Network Flow*

Xiaofeng Gao

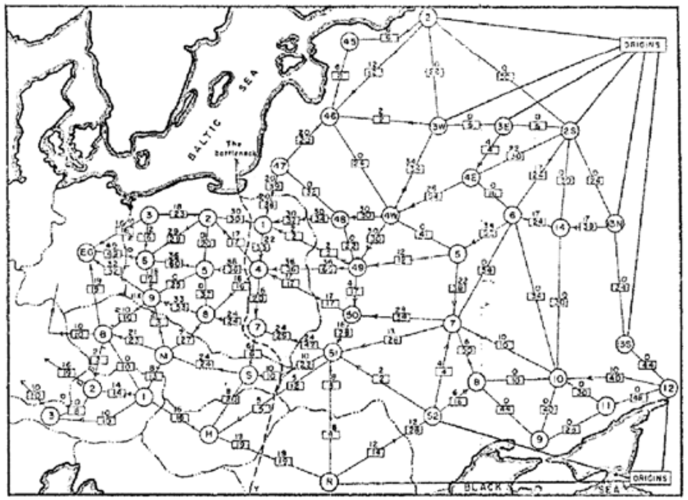
Department of Computer Science and Engineering
Shanghai Jiao Tong University, P.R.China

Algorithm Course: Shanghai Jiao Tong University

* Special thanks is given to *Prof. Kevin Wayne@Princeton*, *Prof. Charles E. Leiserson@MIT* for sharing their teaching materials, and also given to Mr. Mingding Liao from CS2013@SJTU for producing this lecture.

Outline

Soviet Rail Network, 1955



Reference: On the history of the transportation and maximum flow problems.

Alexander Schrijver in Math Programming, 91: 3, 2002.

Introduction: Maximum Flow and Minimum Cut

Maximum Flow and Minimum Cut

- Two very rich algorithmic problems.
- Cornerstone problems in combinatorial optimization.
- Beautiful mathematical duality.

Nontrivial Applications / Reductions

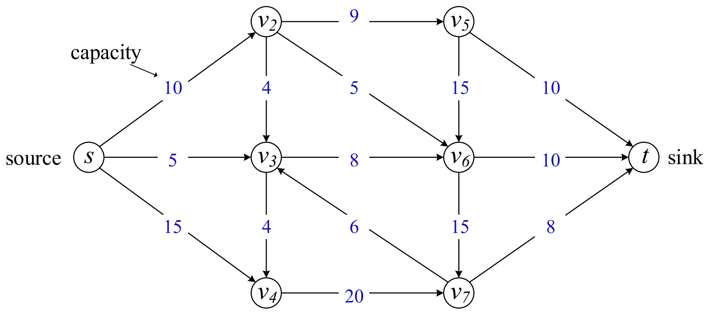
- Data mining.
- Open-pit mining.
- Project selection.
- Airline scheduling.
- Bipartite matching.
- Baseball elimination.
- Network connectivity.
- Network reliability.
- Distributed computing.
- Egalitarian stable matching.
- Security of statistical data.
- Network intrusion detection.
- Multi-camera scene reconstruction.
- Many many more ?

Flow Network

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

- Directed graph $G = (V, E)$, with source $s \in V$ and sink $t \in V$.
- Assume all nodes are reachable from s, no parallel edges.
- Capacity $c(e) > 0$ for each edge $e \in E$.

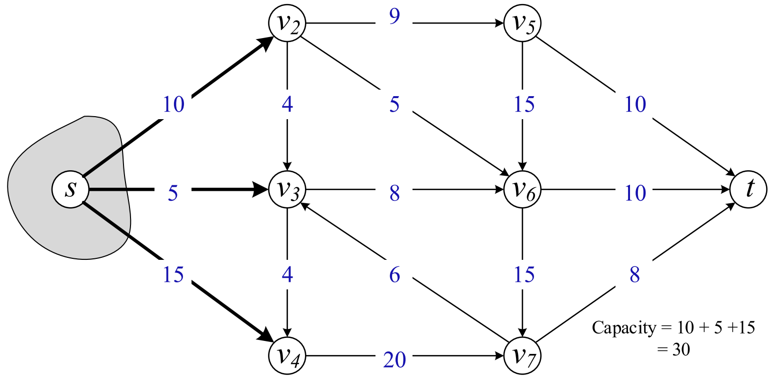
Intuition. Material flowing through a transportation network; material originates at source and is sent to sink.



Cuts

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

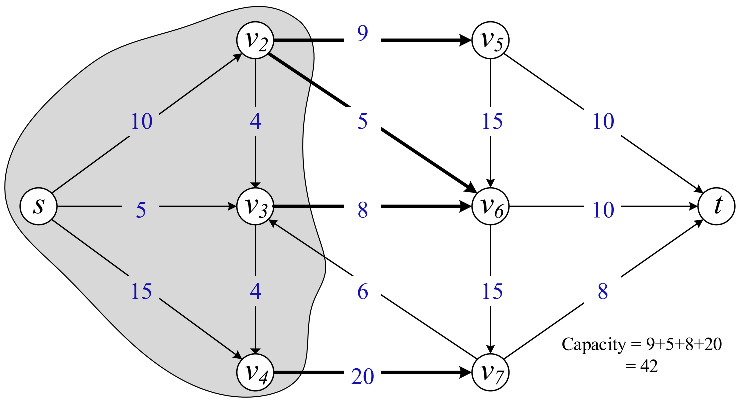
Def. The capacity of a cut (A, B) is $cap(A, B) = \sum_{e \text{ out of } A} c(e)$



