Algorithms - Fall 2023

Applications of Flow

Recap

- Sign up for oral grading + HW & groups
- Sample final: I'll bring next week
- Final readings added to Canvas
- Review session: Wed. of finals week at either 2 or 3pm

Max flow/Min Cut:

- FF: Residual graphs: $O((V+E)\cdot f^*)$
- Edmonds-Karp: $O(E^2 \log E \log f^*)$
- BFS based: $O(VE^2)$

Technique	Direct	With dynamic trees	Source(s)
Blocking flow	$O(V^2E)$	$O(VE\log V)$	[Dinitz; Karzanov; Even and Itai; Sleator and Tarjan]
Network simplex	$O(V^2E)$	$O(VE\log V)$	[Dantzig; Goldfarb and Hao; Goldberg, Grigoriadis, and Tarjan]
Push-relabel (generic)	$O(V^2E)$	—	[Goldberg and Tarjan]
Push-relabel (FIFO)	$O(V^3)$	$O(VE\log(V^2/E))$	[Goldberg and Tarjan]
Push-relabel (highest label)	$O(V^2\sqrt{E})$	—	[Cheriyan and Maheshwari; Tunçel]
Push-relabel-add games	—	$O(VE\log_{E/(V\log V)} V)$	[Cheriyan and Hagerup; King, Rao, and Tarjan]
Pseudoflow	$O(V^2E)$	$O(VE\log V)$	[Hochbaum]
Pseudoflow (highest label)	$O(V^3)$	$O(VE\log(V^2/E))$	[Hochbaum and Orlin]
Incremental BFS	$O(V^2E)$	$O(VE\log(V^2/E))$	[Goldberg, Held, Kaplan, Tarjan, and Werneck]
Compact networks	—	$\boxed{O(VE)}$	[Orlin]

Figure 10.10. Several purely combinatorial maximum-flow algorithms and their running times.

Many use very different techniques

- linear programming
- complex data structures
- not residual graphs

Topics in Ch. 11

A mess of different ideas!

① Matchings:

identify a way to pair up items

Build G': More pairs $\Longleftrightarrow$ larger flow

② Disjoint paths:

Modify G:

Find paths that avoid each other.

③ "Tuple" Selection

Build a graph! flow paths give selection

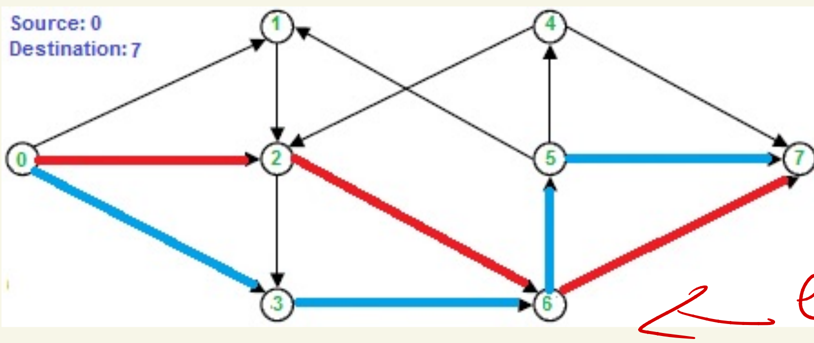
Magic:

First problem

What if we want non-intersecting paths from s to t?

Two variants:

- Edge disjoint: No 2 paths visit the same edge



← edge disjoint but not vertex disjoint

- Vertex disjoint: no 2 paths visit the same vertex

Note: different! And both model useful cases

Key: Flow will decompose into paths!

Edge disjoint:

Input: unweighted directed graph $G=(V,E)$
plus $s,t \in V$.

For each edge $e=\overrightarrow{uv}$:
set capacity$(e) \leftarrow 1$] $O(E)$

$f \leftarrow$ Call flow algorithm] $O(V \cdot E)$

How to get paths?

