CSE 421

Max Flow Algorithms

Yin Tat Lee

Notations

Given a directed graph G with integral capacity $0 \le c_e \le U$.
Input size is $O(m \log U)$.

We call a runtime

- $(mU)^{O(1)}$ is pseudo polynomial.
- $(m \log U)^{O(1)}$ is weakly polynomial.
- $m^{O(1)}$ is strongly polynomial.

Ford Fulkerson takes $O(mF) = O(mnU)$. (Pseudo Polynomial).

Dependence on U is bad because U can be much larger than m.

Weakly polynomial:
Capacity Scaling

Question: How to improve U dependence?

Idea: Handle edges with large capacity first.

Capacity Scaling

Let $G_f(D)$ be the residual G_f with all edges $< D$ capacity removed.

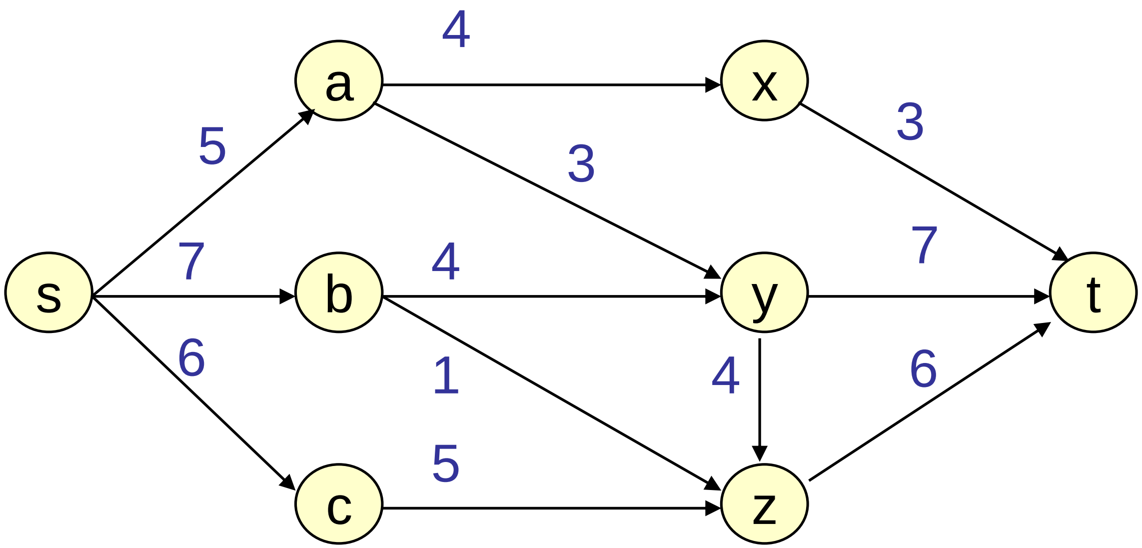
Algorithm:

- Set $D = U$
- While $D \geq 1$
  - While there is s-t path p in $G_f(D)$
    - Augment along p by lowest edge capacity in $G_f(D)$
  - Set $D \leftarrow \lfloor D/2 \rfloor$

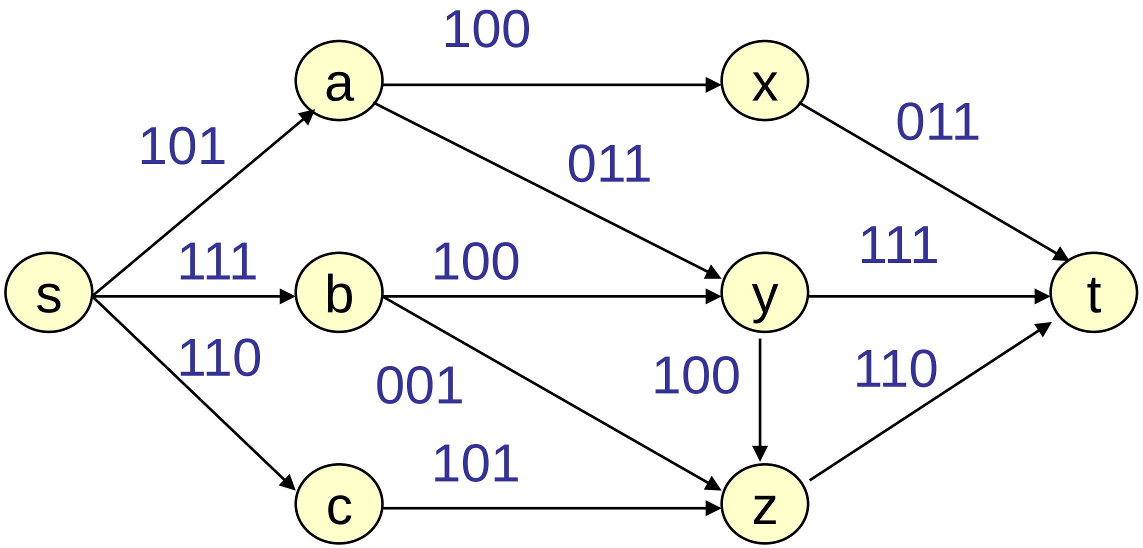
Correctness:

When $D = 1$, the inner loop is simply Ford Fulkerson.
Hence, at termination, we have a maxflow.

