Algorithm Design and Implementation

Principle of Algorithms X

Network Flow I

Guoqiang Li

School of Software, Shanghai Jiao Tong University

Max-Flow and Min-Cut Problems
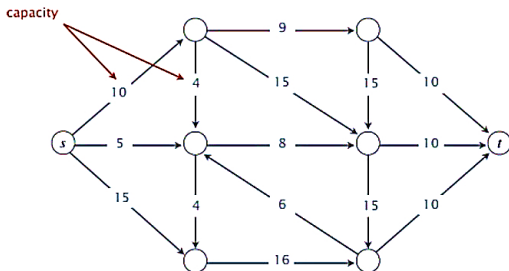
A Flow network

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

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

Intuition. Material flowing through a transportation network, which originates at source and is sent to sink.
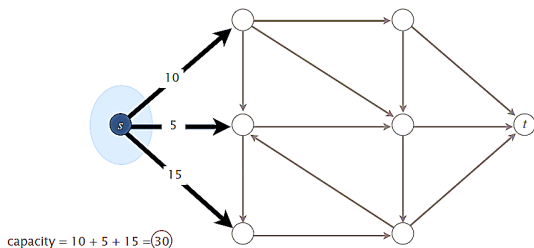
Minimum-cut problem

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

> **Definition**
>
> Its capacity is the sum of the capacities of the edges from A to B.
> $$\text{cap}(A, B) = \sum_{e \text{ out of } A} c(e)$$

10

5

15

capacity = 10 + 5 + 15 = 30

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

> **Definition**
>
> Its capacity is the sum of the capacities of the edges from A to B.
>
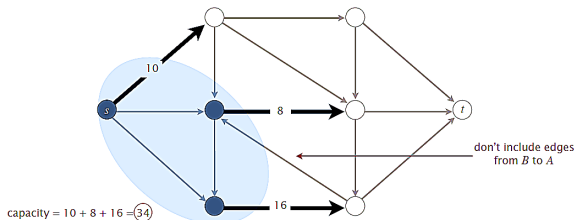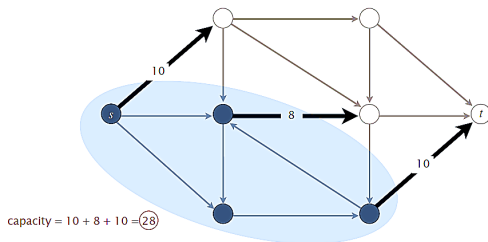> $$\text{cap}(A, B) = \sum_{e \text{ out of } A} c(e)$$

10

8

s t
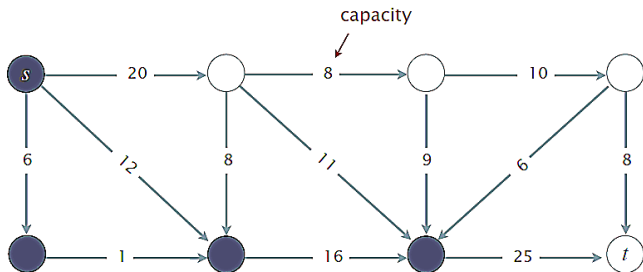
16

don't include edges
from B to A

capacity = 10 + 8 + 16 = 34

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

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 $(20 + 25 + 8 + 11 + 9 + 6)$

Definition

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

- For each $e \in E$: $0 \le f(e) \le c(e)$
- For each $v \in V - \{s, t\}$: $\sum\limits_{e \text{ in to } v} f(e) = \sum\limits_{e \text{ out of } v} f(e)$

inflow at $v = 5 + 5 + 0 = 10$
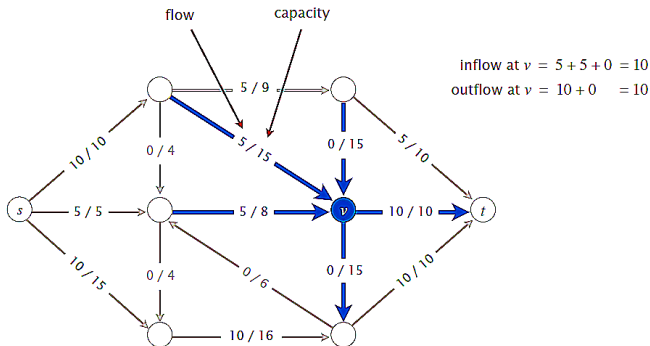outflow at $v = 10 + 0 = 10$
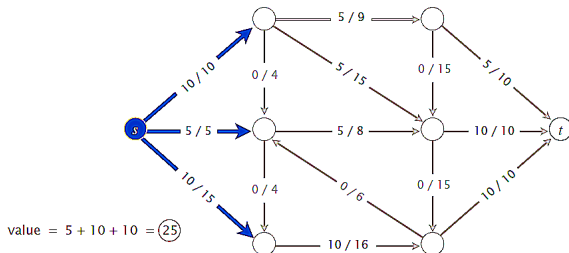
Maximum-flow problem