Cuts

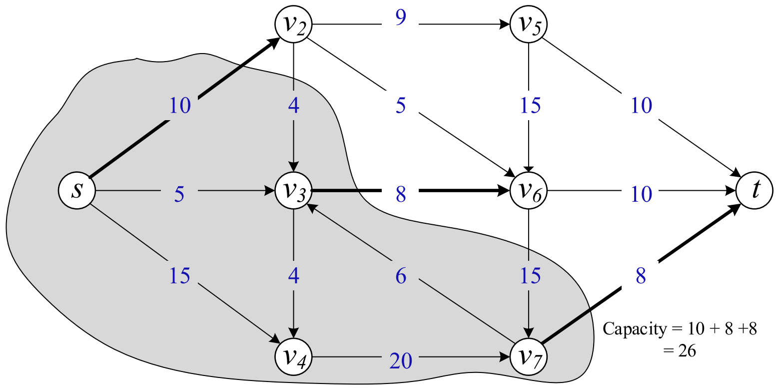
Def. An s-t cut is a partition (A, B) of V with $s \in A$ and $t \in B$.

Def. The capacity of a cut (A, B) is $cap(A, B) = \sum_{e \text{ out of } A} c(e)$



Minimum Cut Problem

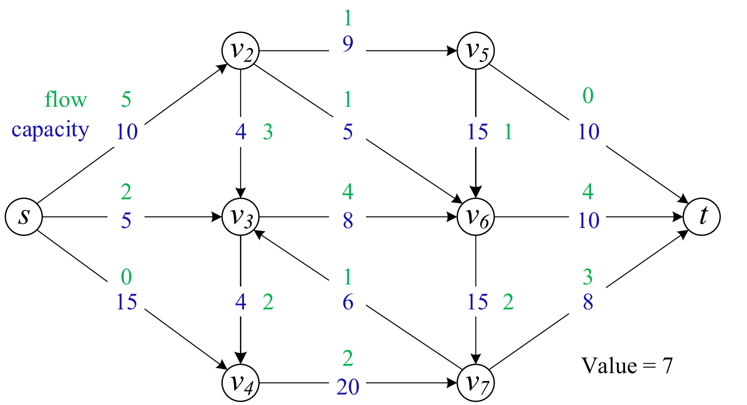
Minimum s-t Cut: Find an s-t cut of minimum capacity.



Flows

Def. An *s-t* flow f is a function that satisfies:

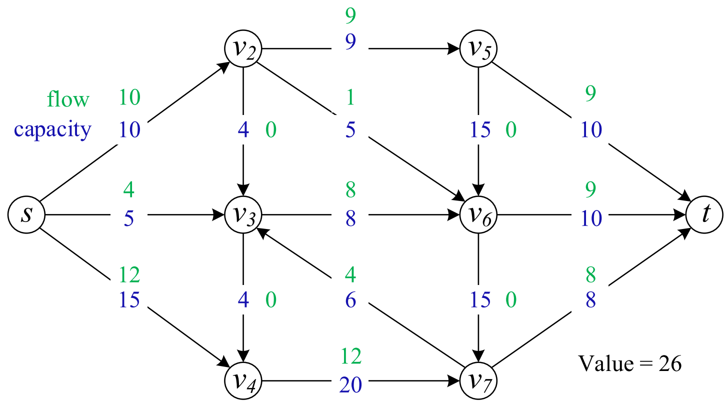
- Capacity: for each $e \in E$: $0 \leq f(e) \leq c(e)$
- Conservation: for each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$



Maximum Flow Problem

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

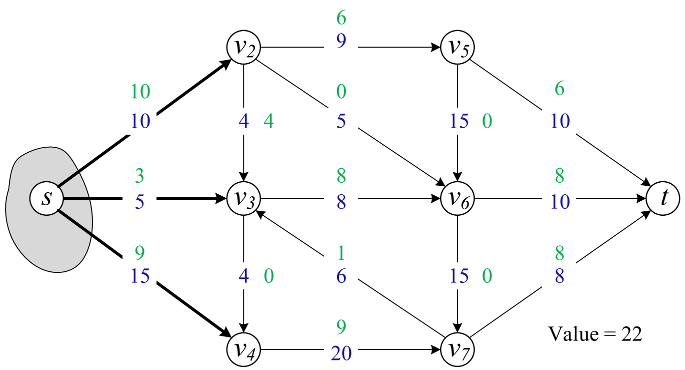
Max flow problem. Find an *s-t* flow of maximum value.



Flows and Cuts

Flow Value Lemma. Let f be any flow, and let (A, B) be any *s-t* cut. Then, the value of f equals to the net flow across the cut (A, B).