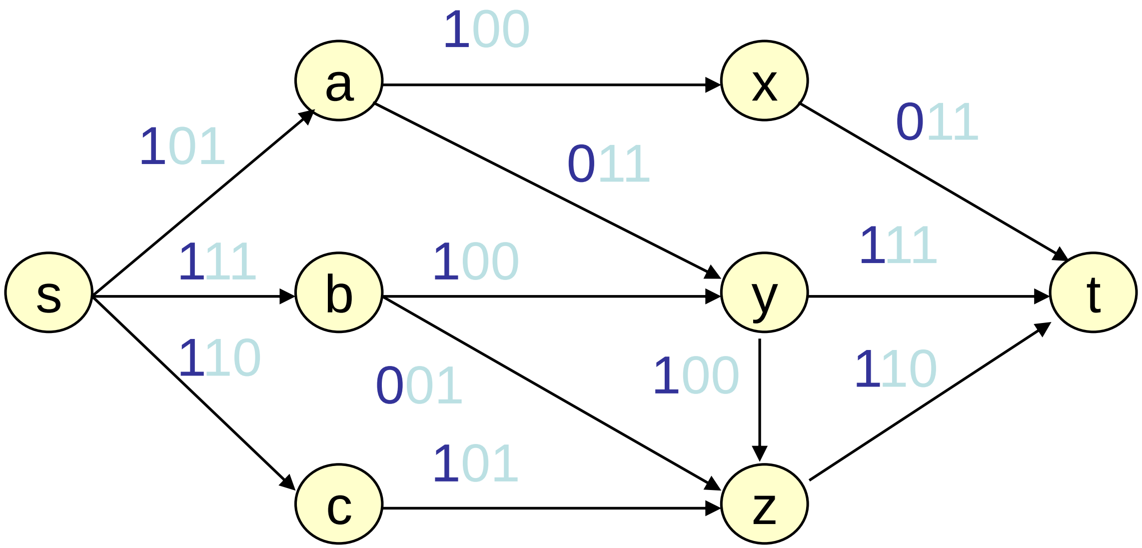
Capacity Scaling



Capacity Scaling

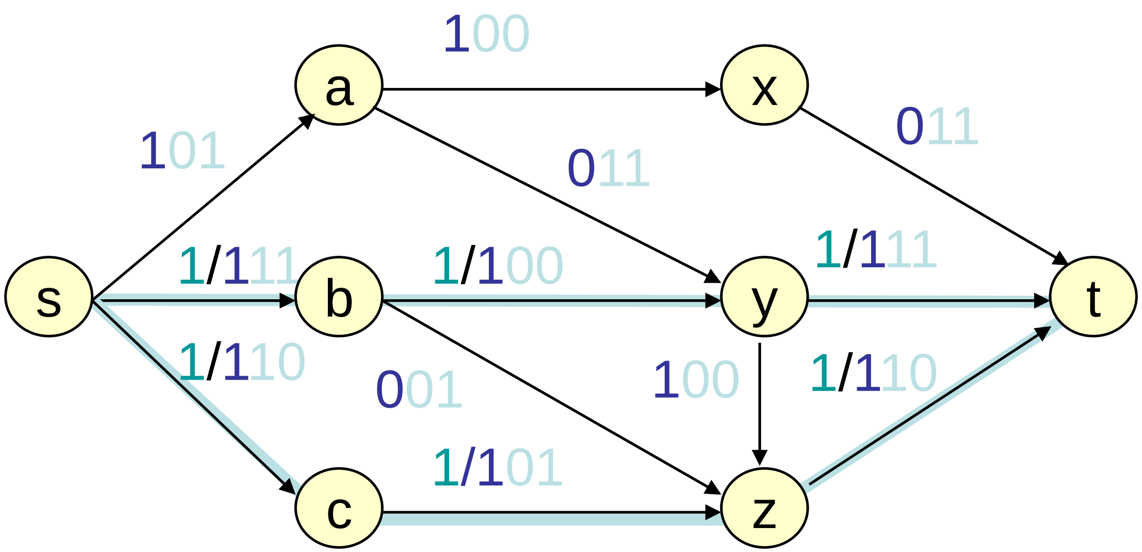


Capacity Scaling Bit 1



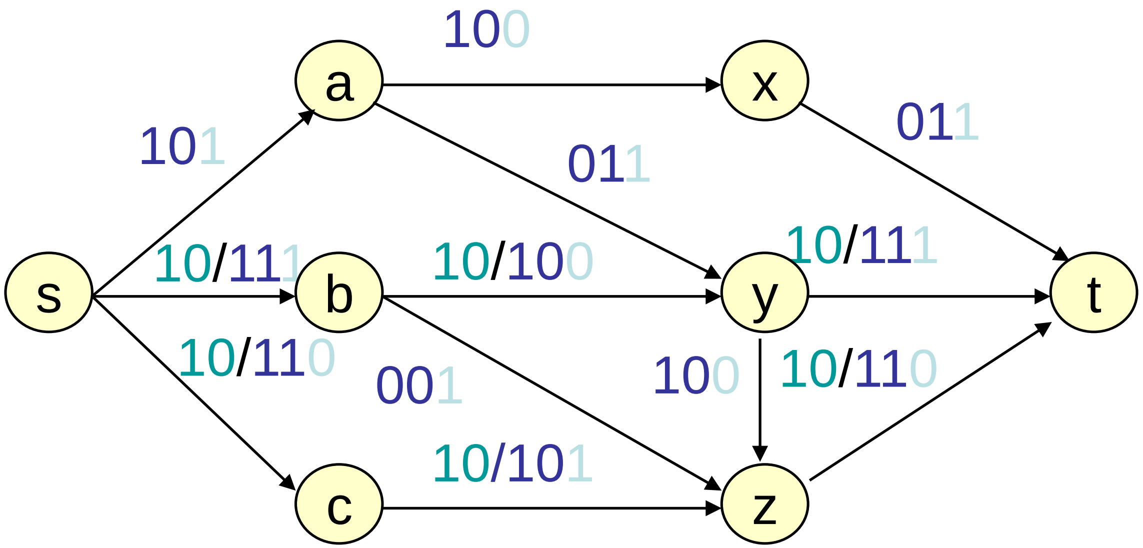
Capacity on each edge is at most **1**
