W, 10/30/19

Fall’19 CSCE 629

# Analysis of Algorithms


Fang Song

Texas A&M U


## Lecture 23

- Remarks on Ford-Fulkerson
- Intro to linear programming

Credit: based on slides by A. Smith & K. Wayne

# Ford-Fulkerson augmenting-path algorithm

$\boldsymbol{Ford\text{—}Fulkerson}(G, s, t, c)$
  For each $e \in E$ $f(e) \leftarrow 0$, $G_f \leftarrow residual\ graph$
  While there is an augmenting path $P$ in $G_f$
    $f \leftarrow Augment(f, c, P)$
    Update $G_f$
  return $f$

Theorem. Ford-Fulkerson terminates in at most $nC$ iterations.

Running time. $O(mnC)$

Exponential in input size:
$\log C$ bits (to represent $C$)

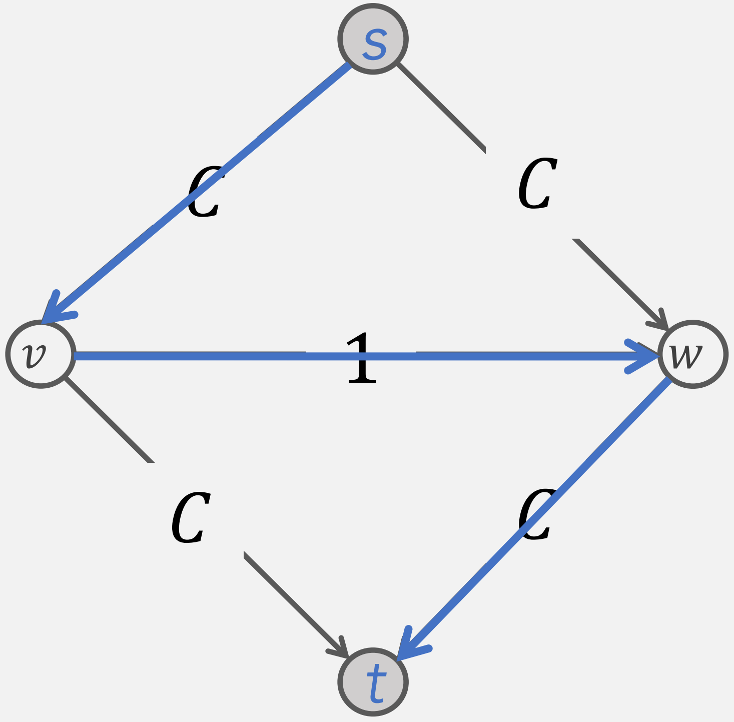
Can it be this bad?

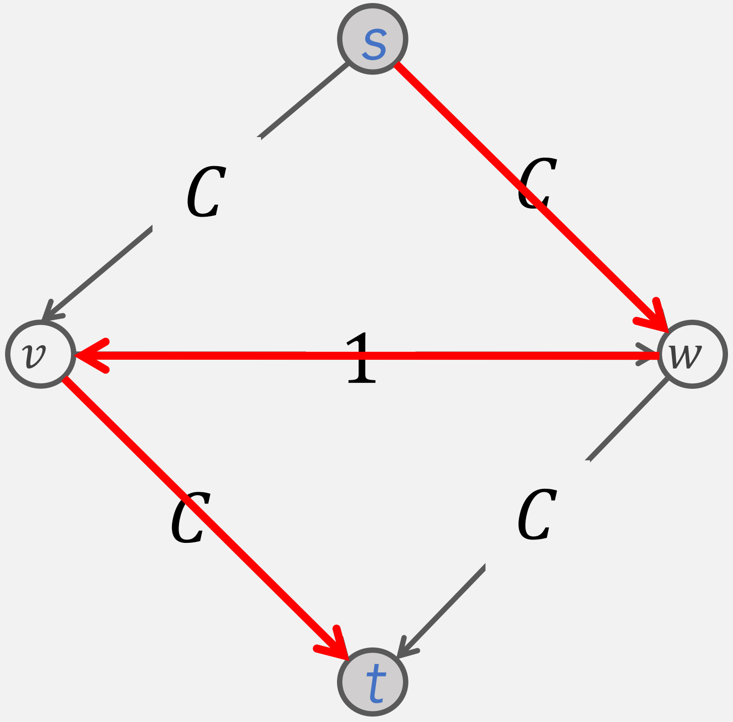
# Ford-Fulkerson: exponential example

Obs. If max capacity is $C$, then FF can take $\geq C$ iterations.

