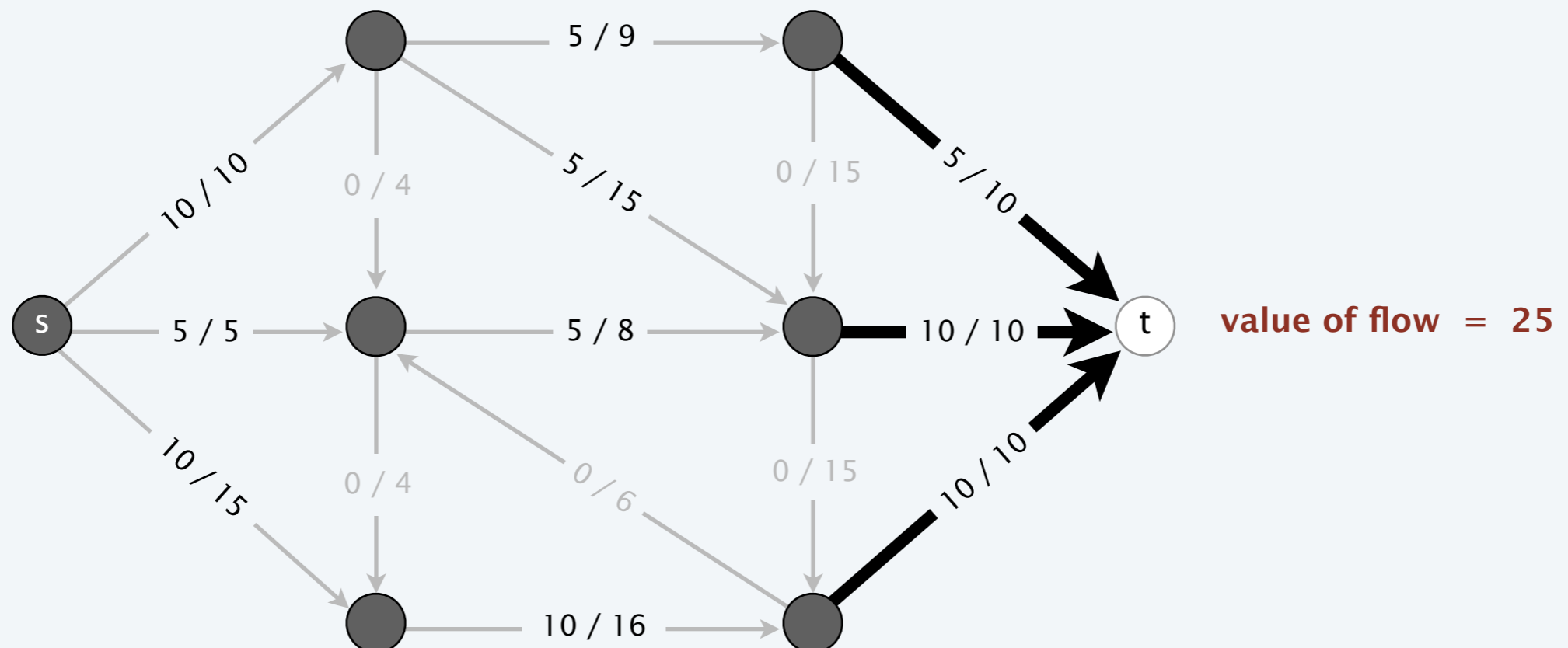
- ▶ *max-flow and min-cut problems*
- ▶ *Ford–Fulkerson algorithm*
- ▶ ***max-flow min-cut theorem***
- ▶ *capacity-scaling algorithm*
- ▶ *shortest augmenting paths*

Relationship between flows and cuts

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) .

$$\text{val}(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$

net flow across cut = 5 + 10 + 10 = 25

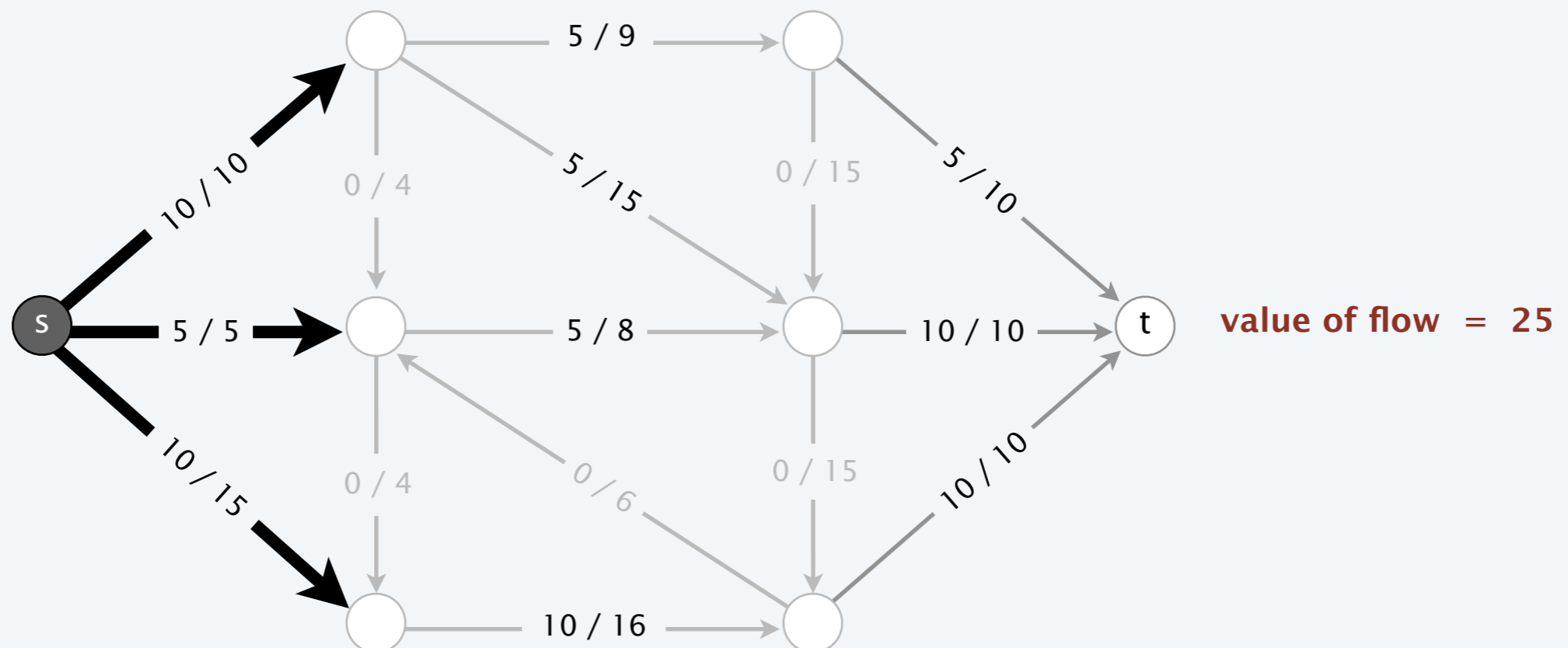


Relationship between flows and cuts

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) .