Definition

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

- For each $e \in E$: $\quad 0 \le f(e) \le c(e)$
- For each $v \in V - \{s, t\}$: $\quad \sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$
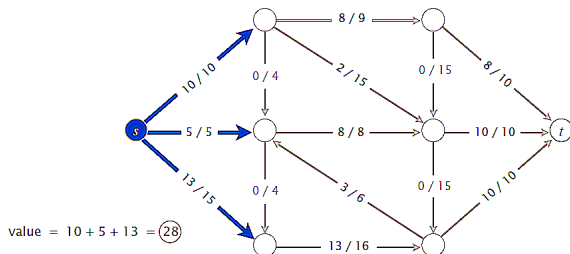
Definition

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

value = 5 + 10 + 10 = (25)

Max-flow problem. Find a flow of maximum value.
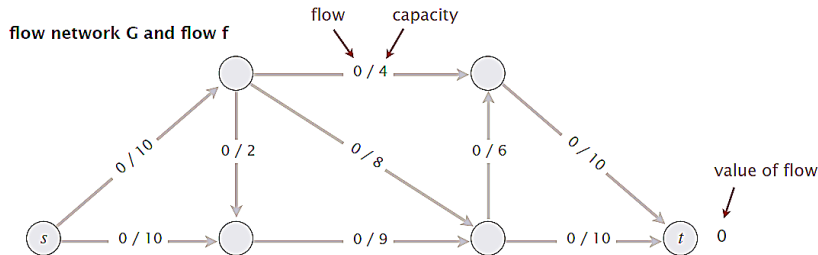
Ford-Fulkerson Algorithm

Toward a max-flow algorithm

Greedy algorithm.
- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \leadsto t$ 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

flow   capacity

value of flow

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

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.

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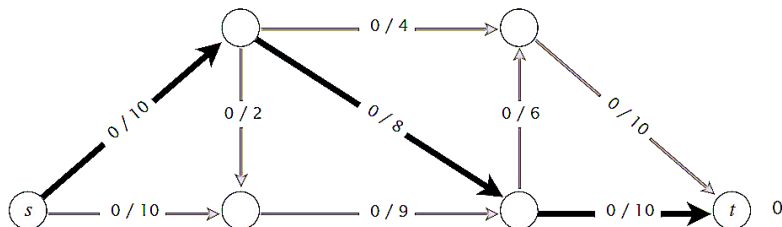- Augment flow along path P.
- Repeat until you get stuck.

Greedy algorithm.

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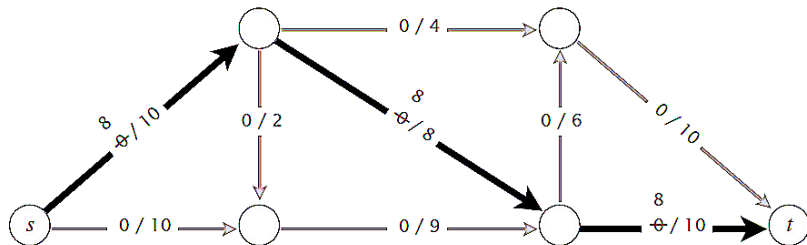- Augment flow along path P.
- Repeat until you get stuck.

