

# 07. Maximum Flow Formulations and Bipartite Matching

## CPSC 535

Kevin A. Wortman



CALIFORNIA STATE UNIVERSITY  
**FULLERTON**



This work is licensed under a [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/).

## Big Idea: Problem Reduction

*problem A reduces to problem B* = can use an algorithm for  $B$  to do all the hard work of solving problem  $A$   
=  $A$  is easier than  $B$  (or tied)

Sometimes  $A, B$  are closely related  
e.g.  $A$  = sorting bounded integers,  $B$  = general sorting

More interesting: problems seem completely unrelated (e.g. SAT, CLIQUE; max-flow, bipartite matching)

## Reduction Algorithm Pseudocode

*problem A reduces to problem B* = can use an algorithm for *B* to do all the hard work of solving problem *A*

```
1: function SOLVE-A(input-for-A)
2:   input-for-B = pre-process input-for-A
3:   solution-for-B = solve-B(input-for-B)
4:   solution-for-A = post-process solution-for-B
5:   return solution-for-A
6: end function
```

In spirit

- ▶ the **solve-B** part is complex and the bottleneck
- ▶ the **overhead** (pre-process and post-process parts) is simple and fast

## Reducing to Max-Flow

### maximum flow problem

*input:* a flow network  $G$

*output:* a flow  $f$  of maximum value  $|f|$

```
1: function SOLVE-A(input-for-A)
2:    $G' =$  flow network based on input-for-A
3:    $f =$  SOLVE-MAX-FLOW( $G'$ )
4:   solution-for-A = post-process  $f$ 
5:   return solution-for-A
6: end function
```

(use  $G'$  because sometimes input-for-A is already a graph  $G$ )

The fastest max-flow alg. in CLRS takes  $O(|V|^3)$  time; overhead usually takes linear time; so SOLVE-MAX-FLOW is usually the bottleneck;  $O(n^3)$  time

## Formulation in General

To formulate  $A$  as a  $B$  instance, need to explain

- ▶ *Input mapping*: how to map input-for- $A$  to input-for- $B$
- ▶ *Output mapping*: how to map solution-for- $B$  to solution-for- $A$
- ▶ *Correctness*: how
  - ▶ all the “rules” of  $A$  are enforced...
  - ▶ .....by the  $B$  problem itself, and/or our input mapping
- ▶ “rules:” constraints on the output
- ▶ *Overhead*: analyze whether the input/output mapping becomes the bottleneck

**This is a valuable problem-solving skill!**

## Max-Flow Formulation

*Max-Flow Formulation:* details of how an algorithm for problem A

- ▶ maps an input into a flow network  $G'$
- ▶ recovers a solution from the flow  $f$

Also: analyze these steps to determine whether

- ▶ overhead is  $O(|V|^3) \implies$  SOLVE-MAX-FLOW is the bottleneck in SOLVE-A (usually yes)
- ▶ or, overhead is  $\Omega(|V|^3)$  and is the bottleneck

Usually we only discuss these parts, and don't write out the SOLVE-A pseudocode explicitly.

## Max-Flow Formulation

“Rules” of problem  $A$  must be implemented by the “rules” built into the max-flow problem and/or the edges and weights we introduce

- ▶ directed graph  $G = (V, E)$ , source  $s \in V$ , sink  $t \in V$
- ▶ none of: self-loop, antiparallel edge, unreachable vertex
- ▶ non-negative capacity on every edge
- ▶ flow is function  $f(u, v)$  over vertices  $u, v$
- ▶ **nonexistent edges**: if  $(u, v) \notin E$  then  $f(u, v) = 0$
- ▶ **capacity constraint**:  $0 \leq f(u, v) \leq c(u, v)$
- ▶ **flow conservation**: (flow-in) = (flow-out), except for source and sink; formally,  $\forall u \in V - \{s, t\}$ ,

$$\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$$

- ▶ *value*  $|f|$  = net flow into sink

## A Straightforward Formulation: Evacuation

Suppose we are working with safety authorities to determine how quickly CSUF could be evacuated in a natural disaster such as a wildfire.