$$\text{val}(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$

net flow across cut = 10 + 5 + 10 = 25

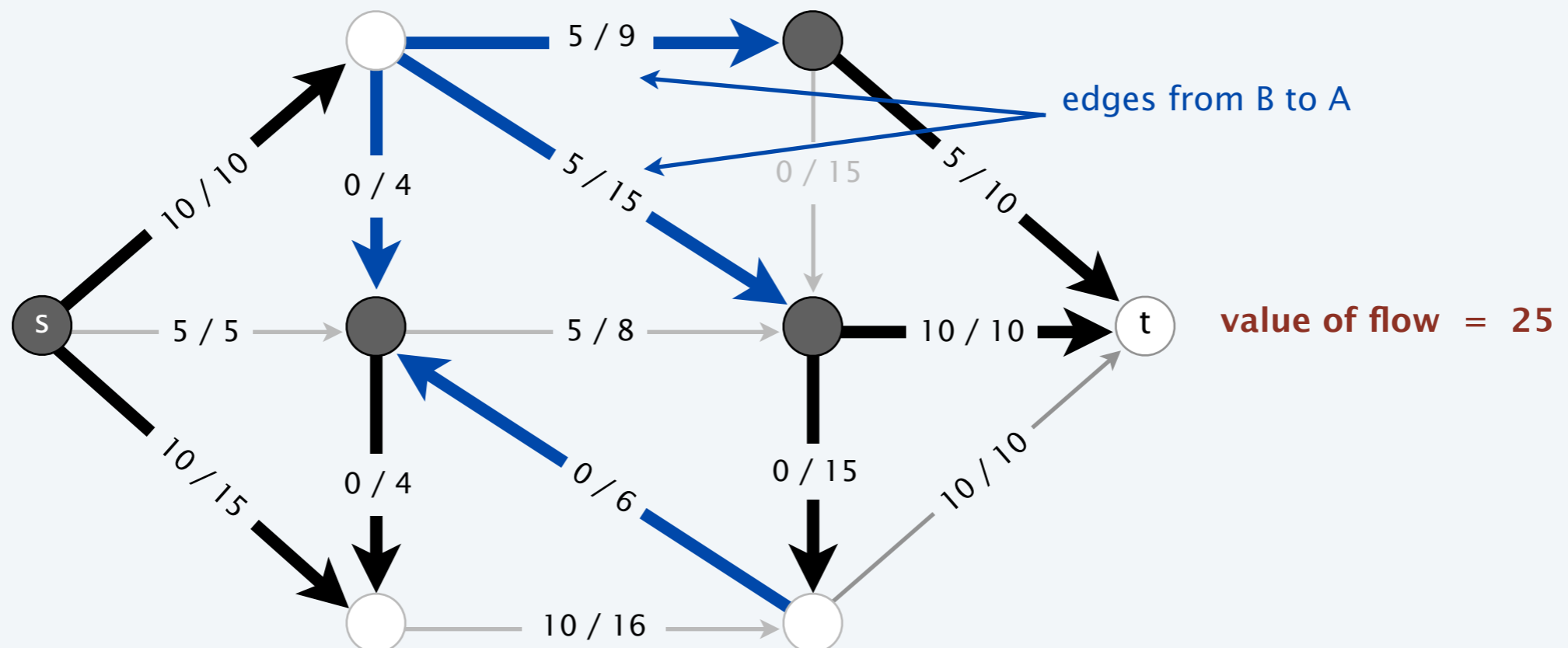


Relationship between flows and cuts

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) .

$$val(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$

$$\text{net flow across cut} = (10 + 10 + 5 + 10 + 0 + 0) - (5 + 5 + 0 + 0) = 25$$

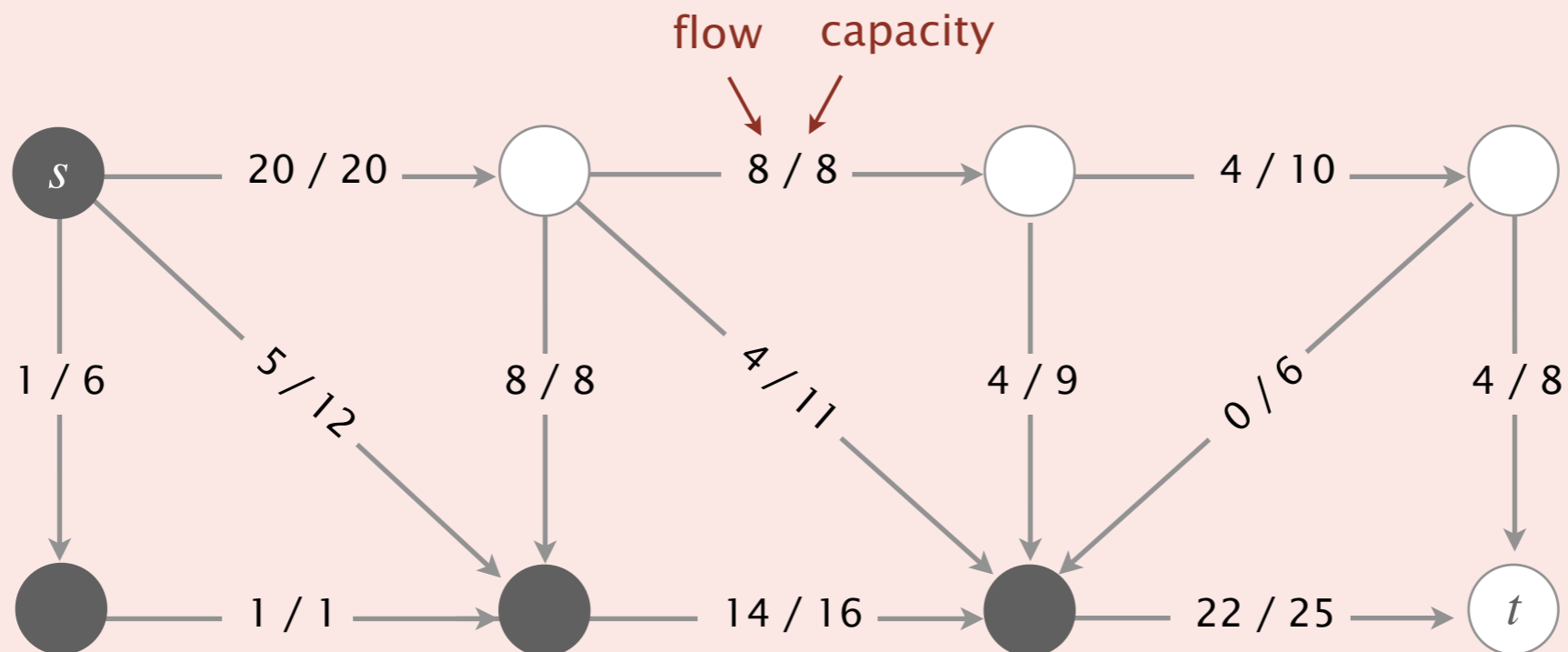


Network flow: quiz 3



Which is the net flow across the given cut?

- A. 11 (20 + 25 - 8 - 11 - 9 - 6)
- B. 26 (20 + 22 - 8 - 4 - 4)
- C. 42 (20 + 22)
- D. 45 (20 + 25)



Relationship between flows and cuts

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) .

$$\text{val}(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$

Pf.

$$\text{val}(f) = \sum_{e \text{ out of } s} f(e) - \sum_{e \text{ in to } s} f(e)$$

by flow conservation, all terms
except for $v = s$ are 0

$$\longrightarrow = \sum_{v \in A} \left(\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to } v} f(e) \right)$$

$$= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \quad \blacksquare$$

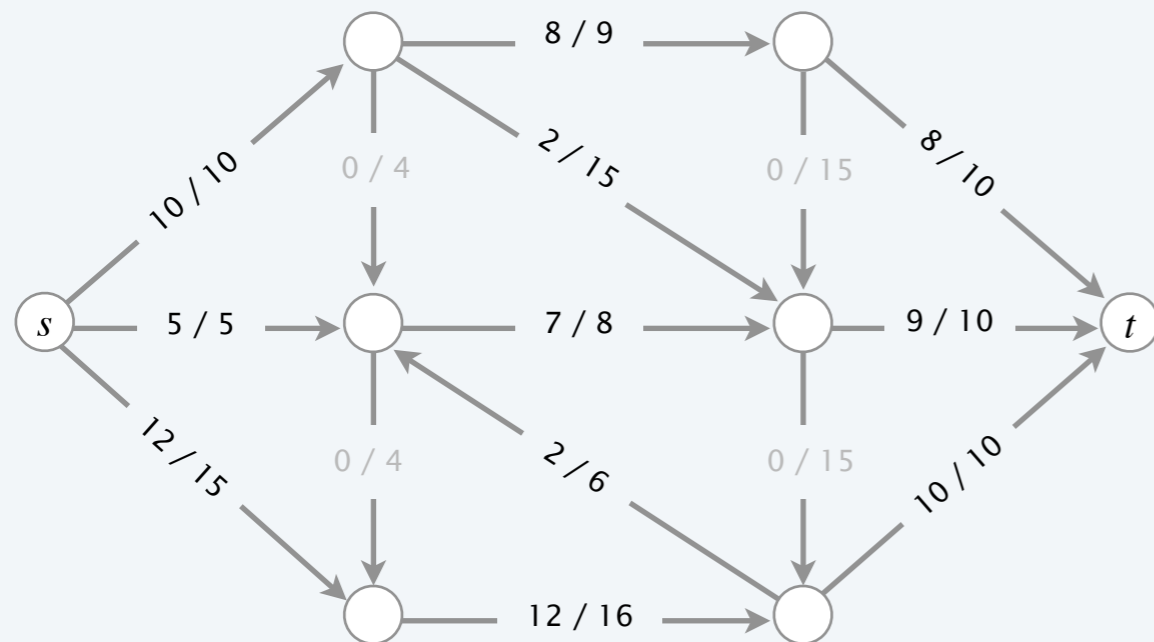
Relationship between flows and cuts

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

Pf.

