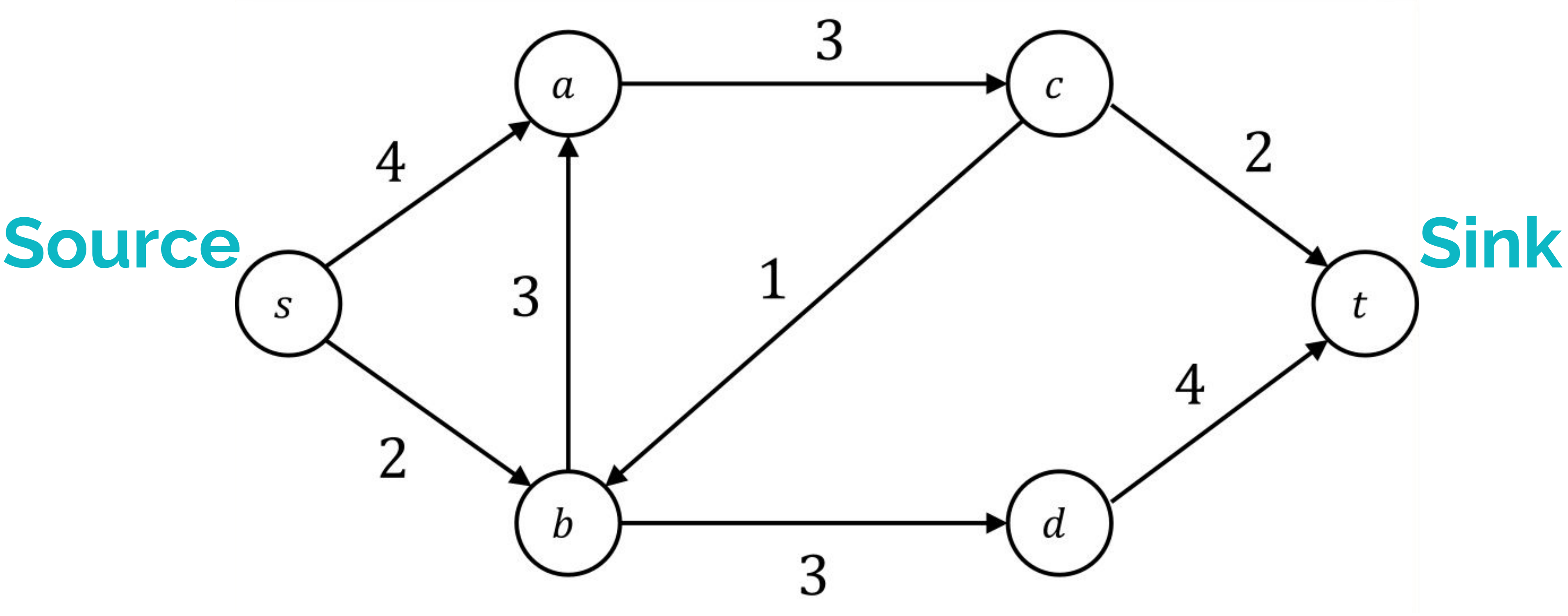
15451 Spring 2023

Max Flow: polynomial-time algorithms

Elaine Shi

Recall the max flow problem



Directed graph, each edge **e** has a **capacity c(e)**

Recall Ford-Fulkerson

While exists augmenting path P of positive residual capacity

Push the maximum possible flow along P

Running time: $O(\underbrace{m}_{\#\text{edges}}\ \underbrace{F}_{\text{flow value}})$