(either **0** or **1** times **Δ=4**)

Capacity Scaling Bit 1



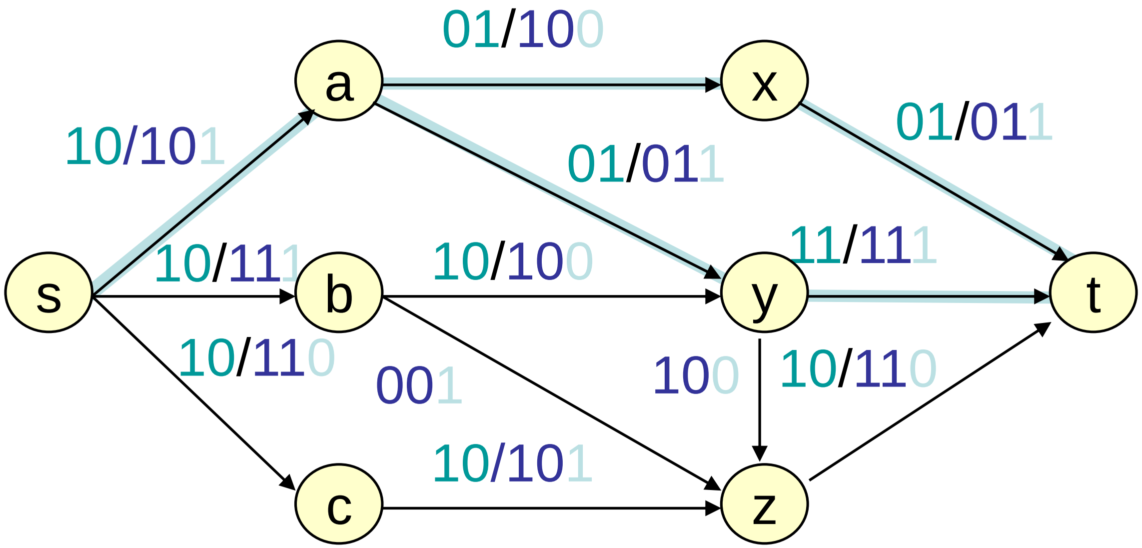
O(nm) time

Capacity Scaling Bit 2



Residual capacity across min cut is at most **m**
(either **0** or **1** times **Δ=2**)