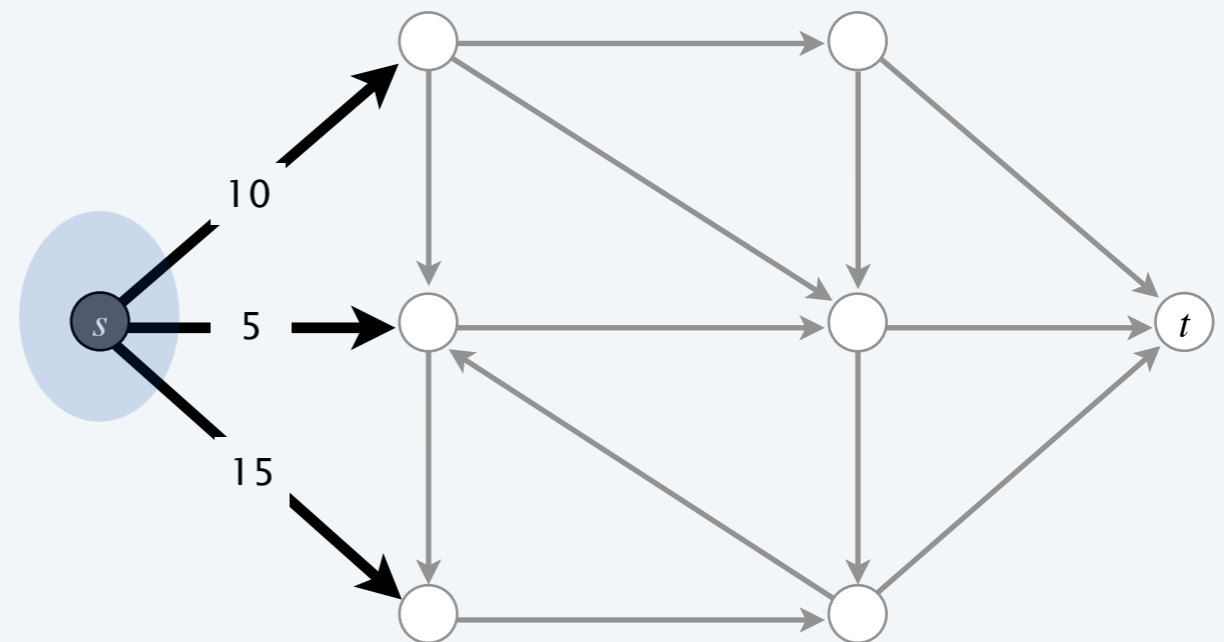
$$\begin{aligned}
 val(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\
 &\leq \sum_{e \text{ out of } A} f(e) \\
 &\leq \sum_{e \text{ out of } A} c(e) \\
 &= cap(A, B) \quad \blacksquare
 \end{aligned}$$

flow value lemma



value of flow = 27

\leq



capacity of cut = 30

Certificate of optimality

Corollary. Let f be a flow and let (A, B) be any cut.

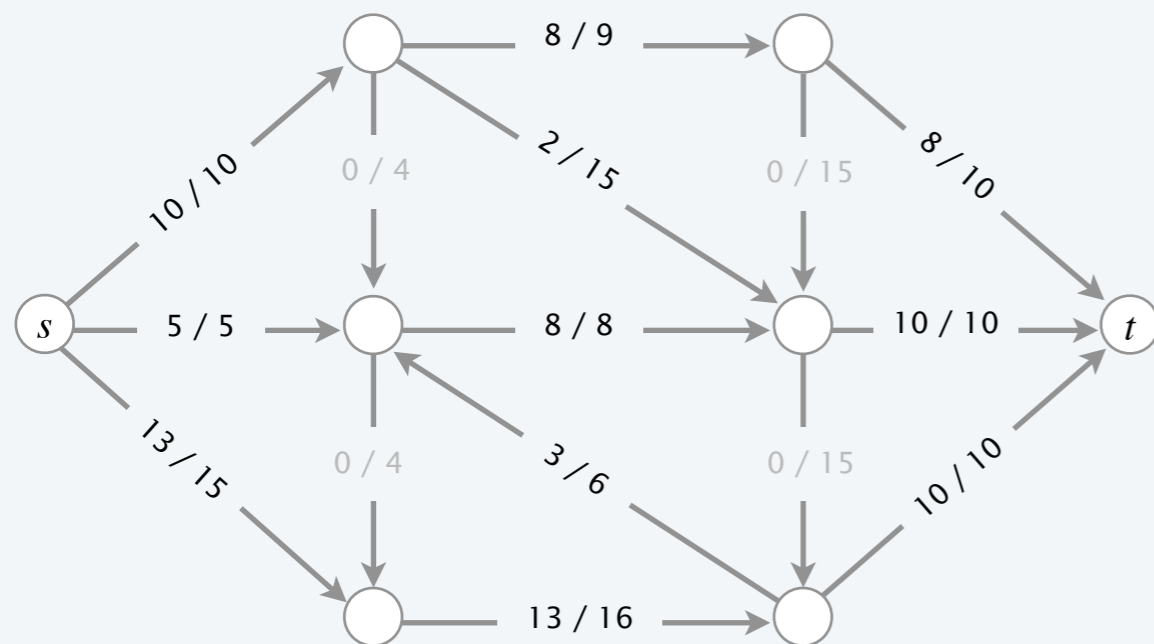
If $val(f) = cap(A, B)$, then f is a max flow and (A, B) is a min cut.

Pf.

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

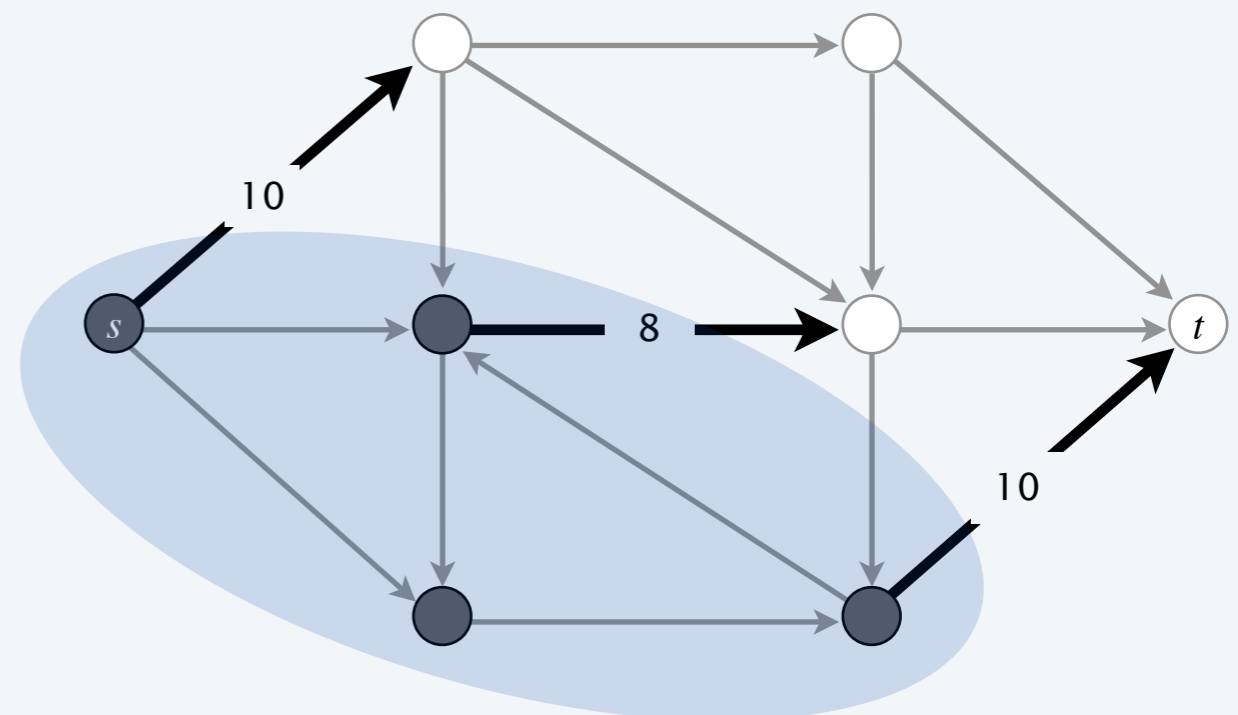
weak duality

weak duality



value of flow = 28

=



capacity of cut = 28

Max-flow min-cut theorem

Max-flow min-cut theorem. Value of a max flow = capacity of a min cut.

← strong duality

MAXIMAL FLOW THROUGH A NETWORK

L. R. FORD, JR. AND D. R. FULKERSON

Introduction. The problem discussed in this paper was formulated by T. Harris as follows:

“Consider a rail network connecting two cities by way of a number of intermediate cities, where each link of the network has a number assigned to it representing its capacity. Assuming a steady state condition, find a maximal flow from one given city to the other.”

ON THE MAX FLOW MIN CUT THEOREM OF NETWORKS

G. B. Dantzig
D. R. Fulkerson

P-826 5/4

April 15, 1955

A Note on the Maximum Flow Through a Network*

P. ELIAS†, A. FEINSTEIN‡, AND C. E. SHANNON§

Summary—This note discusses the problem of maximizing the rate of flow from one terminal to another, through a network which consists of a number of branches, each of which has a limited capacity. The main result is a theorem: The maximum possible flow from left to right through a network is equal to the minimum value among all simple cut-sets. This theorem is applied to solve a more general problem, in which a number of input nodes and a number of output nodes are used.

from one terminal to the other in the original network passes through at least one branch in the cut-set. In the network above, some examples of cut-sets are (d, e, f) , and (b, c, e, g, h) , (d, g, h, i) . By a *simple cut-set* we will mean a cut-set such that if any branch is omitted it is no longer a cut-set. Thus (d, e, f) and (b, c, e, g, h) are simple cut-sets while (d, a, b, c) is not. When a simple cut set is

Max-flow min-cut theorem

Max-flow min-cut theorem. 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 :

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

[i \Rightarrow ii]

- This is the weak duality corollary. ■

Max-flow min-cut theorem

Max-flow min-cut theorem. 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 :

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

[ii \Rightarrow iii] We prove contrapositive: \neg iii \Rightarrow \neg ii.