*evacuation rate problem*

**input:** directed graph  $G$  representing a road map of Fullerton, each edge weighted with the number of autos/hour that may travel on that road

**output:** the maximum number of autos/hour that could travel from CSUF to a 57 or 91 freeway onramp

(Straightforward because this is clearly about flow in a directed graph.)



## A Straightforward Formulation: Evacuation

For a clear formulation, need to specify

- ▶ how to convert road map into flow network  $G'$ ; needs
  - ▶ to be a directed graph
  - ▶ source  $s$  and sink  $t$
  - ▶ non-negative capacity on each edge
  - ▶ no self-loops, antiparallel edges, or disconnected vertices
- ▶ how to decode flow  $f$  into a solution for our problem (# autos/hour evacuated)
- ▶ overhead time efficiency

## A Straightforward Formulation: Evacuation

- ▶ suppose for sake of discussion, road map  $G$  has none of the taboo components (self-loops etc.)
- ▶ start with  $G' = G$
- ▶ define source  $s$  in  $G'$  as the Gymnasium-Campus intersection on campus
- ▶ create new sink  $t$  in  $G'$  that represents “on either freeway;” create edges from highway onramps to  $t$ , each with capacity  $\infty$
- ▶ after finding max-flow in  $G'$ , examine flow function  $f$  to compute evacuation rate as

$$\sum_{\text{onramp vertex } o} f(o, t)$$

- ▶ overhead is  $O(|V| + |E|)$ , not bottleneck

## Robust Max-Flow

Goal: eliminate some of the pesky constraints of the classical max-flow problem

### **robust maximum flow problem**

*input:* a flow network  $G = (V, E)$ , which may contain unreachable vertices, antiparallel edges, a set  $S \subseteq V$  of sources, and a set  $T \subseteq V$  of sinks

*output:* a flow  $f$  of maximum value  $|f|$

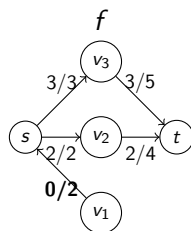
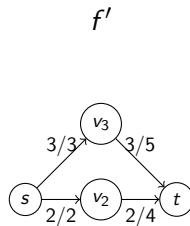
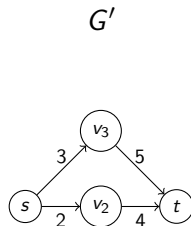
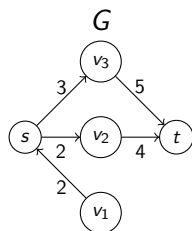
- ▶ unreachable vertices are allowed
- ▶ antiparallel edges are allowed
- ▶ **set** of sources/sinks instead of just one vertex each

## Reformulating to Eliminate Unreachable Vertices

given flow network  $G$  that may contain unreachable vertices,

- ▶ use BFS (or DFS) to mark every vertex that is reachable from  $s$
- ▶ use BFS again, following edges backwards, to mark every vertex that is reachable from  $t$
- ▶ if a vertex was not marked both times, it is redundant
- ▶  $G' =$  induced subgraph of  $G$  with all redundant vertices removed
- ▶ compute flow  $f'$  in  $G'$
- ▶ to convert  $f'$  to flow  $f$  in  $G$ , set flow along all redundant edges to 0
- ▶ overhead is  $2 \times \text{BFS} = O(|V| + |E|)$ , not bottleneck

# Reformulating to Eliminate Unreachable Vertices

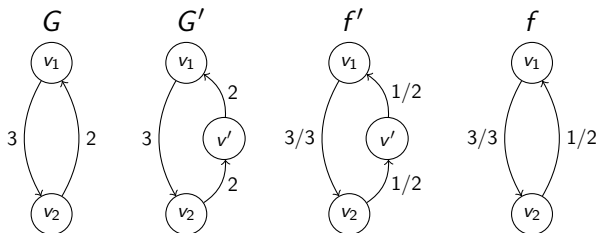


## Reformulating to Eliminate Antiparallel Edges

given a flow network  $G$  that may contain antiparallel edges,

- ▶ initially  $G' = G$
- ▶ identify all antiparallel edges
- ▶ when  $\exists$  antiparallel edges between vertices  $v_1, v_2$ ,
  - ▶ create new vertex  $v'$  in  $G'$  between  $v_1, v_2$
  - ▶ replace edge  $(v_1, v_2)$  with edges  $(v_1, v')$  and  $(v', v_2)$
  - ▶ set  $c(v_1, v') = c(v', v_2) = c(v_1, v_2)$
- ▶ observe that flow between  $v_1, v_2$  is identical but antiparallel edge is eliminated
- ▶ to convert flow  $f'$  in  $G'$  to equiv. flow in  $G$  : for each  $v'$  introduced above, set  $f(v_1, v_2) = f'(v_1, v')$
- ▶ overhead is  $O(|E|)$ ,  $|E'| < 2|E| \in \Theta(|E|)$ , not bottleneck