Initialize empty list for paths

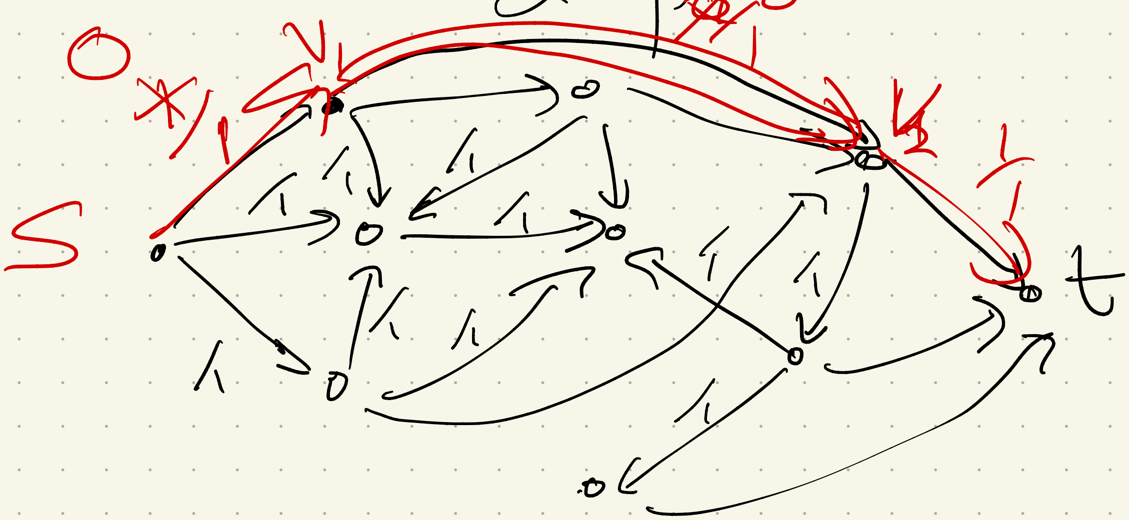
$i \leftarrow 1$

while (value$(f) > 0$)
  find $s \to v$ edge w/ $f(s \to v) = 1$
  add $(s \to v)$ to paths[i]
  set $f(s \to v) \leftarrow 0$
  while $(v \neq t)$
    find edge $v \to u$
      with $f(v \to u) = 1$
    add $(v \to u)$ to paths[i]
    set $f(v \to u) \leftarrow 0$
    $v \leftarrow u$
  i++

list of edges

Since all flows are integral, + capacity of every edge is $= 1$

$\Rightarrow$ no edge will be in 2 paths.



paths: 1 : $s \to v_1$, $v_1 \to v_2$, $v_2 \to t$

2 :

correctness: Any set of k disjoint paths

$\Rightarrow$ flow of value k

any flow of value k $\Rightarrow$ set of k paths

Vertex disjoint: ← find paths which don't share a vertex

Build a new graph $\tilde{G}$:

unweighted

$G=(V,E)$

$s, t \in V$

S for flow

t for flow

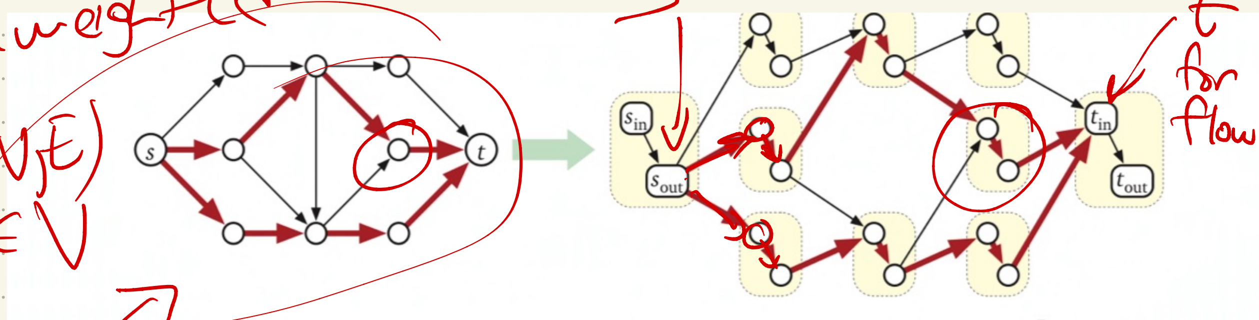


Figure 11.1. Reducing vertex-disjoint paths in G to edge-disjoint paths in $\bar{G}$.

V vertices

Edges

$2V \to V(\tilde{G}) = 2$ vertices v_{in}, v_{out} for every $v \in G$

$V+E \to E(\tilde{G}) =$ for any $u \to v \in E$, add $u_{out} \to v_{in}$ to $\bar{E}(\tilde{G})$

add $v_{in} \to v_{out}$ for every $v \in G$

Add capacity $= 1$ to each edge.

Any flow path that enters v_{in} will exit v_{out}, so

$$f(v_{in} \to v_{out}) = 1 = c(v_{in} \to v_{out})$$

Result:

Another (his) variant

Suppose edges are unlimited, but vertices have capacities.

<u>Ex</u>: internet routing: packets queue up at routers/switches

<u>So</u>: $G=(V,E)$ + $c(v)$ give capacities on <u>vertices</u>

How to do flow?