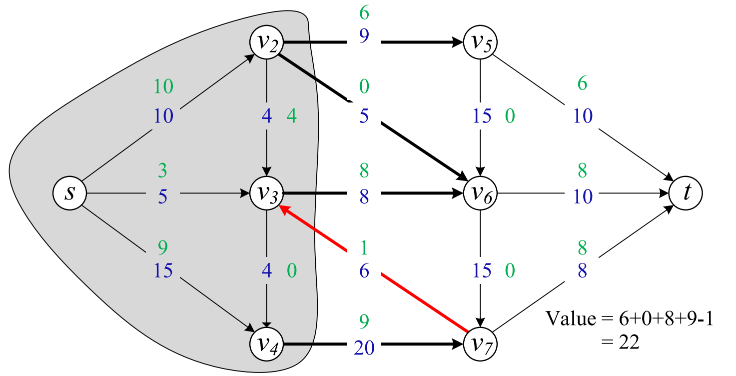
$$v(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$



Flows and Cuts (cont.)

Flow Value Lemma. Let f be any flow, and let (A, B) be any *s-t* cut. Then, the value of f equals to the net flow across the cut (A, B).

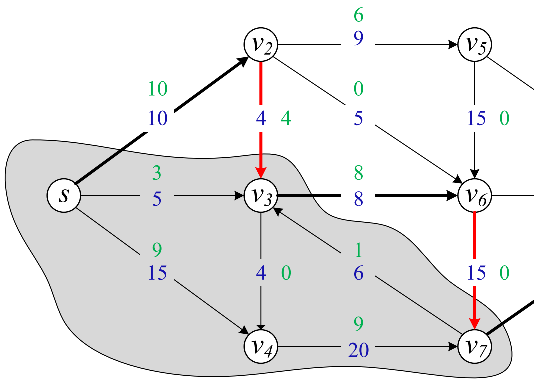
$$v(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$



Flows and Cuts (cont.)

Flow Value Lemma. Let f be any flow, and let (A, B) be any s-t cut. Then, the value of f equals to the net flow across the cut (A, B).

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



Value = 10+8+8-4-0
= 22

Flows and Cuts (cont.)

Flow Value Lemma. Let f be any flow, and let (A, B) be any s-t cut. Then, the value of f equals to the net flow across the cut (A, B).

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

Proof. By flow conservation, $\sum_{e \text{ out of } v} f(e) = \sum_{e \text{ in to } v} f(e)$ if $v \neq s, t$.

Thus,

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

□

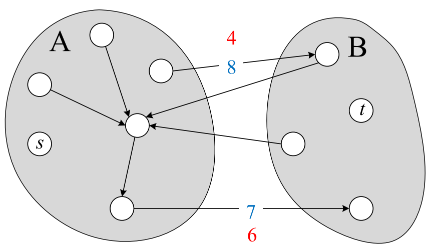
Flows and Cuts: Duality

Weak Duality. Let f be any flow. Then, for any s-t cut (A, B) we have

$$v(f) \leq cap(A, B).$$

Proof.

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



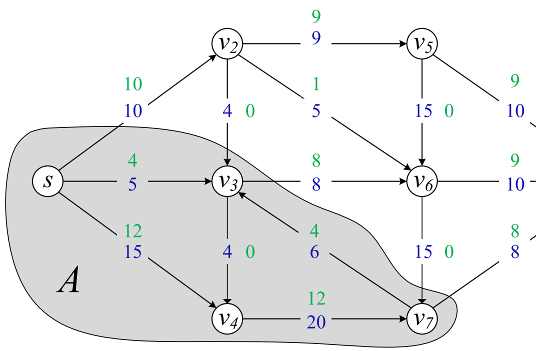
□

Certificate of Optimality

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

Proof. For any flow f': $val(f') \leq cap(A, B) = val(f)$. $\leftarrow$ max flow

For any cut (A', B'): $cap(A', B') \geq val(f) = cap(A, B)$. $\leftarrow$ min cut

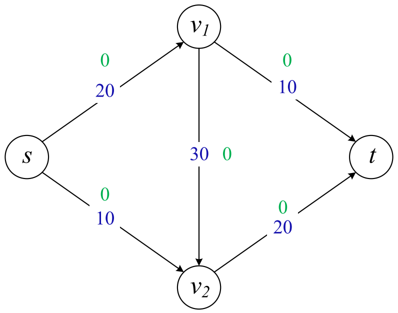


Max Flow = 26
Min Cut = 26

Towards a Max Flow Algorithm

Greedy algorithm.

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

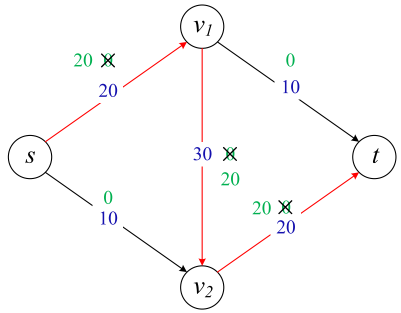


Value = 0

Towards a Max Flow Algorithm (cont.)

Greedy algorithm.

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

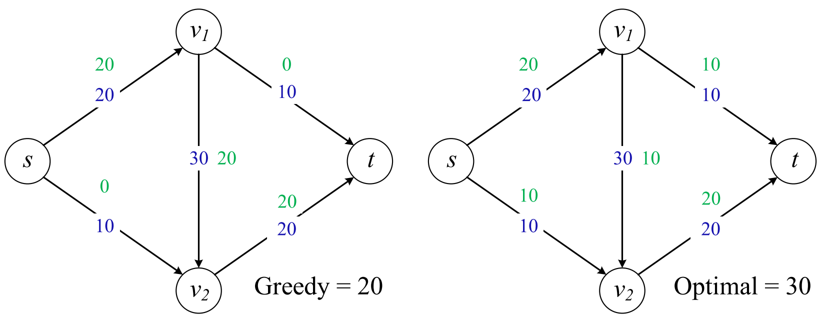


Value = 20

Towards a Max Flow Algorithm (cont.)