# Reformulating to Eliminate Antiparallel Edges

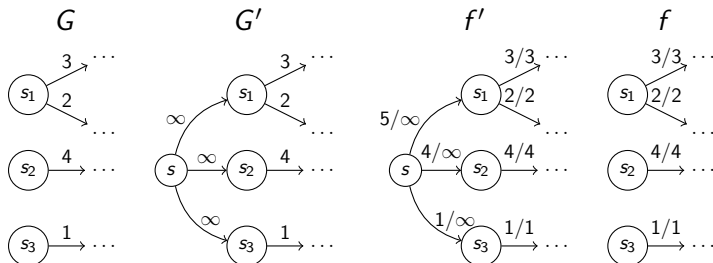


## Reformulating to Accommodate Multiple Sinks or Sources

- ▶ initially  $G' = G$
- ▶ create in  $G'$  a *super-source* vertex  $s$  and *super-sink*  $t$
- ▶ for each source  $s_i \in G$ , create an edge  $(s, s_i)$  in  $G'$  with capacity  $c(s, s_i) = \infty$
- ▶ for each sink  $t_i \in G$ , create an edge  $(t_i, t)$  in  $G'$  with capacity  $c(t_i, t) = \infty$
- ▶ to convert flow  $f'$  in  $G'$  to equiv. flow  $f$  in  $G$  : delete flow info. along any of the new edges
- ▶ overhead is  $O(|V|)$ ,  $|V'| = |V| + 2 \in \Theta(|V|)$ ,  $|E'| \leq |V| + |E|$ , not bottleneck



# Reformulating to Accommodate Multiple Sinks or Sources



## Formulations for Robust Max-Flow

From now on, we have the option of formulating problems as instances of the more robust max-flow problem:

### **robust maximum flow problem**

*input:* a flow network  $G = (V, E)$ , which may contain unreachable vertices, antiparallel edges, a set  $S \subseteq V$  of sources, and a set  $T \subseteq V$  of sinks

*output:* a flow  $f$  of maximum value  $|f|$

## Bipartite Matching

So far, all our reductions to max-flow have been either straightforward flow simulations, or variations on max-flow.

Now we'll see a quite-different problem that also reduces to max-flow.

## Partition of a Set

Intuitively: if  $X = L \cup R$  is a *partition*, then every element of  $X$  is placed in  $L$  or  $R$  (but not both).

Formally:  $L$  and  $R$  partition  $X$  if

- ▶  $X = L \cup R$ ,
- ▶  $L \cap R = \emptyset$ , and
- ▶  $L \neq \emptyset, R \neq \emptyset$ .

## Bipartite Matching

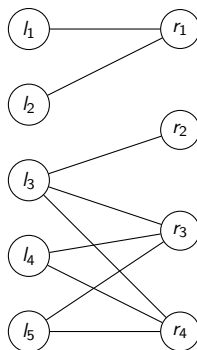
### **bipartite maximum matching**

*input:* an undirected bipartite graph  $G = (V, E)$  with parts  $V = L \cup R$

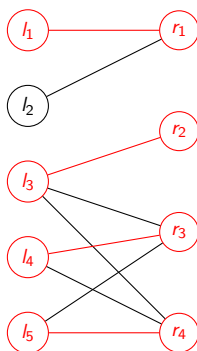
*output:* a matching  $M \subseteq E$  where the number of matched vertices is maximum

- ▶ *bipartite:*  $L, R$  are disjoint and edges only go between  $L, R$
- ▶ *matching:* pick edges that “pair off” two vertices; goal is to maximize #paired-off
- ▶ intuitively,  $L$  is one kind of thing and  $R$  is another kind of thing