- $s \to v \to w \to t$
- $s \to w \to v \to t$
- $s \to v \to w \to t$
- $s \to w \to v \to t$
- $\ldots$
- $s \to v \to w \to t$
- $s \to w \to v \to t$

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



# Choosing good augmenting paths

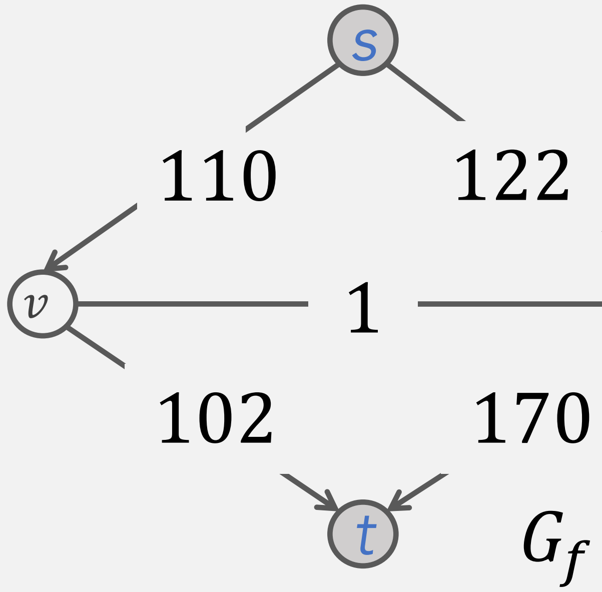
- Use care when selecting augmenting paths
  - Some choices lead to exponential algorithms
  - Clever choices lead to polynomial algorithms
  - If capacities are irrational, algorithm not guaranteed to terminate!

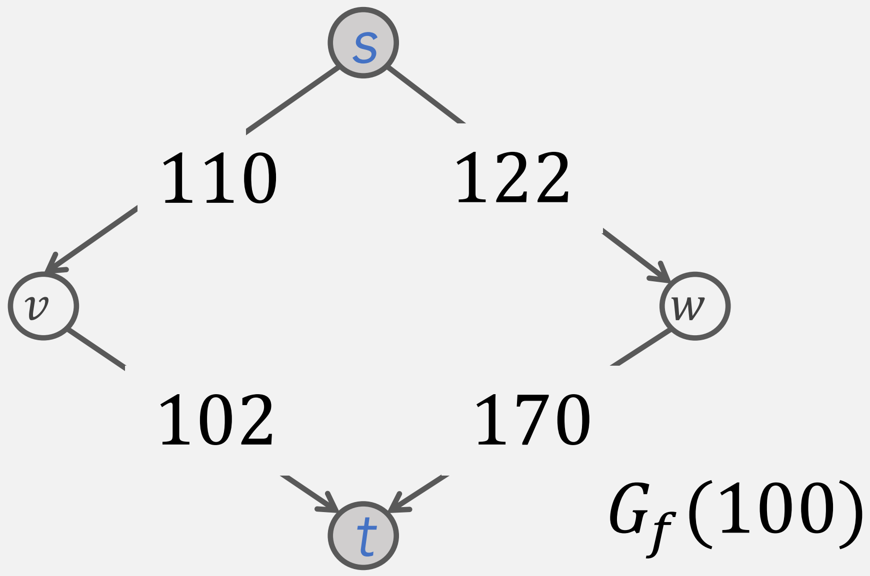
- Good choices of augmenting paths [EdmondsKarp'72,Dinitz'70]
  - Max bottleneck capacity [Next]
  - Fewest edges (shortest) [CLRS 26.2]

# Capacity scaling

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

- OK to choose sufficiently large bottleneck: scaling parameter $\Delta$
- Let $G_f(\Delta)$ be the subgraph of the residual graph consisting of only arcs with capacity at least $\Delta$



# Capacity scaling algorithm

**Scaling–Max–Flow** $(G, s, t, c)$

For each $e \in E$ $f(e) \leftarrow 0$,

$G_f \leftarrow residual\ graph$

$\Delta \leftarrow$ smallest power of 2 & $\geq C$

While $\Delta \geq 1$

  $G_f(\Delta) \leftarrow \Delta$–residual graph

  While there is an augmenting path $P$ in $G_f(\Delta)$

    $f \leftarrow Augment(f, c, P)$ // augment flow by $\geq \Delta$

    Update $G_f(\Delta)$

  $\Delta \leftarrow \Delta/2$

return $f$

**Exercise.** Prove correctness

# Capacity scaling algorithm: running time

While $\Delta \geq 1$
  $G_f(\Delta) \leftarrow \Delta$−residual graph
  While there is $P$ in $G_f(\Delta)$
    $f \leftarrow Augment(f, c, P)$
    Update $G_f(\Delta)$
  $\Delta \leftarrow \Delta/2$
…

Lemma1 Outer loop runs $1 + \log C$ times.

