

Algorithms - Fall 2023

Applications
of Flow



Recap

- Sign up for oral grading
+ HW of 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: $\mathcal{O}(V+E) \cdot f^*$
- Edmonds - Karp: $\mathcal{O}(E^2 \log E \log f^*)$
- BFS based: $\mathcal{O}(VE^2)$

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

② Disjoint paths:

Modify G :

Find paths that avoid each other.

③ "Tuple" Selection

Build a graph: flow paths give selection

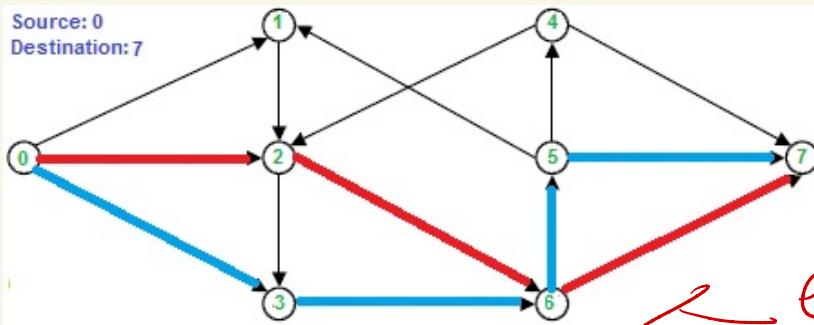
Magic:

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 graph $G = (V, E)$
plus $s, t \in V$.

For each edge $e = \overrightarrow{uv}$:

set $\text{capacity}(e) \leftarrow 1$] $O(E)$

$\uparrow f \leftarrow \text{Call flow algorithm}$] $O(VE)$

How to get paths?

Initialize empty list for paths

while $(\text{value}(f) > 0)$

find $s \rightarrow v$ edge w/ $f(s \rightarrow v) > 1$

add $(s \rightarrow v)$ to paths[i]

set $f(s \rightarrow v) \leftarrow 0$

while ($v \neq t$)

find edge $v \rightarrow u$

with $f(v \rightarrow u) = 1$

add $(v \rightarrow u)$ to paths[i]

set $f(v \rightarrow u) \leftarrow 0$

$v \leftarrow u$

i++

Since all flows are integral,
+ capacity of every edge
is $\underline{= 1}$

\Rightarrow no edge will be in
2 paths.



paths: 1: $s \rightarrow v_1, v_1 \rightarrow v_2, v_2 \rightarrow t$
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} :

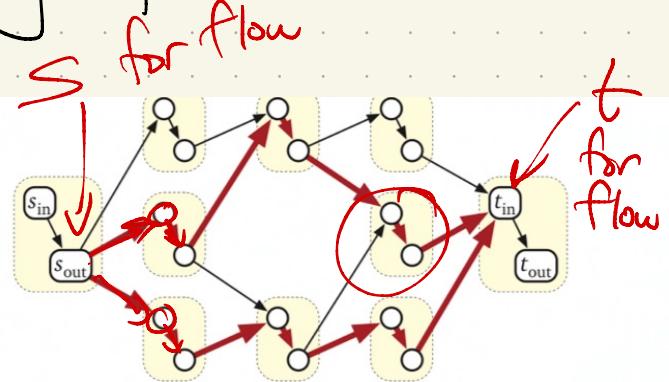
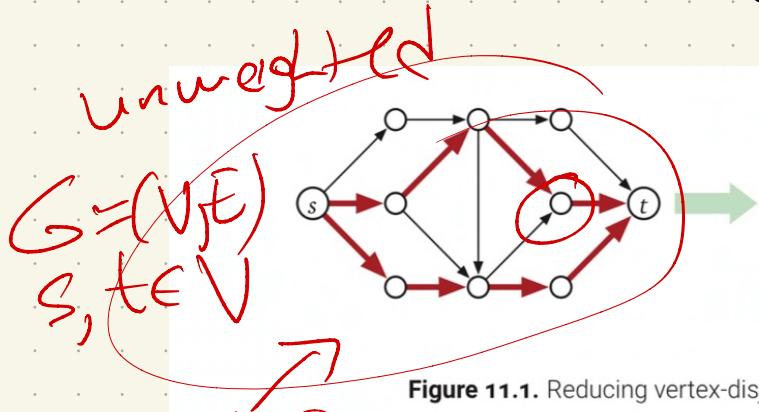


Figure 11.1. Reducing vertex-disjoint paths in G to edge-disjoint paths in \tilde{G} .

$V \rightarrow V(\tilde{G}) = 2$ vertices for every $v \in G$

$E \rightarrow E(\tilde{G}) =$ for any $u \rightarrow v \in E$, add $v_{in} \rightarrow v_{out}$ for every $v \in G$

Add capacity = 1 to each edge.