- 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. ■

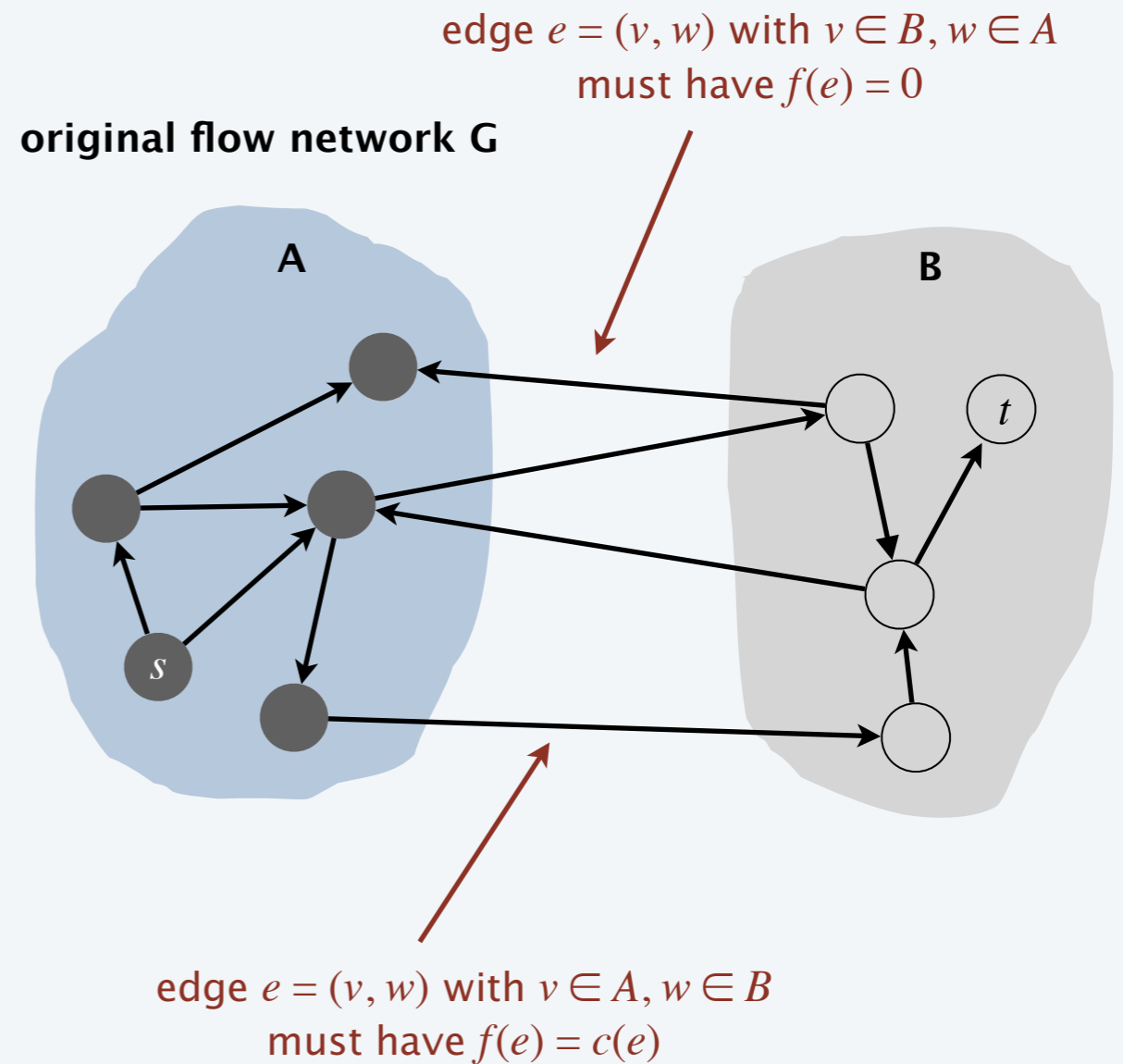
Max-flow min-cut theorem

[iii \Rightarrow i]

- 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 : $t \notin A$.