Build $\tilde{G}$: (with only edge capacities)

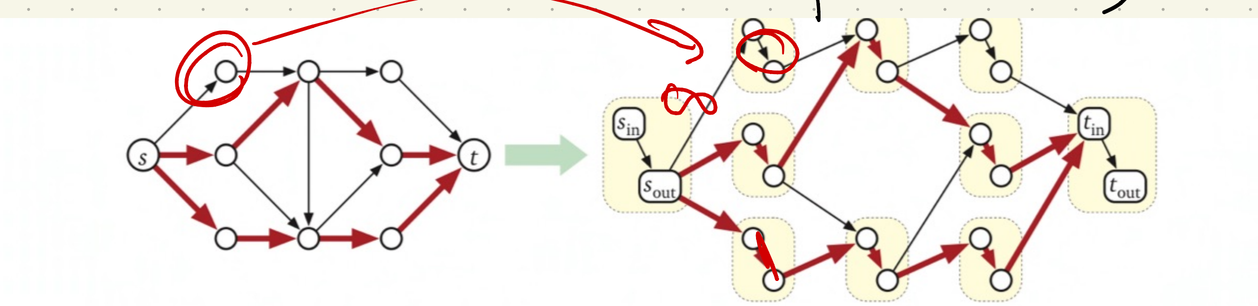


Figure 11.1. Reducing vertex-disjoint paths in G to edge-disjoint paths in $\bar{G}$.

Reductions:

Correctness

Solution in G

$\Rightarrow$ max flow in $\tilde{G}$

max flow in $\tilde{G}$

$\Rightarrow$ ~~max flow~~ in G

solution

So: compute in $\tilde{G}$

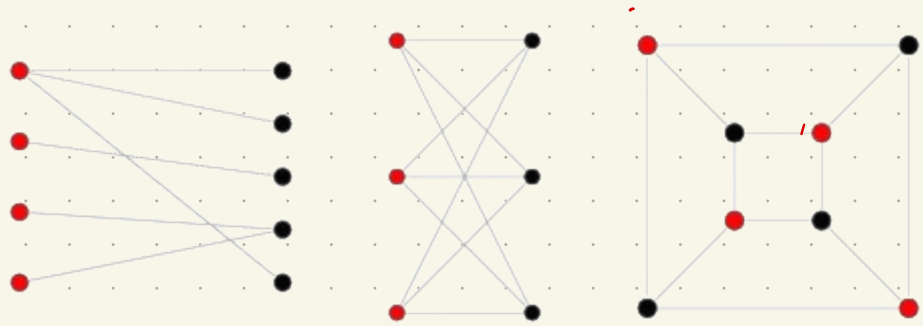
Runtime: $\boxed{O(V(\tilde{G}) \cdot E(\tilde{G}))}$

=

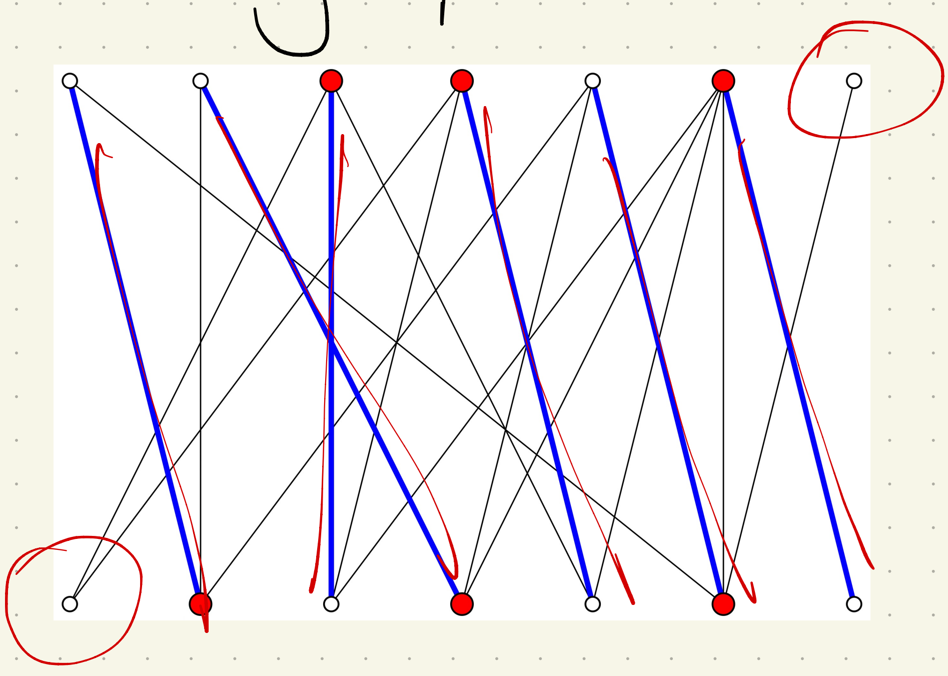
Bipartite Graphs

Any graph where vertices can be divided into 2 sets (usually L & R) s.t. no edges exist inside L or R

Ex:



Maximum matching: find edges (no 2 edges per vertex)



Instead, use flows:

Convert G to $\tilde{G}$:

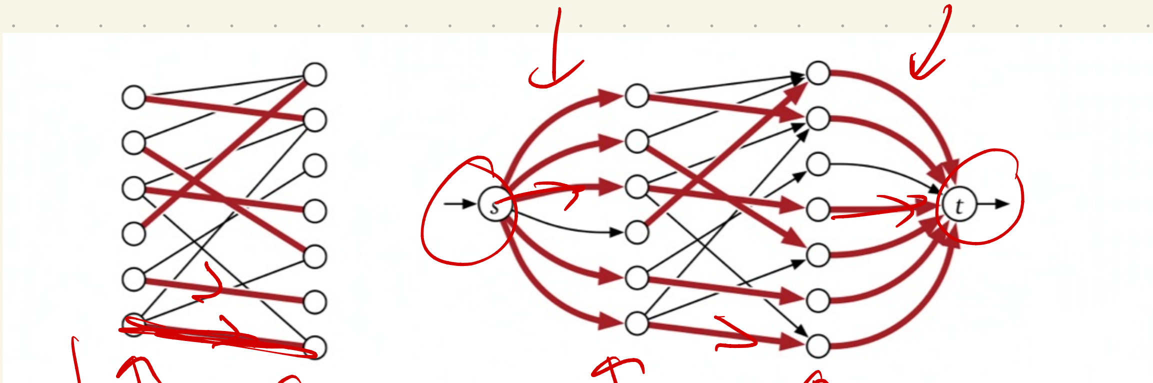


Figure 11.2. A maximum matching in a bipartite graph G, and the corresponding maximum flow in G'.

L R

G

$\tilde{G}$: add 2 new vertices, s & t, all all $v \in G$ to $\tilde{G}$

Edges for $\tilde{G}$: every edge in G, add directed version in $\tilde{G}$

for all $v \in L$, add $s \to v$

for all $u \in R$, add $u \to t$

give every edge cap $= 1$

<u>Algorithm</u>: Given $G = (V, E)$
with $V = L \cup R$ (bipartite)

// build $\tilde{G}$

(prev slide)

$\tilde{V} = V + 2$

$\tilde{E} = V + E$

// run flow

$O(\tilde{V}\tilde{E}) = O(V(V+E))$

// get matching

track flow paths

↳ any $L \to R$ edge w/ flow $= 1$
is in matching

<u>Runtime</u>:

Correctness:

① Any matching in G
$\Rightarrow$ flow in $\tilde{G}$

② Any flow in $\tilde{G}$ $\Rightarrow$ matching in G

therefore, max flow in $\tilde{G}$ $\Leftrightarrow$ max matching in G

Aside: How??

(FF is somehow improving matching...)

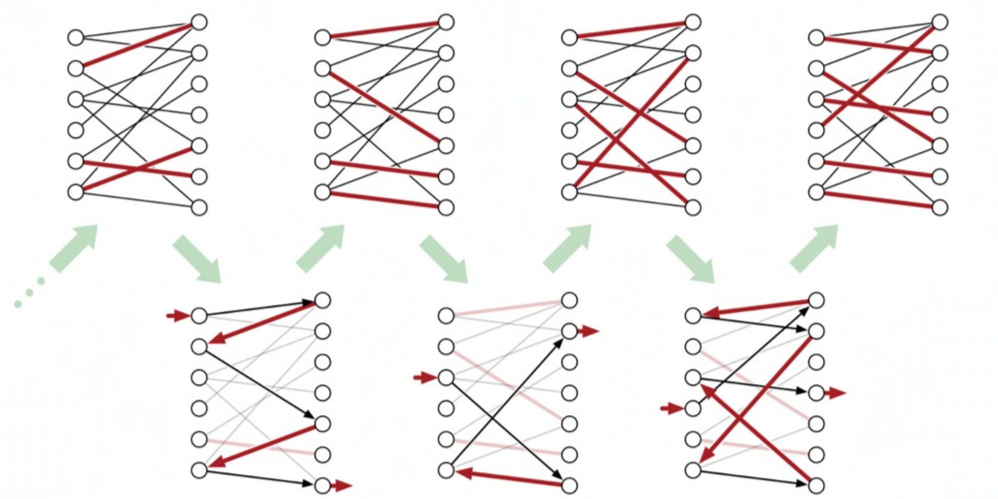


Figure 11.3. An increasing sequence of matchings connected by alternating paths.

Crazier "word problem" examples

A company sells k products, & ~~keeps~~ records on customers.

Goal: Design a survey to send to n customers, to get feedback.

- Each customer's survey shouldn't be too long, & should ask only about products they purchased
- Each product needs some # of reviews from different customers

Input: - k products
- n customers
- records of who bought what: a_{ij} for $i \le k$, $j \le n$

- For each customer, C_i is max # of products to ask them about

- for each product, P_i is minimum # of reviews needed.

Can we design a survey?

Algorithm

Runtime + correctness