Greedy algorithm.

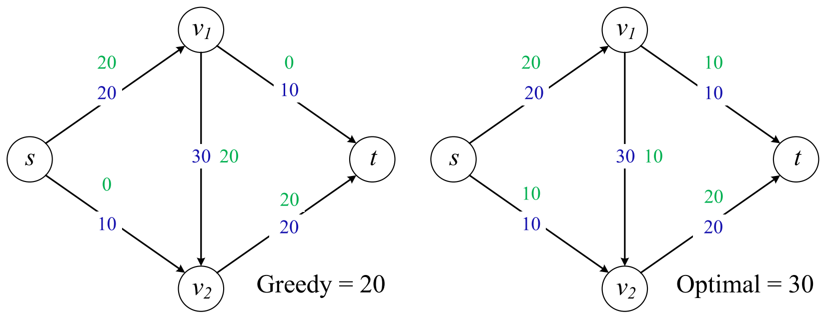
- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an s-t path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.
- However, locally optimality $\neq$ global optimality



Why the Greedy Algorithm Fails?

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

Consider this example: the unique max flow has $f^*(v_1, v_2) = 10$, but the greedy algorithm could choose $s \rightarrow v_1 \rightarrow v_2 \rightarrow t$ as the first augmenting path, each with $f(e) = 20$.

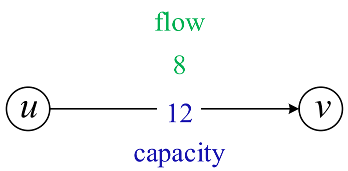


Bottom Line: Need some mechanism to “undo” a bad decision.

Solution: Residual Graph

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

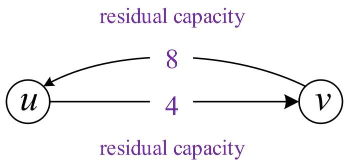
- Flow $f(e)$, capacity $c(e)$.



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

- "Undo" flow sent.
- $e = (u, v)$ and $e^R = (v, u)$.
- Residual capacity:

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



Residual Network

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

- Residual edges with positive residual capacity.
- $E_f = \{e \mid f(e) < c(e)\} \cup \{e^R \mid f(e) > 0\}$

Greedy Algorithm: Run the greedy algorithm on G_f to get a max flow f'. For G, when there is a flow on a reverse edge, negates flow on the corresponding forward edge (f).

$\Rightarrow$ the key idea for Ford-Fulkerson algorithm.

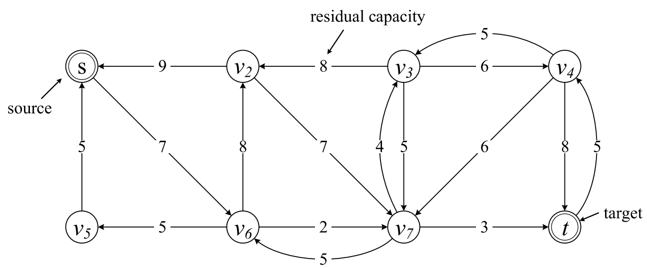
Key Property: f' is a flow in G_f iff f is a flow in G.

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



AUGMENT Subroutine

Algorithm 1: AUGMENT(f, c, P)

```
δ ← bottleneck capacity of augmenting path P;
foreach e ∈ P do
    if e ∈ E then
        f(e) ← f(e) + δ ;          /* forward edge */
    else
        f(e^R) ← f(e^R) − δ ;      /* reverse edge */
return f;
```

Ford-Fulkerson Algorithm

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

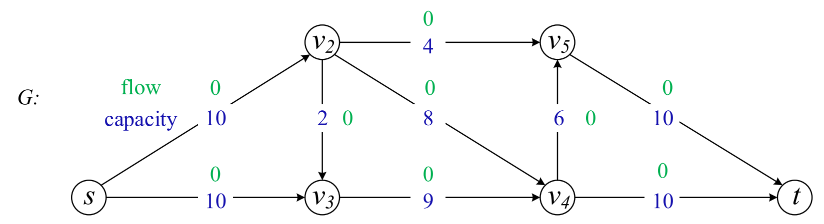
Algorithm 2: Ford-Fulkerson Algorithm

Input: $G = (V, E), c, s, t$

foreach $e \in E$ **do**
  $f(e) \leftarrow 0$;
$G_f \leftarrow$ residual graph;
while *there exists augmenting path* P **do**
  $f \leftarrow \textsc{Augment}(f, c, P)$;
  update G_f;
return f;

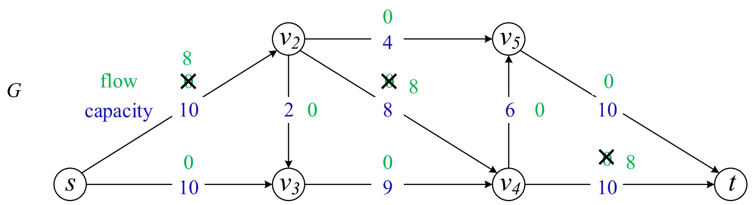
Example of Ford-Fulkerson Algorithm

Initial:

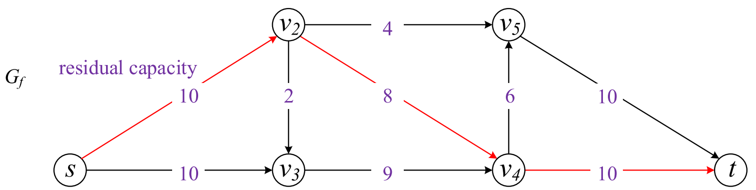


Flow = 0

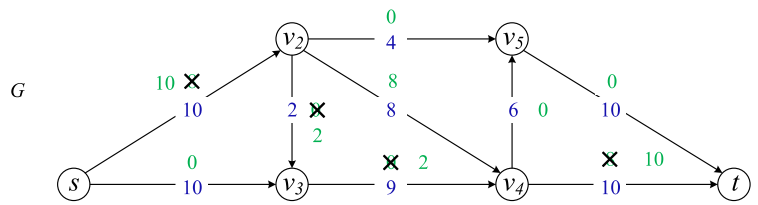
Example of Ford-Fulkerson Algorithm



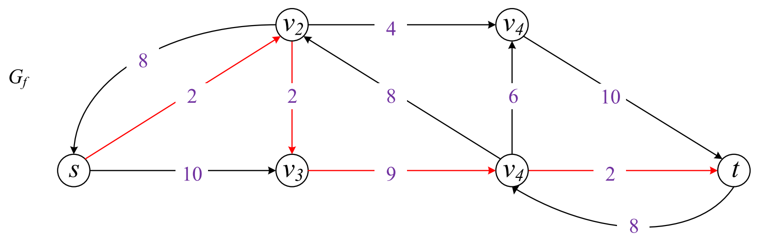
Flow = 0



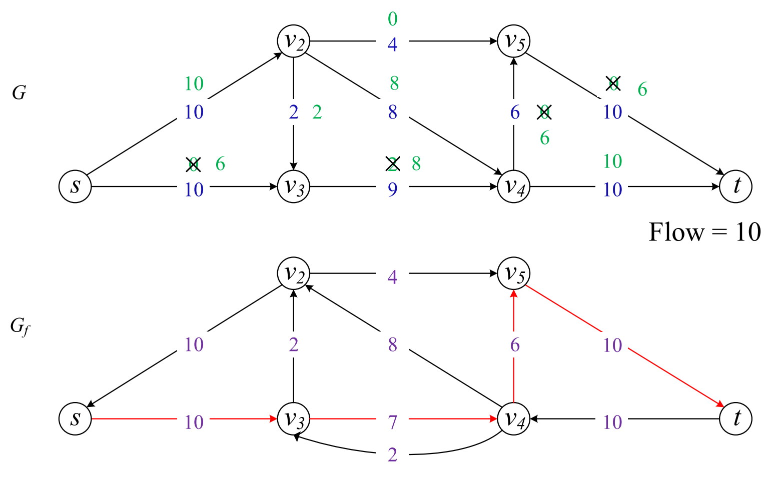
Example of Ford-Fulkerson Algorithm (cont.)



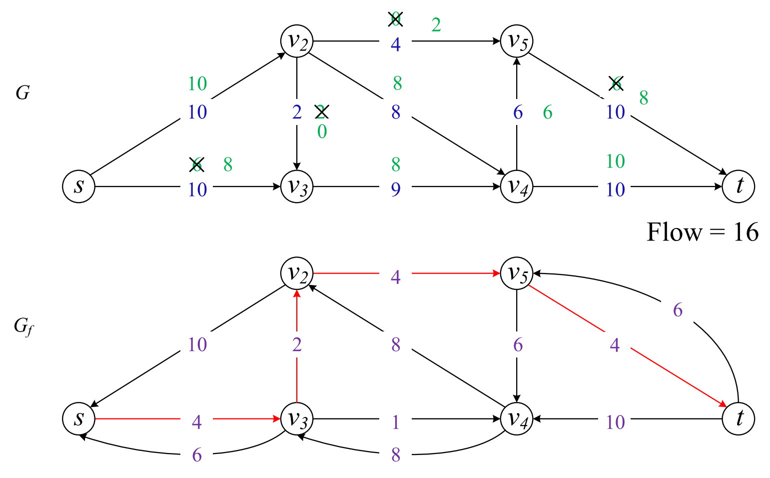
Flow = 8



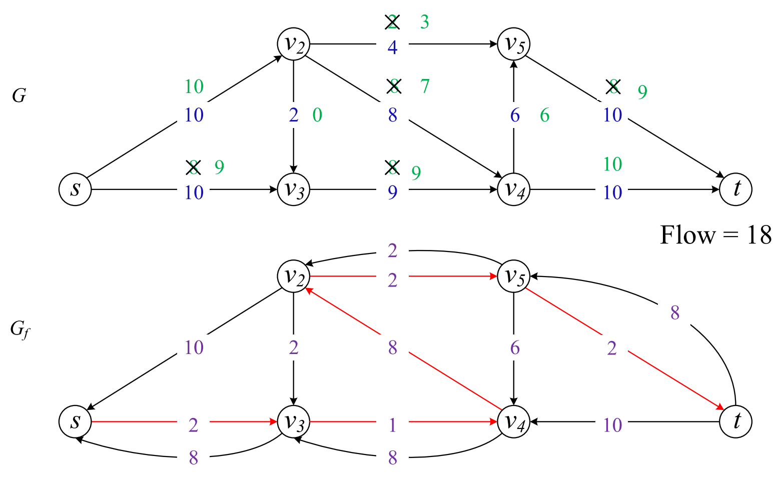
Example of Ford-Fulkerson Algorithm (cont.)