flow value lemma \nearrow

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

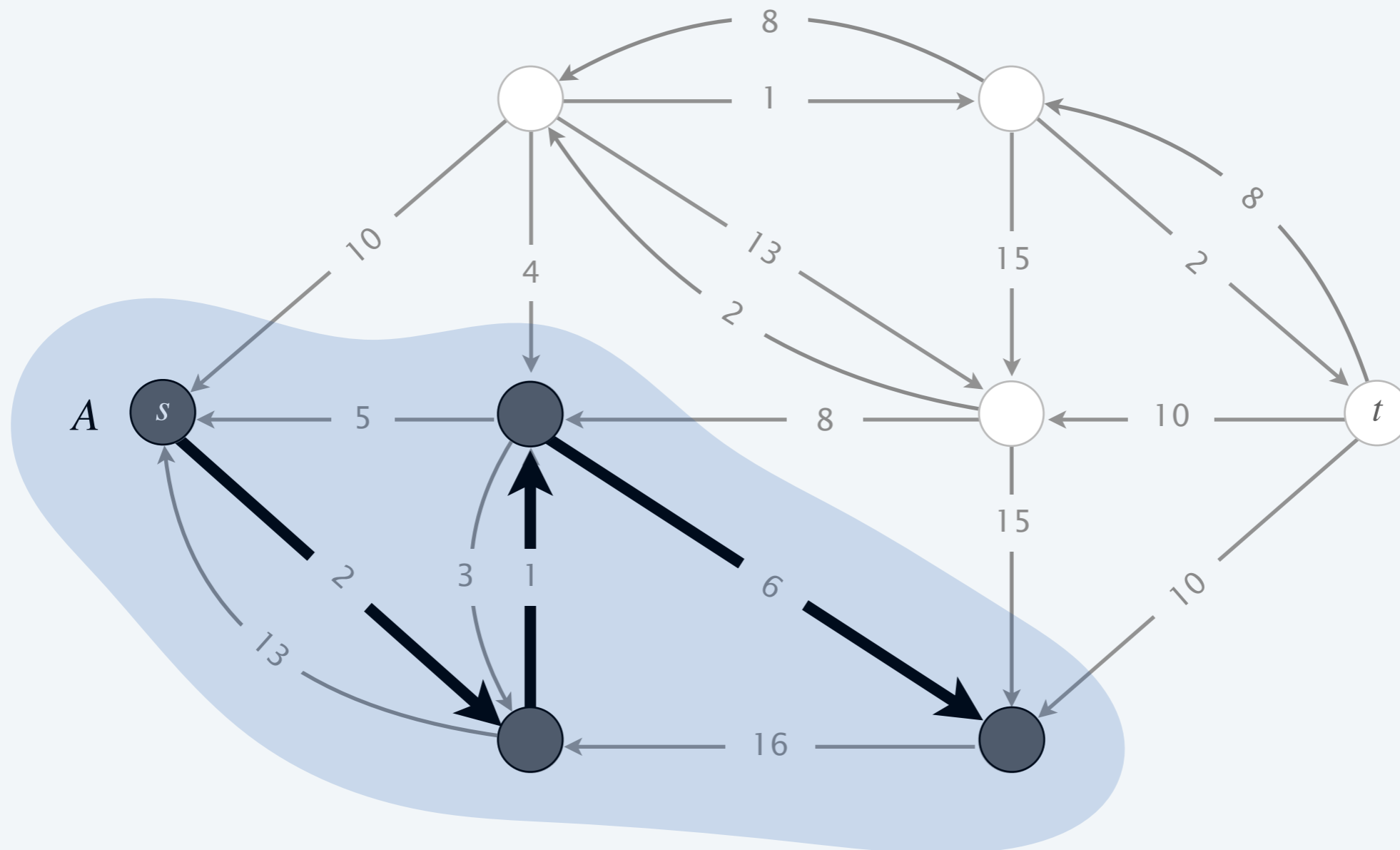


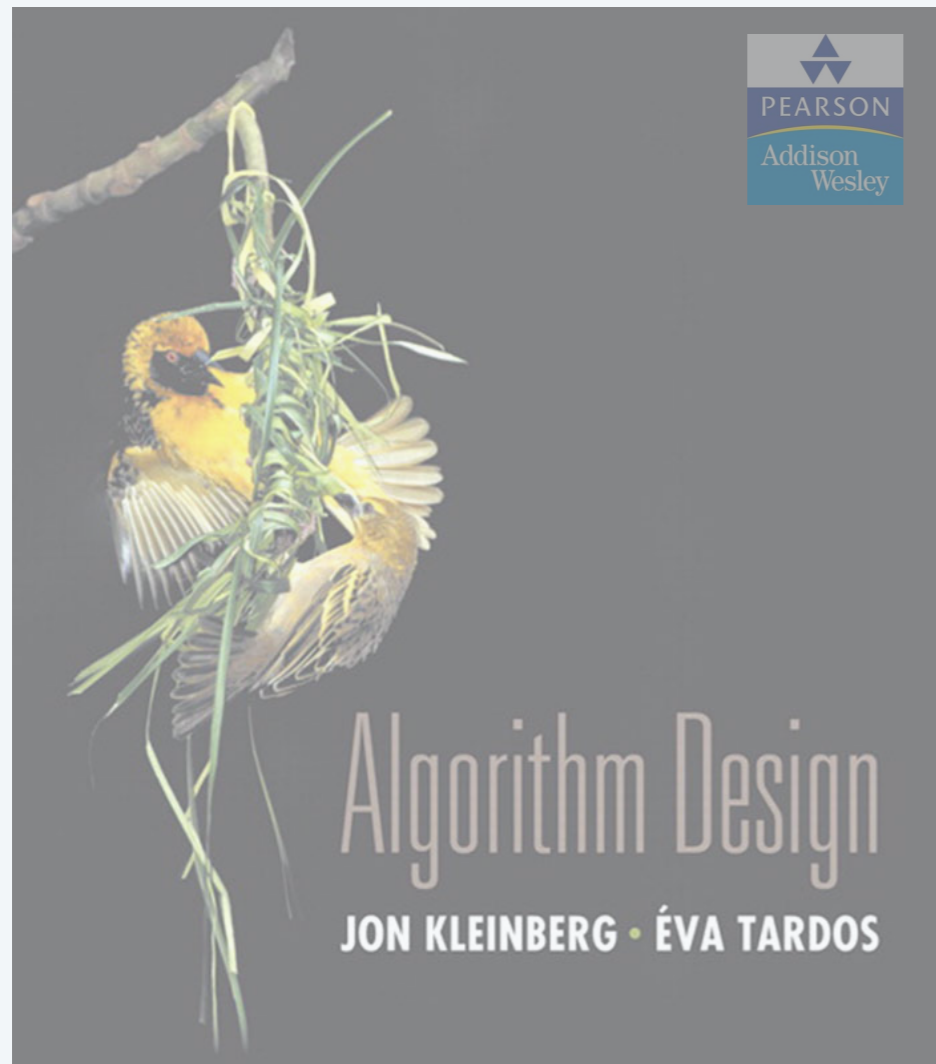
Computing a minimum cut from a maximum flow

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





SECTION 7.3

7. NETWORK FLOW I

- ▶ *max-flow and min-cut problems*
- ▶ *Ford–Fulkerson algorithm*
- ▶ *max-flow min-cut theorem*
- ▶ ***capacity-scaling algorithm***
- ▶ *shortest augmenting paths*

Analysis of Ford–Fulkerson algorithm (when capacities are integral)

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. ■

consider cut $A = \{s\}$
(assumes no parallel edges)

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. ■

$f(e)$ is an integer for every e

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?

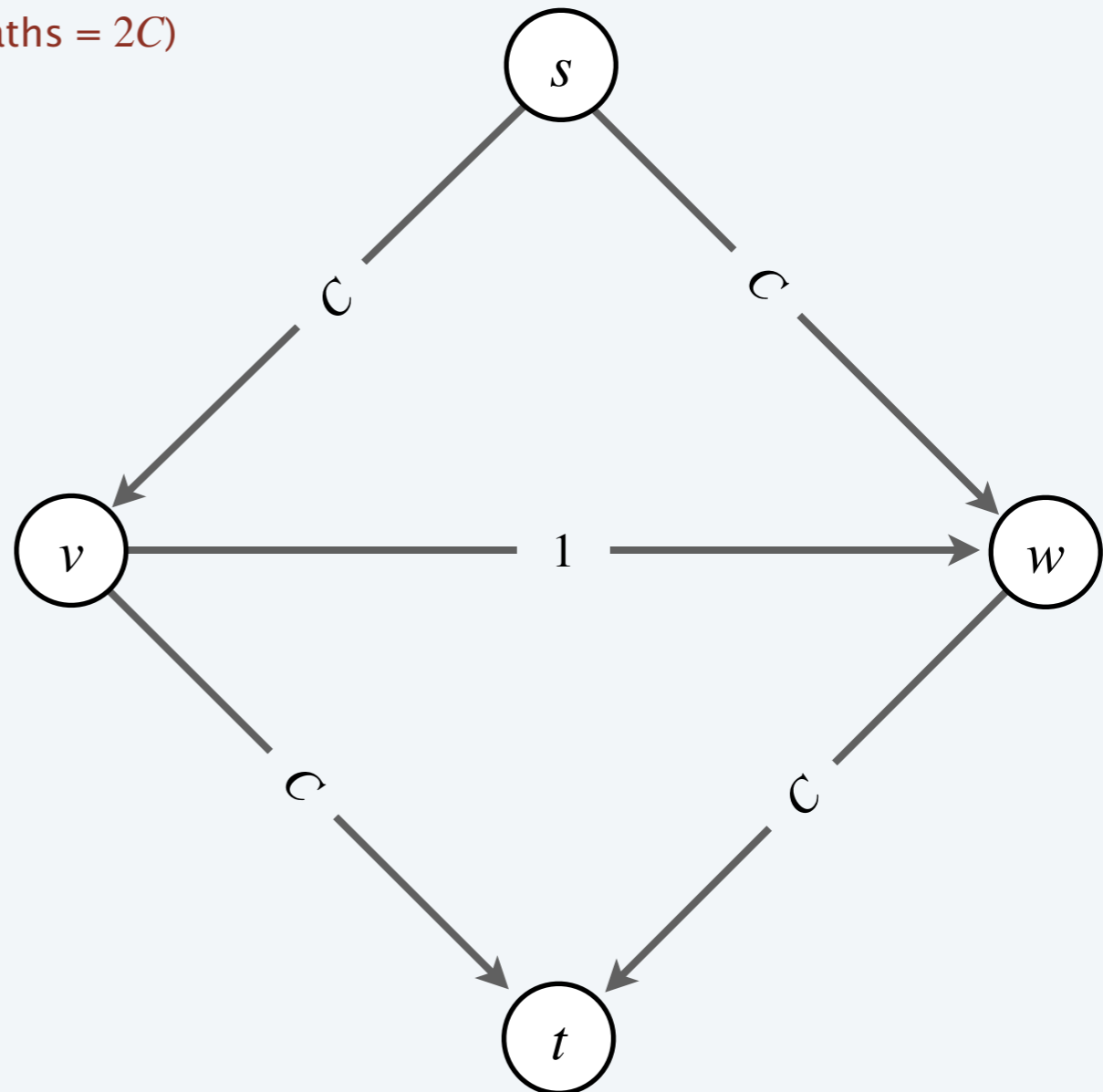
$m, n,$ and $\log C$



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

- $s \rightarrow v \rightarrow w \rightarrow t$
- $s \rightarrow w \rightarrow v \rightarrow t$
- $s \rightarrow v \rightarrow w \rightarrow t$
- $s \rightarrow w \rightarrow v \rightarrow t$
- ...
- $s \rightarrow v \rightarrow w \rightarrow t$
- $s \rightarrow w \rightarrow v \rightarrow t$

each augmenting path
sends only 1 unit of flow
(# augmenting paths = $2C$)





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.

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)!



Goal. Choose augmenting paths so that:

- Can find augmenting paths efficiently.
- Few iterations.

Choosing good augmenting paths

Choose augmenting paths with:

- Max bottleneck capacity (“fattest”). ← how to find?
- Sufficiently large bottleneck capacity. ← next
- Fewest edges. ← ahead

Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems

JACK EDMONDS

University of Waterloo, Waterloo, Ontario, Canada

AND

RICHARD M. KARP

University of California, Berkeley, California

ABSTRACT. This paper presents new algorithms for the maximum flow problem, the Hitchcock transportation problem, and the general minimum-cost flow problem. Upper bounds on the numbers of steps in these algorithms are derived, and are shown to compare favorably with upper bounds on the numbers of steps required by earlier algorithms.

Edmonds–Karp 1972 (USA)

Dokl. Akad. Nauk SSSR
Tom 194 (1970), No. 4

Soviet Math. Dokl.
Vol. 11 (1970), No. 5

ALGORITHM FOR SOLUTION OF A PROBLEM OF MAXIMUM FLOW IN A NETWORK WITH POWER ESTIMATION

UDC 518.5

E. A. DINIC

Different variants of the formulation of the problem of maximal stationary flow in a network and its many applications are given in [1]. There also is given an algorithm solving the problem in the case where the initial data are integers (or, what is equivalent, commensurable). In the general case this algorithm requires preliminary rounding off of the initial data, i.e. only an approximate solution of the problem is possible. In this connection the rapidity of convergence of the algorithm is inversely proportional to the relative precision.