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

$$f(v_{in} \rightarrow v_{out}) = 1 = C(v_{in} \rightarrow v_{out})$$

Result:

Another (his) variant

Suppose edges are unlimited,
but vertices have capacities.

Ex: Internet routing:

packets queue up at
routers/switches

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

How to do flow?

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

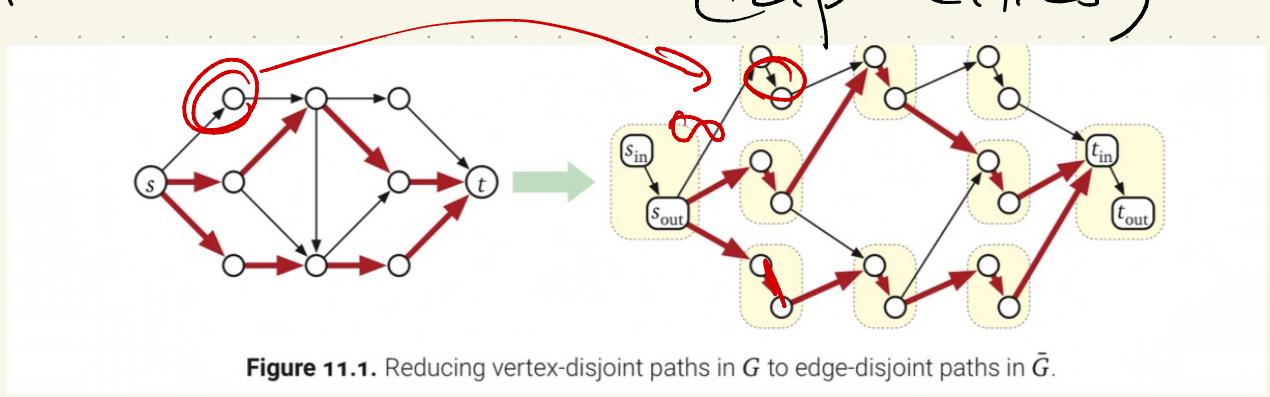


Figure 11.1. Reducing vertex-disjoint paths in G to edge-disjoint paths in \bar{G} .

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}

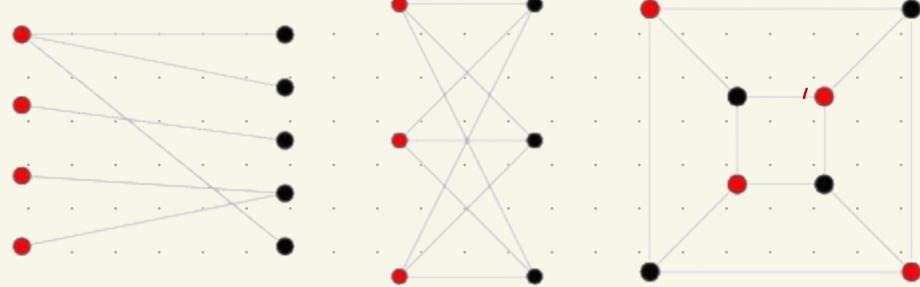
Runtime: $O(V(\tilde{G}) \cdot E(\tilde{G}))$

=

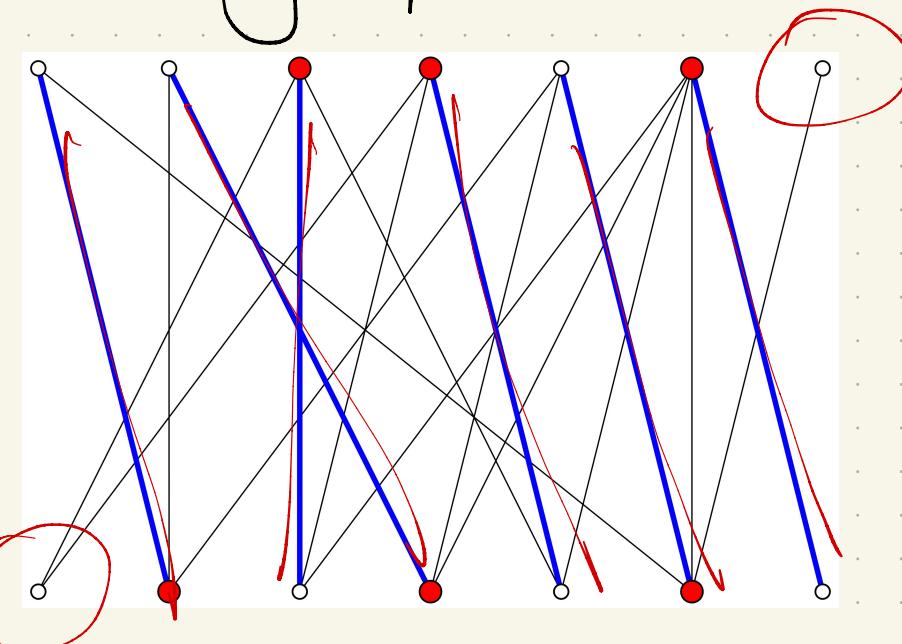
Bipartite Graphs

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} :