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.
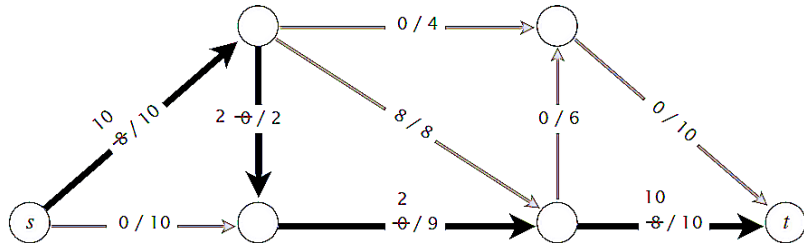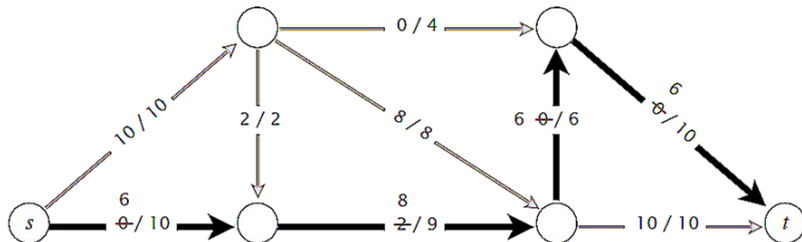- Repeat until you get stuck.

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.

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

flow network G
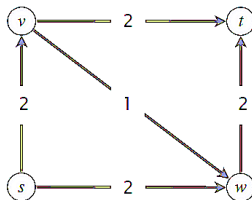
Bottom line. Need some mechanism to undo a bad decision.

Residual network

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

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

original flow network G

u —— 6 / 17 —— v
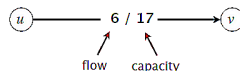
flow   capacity

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

- Undo flow sent.

Residual capacity

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

residual network G_f

u —— 11 —— v

u —— 6 —— v

residual capacity

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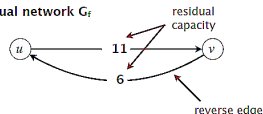
Residual network $G_f = (V, E_f, s, t, c_f)$

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

Definition

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

Definition

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

Augment(f,c,P)

$\delta \leftarrow$ bottleneck capacity of augmenting path P;

for *each edge* $e \in P$ **do**

    if ($e \in E$) **then** $f(e) \leftarrow f(e) + \delta$;

    else

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

    end

end

Return f;

Which is the augmenting path of highest bottleneck capacity?

1. $A \rightarrow F \rightarrow G \rightarrow H$
2. $A \rightarrow B \rightarrow C \rightarrow D \rightarrow H$
3. $A \rightarrow F \rightarrow B \rightarrow G \rightarrow H$
4. $A \rightarrow F \rightarrow B \rightarrow G \rightarrow C \rightarrow D \rightarrow H$
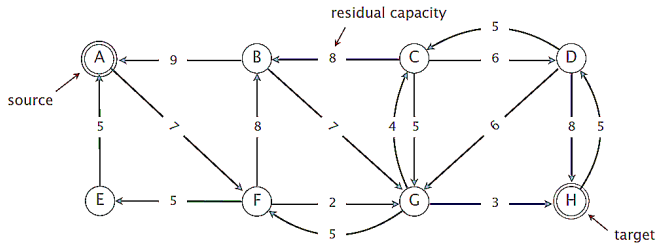
Ford–Fulkerson algorithm

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)

for *each edge $e \in E$* **do**
| $f(e) \leftarrow 0$
end
$G_f \leftarrow$ residual network of G with respect to flow f;
while *there exists an $s \rightsquigarrow t$ path P in G_f* **do**
| $f \leftarrow$ Augment(f,c,P);
| Update(G_f);
end
Return f;

Max-Flow Min-Cut Theorem

Relationship between flows and cuts

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).

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

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

value of flow = 25

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).

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

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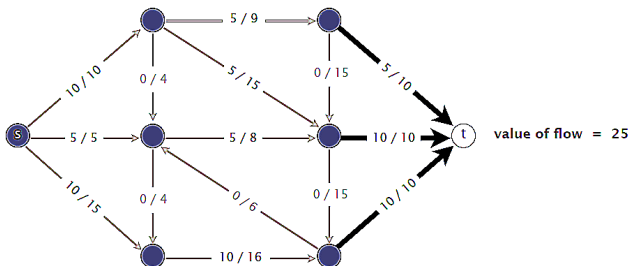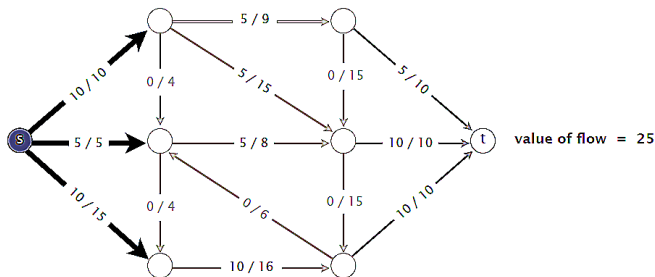
value of flow = 25