Dinitz 1970 (Soviet Union)

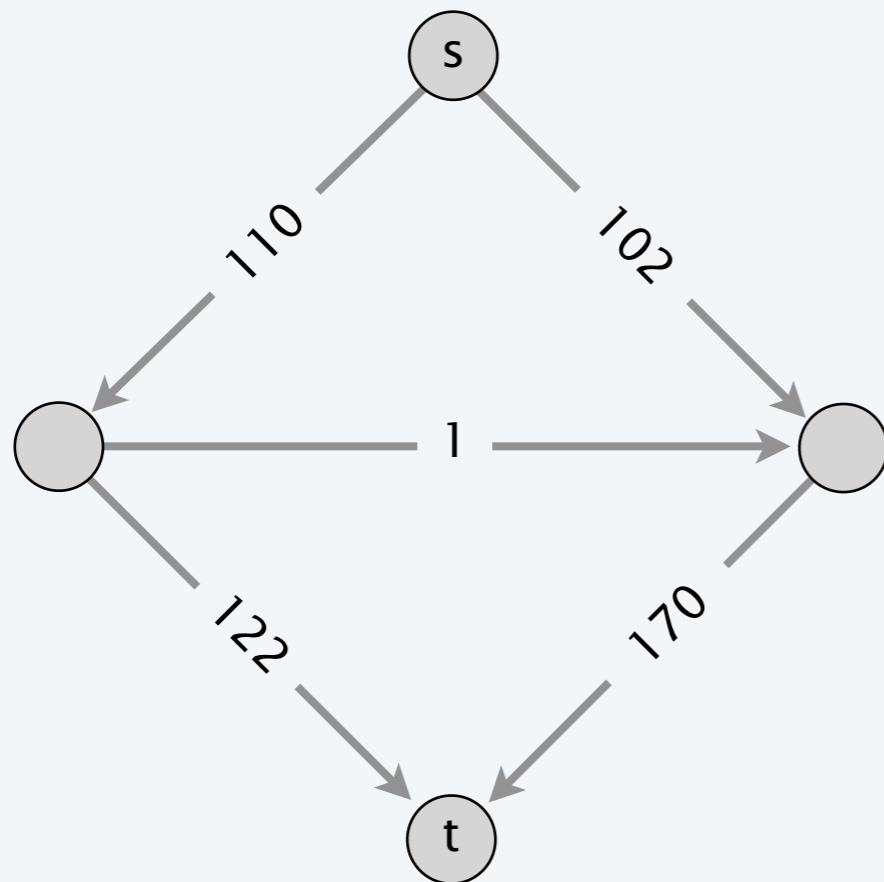
invented in response to a class
exercises by Adel'son-Vel'skiĭ

Capacity-scaling algorithm

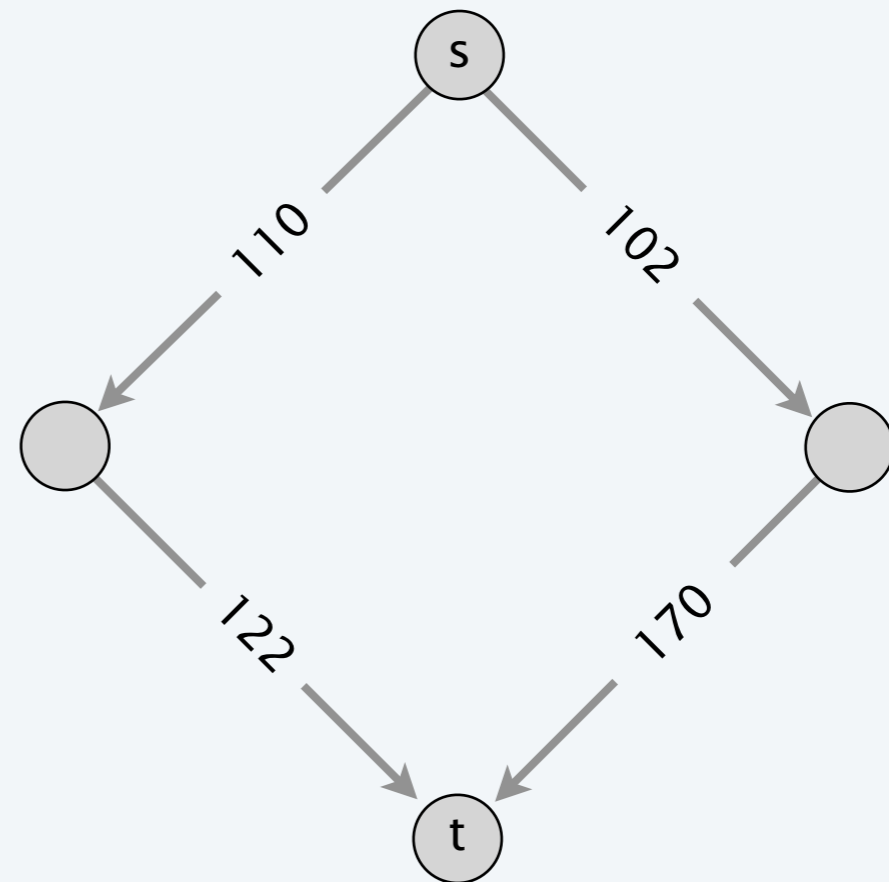
Overview. Choosing augmenting paths with “large” bottleneck capacity.

- 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 $\geq \Delta$.

though not necessarily largest



G_f



$G_f(\Delta), \Delta = 100$

Capacity-scaling algorithm

CAPACITY-SCALING(G)

FOREACH edge $e \in E : f(e) \leftarrow 0$.

$\Delta \leftarrow$ largest power of 2 $\leq C$.

WHILE ($\Delta \geq 1$)

$G_f(\Delta) \leftarrow$ Δ -residual network of G with respect to flow f .

WHILE (there exists an $s \rightsquigarrow t$ path P in $G_f(\Delta)$)

$f \leftarrow$ AUGMENT(f, c, P).

Update $G_f(\Delta)$.

$\Delta \leftarrow \Delta / 2$.

Δ -scaling phase

RETURN f .

Capacity-scaling algorithm: proof of 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 algorithm: analysis of running time

Lemma 1. There are $1 + \lceil \log_2 C \rceil$ 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 $\leq \text{val}(f) + m \Delta$.

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 $\leq \text{val}(f) + m (2 \Delta)$.
- Each augmentation in a Δ -phase increases $\text{val}(f)$ by at least Δ . ■

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

Pf.

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

Capacity-scaling algorithm: analysis of running time

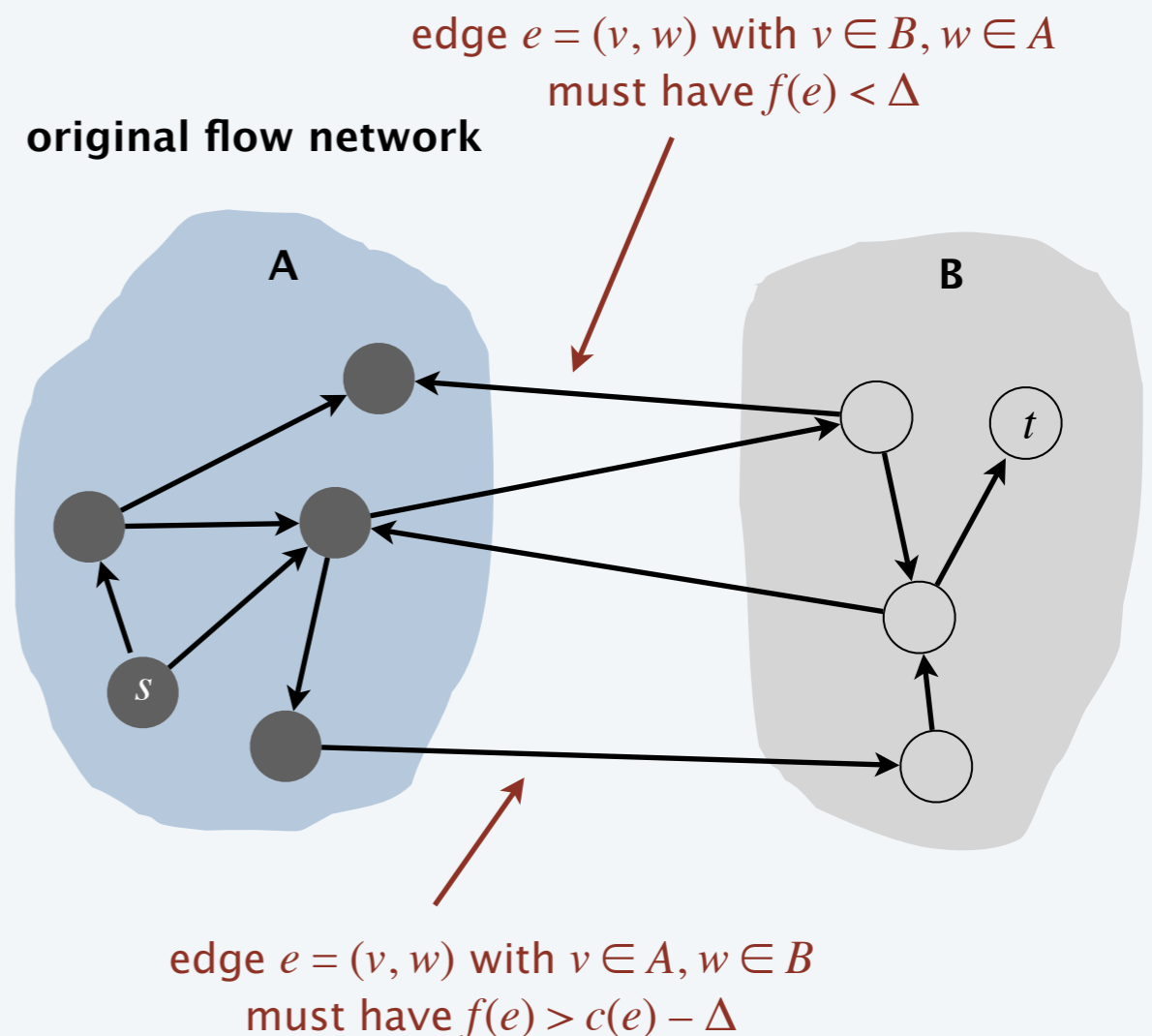
Lemma 2. Let f be the flow at the end of a Δ -scaling phase.