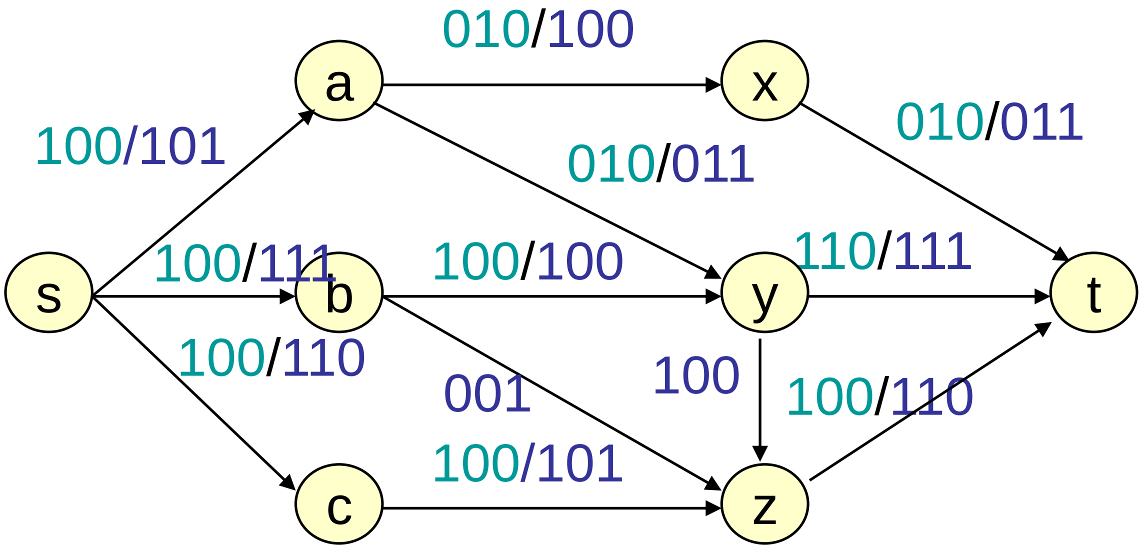
Capacity Scaling Bit 2



Residual capacity across min cut is at most **m**

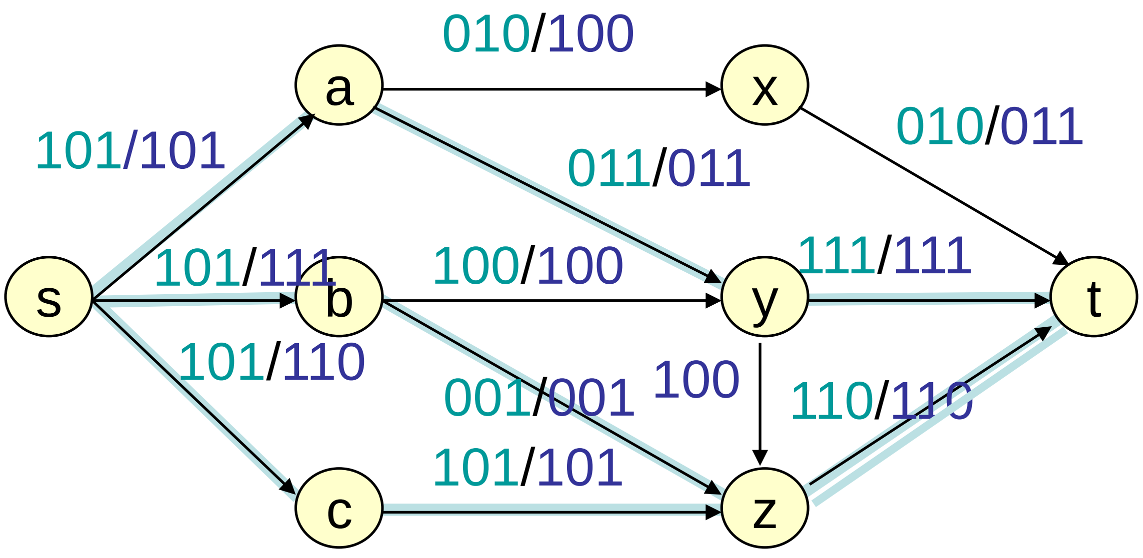
$\Rightarrow$ $\leq$ **m** augmentations

Capacity Scaling Bit 3



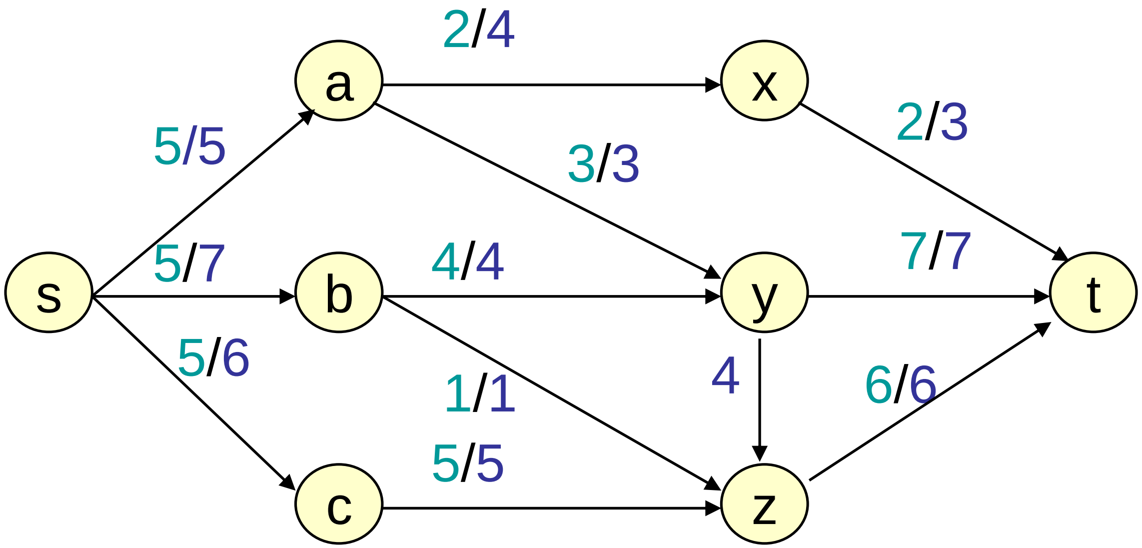
Residual capacity across min cut is at most **m**
(either **0** or **1** times **Δ=1**)

Capacity Scaling Bit 3



After $\leq$ **m** augmentations

Capacity Scaling Final



Capacity Scaling

Lemma: When the inner loop terminates for some D,

$$val(f^*) \leq val(f) + Dm$$

Proof:

When the inner loop terminates, there is no s-t path in $G_f(D)$.

Let S be the set of vertices reachable from s in $G_f(D)$.

Note that $cap(S, \overline{S}) \leq val(f) + Dm$.

Hence, $val(f^*) \leq val(f) + Dm$.