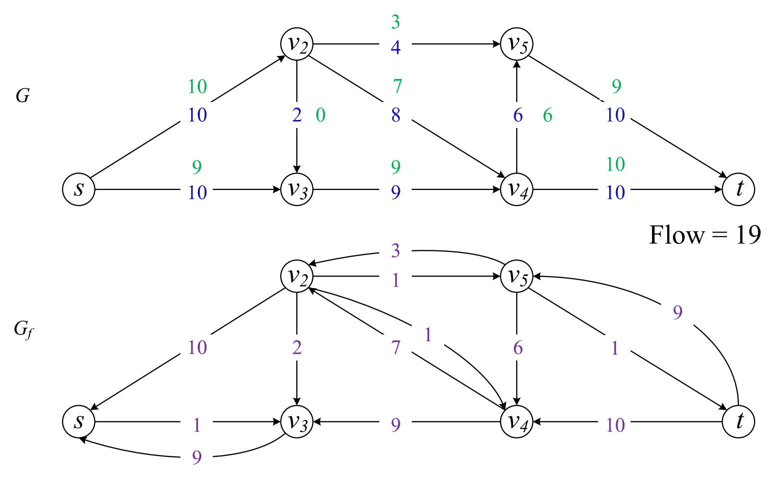
Example of Ford-Fulkerson Algorithm (cont.)



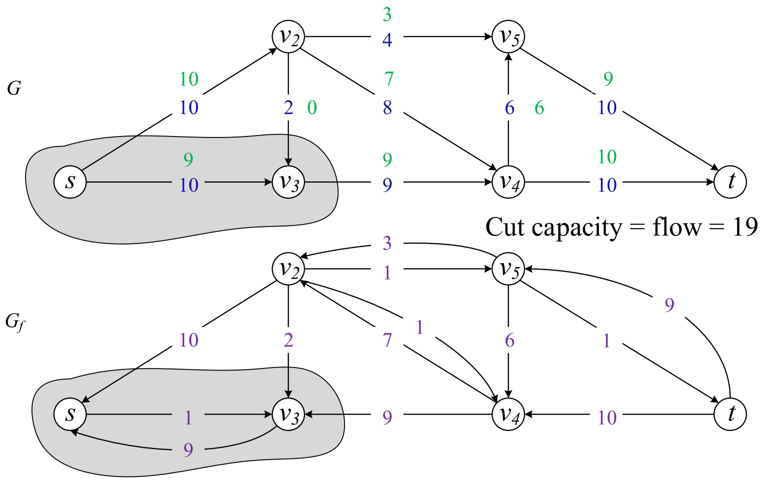
Example of Ford-Fulkerson Algorithm (cont.)



Example of Ford-Fulkerson Algorithm (cont.)



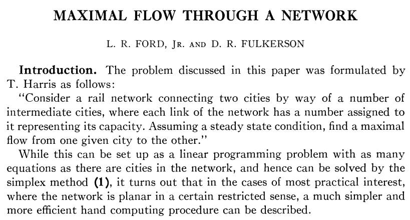
Example of Ford-Fulkerson Algorithm (cont.)



Max-Flow Min-Cut Theorem (Strong Duality)

Augmenting Path Theorem. Flow f is a max flow iff. there are no augmenting paths.

Max-flow Min-cut Theorem. [Elias-Feinstein-Shannon 1956, Ford-Fulkerson 1956]
The value of the max flow is equal to the value of the min cut.



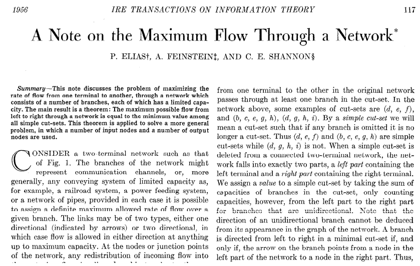
MAXIMAL FLOW THROUGH A NETWORK

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."

While this can be set up as a linear programming problem with as many equations as there are cities in the network, and hence can be solved by the simplex method **(1)**, it turns out that in the cases of most practical interest, where the network is planar in a certain restricted sense, a much simpler and more efficient hand computing procedure can be described.



1956 IRE TRANSACTIONS ON INFORMATION THEORY 117

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.