Recall Ford-Fulkerson

While exists augmenting path P of positive residual capacity

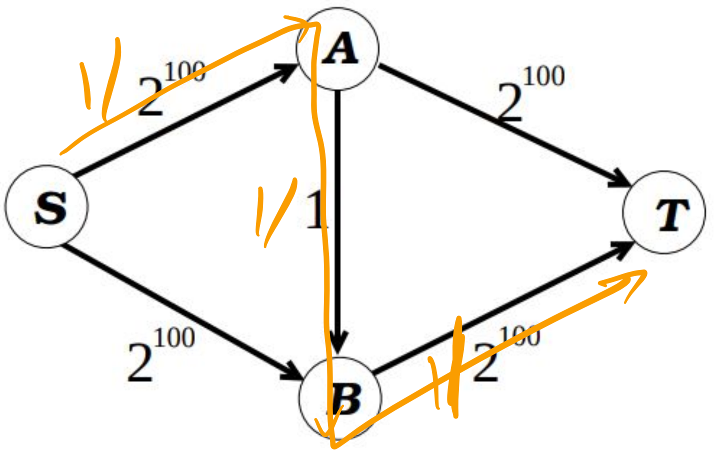
Push the maximum possible flow along P

Running time: **O(m F)**

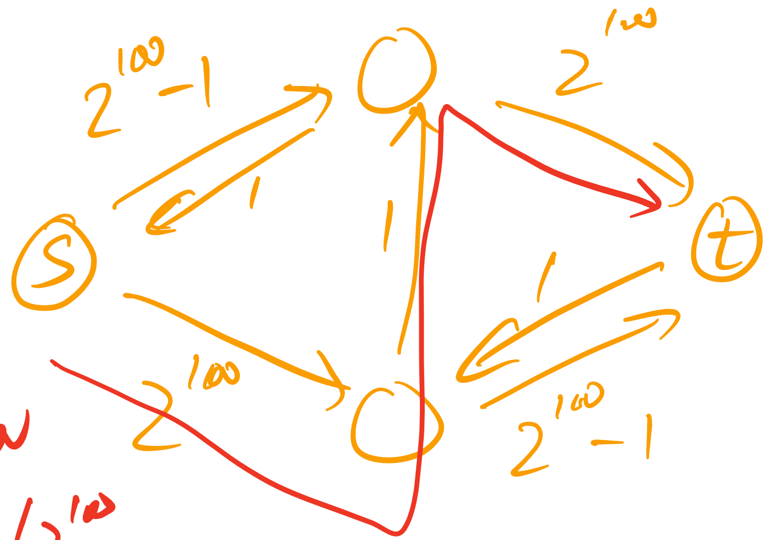
"Pseudo-polynomial time"

Can we have an algorithm whose running time is polynomial in only **m** and **n**?

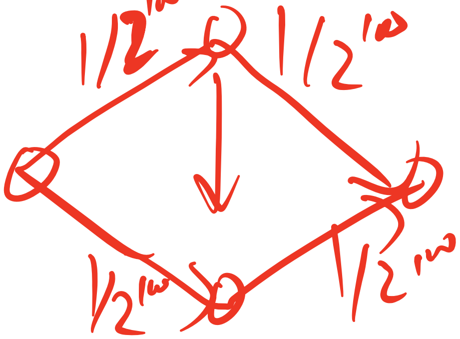
Pathological example for FF



Residual graph



new flow



Can we avoid the pathological cases if we select the augmenting path more cleverly?

Select the “widest” path

Edmonds-Karp 1972

When people implemented FF using **BFS**, the performance is always polynomial.

Why?

Edmonds-Karp

When people implemented FF using **BFS**, the performance is always polynomial.

Why?

BFS → shortest path

Edmonds-Karp's Algorithm

Run FF but select the **shortest** augmenting path

Theorem: Edmonds-Karp makes at most mn iterations, i.e. E-K completes in $O(n \cdot m^2)$ time

Theorem: Edmonds-Karp makes at most **mn** iterations

Proof: let **d** be the distance from s to t in the residual graph

- Claim 1: d never ~~increases~~ decreases
- Claim 2: every m iters, d increases by at least 1

$\Rightarrow$ #iters $= O(n \cdot m)$

Claim 1: d never ~~in~~ de creases

$d(s, v')$: shortest distance from s to v'

Proof: We will $\forall v'$, $d(s, v')$ never decreases

during each iter, how the residual graph can change

① some edges may get removed b/c they are saturated

② add a backward edge (w.r.t. current residual graph)

(u, v). (v, u) is on the shortest path from s-t in the current residual graph

$d(s, u) = d(s, v) + 1$

(u, v) will not decrease $d(s, v')$ for any v'

Claim 2: every m iterations, d increases by at least 1

i: iter

G^i : residual graph in iter G^i.