Pf. Initially $C \leq \Delta \leq 2C$, decreases by a factor of 2 each iteration

Lemma2. Let $f$ be the flow at the end of a Δ-scaling phase. Then the value of the maximum flow $f^*$ is at most $v(f) + m\Delta$.

Lemma3. There are at most $2m$ augmentations per scaling phase.

Pf. Let $f$ be the flow at end of previous scaling

- [Lemma2] $\Rightarrow v(f^*) \leq v(f) + m(2\Delta)$
- Each augmentation in Δ-scaling increases $f$ by $\Delta$

**Theorem.** Scaling-max-flow finds a max flow in $O(m^2 \log C)$ time.

# Completing the proof

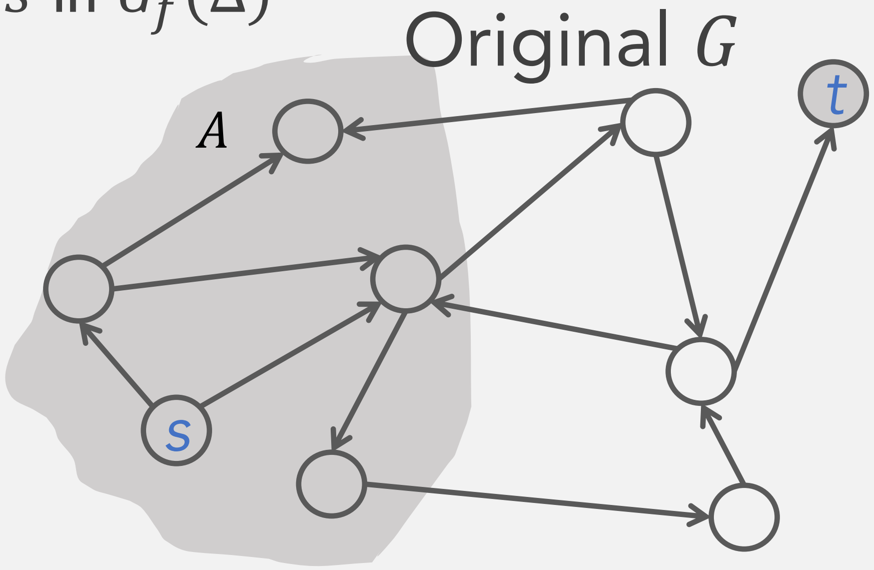
Lemma2. Let $f$ be the flow at the end of a $\Delta$-scaling phase. Then the value of the maximum flow $f^*$ is at most $v(f) + m\Delta$.

Pf. [Almost identical to proof of max-flow min-cut theorem]

Show cut $(A, B)$ w. $cap(A, B) \le v(f) + m\Delta$ at the end of a $\Delta$–phase.

- Choose $A$ to be the set of nodes reachable from $s$ in $G_f(\Delta)$
- By definition $s \in A$ & $t \notin A$

$$\begin{aligned} v(f) &= \textstyle\sum_{e \text{ outof } A} f(e) - \sum_{e \text{ into } A} f(e) \\ &\ge \textstyle\sum_{e \text{ outof } A} (c(e) - \Delta) - \sum_{e \text{ into } A} \Delta \\ &= \textstyle\sum_{e \text{ outof } A} c(e) - \sum_{e \text{ outof } A} \Delta - \sum_{e \text{ into } A} \Delta \\ &= cap(A, B) - m\Delta \end{aligned}$$



# 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 CapS | $m \log C$ | $O(mn \log C)$ |
| 1970 | Shortest path | $mn$ | $O(m^2 n)$ |
| 1970 | level graph | $mn$ | $O(mn^2)$ |
| 1983 | dynamic trees | $mn$ | $O(mn \log n)$ |