Relationship between flows and cuts

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

net flow across cut = (10 + 10 + 5 + 10 + 0 + 0) – (5 + 5 + 0 + 0) = 25

edges from B to A

value of flow = 25
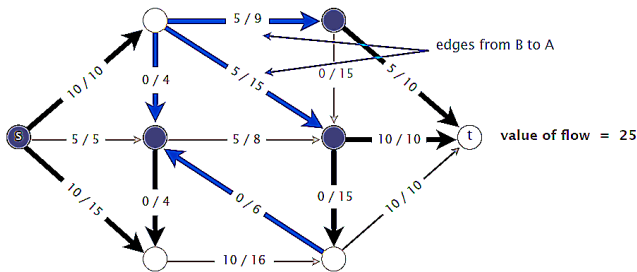
Which is the net flow across the given cut?

1. 11 $(20 + 25 - 8 - 11 - 9 - 6)$
2. 26 $(20 + 22 - 8 - 4 - 4)$
3. 42 $(20 + 22)$
4. 45 $(20 + 25)$
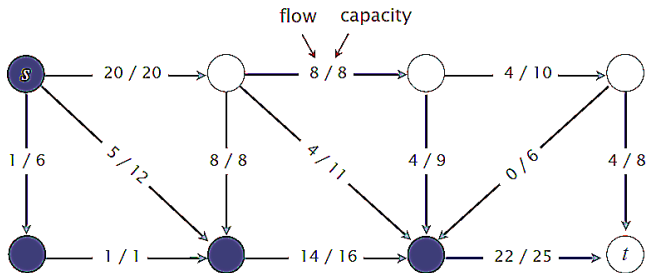
Lemma

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).

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

Proof.

$$
\begin{aligned}
val(f) &= \sum_{e \text{ out of } s} f(e) - \sum_{e \text{ in to } s} f(e) \\
&= \sum_{v \in A} \left(\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to } v} f(e) \right) \\
&= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e).
\end{aligned}
$$

Theorem

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

Proof.

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

$$\leq \sum_{e \text{ out of } A} f(e)$$

$$\leq \sum_{e \text{ out of } A} c(e)$$

$$= cap(A, B)$$
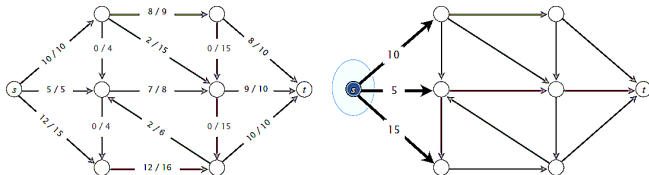
Corollary

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.

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

value of flow = 28   =   capacity of cut = 28

Max-Flow Min-Cut Theorem

Value of a max flow = capacity of a min cut.

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."
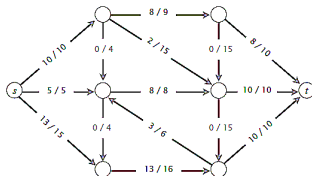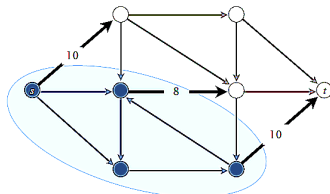
ON THE MAX FLOW MIN CUT THEOREM OF NETWORKS

G. B. Dantzig
D. R. Fulkerson

P-826

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, g, h, i) 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.

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

i. There exists a cut (A, B) such that $\mathsf{cap}(A, B) = \mathit{val}(f)$.

ii. f is a max flow.

iii. There is no augmenting path with respect to f.

[i$\Rightarrow$ ii]   This is the weak duality corollary.

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.

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

   i. There exists a cut (A, B) such that $\text{cap}(A, B) = val(f)$.

  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.