~~Claim:~~

G^{i+1}

$\vdots$

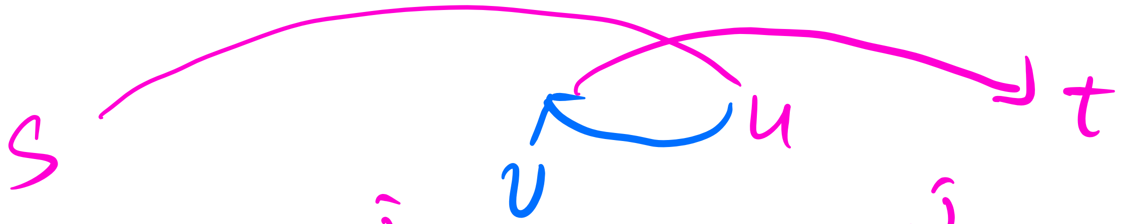
G^j

Suppose that in G^j, we first start to use a back edge w.r.t G^i, then $d^{G^j}(s,t) > d^{G^i}(s,t)$

if this is true, Claim 2 follows b/c every aug. path saturates at least one edge. there are only $O(m)$ edges in G^i

Claim: Suppose in G^j, we first start to use a back edge w.r.t. G^i, then $d^{G^j}(s,t) > d^{G^i}(s,t)$

in G^j



(u,v) is the last back edge w.r.t G^i in aug. path in G^j

$$d^{G^j}(s,t) = \underbrace{d^{G^j}(s,u)+1}_{\geq \underline{d^{G^i}(s,u)}+1} + \underbrace{d^{G^j}(v,t)}_{\geq d^{G^i}(v,t)}$$

$$d^{G^i}(s,t) = \underbrace{d^{G^i}(s,v)} + \underline{d^{G^i}(v,t)}$$

(v,u) is on shortest path in G^i

Running time of Edmonds-Karp: $O(nm^2)$

n phases (in term of d)

each phase : at most m iters

each iter : BFS : $O(m)$

Dinic's algorithm: $O(n^2m)$ 1970, "Dinitz"

EK $O(n \cdot m^2)$

Dinic's algorithm: $O(n^2m)$ 1970

A single BFS finds distances of all vertices from s, which encodes every shortest path

Let's not waste this work!

Dinic's algorithm

For each d: do a BFS, find multiple shortest augmenting paths to form a "**blocking flow**"

A **blocking flow** in a residual graph G_f is a union of flows on shortest augmenting paths such that every shortest path in G_f has at least one edge that is saturated

Dinic's algorithm

- Initially, the residual graph is the original graph
- Repeat at most **n phases**:
  - Construct the layered graph of the residual graph
  - Find a blocking flow on the layered graph
  - Augment current flow using blocking flow, update residual graph

Goal: find blocking flow in **O(n m)** time $\Rightarrow O(n^2 \cdot m)$ time in total