CONSIDER a two-terminal network such as that of Fig. 1. The branches of the network might represent communication channels, or, more generally, any conveying system of limited capacity as, for example, a railroad system, a power feeding system, or a network of pipes, provided in each case it is possible to assign a definite maximum allowed rate of flow over a given branch. The links may be of two types, either one directional (indicated by arrows) or two directional, in which case flow is allowed in either direction at anything up to maximum capacity. At the nodes or junction points of the network, any redistribution of incoming flow into

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 deleted from a connected two-terminal network, the network falls into exactly two parts, a *left part* containing the left terminal and a *right part* containing the right terminal. We assign a *value* to a simple cut-set by taking the sum of capacities of branches in the cut-set, only counting capacities, however, from the left part to the right part for branches that are unidirectional. Note that the direction of an unidirectional branch cannot be deduced from its appearance in the graph of the network. A branch is directed from left to right in a minimal cut-set if, and only if, the arrow on the branch points from a node in the left part of the network to a node in the right part. Thus,

Max-Flow Min-Cut Theorem (cont.)

Proof. We prove both simultaneously by showing TFAE:
(1) There exists a cut (A, B) such that $v(f) = cap(A, B)$.
(2) Flow f is a max flow.
(3) There is no augmenting path relative to f.

(1) $\Rightarrow$ (2) This was the corollary to weak duality lemma.

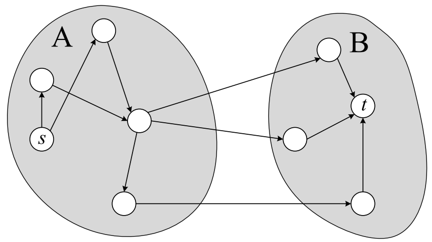
(2) $\Rightarrow$ (3) We show contrapositive. Let f be a flow. If there exists an augmenting path, then we can improve f by sending flow along path.

Max-Flow Min-Cut Theorem (cont.)

(3) $\Rightarrow$ (1) Let f be a flow with no augmenting paths. Let A be set of vertices reachable from s in residual graph.

By definition of A, $s \in A$. By definition of f, $t \notin A$.

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



Edge $e = (v, w)$ with $v \in B$, $w \in A$ must have $f(e) = 0$;
Edge $e = (v, w)$ with $v \in A$, $w \in B$ must have $f(e) = c(e)$.

Running Time of Ford-Fulkerson Algorithm

Assumption. All capacities are integers between 1 and C.

Invariant. Every flow value $f(e)$ and every residual capacity $c_f(e)$ remains an integer throughout the algorithm.
Proof. By induction on the number of augmenting paths. $\square$

Theorem. The algorithm terminates at most $val(f^*) \leq n \times C$ augmenting pathes, where f^* is a max flow.
Proof. Each augmentation increase value by at least 1. $\square$

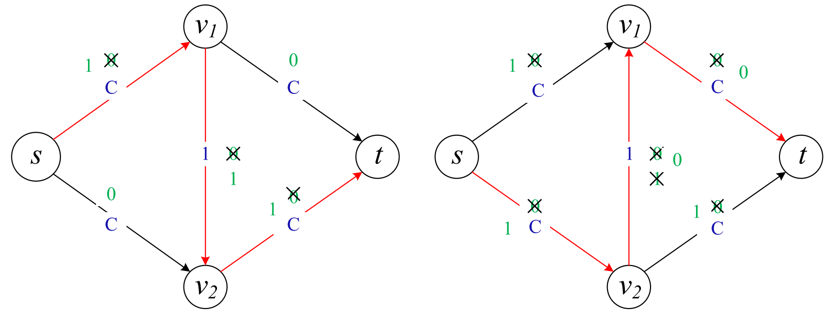
Corollary. Ford-Fulkerson runs in $O(mnC)$ time.
Proof. Can use either BFS or DFS to find an augmenting path in $O(m)$ times. $\square$

Integrality Theorem. There exists an integral max flow f.
Proof. Since algorithm terminates, theorem follows from integrality invariant (and augmenting path theorem). $\square$

Ford-Fulkerson: Exponential Number of Augmentations

Q. Is generic Ford-Fulkerson algorithm polynomial in input size?
Input size: $m, n, \log C$

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



Choosing Good Augmenting Paths

Use care when selecting augmenting paths.

- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.
- (Pathology) If capacities are irrational, algorithm not guaranteed to terminate (or converges to a maximum flow)!

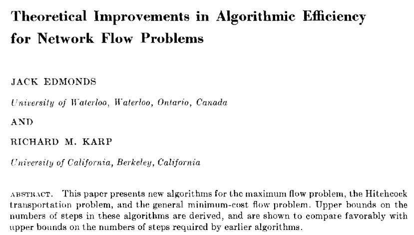
Goal: choose augmenting paths so that:

- Can find augmenting paths efficiently.
- Few iterations.

Choose augmenting paths with: [Edmonds-Karp 1972, Dinitz 1970]

- Max bottleneck capacity.
- Sufficiently large bottleneck capacity.
- Fewest number of edges.

Choosing Good Augmenting Paths



Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems

JACK EDMONDS

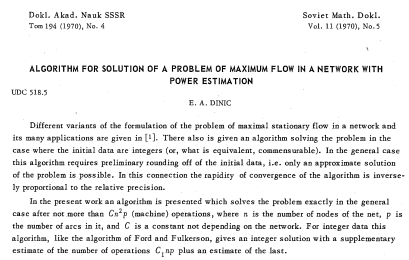
University of Waterloo, Waterloo, Ontario, Canada

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.



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.