# Bipartite Matching



## Bipartite Matching



matching

$$M = \{\text{included edges}\} = \{\{l_1, r_1\}, \{l_3, r_2\}, \{l_4, r_3\}, \{l_5, r_4\}\}$$
$$|M| = 4$$

(other optimal matchings exist)

## Bipartite Matching Applications

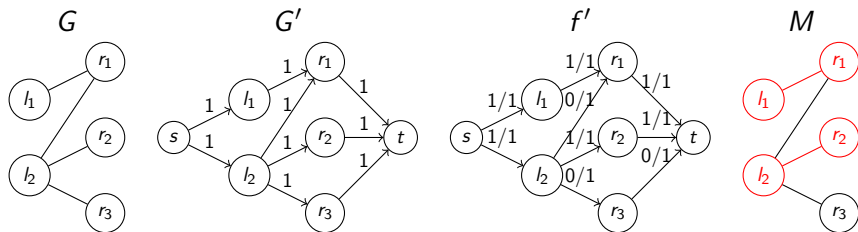
- ▶ any scenario where there are two kinds of things that can be paired
- ▶ goal is simply maximum number of pairings
- ▶ casting for a play:  $L$  = set of actors;  $R$  = set of roles; edge  $\{l, r\}$  exists when  $l$  could play role  $r$
- ▶ packing leftover food (one item/container):  $L$  = set of food items;  $R$  = available containers; edge  $\{l, r\}$  exists when food  $l$  could fit in container  $r$
- ▶ scheduling appointments:  $L$  = set of clients;  $R$  = set of time slots; edge  $\{l, r\}$  exists when client  $l$  could meet appointment  $r$
- ▶ might feel *NP*-hard, but actually in *P*



## Formulating Bipartite Matching as Flow

- ▶ let  $G = (V, E)$  be bipartite matching instance
- ▶ create  $G' = (V', E')$  with  $V' = V \cup \{s, t\}$  where  $s, t$  are new source/sink
- ▶ create edges in  $G'$ :
  - ▶  $(l, r) \forall l \in L, r \in R, \{l, r\} \in E$
  - ▶  $(s, l) \forall l \in L$
  - ▶  $(r, t) \forall r \in R$
- ▶ every edge  $(v, w)$  has capacity  $c(v, w) = 1$
- ▶ post-processing: edge  $(l, r) \in M$  iff  $f(l, r) = 1$
- ▶ observe  $|V'| \in O(|V|), |E'| \in O(|E|)$ , overhead is  $O(|V| + |E|)$
- ▶  $\implies$  if this is correct, can solve bipartite matching in  $O(|V|^3)$  time

# Formulating Bipartite Matching as Flow



$$M = \{\{l, r\} \mid f'(l, r) = 1\} = \{\{l_1, r_1\}, \{l_2, r_2\}\}$$

(other max flows  $\Leftrightarrow$  matchings exist)

## Correctness of this Formulation

Technical details:

- ▶ *integrality theorem*: if every capacity  $c(u, v) \in \mathbb{Z}$  then every  $f(u, v) \in \mathbb{Z}$  and  $|f| \in \mathbb{Z}$
- ▶  $\exists$  matching  $M$  with cardinality  $k = |M|$  iff  $\exists$  some flow  $f$  with value  $k = |f|$ 
  - ▶ key idea: pairing two vertices in the matching adds exactly one flow from  $s \rightsquigarrow t$
  - ▶ there are no opportunities for flow aside from matched vertices
- ▶  $\implies$  a maximum flow in  $G'$  corresponds to a maximum matching in  $G$

## Summary

- ▶ classical max-flow problem can be solved in  $O(|V|^3)$  time, in  $P$
- ▶ robust max-flow problem (supports unreachable vertices, antiparallel edges, multiple sinks/sources) also in  $O(|V|^3)$  time w/ worse constant factors, in  $P$
- ▶ bipartite matching reduces to max-flow, so bipartite matching can be solved in  $O(|V|^3)$  time, in  $P$
- ▶ other practical, distinct problems reduce to max-flow or bipartite matching so take  $O(|V|^3)$  time and are in  $P$