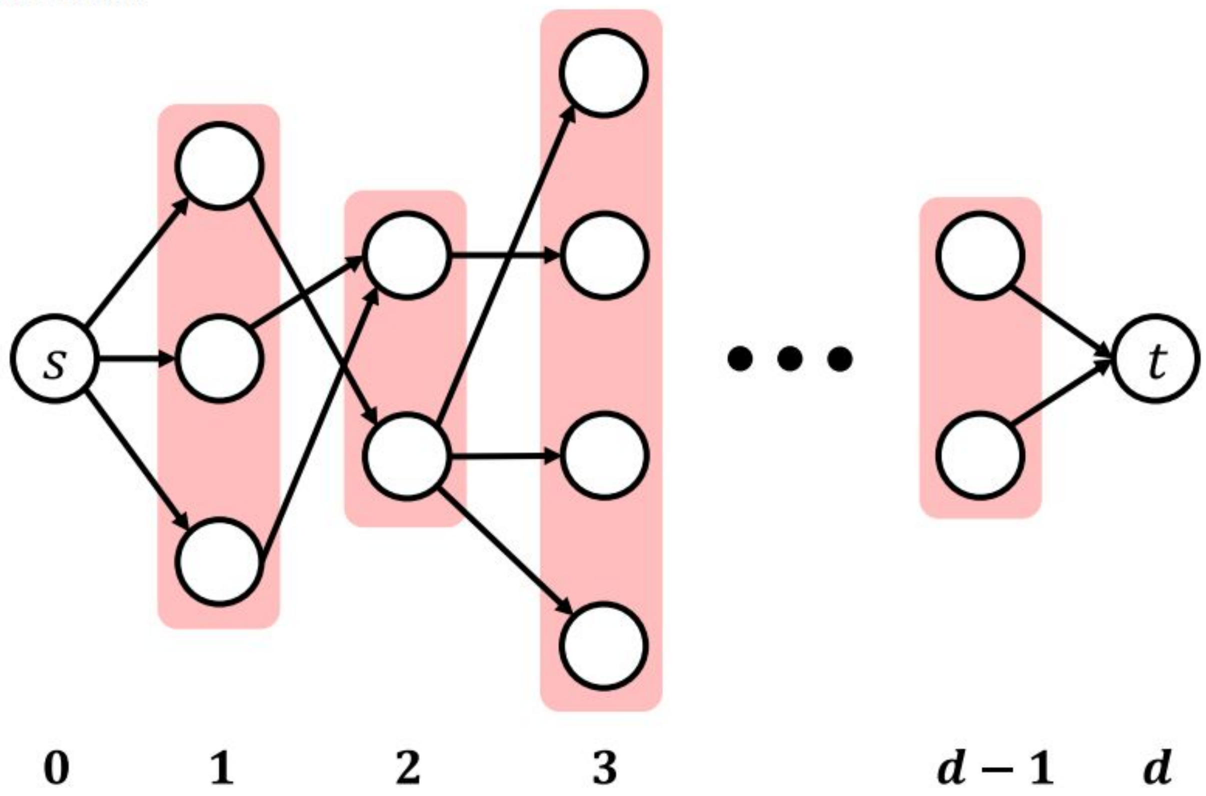
Dinic's algorithm

d increases every phase

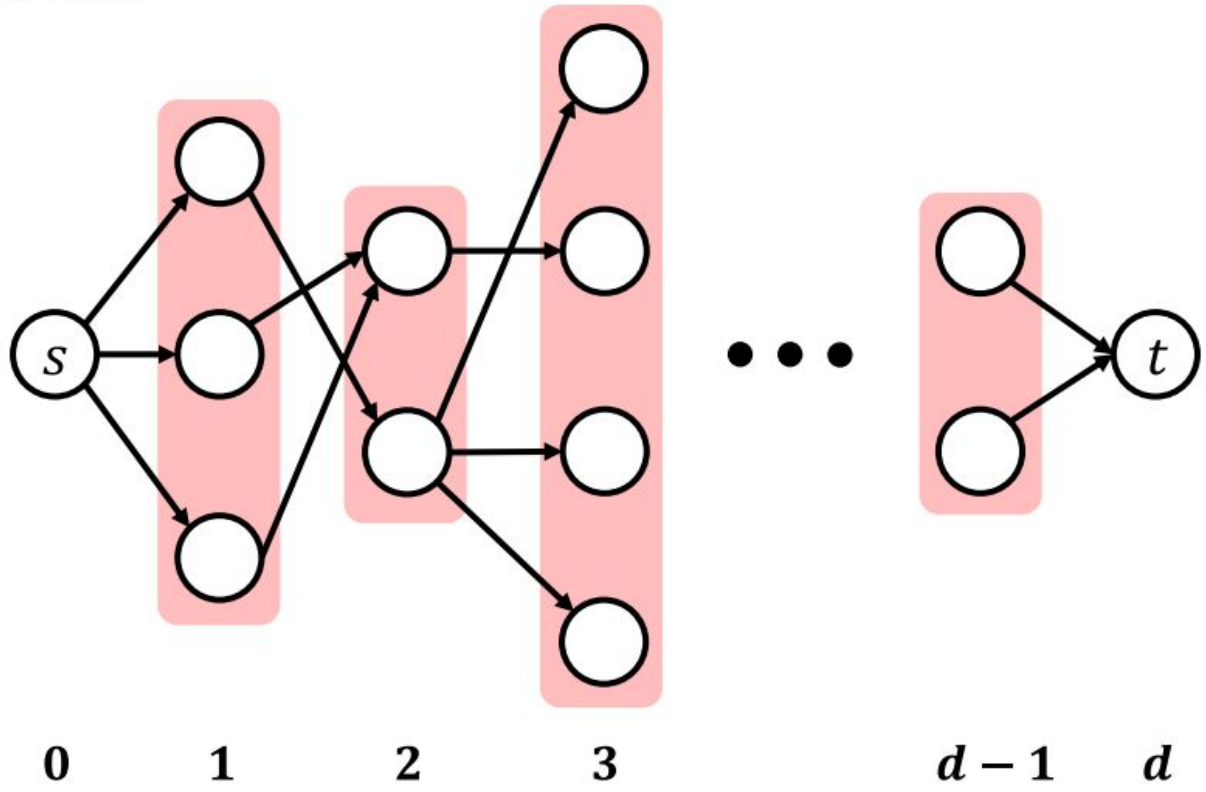
- Initially, the residual graph is the original graph
- Repeat at most **n phases**:
  - Construct the layered graph of the residual graph
  - Find a blocking flow on the layered graph
  - Augment current flow using blocking flow, update residual graph

