

### 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*
- *blocking-flow algorithm*
- *unit-capacity simple networks*

## Running time

---

**Assumption.** Capacities are non-negative integers, where  $C$  is the total capacity from the source. The network is a connected graph.

**Integrity invariant.** Throughout the algorithm, the flow values  $f(e)$  and the residual capacities  $c_f(e)$  are integers.

**Theorem.** The algorithm terminates in at most  $val(f^*) \leq C$  iterations.

**Pf.** Each augmentation increases the value by at least 1. ■

**Corollary.** The running time of Ford-Fulkerson is  $O(m C)$ .

**Pf.** Each iteration costs  $O(m)$  by breadth first search.

**Integrity theorem.** Then exists a max-flow  $f^*$  for which every flow value  $f^*(e)$  is an integer.

**Pf.** Since algorithm terminates, theorem follows from invariant. ■

## Choosing good augmenting paths

---

Use care when selecting augmenting paths.

- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.
- If capacities are irrational, algorithm not guaranteed to terminate!

**Goal.** Choose augmenting paths so that:

- Can find augmenting paths efficiently.
- Few iterations.

## Choosing good augmenting paths

---

### Choose augmenting paths with:

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

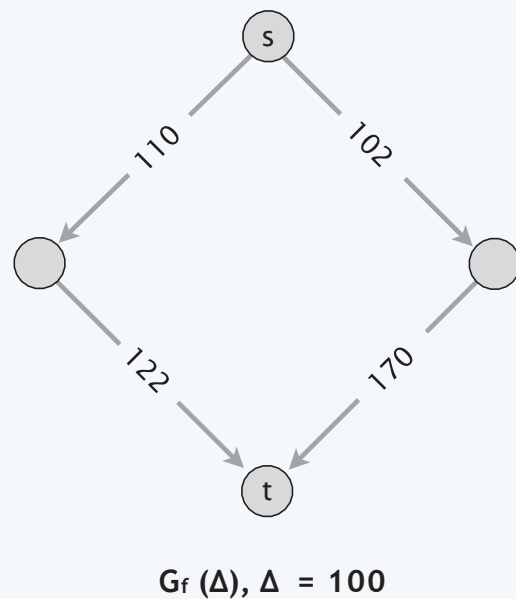
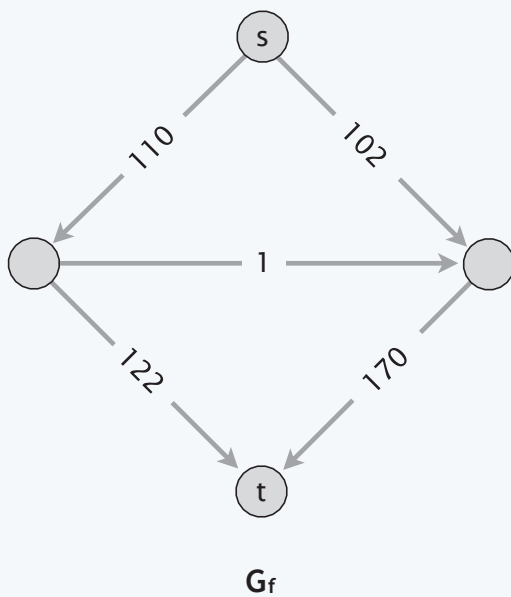
Different variants of the formulation of the problem of maximal stationary flow in a network and its many applications are given in [1]. There also is given an algorithm solving the problem in the case where the initial data are integers (or, what is equivalent, commensurable). In the general case this 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.

**Dinic 1970 (Soviet Union)**

## Capacity-scaling algorithm

**Intuition.** Choose augmenting path with highest bottleneck capacity:  
it increases flow by max possible amount in given iteration.

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



## Capacity-scaling algorithm

---

**CAPACITY-SCALING**( $G, s, t, c$ )

---

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

$\Delta \leftarrow$  largest power of 2 that is  $\leq \max_{e \text{ out of } s} c_e$ .

**WHILE** ( $\Delta \geq 1$ )

$G_f(\Delta) \leftarrow \Delta$ -residual graph.

**WHILE** (there exists an augmenting path  $P$  in  $G_f(\Delta)$ )

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

        Update  $G_f(\Delta)$ .

$\Delta \leftarrow \Delta / 2$ .

**RETURN**  $f$ .

---

## Capacity-scaling algorithm: proof of correctness

---

**Assumption.** All edge capacities are integers between 1 and  $\tilde{C}$ .

**Integrality invariant.** All flow and residual capacity values are integral.

**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. ■

## Capacity-scaling algorithm: analysis of running time

---

**Lemma 1.** The outer while loop repeats  $1 + \log_2 C$  times.




## Capacity-scaling algorithm: analysis of running time

---

**Lemma 1.** The outer while loop repeats  $1 + \log_2 \tilde{C}$  times.

**Pf.** Initially  $\tilde{C}/2 < \Delta \leq \tilde{C}$ ;  $\Delta$  decreases by a factor of 2 in each iteration. ■

**Lemma 2.** Let  $f$  be the flow at the end of a  $\Delta$ -scaling phase. Then, the value of the max-flow  $\leq \text{val}(f) + m \Delta$ .  proof on next slide

## Capacity-scaling algorithm: analysis of running time

---

**Lemma 1.** The outer while loop repeats  $1 + \log_2 \tilde{C}$  times.

**Pf.** Initially  $\tilde{C}/2 < \Delta \leq \tilde{C}$ ;  $\Delta$  decreases by a factor of 2 in each iteration. ■

**Lemma 2.** Let  $f$  be the flow at the end of a  $\Delta$ -scaling phase. Then, the value of the max-flow  $\leq \text{val}(f) + m \Delta$ . ← proof on next slide

**Lemma 3.** There are at most  $2m$  augmentations per scaling phase.

**Pf.**

## Capacity-scaling algorithm: analysis of running time

---

**Lemma 1.** The outer while loop repeats  $1 + \log_2 \tilde{C}$  times.

**Pf.** Initially  $\tilde{C}/2 < \Delta \leq \tilde{C}$ ;  $\Delta$  decreases by a factor of 2 in each iteration. ■

**Lemma 2.** Let  $f$  be the flow at the end of a  $\Delta$ -scaling phase. Then, the value of the max-flow  $\leq \text{val}(f) + m \Delta$ . ← proof on next slide

**Lemma 3.** There are at most  $2m$  augmentations per scaling phase.

**Proof.** The statement is clearly true in the first scaling phase: we can use each of the edges out of  $s$  only for at most one augmentation in that phase. Now consider a later scaling phase  $\Delta$ , and let  $f_p$  be the flow at the end of the *previous* scaling phase. In that phase, we used  $\Delta' = 2\Delta$  as our parameter. By (7.18), the maximum flow  $f^*$  is at most  $v(f^*) \leq v(f_p) + m\Delta' = v(f_p) + 2m\Delta$ . In the  $\Delta$ -scaling phase, each augmentation increases the flow by at least  $\Delta$ , and hence there can be at most  $2m$  augmentations. ■

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

**Pf.** Follows from LEMMA 1 and LEMMA 3. ■

## Capacity-scaling algorithm: analysis of running time

---

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

## Capacity-scaling algorithm: analysis of running time

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

**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 cut  $A, s \in A$ .
- By definition of flow  $f, t \notin A$ .

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