Capacity Scaling

Corollary: Inner loop takes $O(m)$ steps.

Proof:
Each step, $val(f)$ is increased by at least D.
But $val(f^*) \leq val(f) + 2Dm$ at the beginning.
Hence, there are at most $2m$ steps.

Theorem: Capacity scaling takes $O(m^2 \log U)$ time.
Proof:
Outer loop has $O(\log U)$ steps.
Inner loop has $O(m)$ steps.
Each step takes $O(m)$ time.

Algorithm:
Set $D = U$
While $D \geq 1$
- While there is s-t path p in $G_f(D)$
  - Augment along p by lowest edge capacity in $G_f(D)$
- $D \leftarrow \lfloor D/2 \rfloor$

Strongly polynomial: Edmonds-Karp Algorithm

Question: Can we pick better paths to augment?

Idea: Shortest Path!

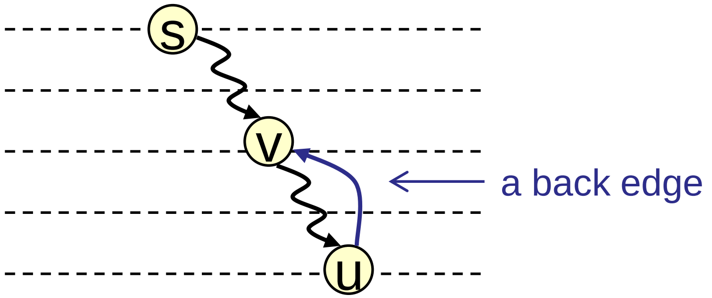
Edmonds-Karp Algorithm

Use a shortest augmenting path
(via Breadth First Search in residual graph)

Lemma:

Let f be a flow and P a shortest augmenting path.

Then no vertex is closer to s in G_f after augmentation along P.



Proof: Augmentation along P only

- deleting forward edges

  no new (hence no shorter) path created

- adds back edges that go to previous vertices along P

  BFS is unchanged, since v visited before (u, v) examined

Theorem

Edmonds-Karp performs $O(mn)$ flow augmentations

Proof:

Call (u, v) critical for augmenting path P if it's closest to s with min residual capacity.

It will disappear from G_f after augmenting along P.

For (u, v) to be critical again, the (u, v) edge must re-appear in G_f but that will only happen when the distance to u has increased by 2 (next slide)

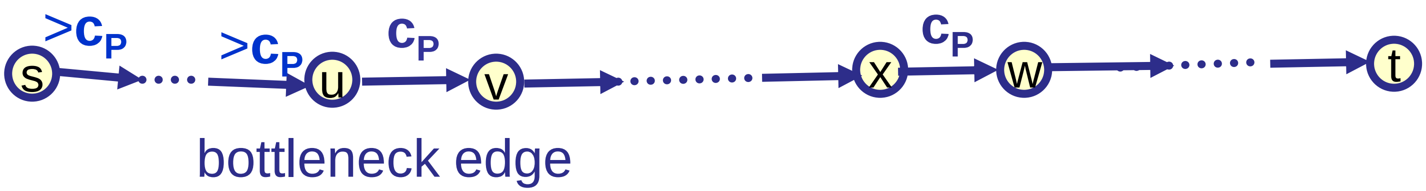
It won't be critical again until farther from s so each edge critical at most $n/2$ times.

Corollary: Total time is $O(m^2 n)$.

Distance for bottleneck edges

Let $d_f(s, v)$ be the distance from s to v on G_f.

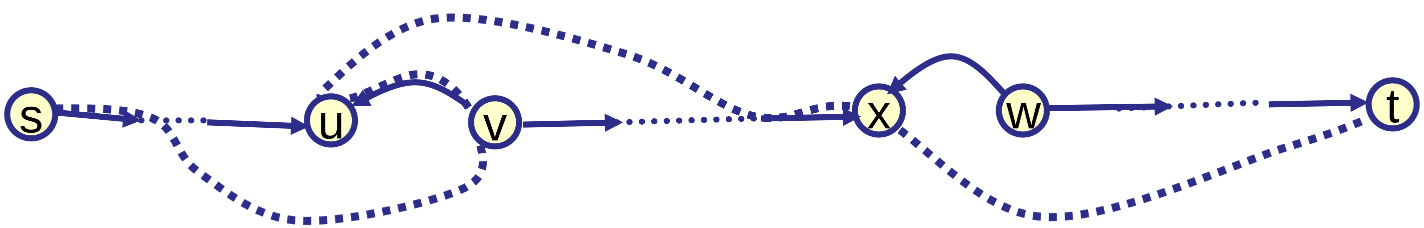
Shortest s-t path P in G_f



After augmenting along P

$d_f(s, v) = d_f(s, u) + 1$ since this is a shortest path

For (u, v) to be bottleneck again for some flow f'



$$d_{f'}(s, u) = d_{f'}(s, v) + 1 \geq d_f(s, v) + 1 = d_f(s, u) + 2$$

Faster Strongly polynomial: Dinic algorithm

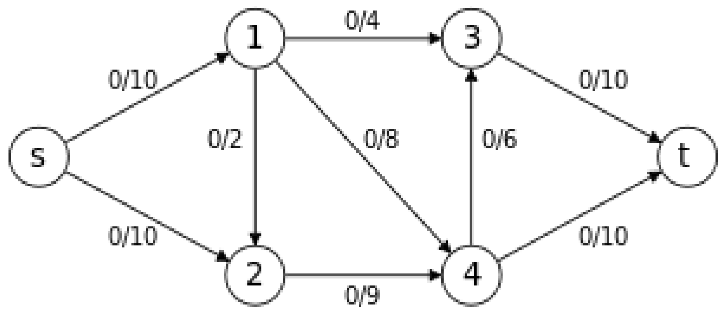
Question: What is better than 1 shortest paths?