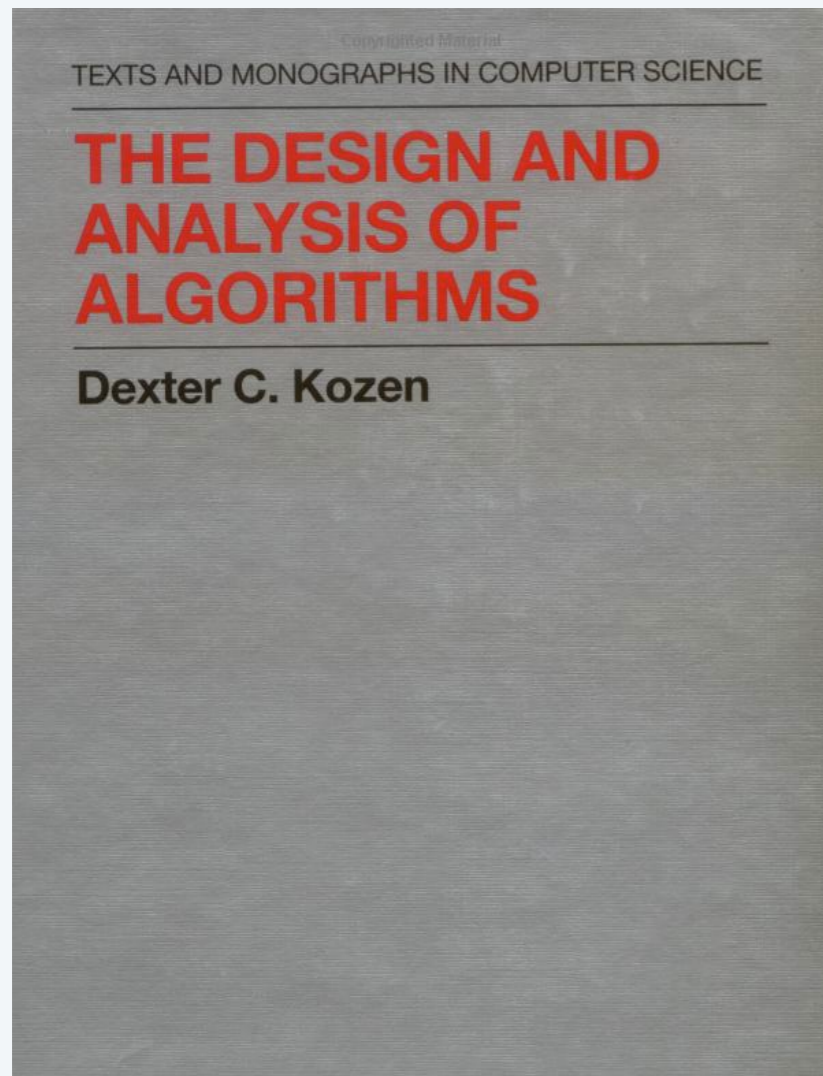
Then, the max-flow value $\leq \text{val}(f) + m \Delta$.

Pf.

- We show there exists a cut (A, B) such that $\text{cap}(A, B) \leq \text{val}(f) + m \Delta$.
- 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 : $t \notin A$.

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





SECTION 17.2

7. NETWORK FLOW I

- ▶ *max-flow and min-cut problems*
- ▶ *Ford–Fulkerson algorithm*
- ▶ *max-flow min-cut theorem*
- ▶ *capacity-scaling algorithm*
- ▶ ***shortest augmenting paths***

Shortest augmenting path

Q. How to choose next augmenting path in Ford–Fulkerson?

A. Pick one that uses the fewest edges.

 can find via BFS

SHORTEST-AUGMENTING-PATH(G)

FOREACH $e \in E$: $f(e) \leftarrow 0$.

$G_f \leftarrow$ residual network of G with respect to flow f .

WHILE (there exists an $s \rightsquigarrow t$ path in G_f)

$P \leftarrow$ BREADTH-FIRST-SEARCH(G_f).


$f \leftarrow$ AUGMENT(f, c, P).

Update G_f .

RETURN f .

Shortest augmenting path: overview of analysis


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


 number of edges

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.

Pf.

- $O(m)$ time to find a shortest augmenting path via BFS.
- There are $\leq m n$ augmentations.
 - at most m augmenting paths of length k  Lemma 1 + Lemma 2
 - at most $n-1$ different lengths ■

 augmenting paths are simple paths

Augmenting-path algorithms: summary

year	method	# augmentations	running time	
1955	augmenting path	$n C$	$O(m n C)$	
1972	fattest path	$m \log (mC)$	$O(m^2 \log n \log (mC))$	fat paths
1972	capacity scaling	$m \log C$	$O(m^2 \log C)$	
1985	improved capacity scaling	$m \log C$	$O(m n \log C)$	
1970	shortest augmenting path	$m n$	$O(m^2 n)$	shortest paths
1970	level graph	$m n$	$O(m n^2)$	
1983	dynamic trees	$m n$	$O(m n \log n)$	

augmenting-path algorithms with m edges, n nodes, and integer capacities between 1 and C

Maximum-flow algorithms: theory highlights

year	method	worst case	discovered by
1951	simplex	$O(m n^2 C)$	Dantzig
1955	augmenting paths	$O(m n C)$	Ford–Fulkerson
1970	shortest augmenting paths	$O(m n^2)$	Edmonds–Karp, Dinitz
1974	blocking flows	$O(n^3)$	Karzanov
1983	dynamic trees	$O(m n \log n)$	Sleator–Tarjan
1985	improved capacity scaling	$O(m n \log C)$	Gabow
1988	push-relabel	$O(m n \log (n^2 / m))$	Goldberg–Tarjan
1998	binary blocking flows	$O(m^{3/2} \log (n^2 / m) \log C)$	Goldberg–Rao
2013	compact networks	$O(m n)$	Orlin
2014	interior–point methods	$\tilde{O}(m n^{1/2} \log C)$	Lee–Sidford
2016	electrical flows	$\tilde{O}(m^{10/7} C^{1/7})$	Mądry
20xx		???	

max-flow algorithms with m edges, n nodes, and integer capacities between 1 and C

Maximum-flow algorithms: practice

Push–relabel algorithm (SECTION 7.4). [Goldberg–Tarjan 1988]

Increases flow one edge at a time instead of one augmenting path at a time.

A New Approach to the Maximum-Flow Problem

ANDREW V. GOLDBERG

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

ROBERT E. TARJAN

Princeton University, Princeton, New Jersey, and AT&T Bell Laboratories, Murray Hill, New Jersey

Abstract. All previously known efficient maximum-flow algorithms work by finding augmenting paths, either one path at a time (as in the original Ford and Fulkerson algorithm) or all shortest-length augmenting paths at once (using the layered network approach of Dinic). An alternative method based on the *preflow* concept of Karzanov is introduced. A preflow is like a flow, except that the total amount flowing into a vertex is allowed to exceed the total amount flowing out. The method maintains a preflow in the original network and pushes local flow excess toward the sink along what are estimated to be shortest paths. The algorithm and its analysis are simple and intuitive, yet the algorithm runs as fast as any other known method on dense graphs, achieving an $O(n^3)$ time bound on an n -vertex graph. By incorporating the dynamic tree data structure of Sleator and Tarjan, we obtain a version of the algorithm running in $O(nm \log(n^2/m))$ time on an n -vertex, m -edge graph. This is as fast as any known method for any graph density and faster on graphs of moderate density. The algorithm also admits efficient distributed and parallel implementations. A parallel implementation running in $O(n^2 \log n)$ time using n processors and $O(m)$ space is obtained. This time bound matches that of the Shiloach–Vishkin algorithm, which also uses n processors but requires $O(n^2)$ space.