In the present work an algorithm is presented which solves the problem exactly in the general case after not more than Cn^2p (machine) operations, where n is the number of nodes of the net, p is the number of arcs in it, and C is a constant not depending on the network. For integer data this algorithm, like the algorithm of Ford and Fulkerson, gives an integer solution with a supplementary estimate of the number of operations C_1np plus an estimate of the last.

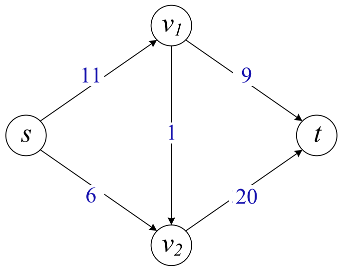
Edmond-Karp 1972 (USA) Dinitz 1970 (Soviet Union)

Note: Dinitz's paper is invented in response to a class exercise by Adel'son-Vel'skiĭ !!

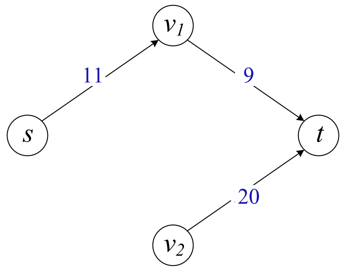
Capacity Scaling

Intuition. Choosing path with highest bottleneck capacity increases flow by max possible amount.

- Don't worry about finding exact highest bottleneck path.
- Maintain scaling parameter Δ.
- Let $G_f(\Delta)$ be the subgraph of the residual graph consisting of only arcs with capacity at least Δ.



G



$G_f(8)$

Capacity Scaling (cont.)

Algorithm 3: Scaling Max-Flow Algorithm

Input: $G = (V, E), c, s, t$

foreach $e \in E$ **do**
  $f(e) \leftarrow 0;$
$G_f \leftarrow$ residual graph;
$\Delta \leftarrow$ smallest power of 2 greater than or equal to C;
while $\Delta \geq 1$ **do**
  $G_f(\Delta) \leftarrow \Delta$-residual graph;
  while *there exists augmenting path P in $G_f(\Delta)$* **do**
    $f \leftarrow \text{ARGUMENT}(f, c, P)$;
    update $G_f(\Delta)$;
  $\Delta \leftarrow \Delta/2$;
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.

Proof. Initially a power of 2; each phase divides Δ by exactly 2. □

Integrality Invariant. Throughout the algorithm, all flow and residual capacity values are integral.

Proof. Same as for generic Ford-Fulkerson. □

Correctness. If the 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. □

Capacity Scaling: Running Time

Lemma 1. The outer while loop repeats $1 + \lceil \log_2 C \rceil$ times.
Proof. Initially $C \leq \Delta < 2C$. Δ decreases by a factor of 2 each iteration.

Lemma 2. Let f be the flow at the end of a Δ-scaling phase. Then the value of the maximum flow is at most $v(f) + m\Delta$ (proof on next slide).

Lemma 3. There are at most $2m$ augmentations per scaling phase.
Proof. Let f be the flow at the end of the previous scaling phase.
Lemma 2 $\Rightarrow v(f^*) \leq v(f) + m(2\Delta)$
Each augmentation in a Δ-phase increases $v(f)$ by at least Δ. □

Theorem. The scaling max-flow algorithm finds a max flow in $O(m \log C)$ augmentations. It can be implemented to run in $O(m^2 \log C)$ time.

Capacity Scaling: Running Time (cont.)

Lemma 2. Let f be the flow at the end of a Δ-scaling phase. Then the value of the maximum flow is at most $v(f) + m\Delta$.

Proof. (almost identical to proof of max-flow min-cut theorem)
We show that at the end of a Δ-phase, there exists a cut (A, B) such that $cap(A, B) \leq v(f) + m\Delta$.

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 f, $t \notin A$.

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

Augmenting-Path Algorithms: Summary

Year	Method	# Augmentations	Running Time
1955	augmenting path	nC	$O(mnC)$
1972	fattest path	$m\log(mC)$	$O(m^2 \log n \log(mC))$
1972	capacity scaling	$m\log C$	$O(m^2 \log C)$
1985	improved capacity scaling	$m\log C$	$O(mn\log C)$
1970	shortest augmenting path	mn	$O(m^2 n)$
1970	level graph	mn	$O(mn^2)$
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	$O(mn^2C)$	Dantzig
1955	augmenting path	$O(mnC)$	Ford-Fulkerson
1970	shortest augmenting path	$O(mn^2)$	Edmonds-Karp, Dinitz
1974	blocking flows	$O(n^3)$	Karzanov
1983	dynamic trees	$O(mn\log n)$	Sleator-Tarjan
1985	improved capacity scaling	$O(mn\log C)$	Gabow
1988	push-relabel	$O(mn\log(\frac{n^2}{m}))$	Goldberg-Tarjan
1998	binary blocking flows	$O(m^{\frac{3}{2}}\log(\frac{n^2}{m})\log C))$	Goldberg-Rao
2013	compact networks	$O(mn)$	Orlin
2014	interior-point methods	$O(m^{\frac{3}{2}}\log C)$	Lee-Sidford
2016	electrical flows	$O(m^{\frac{10}{7}}C^{\frac{1}{7}})$	Madry
20*xx*	?	?	?

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