Algorithms – Fall 2020

Flows
(final day?)

Recap

- HW due Saturday
  caution: I'm often busy
  Sunday.
- Reading as usual.
- Next HW - probably
  oral grading

Max flow/Min Cut:

- FF: $O((V+E)f^*)$
- Edmonds-Karp: $O(E^2 \log E \log f^*)$
- BFS based: $O(VE^2)$ (no f^*)

based on G_f

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

Still active!

8:46 AM Mon Oct 26 arxiv.org 98%

Showing 1–4 of 4 results

Search v0.5.6 released 2020-02-24 Feedback?

Query: order: -announced_date_first; size: 50; classification: Computer Science (cs); include_cross_list: True; terms: AND title=Max flow algorithm

Simple Search

Refine query New search

50 results per page. Sort results by Announcement date (newest first) Go

1. arXiv:2009.03260 [pdf, ps, other] cs.DS math.OC

   A Potential Reduction Inspired Algorithm for Exact Max Flow in Almost $\widetilde{O}(m^{4/3})$ Time

   Authors: Tarun Kathuria

   Abstract: We present an algorithm for computing s-t maximum flows in directed graphs in $\widetilde{O}(m^{4/3+o(1)}U^{1/3})$ time. Our algorithm is inspired by potential reduction interior point methods for linear programming. Instead of using scaled gradient/Newton steps of a potential function, we take the step which maximizes the decrease in the potential value subject to advancing a certain amount o... ▽ More

   Submitted 7 September, 2020; **originally announced** September 2020.

2. arXiv:1910.04848 [pdf, other] cs.DS cs.CC

   A Fast Max Flow Algorithm

   Authors: James B. Orlin, Xiao-Yue Gong

   Abstract: In 2013, Orlin proved that the max flow problem could be solved in $O(nm)$ time. His algorithm ran in $O(nm + m^{1.94})$ time, which was the fastest for graphs with fewer than $n^{1.06}$ arcs. If the graph was not sufficiently sparse, the fastest running time was an algorithm due to King, Rao, and Tarjan. We describe a new variant of the excess scaling algorithm for the max flow problem whose runn... ▽ More

   Submitted 10 October, 2019; **originally announced** October 2019.

   Comments: 35 pages

3. arXiv:1901.01412 [pdf, other] cs.DS

   New Algorithms and Lower Bounds for All-Pairs Max-Flow in Undirected Graphs

   Authors: Amir Abboud, Robert Krauthgamer, Ohad Trabelsi

   Abstract: We investigate the time-complexity of the All-Pairs Max-Flow problem: Given a graph with n nodes and m edges, compute for all pairs of nodes the maximum-flow value between them. If Max-Flow (the version with a given source-sink pair s, t) can be solved in time $T(m)$, then an $O(n^2) \cdot T(m)$ is a trivial upper bound. But can we do better? For directed graphs, recent results in fine-grai... ▽ More

   Submitted 9 July, 2019; **v1** submitted 5 January, 2019; **originally announced** January 2019.

4. arXiv:1304.2338 [pdf, other] cs.DS

   An Almost-Linear-Time Algorithm for Approximate Max Flow in Undirected Graphs, and its Multicommodity Generalizations

   Authors: Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, Aaron Sidford

   Abstract: In this paper, we introduce a new framework for approximately solving flow problems in capacitated, undirected graphs and apply it to provide asymptotically faster algorithms for the maximum s-t flow and maximum concurrent multicommodity flow problems. For graphs with n vertices and m edges, it allows us to find an ε-approximate maximum s-t flow in time $O(m^{1+o(1)}\varepsilon^{-2})$, improvi... ▽ More

   Submitted 23 September, 2013; **v1** submitted 8 April, 2013; **originally announced** April 2013.

Even more known about "special" graphs:

Computer Science > Data Structures and Algorithms

Minimum Cuts in Surface Graphs

Erin W. Chambers, Jeff Erickson, Kyle Fox, Amir Nayyeri

(Submitted on 9 Oct 2019)

We describe algorithms to efficiently compute minimum (s, t)-cuts and global minimum cuts of undirected surface-embedded graphs. Given an edge-weighted undirected graph G with n vertices embedded on an orientable surface of genus g, our algorithms can solve either problem in $g^{O(g)}n \log\log n$ or $2^{O(g)}n \log n$ time, whichever is better. When g is a constant, our $g^{O(g)}n \log\log n$ time algorithms match the best running times known for computing minimum cuts in planar graphs. Our algorithms for minimum cuts rely on reductions to the problem of finding a minimum-weight subgraph in a given $\mathbb{Z}_2$-homology class, and we give efficient algorithms for this latter problem as well. If G is embedded on a surface with b boundary components, these algorithms run in $(g + b)^{O(g+b)}n \log\log n$ and $2^{O(g+b)}n \log n$ time. We also prove that finding a minimum-weight subgraph homologous to a single input cycle is NP-hard, showing it is likely impossible to improve upon the exponential dependencies on g for this latter problem.