Idea: Multiple Shortest Paths!

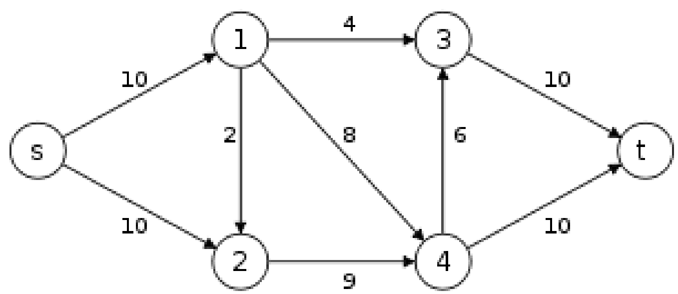
Dinic's algorithm

Send as many shortest paths as possible at the same time.

G

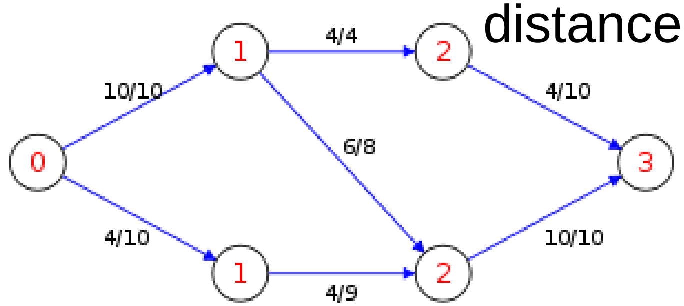


G_f



distance to s

Keep on forward edges



Dinic's algorithm

Let f be some flow.

The level graph L_f is the graph with edges given by

$$\{(u,v) \in G_f : d_{G_f}(s,v) = d_{G_f}(s,u)\}$$

We call f' is a blocking flow in L_f if

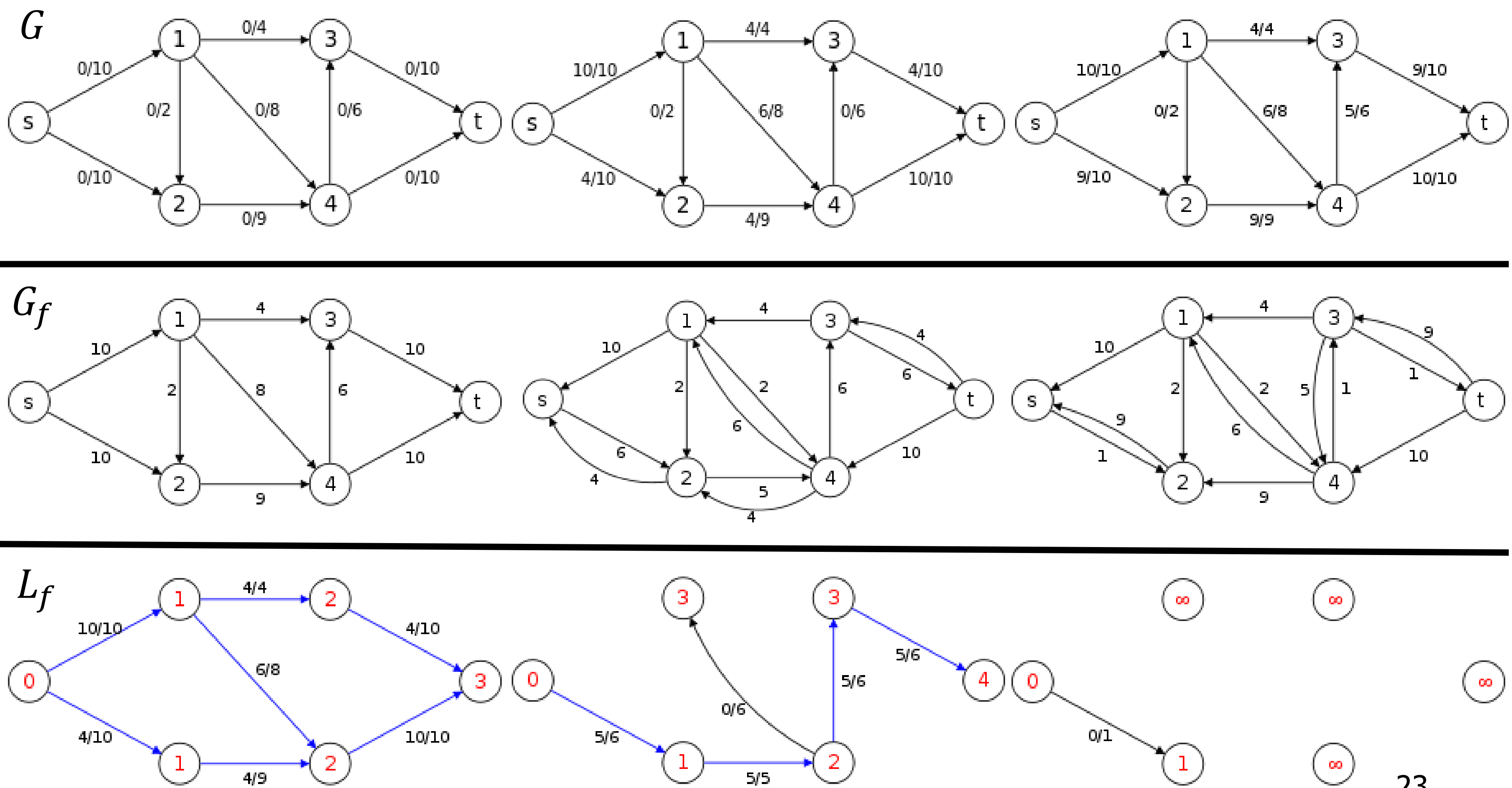
$$\{e \in L_f : f'(e) < c_{G_f}(e)\} \text{ has no } s\text{-}t \text{ path.}$$