[iii ⇒ i]

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

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

$$= \sum_{e \text{ out of } A} c(e) - 0$$

$$= \text{cap}(A, B)$$
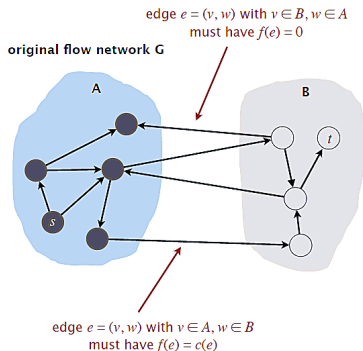
edge $e = (v, w)$ with $v \in B, w \in A$
must have $f(e) = 0$

original flow network G

A

B

edge $e = (v, w)$ with $v \in A, w \in B$
must have $f(e) = c(e)$

Capacity-Scaling Algorithm

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.

Proof. 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.*

Proof. Each augmentation increases the value of the flow by at least 1.

> **Corollary**
>
> *The running time of Ford–Fulkerson is $O(mnC)$.*

Proof. Can use either BFS or DFS to find an augmenting path in $O(m)$ time.

> **Integrality Theorem**
>
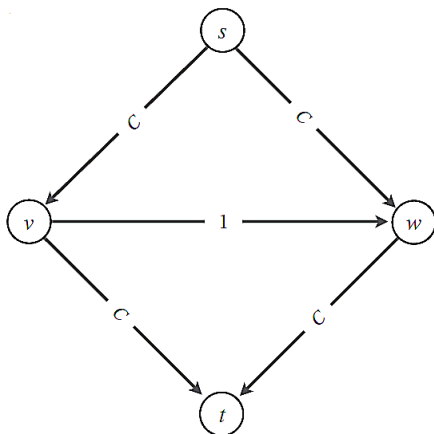> There exists an integral max flow f^*

Proof. Since Ford–Fulkerson terminates, theorem follows from integrality invariant.

Q. Is generic Ford–Fulkerson algorithm poly-time in input size?

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

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

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.

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.

Choose augmenting paths with:

- Max bottleneck capacity("fattest").
- Sufficiently large bottleneck capacity.
- Fewest edges.

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
Том 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.
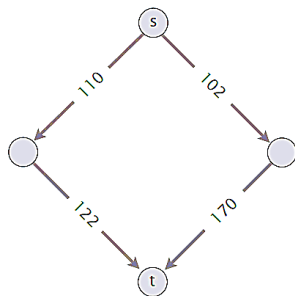
Dinitz 1970 (Soviet Union)

Capacity-scaling algorithm

Overview. Choosing augmented 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 $\geq \Delta$.
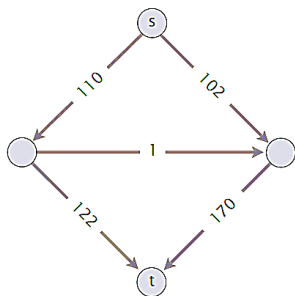
G_f $G_f(\Delta)$, $\Delta = 100$

Capacity-scaling algorithm

Capacity-Scaling(G)

for *each edge* $e \in E$ **do**
| $f(e) \leftarrow 0$
end
$\Delta \leftarrow$ largest power of $2 \leq C$;
while $\Delta \geq 1$ **do**
| $G_f(\Delta) \leftarrow \Delta$-residual network of G with respect to flow f;
| **while** *there exists an* $s \rightsquigarrow t$ *path* P *in* $G_f(\Delta)$ **do**
| | $f \leftarrow$ Augment(f, c, P);
| | Update($G_\Delta(f)$);
| **end**
| $\Delta = \Delta/2$;
end
Return f;

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

Invariant. The scaling parameter Δ is a power of 2.

Proof. 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.

Proof. Same as for generic Ford–Fulkerson.

Theorem

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

Proof.

- 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.

Lemma 1

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

Lemma 2

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

Lemma 3

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

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

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

Proof.

- Lemma 1+ Lemma 2 $\Rightarrow O(m \log C)$ augmentations.
- Finding an augmenting path takes $O(m)$ time.

Lemma 1

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

Proof. Initially $C/2 < \Delta \leq C$; Δ decreases by a factor of 2 in each iteration.

Lemma 2

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

Proof.

- Let f be the flow at the beginning of a Δ-scaling phase.
- Lemma 3 $\Rightarrow$ max-flow value $\leq val(f) + m(2\Delta)$.
- Each augmentation in a Δ-phase increases $val(f)$ by at least Δ.