Topics in Ch. 11

A mess of different ideas!

① Matchings:

Identify a way to pair up items — no G in this!

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

② Disjoint paths:

~~Modify G~~: use flow

Find paths that avoid each other.

③ "Tuple" Selection

Build a graph! flow paths give selection

Magic:

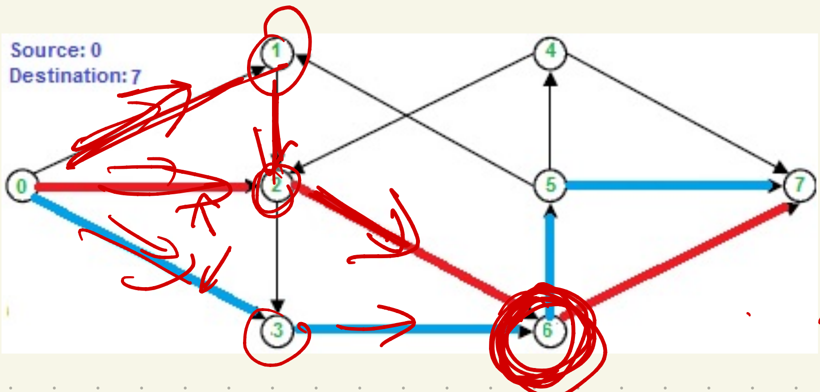
Flows/cuts solve a lot of problems!

First problem

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

Two variants:

- Edge disjoint: No 2 paths visit the same edge



← only 1 vertex disjoint path

- Vertex disjoint: no 2 paths visit the same vertex

stricter!

Note: different! And both model useful cases.

3 5 7 3 2 0 2

f(e)/c(e)

in CH10

Key: Flow will decompose into paths!

Edge disjoint:

Input: unweighted graph G, s,t.

Goal: Compute max # of e-d s~>t paths

$\tilde{G}$: Simply add capacity = 1 to each edge.

Run max flow, & ~~trace~~ decompose into paths from s ~> t.

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

⟹ no edge will be in 2 paths.

max flow ⟺ max # of e.d. paths

Vertex disjoint:

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

$u \; G \; v$ $\quad u_{out} \; v_{in}$ $\quad \tilde{G}$

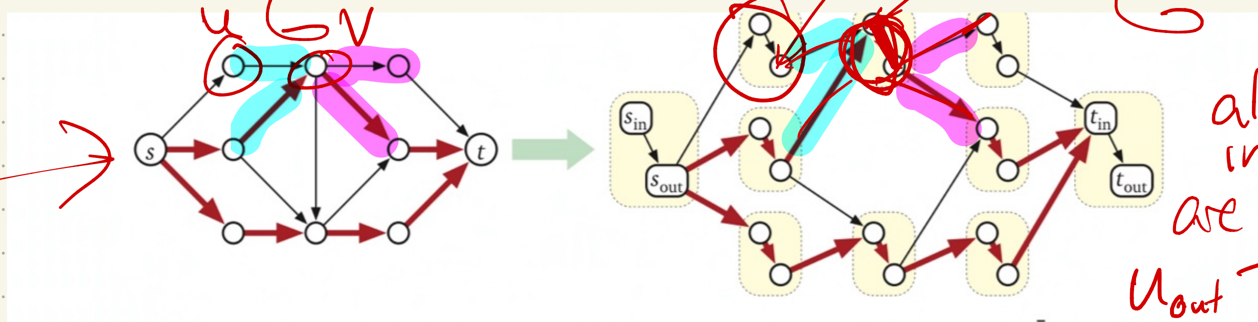


Figure 11.1. Reducing vertex-disjoint paths in G to edge-disjoint paths in $\bar{G}$.