Algorithm:

- $f = 0$
- While there is s-t path p in G_f
  - Compute the level graph L_f
  - Find a blocking flow f'.
  - Update $f \leftarrow f + f'$.

Dinic's algorithm

Send as many shortest paths as possible at the same time.



Dinic's algorithm

Lemma: $d_{G_f}(s,t)$ increase every iteration.

Proof (Draft):

If distance doesn't change, a shortest path on the new graphs must be on the level graph.

But there is no s-t path in the level graph after sending the blocking flow.

Corollary: There are at most n iterations.

Proof:

s-t distance is bounded by n and is increased by 1 every step.

We can find blocking flow in $O(mn)$ time picking path 1 by 1.

Hence, Dinic's algorithm takes $O(mn^2)$ time.

Link Cut Tree

There are a data structure that supports the following.

- make_tree(): Return a new vertex in a singleton tree.
- link(v,w,x): Make vertex v a new child of vertex w. Set the edge capacity to x.
- cut(v): Delete the edge between v and its parent.
- find_root(v) – Return the root of the tree that contains v.
- find_min(v) – Return the edge with minimum capacity on the v-root path.
- subtract(v,x) – Subtract x from the capacity on the v-root path.

Furthermore, all steps takes $O(\log n)$ time.

With this, we can send a flow in $O(\log n)$ time in the level graph.

Hence, Dinic's algorithm takes $O(mn \log n)$ time.

Can we do it even faster?

Runtimes…

Previous Best: $\tilde{O}(\min(m^{1.5}, mn^{2/3}))$ [Even-Tarjan 75, Goldberg-Rao 89]

Unfortunately, this slide was made in 2011 (during my undergrad).

Undirected graph

$mn^{1/3}/\epsilon^{11/3}$ [Christiano-Kelner-Madry-Spielman-Teng 2011]

$mn^{1/3}/\epsilon^{2/3}$ [Lee-Rao-Srivastava 2013]

m/ϵ^2 [Sherman 2013, Kelner-Lee-Orecchia-Sidford 2014]

m/ϵ [Sherman 2017]

$m + \sqrt{mn}/\epsilon$ [Sidford-Tian 2020]

Directed graph

$m^{10/7}U^{1/7}$ [Madry 2013, Madry 2016]

$m\sqrt{n}$ [Lee-Sidford 2013]

$m^{11/8}U^{1/4}$ [Liu-Sidford 2020]

$m^{4/3}U^{1/3}$ [Liu-Sidford 2020, Kathuria 2020]

$m + n^{1.5}$ [Brand-Lee-Liu-Saranurak-Sidford-Song-Wang 2021]

$m^{3/2-1/328}$ [Gao-Liu-Peng 2021]

$m^{3/2-1/58}$ [Brand-Gao-Jambulapati-Lee-Liu-Peng-Sidford 2021]

Shortest Path and Maxflow?

Given an undirected graph with unit capacity and unit length.

What is the relation between shortest path and maxflow?

Let $\mathcal{F}$ be the set of s-t flow with value 1.

Shortest path problem: $\min_{f \in \mathcal{F}} ||f||_1$.