# and the show goes on …

| Year | Method | Worst case | Discovered by |
|---|---|---|---|
| 1951 | Simplex | $O(mn^2C)$ | Dantzig |
| 1955 | Augmenting path | $O(mn^2W)$ | Ford-Fulkerson |
| … | | | |
| 1988 | Push-relabel | $O(mn\log(n^2/m))$ | Goldberg-Tarjan |
| … | | | |
| 2013 | Compact networks | $O(mn)$ | Orlin |
| 2016 | Electrical flows | $\tilde{O}(m^{10/7}C^{1/7})$ | Madry |
| 20XX | | | |

To keep it simple, cite below when you invoke a max-flow subroutine in hw/exam

Maximum flows can be computed in $O(mn)$ time

# Another formulation of max-flow problem

Recall. An $s\text{–}t$ flow is a function $f: E \to \mathbb{R}$ satisfying

- [Capacity] $\forall e \in E: 0 \le f(e) \le c(e)$
- [Conservation] $\forall v \in V\backslash\{s,t\}$: $\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$

The value of a flow $f$ is $v(f) \coloneqq \sum_{e \text{ out of } s} f(e)$

**Max–Flow Problem**

Real-value variables $\vec{f} = \{f_e : e \in E\}$

Maximize: $v(\vec{f})$

Subject to:

$$0 \le f_e \le c(e),\ \forall e \in E$$

$$\sum_{e \text{ into } v} f_e - \sum_{e \text{ out of } v} f_e = 0,\ \forall v \in V\backslash\{s,t\}$$

Linear constraints: no $x^2, xy, \sin(x)$, ...

# Grade maximization

Input. HW from two courses (xxx & 629) due in one day

- Every hour you spend, you earn 1pts on xxx or 5pts on 629
- Your brain will explode if you work more than 12hrs on xxx or 15hrs on 629
- Of course, there are only 24 hrs in a day

Goal. Maximize the total pts you can earn

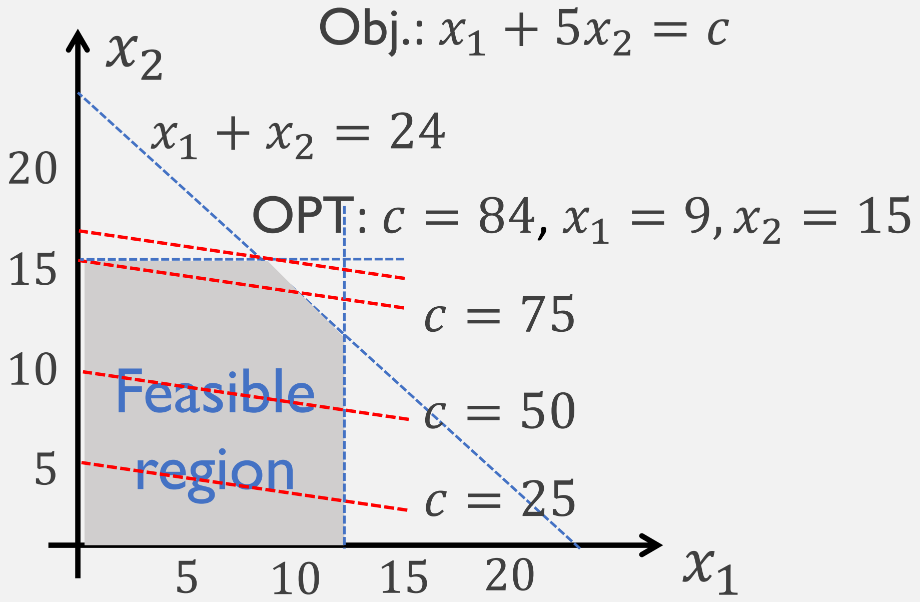
**Grade—Maximization**

Variables: $x_1$ (xxx hrs); $x_2$ (629 hrs)

Maximize: $x_1 + 5x_2$

Subject to: // linear constraints

$$0 \le x_1 \le 12$$
$$0 \le x_2 \le 15$$
$$x_1 + x_2 \le 24$$



# Linear programming

Linear programming. Optimize a linear objective function subject to linear inequalities.

- Formal definition and representations
- Duality
- Algorithms: simplex, ellipsoid, interior point

## Why significant?

- Design poly-time algorithms & approximation algorithms
- Wide applications: math, economics, business, transportation, energy, telecommunications, and manufacturing

Ranked among most important scientific advances of 20th century


# Happy Halloween!
# &
# Enjoy the treats!