all $u \to v$ in G are now $u_{out} \to v_{in}$

$V(\tilde{G}) =$ vert. of $\tilde{G} = 2 \times V(G)$

v_{in}, v_{out}

$E(\tilde{G}) =$ edge of $\tilde{G}$ = edges of G + edge b/t v_{in} & v_{out}

Add capacity = 1 to each edge. Run max flow alg($\tilde{G}$)

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

only 1 out edge to V_{out}

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

Result: each vertex appears in only 1 flow path

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

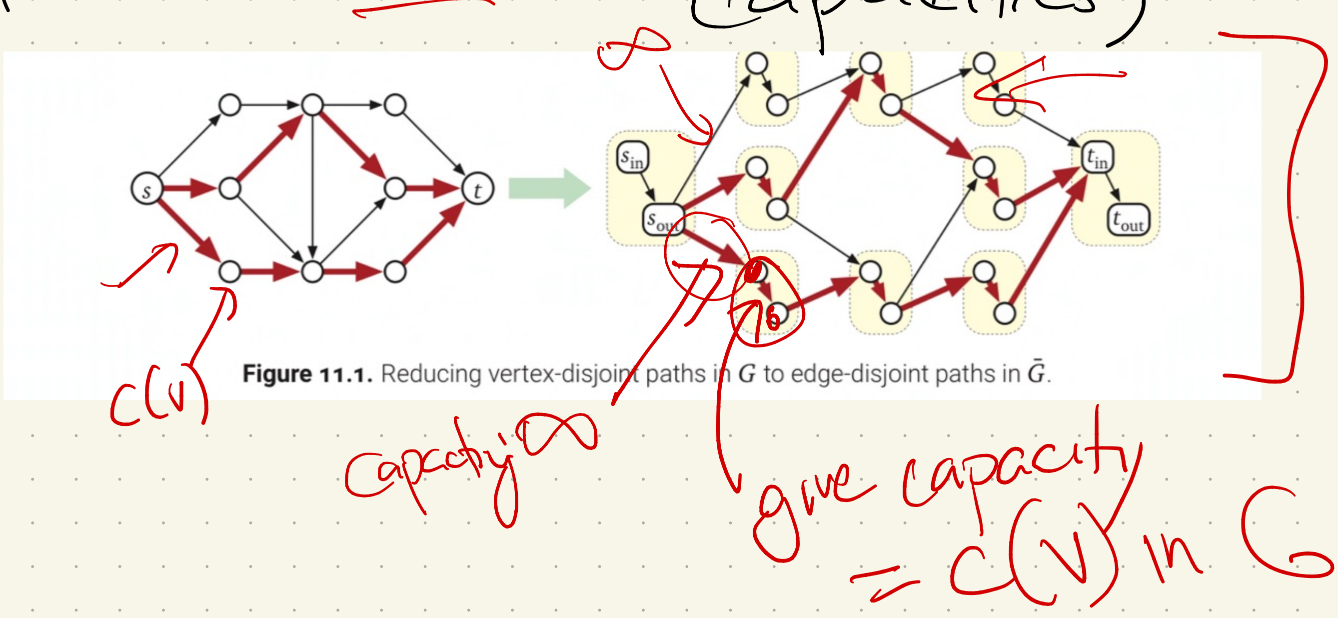


Figure 11.1. Reducing vertex-disjoint paths in G to edge-disjoint paths in $\bar{G}$.

Correctness

max flow in G (not exceeding vertex capacities)

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

∞ edges in $\tilde{G}$ are fine

in G flow, no vertex is exceeded $\rightarrow$ in $\tilde{G}$, $V_{in} \rightarrow V_{out}$ edges are also $\leq$ cap.

max flow in $\tilde{G}$

$\Rightarrow$ max flow in G

∞ edges are same

vertex capacities same as $V_{in} \rightarrow V_{out}$

So: compute in $\tilde{G}$ (but input was G)