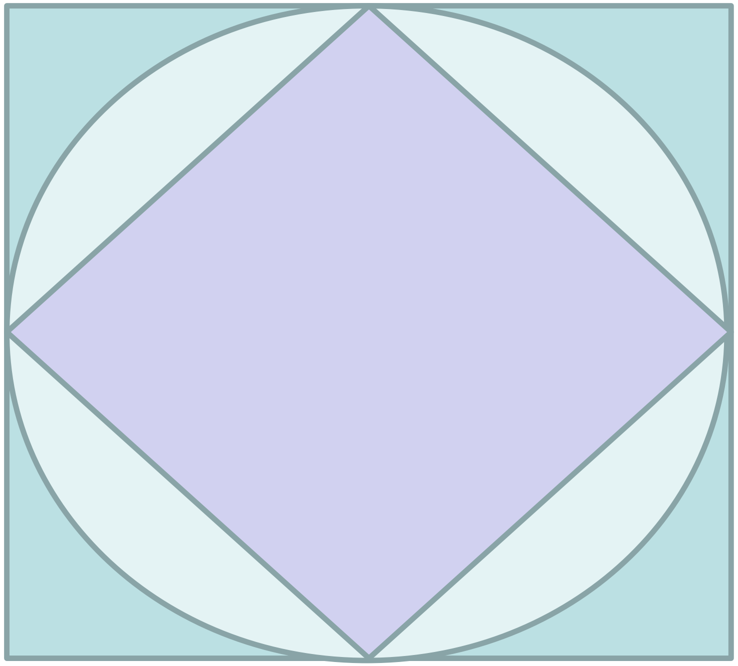
Maxflow problem: $\min_{f \in \mathcal{F}} ||f||_\infty$.

Solving maxflow via shortest path is like using ℓ_1 problem to approximate ℓ_∞ problem.

Shortest Path and Maxflow?

Fact: $\min_{f \in \mathcal{F}} ||f||_2$ can be solved in $O(m \log m)$ time.

[Spielman-Teng 2003]



Instead of augmenting using shortest path,

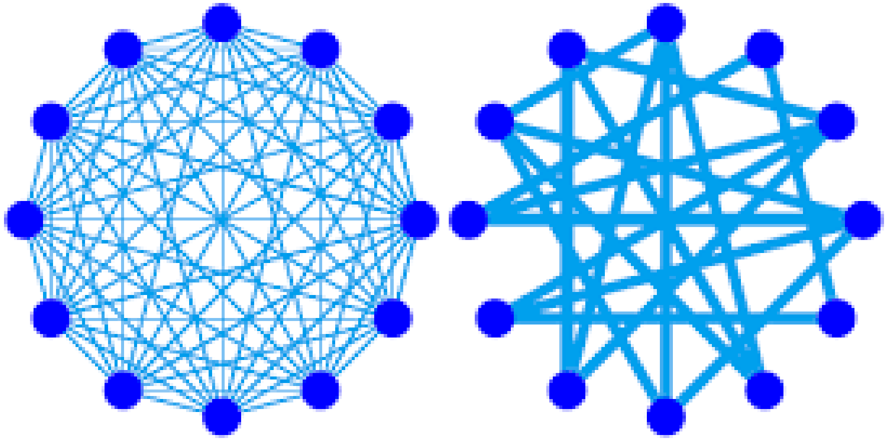
"augment" using ℓ_2 flow.

This gives $m\sqrt{n}$ time algorithm. (2014)

Approximate Graphs with Sparse Graphs

Every graph G can be approximated by a sparse graph G':

- G' has $O(n)$ edges
- $cap_G(S,\overline{S}) = (1 \pm \frac{1}{2})cap_{G'}(S,\overline{S})$ for all set S.

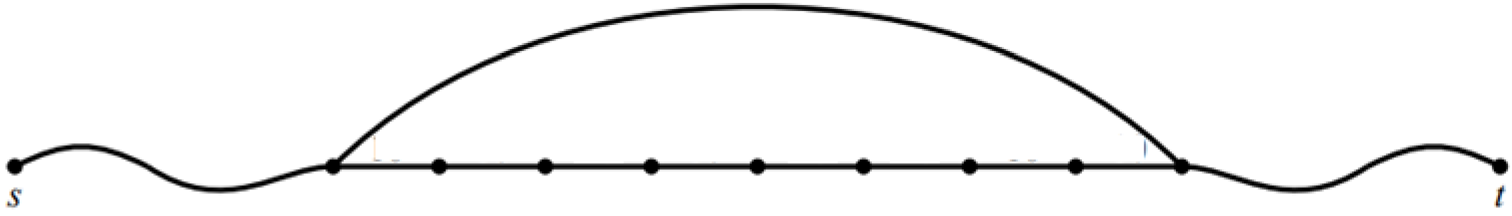


Using this, one can "find" the augmenting path on the sparse graph.

This gives $m + n^{3/2}$ time algorithm. (2020)

Creating shortcut in the graph

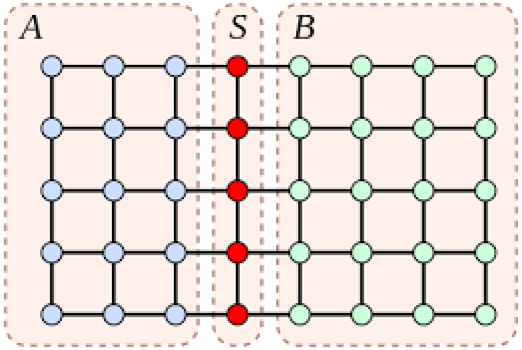
Instead of looking at all vertices all the time,
we can random sample some vertices and shortcut the graph.



This gives $m^{3/2-1/58}$ time algorithm. (2022)

How about simpler graphs?

Many graphs in practice has structures.



One common class is planar graphs.

For these graphs,

- we can solve maxflow in nearly linear time (2009)
- we can solve mincost flow in nearly linear time (2022)

Sally Dong

Guanghao Ye
(was a 421 student)