Goal: find blocking flow in **O(n m)** time

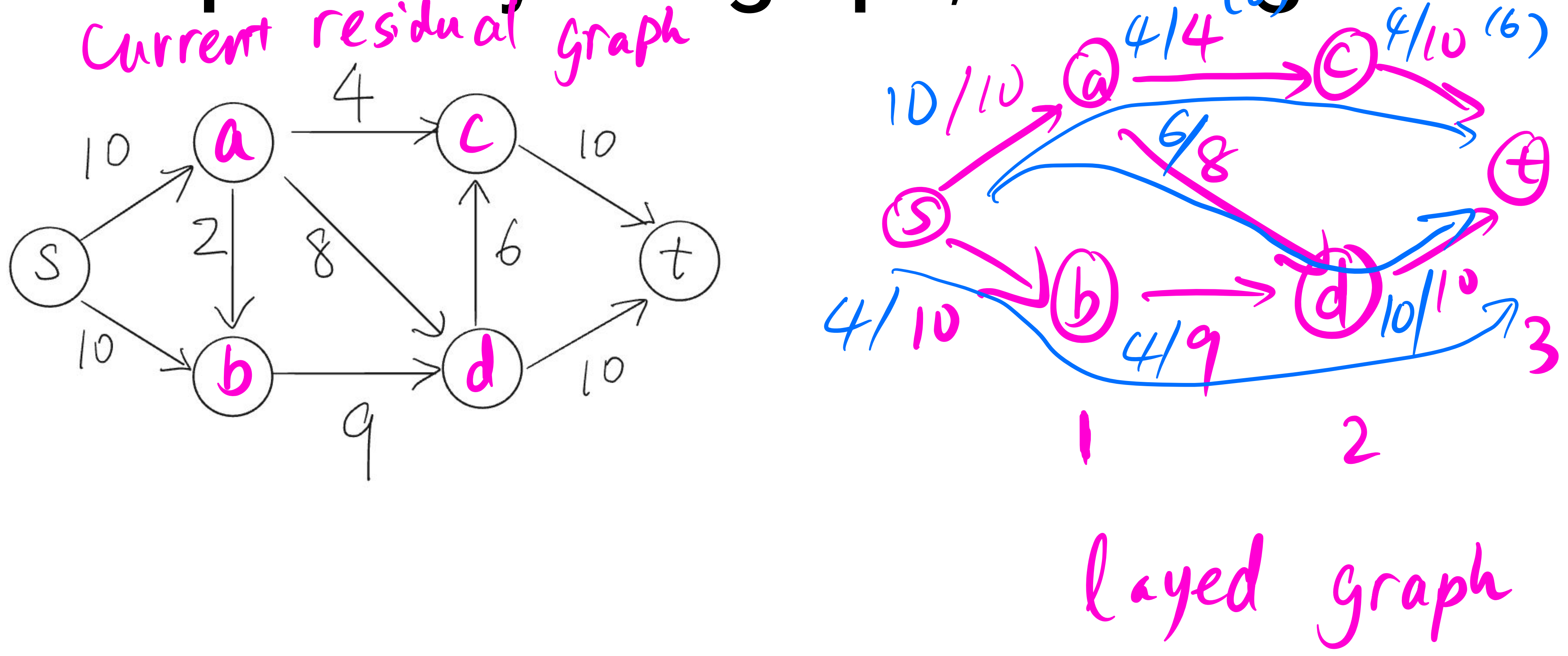
Find blocking flow using the layered graph