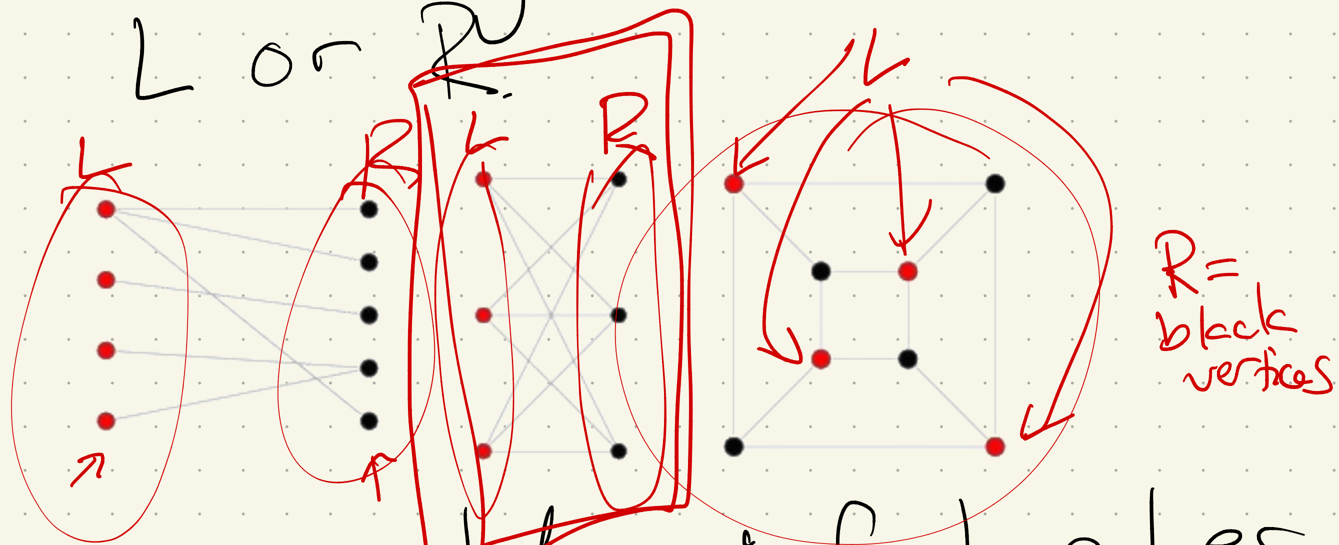
Runtime: $O(V(\tilde{G}) \times E(\tilde{G}))$ Orlin