> **Lemma 3**
>
> Let f be the flow at the end of a Δ-scaling phase.
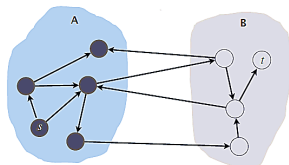> Then, the max-flow value $\leq \mathsf{val}(f) + m\Delta$.

Proof.
- 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 $f : t \notin A$.

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

original flow network

Shortest Augmenting Paths

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

A. Pick one that uses the fewest edges.

```
Shortest-Augmenting-Path(G)

for each edge e ∈ E do
 |  f(e) ← 0
end
Gf ← residual network of G with respect to flow f;
while there exists an s ⤳ t path in Gf do
 |  P ← BFS((Gf));
 |  f ← Augment(f, c, P);
 |  Update(Gf);
end
Return f;
```

Lemma 1

The length 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.

Proof.

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

Definition

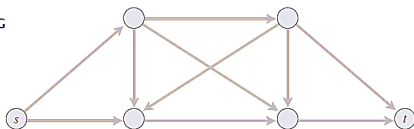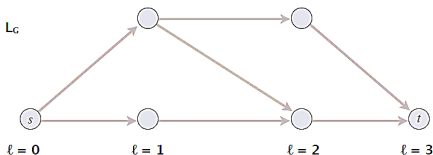Given a digraph $G = (V, E)$ with source s, its level graph is defined by:

- $\ell(v)$ = 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 $\ell(w) = \ell(v) + 1$.
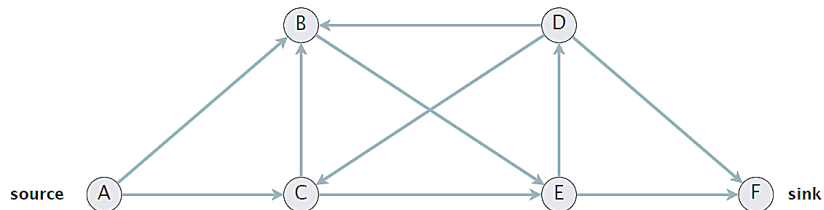

graph G

level graph L_G

$\ell = 0$ $\ell = 1$ $\ell = 2$ $\ell = 3$

Which edges are in the level graph of the following digraph?

A. $D \rightarrow F$

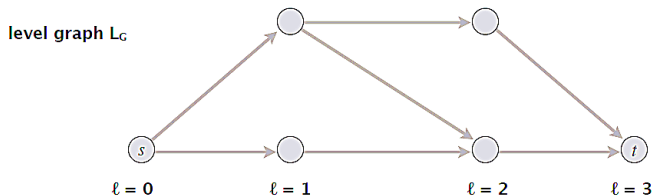B. $E \rightarrow F$

C. Both A and B.

D. Neither A nor B.

Definition

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

- $\ell(v)$ = 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 $\ell(w) = \ell(v) + 1$.

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

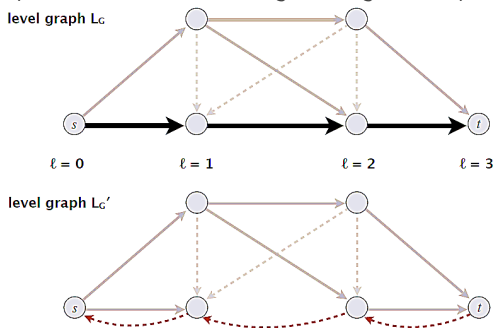level graph L_G

$\ell = 0$ $\ell = 1$ $\ell = 2$ $\ell = 3$

Lemma 1

The length of a shortest augmenting path never decreases.

Proof.

- 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 $G_{f'}$
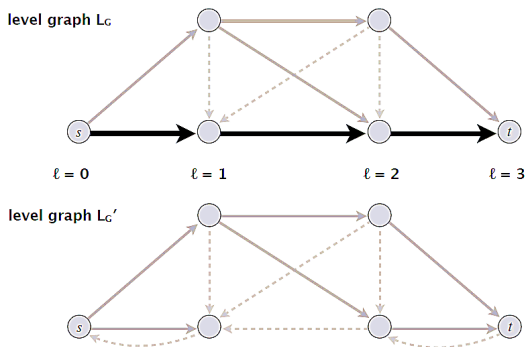 (any $s \rightsquigarrow t$ path that uses a back edge is longer than previous length)

Shortest augmenting path: analysis

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

Proof.
- 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.

Lemma 1

The length 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]

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

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

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