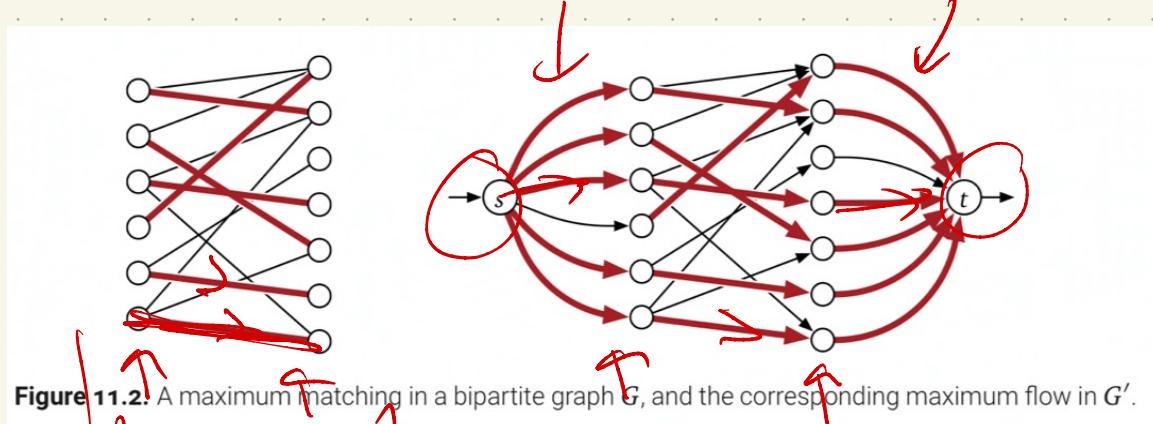


Figure 11.2: A maximum matching in a bipartite graph G , and the corresponding maximum flow in G' .



G

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

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

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

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

give every edge $cap = 1$

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

// build \tilde{G}
(new side)

$$\tilde{V} = V + 2$$

$$\tilde{E} = V + E$$

// run flow

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

// get matching

track flow paths

↳ any $L \rightarrow R$ edge w/ $\text{flow} = 1$
is in matching

Runtime:

Correctness:

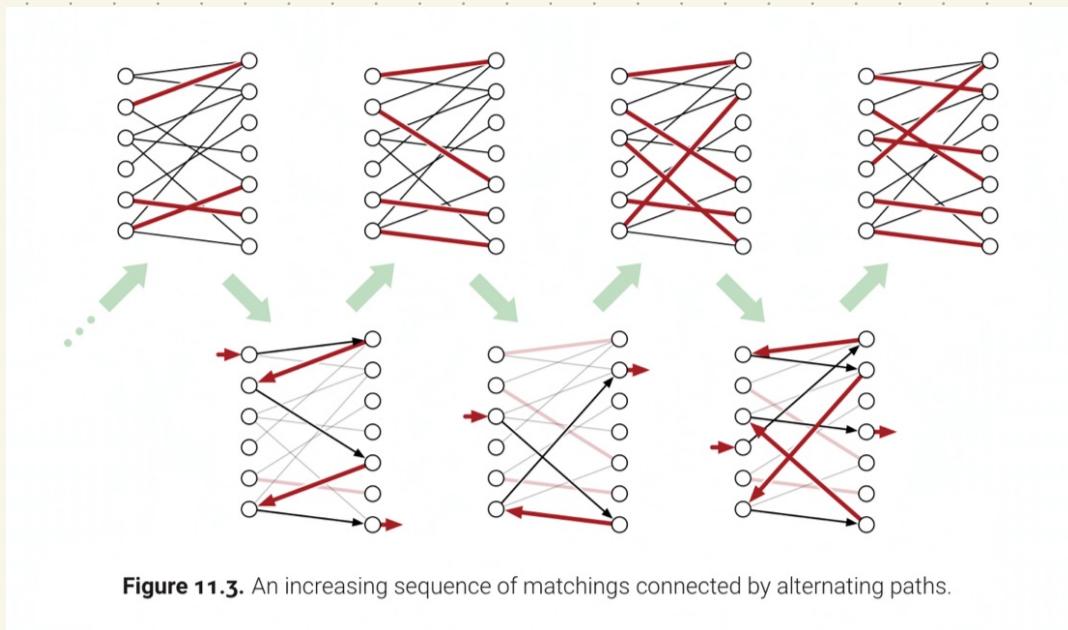
① Any matching in \tilde{G}
 \Rightarrow flow in \tilde{G}

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

therefore, max flow \Leftrightarrow max
matching
in G

Aside: How??

(FF is somehow improving matching...)



Crazier "word problem" examples

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

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

- 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 $i \leq k, j \leq n$
 - 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