$O(V^2 + VE) = O([2 \cdot V(G)] \times [E(G) + V(G)])$

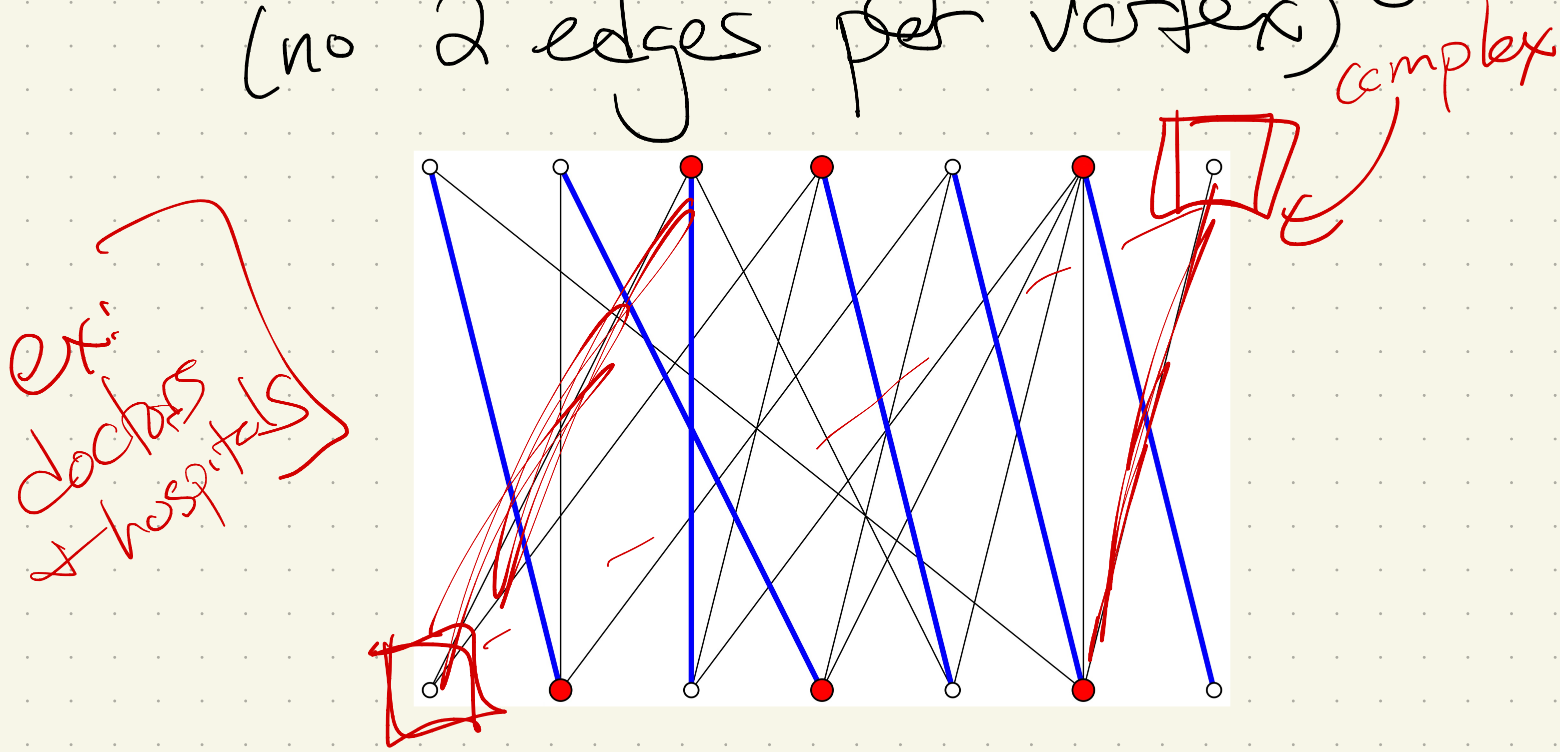
Bipartite Graphs

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)



How to compute?

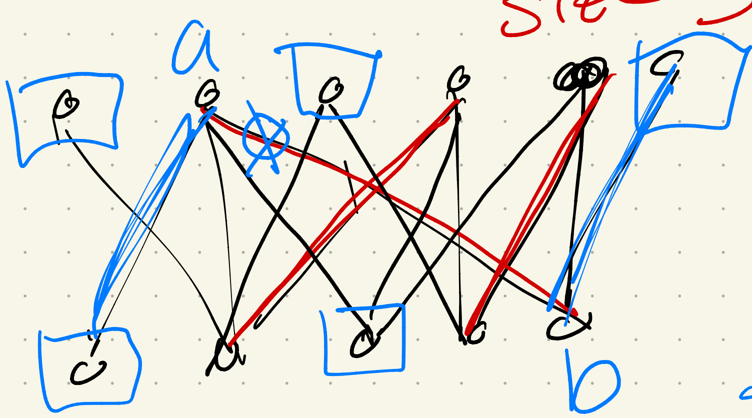
Greedy ideas:

- match vertex with minimum degree first
  - ↳ fails
  - gives a matching but not largest

Iterative improvement: ~~use flow~~

Given a (non-maximal) matching, how to improve?

size 3



remove a-b from matching, + add 2 new edges

↳ size 4

Instead, use flows:

Convert G to $\tilde{G}$:

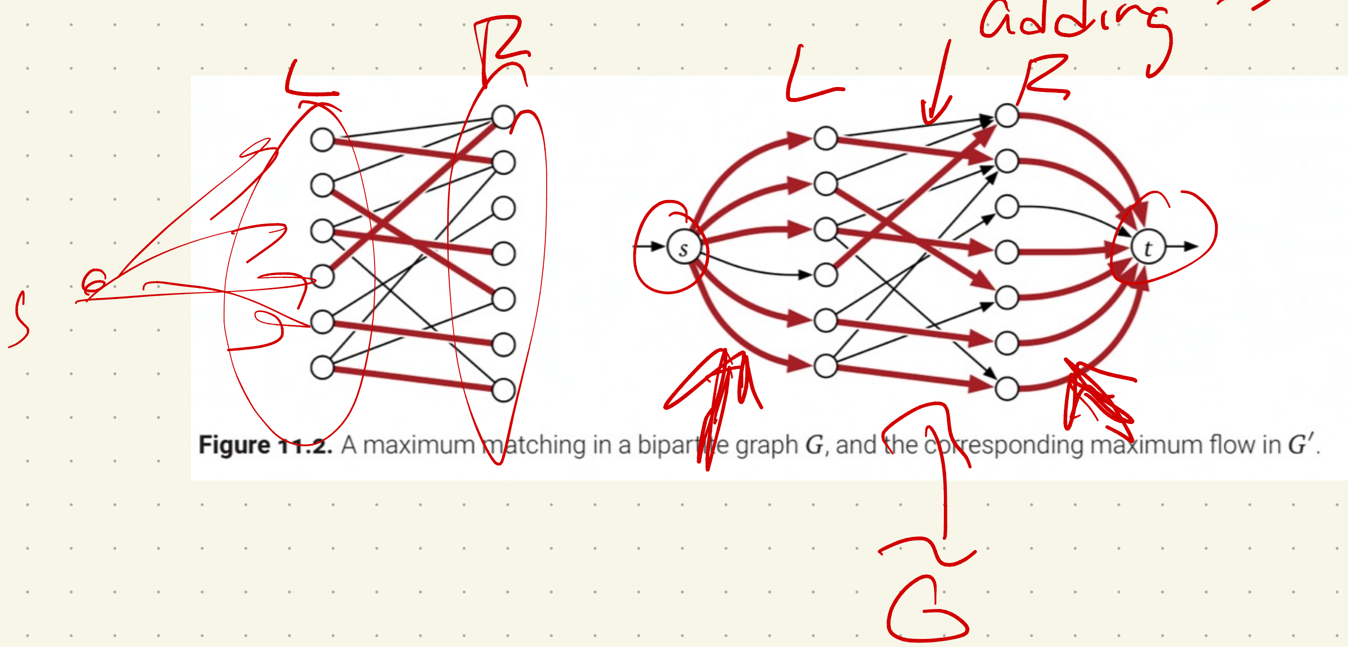


Figure 11.2. A maximum matching in a bipartite graph G, and the corresponding maximum flow in G'.

add s + t

direct edges of G from one set L to set R

add edges $s \to L$ (every vertex in L)

add: edges from vertices in R to t

give unit capacities to all edges

Algorithm: Given $G=(V,E)$

with $V = L \cup R$ (bipartite)

$\overset{L}{u} \to \overset{R}{v}$

// build $\tilde{G}$

add s & t

s's adj list = L

add t to adj list of all $v \in R$

direct & give unit cap

// run flow

call Orlin

// get matching

trace flow paths, build match that consists of any $L \to R$ edge with $f(e)=1$

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

$$= O\left((V+2)(E+V)\right)$$

$$= O(VE + V^2) = O(VE)$$

Correctness:

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

take matching edge $e = u \to v$
Set $f(e) = 1$
& $f(s \to u)$ & $f(v \to t) = 1$
valid, no capacity is too large

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

flow decomposes into unit paths, $s \Rightarrow u \to v \to t$
$u \in L$, $v \in R$

include $u \to v$ in matching
valid matching

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

Aside: How??

(FF is somehow improving matching...)

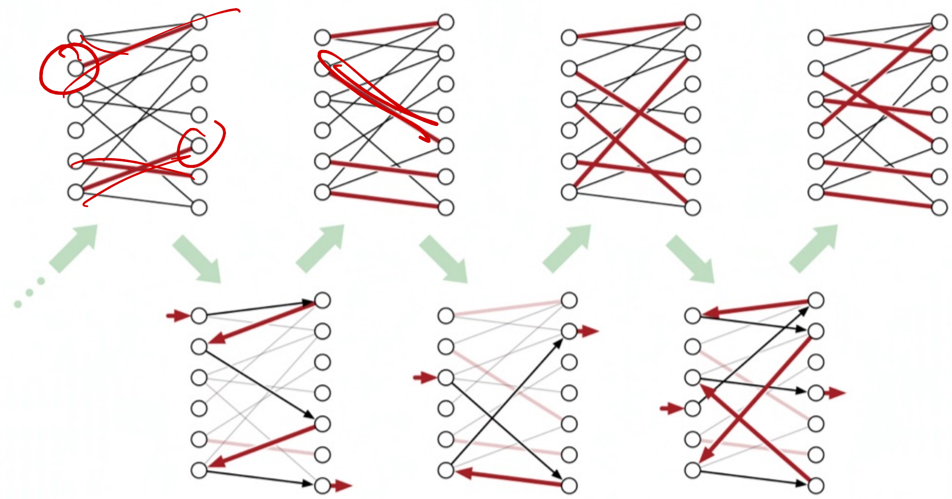


Figure 11.3. An increasing sequence of matchings connected by alternating paths.

this is our iterative improvement idea using G_f

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