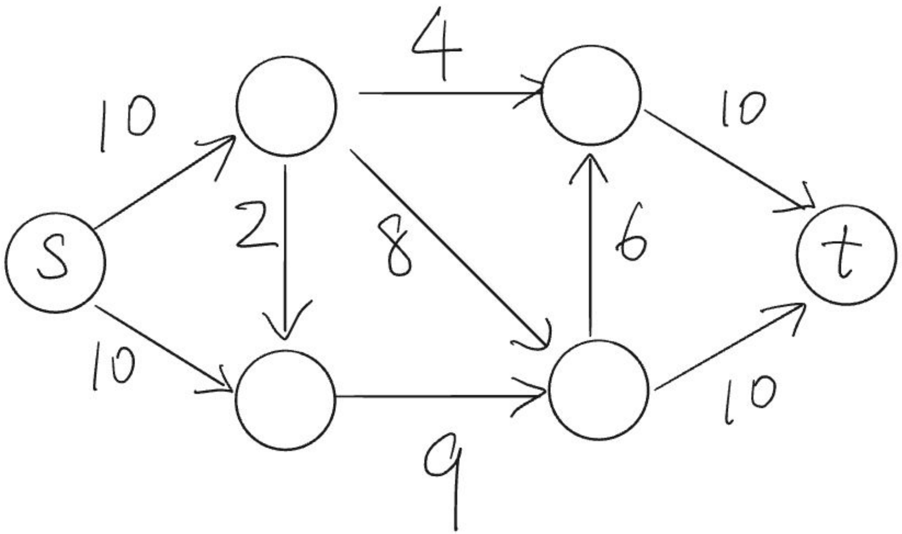
Claim: all augmenting paths of len d must be paths s-t paths in this layered graph

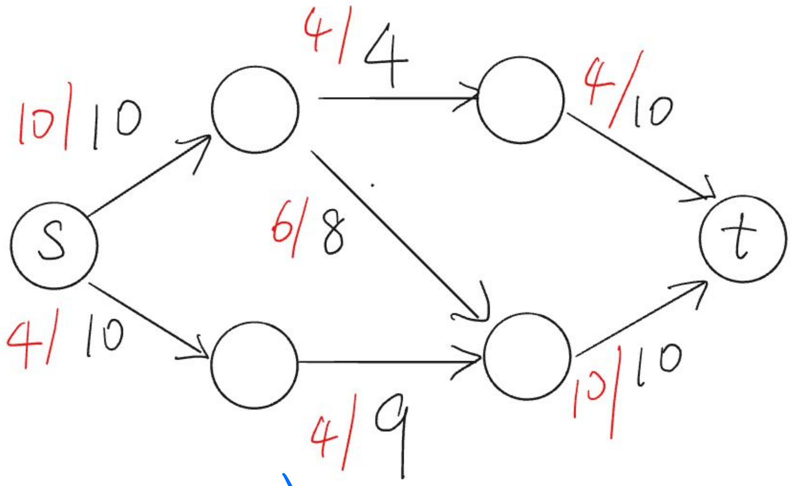


Example of layered graph, blocking flow



Example of layered graph, blocking flow





layered graph
blocking flow

Naive algo. for finding blocking flow:

Repeat until no more path found

m iterations

- Find a shortest path in layered graph using DFS, push maximum flow through it
- Update the residual capacities and remove edges with 0 residual capacity

$O(m)$ time

Total : $O(m^2)$

Want : $O(m \cdot n)$??

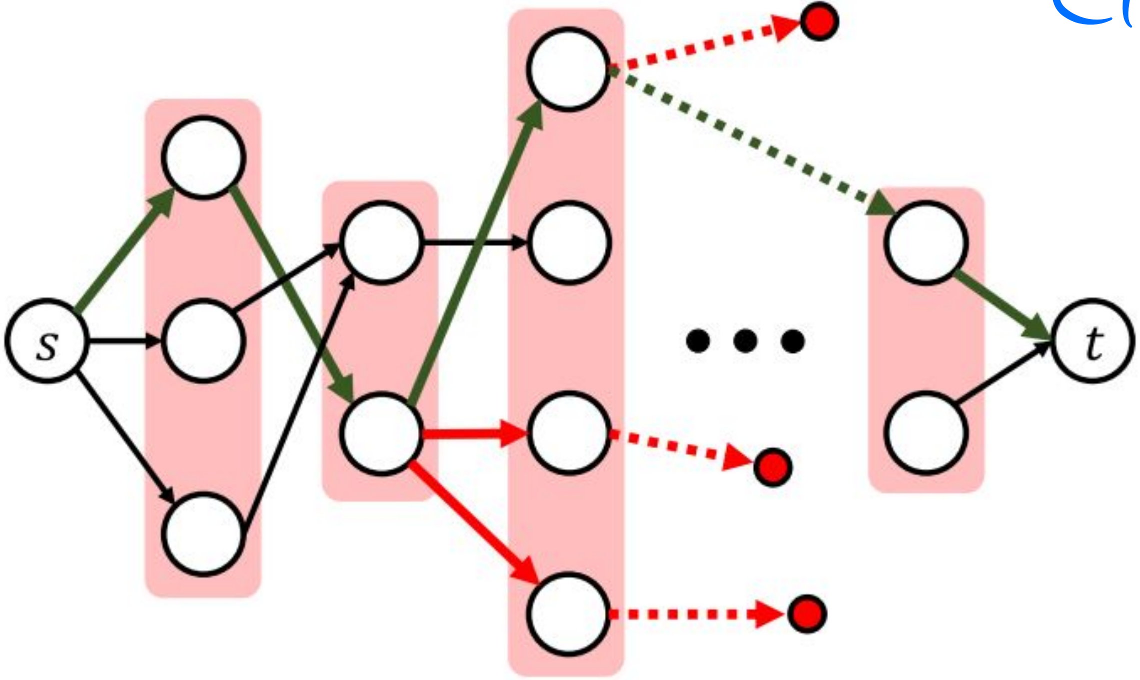
Naive algo. for finding blocking flow:

Repeat until no more path found

- Find a shortest path in layered graph using DFS, push maximum flow through it
- Update the residual capacities and remove edges with 0 residual capacity

Running time: $O(m^2)$

Improved algo: mark dead-end paths and prune them in future searches



Claim: $O(mn)$ time for finding blocking flow

Analysis of improved algorithm

m DFS iters.

cost in iter i = $\underbrace{n}_{\text{max len of shortest path we find}} + \underbrace{(\text{\# new dead-end edges encountered})}_{X_i}$

$$\text{total cost} = \sum_{i=1}^{m} n + X_i = m \cdot n + \sum_{i=1}^{m} X_i$$

$\sum_{i=1}^{m} X_i \leq m$

$\leq mn + m$

Dinic's algorithm often performs better than $O(mn^2)$ in practice

blocking flow : $O(m \cdot n)$ time

n total phases

Dinic's algorithm often performs better than O(mn²) in practice

①

Claim: For a unit capacity graph, a blocking flow can be found in O(m) time

②

Claim: For a unit capacity graph, we need at most $2\sqrt{m}$ blocking flows

Claim: For a unit capacity graph, Dinic finishes in $O(m \cdot \min(\sqrt{m}, n))$ time

Claim: For a unit capacity graph, a blocking flow can be found in O(m) time

b/c residual graph is also unit capacity

every edge can be on at most 1

aug. path in the blocking flow

in the blocking flow, total path len $\leq m$

Claim: For a unit capacity graph, we need at most $2\sqrt{m}$ blocking flows

proof: suppose we have completed k phases so far

$d \geq k$

there are at most $\frac{m}{k}$ aug paths of len $\geq k$

max flow in residual network $\leq \frac{m}{k}$

$\Rightarrow$ need at most m/k additional blocking flows

$\forall k$

blocking flows $\leq k + \frac{m}{k}$

pick $k = \sqrt{m}$

Latest developments

Maximum Flow and Minimum-Cost Flow in Almost-Linear Time

Li Chen[*]
Georgia Tech
lichen@gatech.edu

Rasmus Kyng[†]
ETH Zurich
kyng@inf.ethz.ch

Yang P. Liu[‡]
Stanford University
yangpliu@stanford.edu

Richard Peng
University of Waterloo [§]
y5peng@uwaterloo.ca

Maximilian Probst Gutenberg[†]
ETH Zurich
maxprobst@ethz.ch

Sushant Sachdeva[¶]
University of Toronto
sachdeva@cs.toronto.edu

April 26, 2022

Abstract

We give an algorithm that computes exact maximum flows and minimum-cost flows on directed graphs with m edges and polynomially bounded integral demands, costs, and capacities in $m^{1+o(1)}$ time. Our algorithm builds the flow through a sequence of $m^{1+o(1)}$ approximate undirected minimum-ratio cycles, each of which is computed and processed in amortized $m^{o(1)}$ time using a new dynamic graph data structure.

Our framework extends to algorithms running in $m^{1+o(1)}$ time for computing flows that minimize general edge-separable convex functions to high accuracy. This gives almost-linear time algorithms for several problems including entropy-regularized optimal transport, matrix scaling, p-norm flows, and p-norm isotonic regression on arbitrary directed acyclic graphs.