Maximum-flow algorithms: practice

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.

On Implementing Push-Relabel Method for the Maximum Flow Problem

Boris V. Cherkassky¹ and Andrew V. Goldberg²

¹ Central Institute for Economics and Mathematics,
Krasikova St. 32, 117418, Moscow, Russia
cher@cemi.msk.su

² Computer Science Department, Stanford University
Stanford, CA 94305, USA
goldberg@cs.stanford.edu

Abstract. We study efficient implementations of the push-relabel method for the maximum flow problem. The resulting codes are faster than the previous codes, and much faster on some problem families. The speedup is due to the combination of heuristics used in our implementations. We also exhibit a family of problems for which the running time of all known methods seem to have a roughly quadratic growth rate.



ELSEVIER

European Journal of Operational Research 97 (1997) 509–542

EUROPEAN
JOURNAL
OF OPERATIONAL
RESEARCH

Theory and Methodology

Computational investigations of maximum flow algorithms

Ravindra K. Ahuja^a, Murali Kodialam^b, Ajay K. Mishra^c, James B. Orlin^{d,*}

^a Department of Industrial and Management Engineering, Indian Institute of Technology, Kanpur, 208 016, India

^b AT & T Bell Laboratories, Holmdel, NJ 07733, USA

^c KATZ Graduate School of Business, University of Pittsburgh, Pittsburgh, PA 15260, USA

^d Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

Received 30 August 1995; accepted 27 June 1996

Maximum-flow algorithms: practice

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

An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision

Yuri Boykov and Vladimir Kolmogorov*

Abstract

After [15, 31, 19, 8, 25, 5] minimum cut/maximum flow algorithms on graphs emerged as an increasingly useful tool for exact or approximate energy minimization in low-level vision. The combinatorial optimization literature provides many min-cut/max-flow algorithms with different polynomial time complexity. Their practical efficiency, however, has to date been studied mainly outside the scope of computer vision. The goal of this paper is to provide an experimental comparison of the efficiency of min-cut/max flow algorithms for applications in vision. We compare the running times of several standard algorithms, as well as a new algorithm that we have recently developed. The algorithms we study include both Goldberg-Tarjan style “push-relabel” methods and algorithms based on Ford-Fulkerson style “augmenting paths”. We benchmark these algorithms on a number of typical graphs in the contexts of image restoration, stereo, and segmentation. In many cases our new algorithm works several times faster than any of the other methods making near real-time performance possible. An implementation of our max-flow/min-cut algorithm is available upon request for research purposes.

VERMA, BATRA: MAXFLOW REVISITED

1

MaxFlow Revisited: An Empirical Comparison of Maxflow Algorithms for Dense Vision Problems

Tanmay Verma
tanmay08054@iitd.ac.in
Dhruv Batra
dbatra@ttic.edu

IIT-Delhi
Delhi, India
TTI-Chicago
Chicago, USA

Abstract

Algorithms for finding the maximum amount of flow possible in a network (or max-flow) play a central role in computer vision problems. We present an empirical comparison of different max-flow algorithms on modern problems. Our problem instances arise from energy minimization problems in Object Category Segmentation, Image Deconvolution, Super Resolution, Texture Restoration, Character Completion and 3D Segmentation. We compare 14 different implementations and find that the most popularly used implementation of Kolmogorov [5] is no longer the fastest algorithm available, especially for dense graphs.

Maximum-flow algorithms: Matlab



Documentation



CONTENTS

maxflow

R2018a

Maximum flow in graph

[collapse all in page](#)

Syntax

```
mf = maxflow(G,s,t)
mf = maxflow(G,s,t,algorithm)
[mf,GF] = maxflow(__)
[mf,GF,cs,ct] = maxflow(__)
```

Description

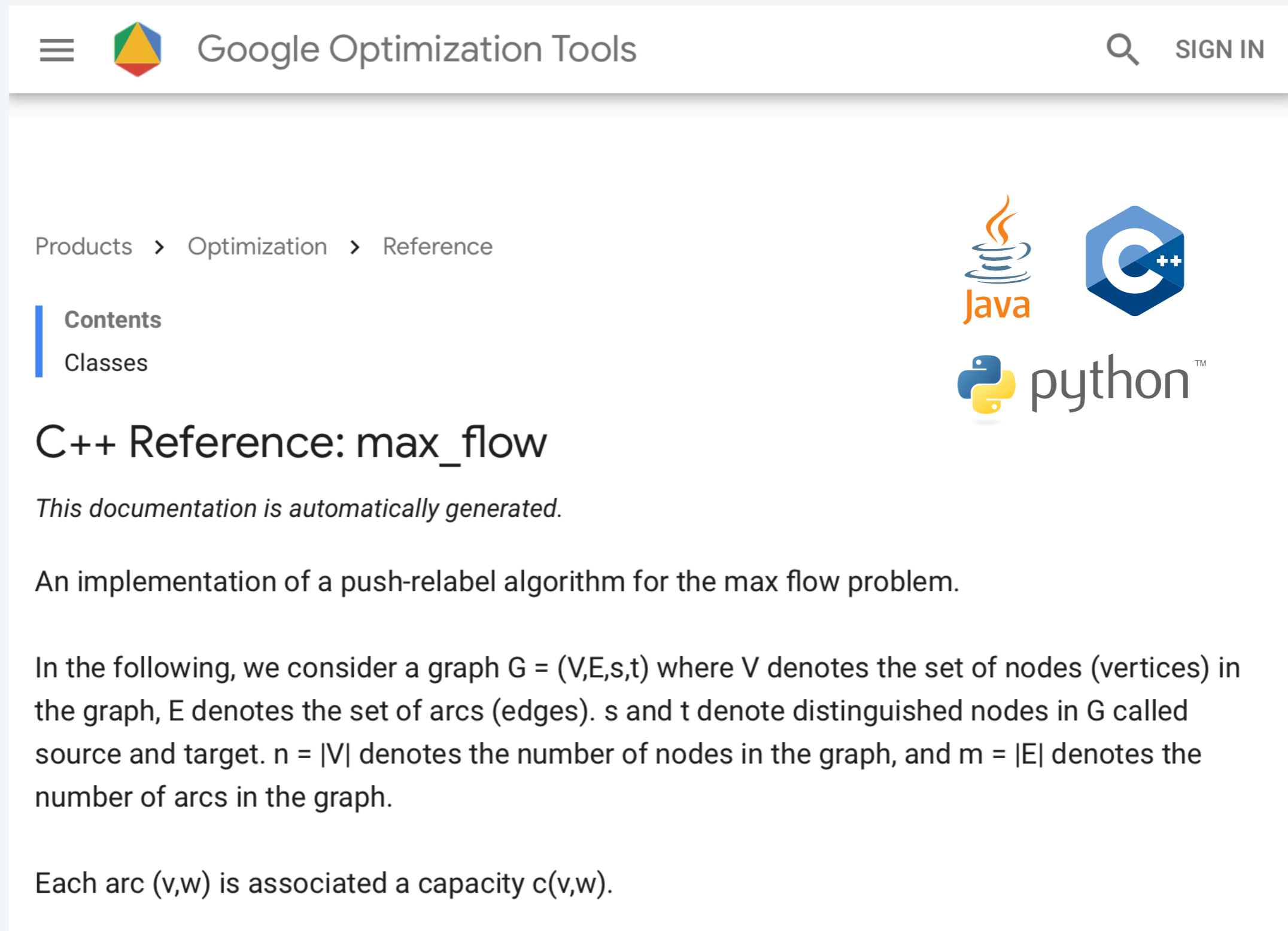
`mf = maxflow(G,s,t)` returns the [maximum flow](#) between nodes `s` and `t`. If graph `G` is unweighted (that is, `G.Edges` does not contain the variable `Weight`), then `maxflow` treats all graph edges as having a weight equal to 1.

[example](#)

`mf = maxflow(G,s,t,algorithm)` specifies the maximum flow algorithm to use. This syntax is only available if `G` is a directed graph.

[example](#)

Maximum-flow algorithms: Google



The screenshot shows the Google Optimization Tools website. At the top, there is a navigation bar with the Google logo, the text "Google Optimization Tools", a search icon, and a "SIGN IN" link. Below the navigation bar, there is a breadcrumb trail: "Products > Optimization > Reference". On the left side, there is a sidebar with a blue vertical bar and the text "Contents" and "Classes". The main content area features the logos for Java, C++, and Python. The title of the page is "C++ Reference: max_flow". Below the title, there is a sub-header: "This documentation is automatically generated." The main text describes the implementation of a push-relabel algorithm for the max flow problem. It defines a graph $G = (V, E, s, t)$ where V is the set of nodes, E is the set of arcs, s is the source, and t is the target. It also defines $n = |V|$ as the number of nodes and $m = |E|$ as the number of arcs. Finally, it states that each arc (v, w) is associated with a capacity $c(v, w)$.