Algorithms and Complexity

Applications of max flow and min cut

Julián Mestre

School of Information Technologies
The University of Sydney

THE UNIVERSITY OF SYDNEY

Recap from last week

An instance of the maximum flow problem is defined by a directed graph $G=(V,E)$, a pair s-t, and capacities $c : E \rightarrow Z^+$

A flow $f : E \rightarrow Z^+$ is feasible if it obeys capacity and flow conservation constraints; its value is $v(f) = f^{out}(s)$

The Ford-Fulkerson algorithm find a feasible flow with maximum value in $O(C\ m)$ time, where $m=|E|$ and $C = c^{out}(s)$

The value of the maximum s-t flow f equal the capacity of the minimum s-t cut (A,B). Given f we can find (A,B) in $O(m)$ time

Edge disjoint paths problem

Motivation:

- a communication network is reliable, if there are several independent ways of routing traffic

Input:

- directed or undirected graph G=(V,E)
- a pair of vertices s-t

Task:

- find the maximum number of edge disjoint s-t paths

Auxiliary network flow

Given a directed graph G=(V,E) we create a new graph G'=(V,E) and capacities c(e)=1 for all e in E

Clearly the capacity of a minimum s-t cut is an upper bound for the maximum number of edge disjoint paths since each path must use at least one edge from the cut

Given a max flow f in G', we construct a set of v(f) disjoint s-t paths, which by the previous remark should be optimal

Finding disjoint paths

Def.: Call an s-t path p a flow path if $f(e) = 1$ for all e in p

We will show that if $v(f) > 0$ then $\exists$ a simple s-t flow path p

Zeroing out flow path decreases $v(f)$ by 1, preserving feasibility

```
def find_disjoint_paths(G,s,t):
  build (G',c) from G
  f = FF(G',s,t,c)
  while v(f) > 0:
    p = some s-t flow path w.r.t. G' and f
    output(p)
    for e in p:
      f(e) = 0
```

Time complexity

Finding a maximum flow takes $O(m\ n)$ time using FF

Extracting a single path takes $O(m)$

We can extract at most n paths

Thm. Given a directed graph, we can find a maximum number of disjoint s-t paths in $O(mn)$ time

Undirected graphs

Although we have shown this for directed graphs, this also holds for undirected graph:

- Use two anti-parallel edges for each undirected edge
- Cancel flow along the trivial cycles defined by a pair of anti-parallel edges
- Carry out the procedure we saw before

Thm. Given an undirected graph, we can find a maximum number of disjoint s-t paths in $O(mn)$ time

Airline scheduling

Airlines face daily the challenging problem of scheduling planes and crews to flights. Here we consider a simplified version.

Flight routes are specified by a tuple (po, td, pd, ta)

- po is the place of origin
- td is the time of departure
- pd is the place of destination
- ta is the time of arrival

A plane can serve two consecutive flights if

- destination and origin match and there is enough time for maintenance
- there is enough time to go from destination of first flight to origin of second

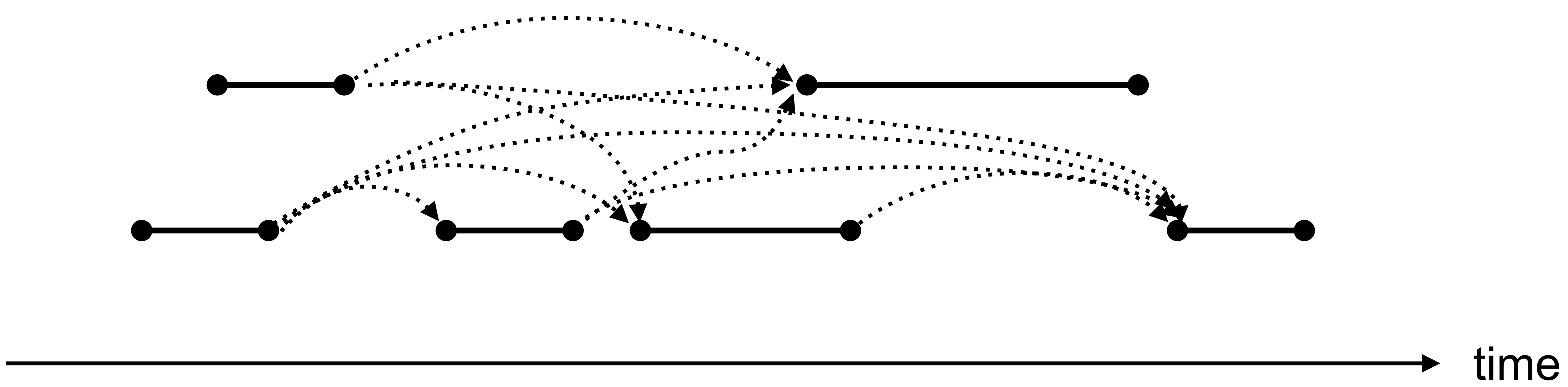
Airline scheduling

Input:

- set of tuples specifying flights (po, td, pd, ta)
- (directed) compatibility graph

Task:

- schedule all flights using the minimum number of planes



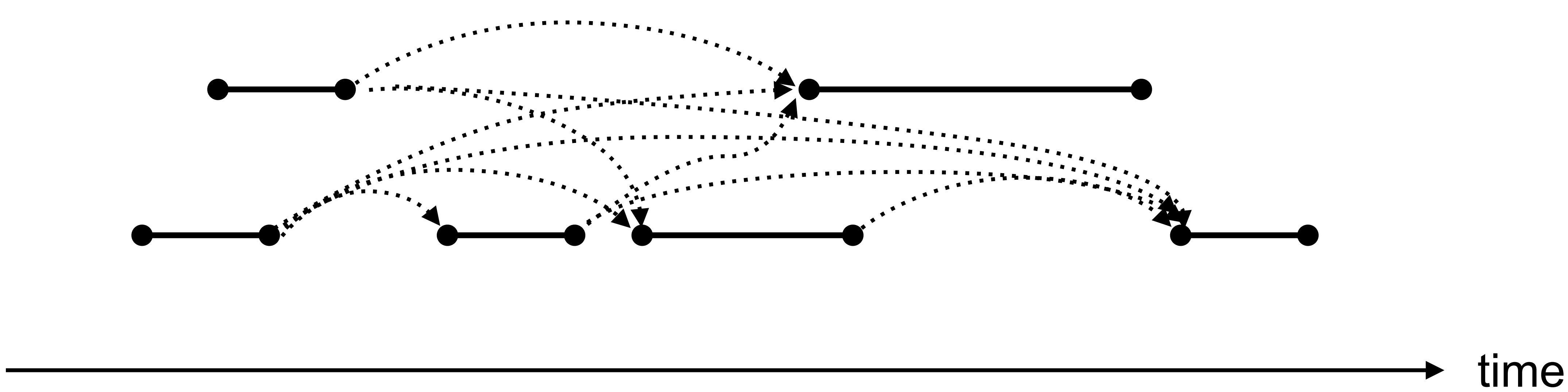
Airline scheduling

Input:

- set of tuples specifying flights (po, td, pd, ta)
- (directed) compatibility graph

Task:

- schedule all flights using the minimum number of planes



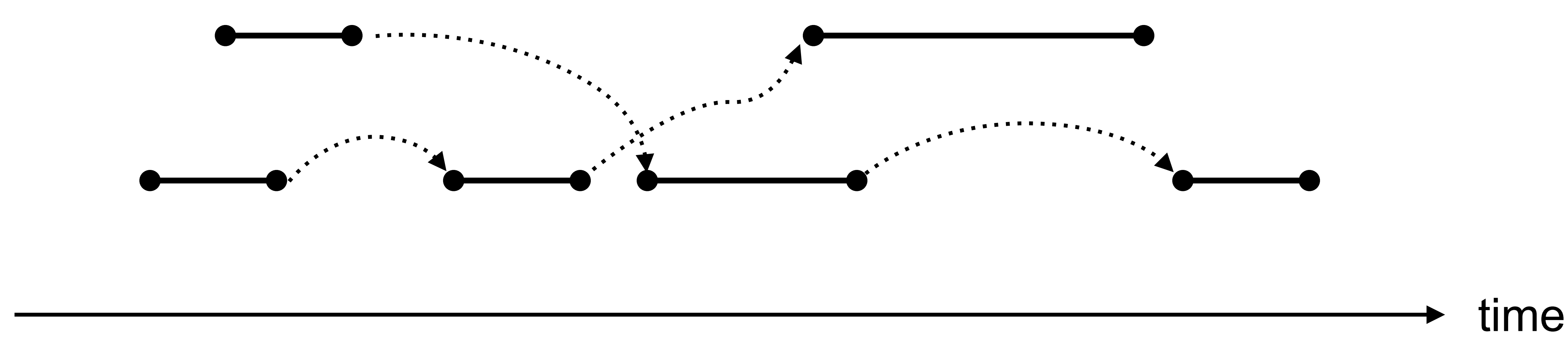
Airline scheduling

Input:

- set of tuples specifying flights (po, td, pd, ta)
- (directed) compatibility graph

Task:

- schedule all flights using the minimum number of planes



Auxiliary bipartite graph

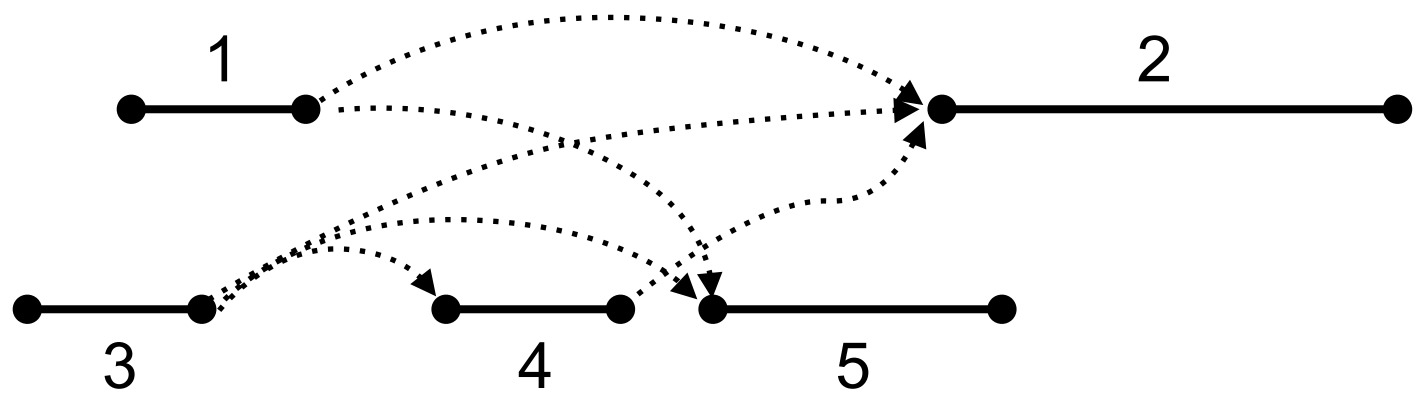
Create a bipartite graph G'=(A, B, E) where

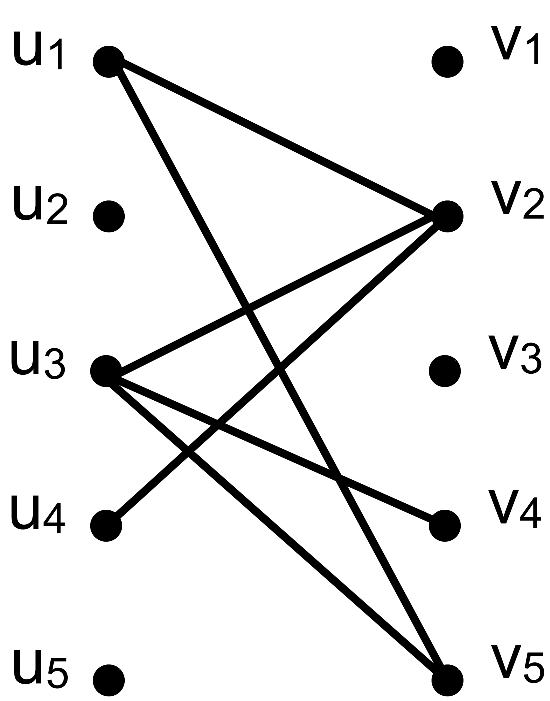
- each flight induces a vertex in A and a vertex in B
- connect left copy a flight x with right copy of a flight y if they are compatible; that is, the same plane can serve x followed by y

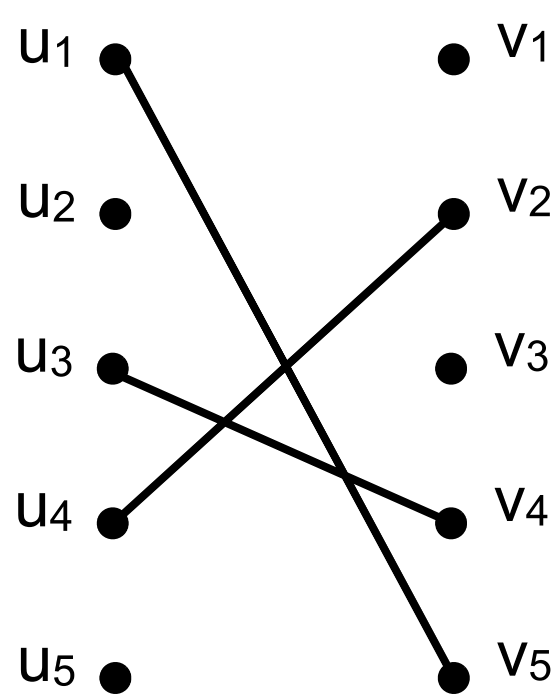
A matching M in this graph induces a schedule with n - |M| planes, where n is the number of flights

Thus, a maximum size matching leads to a plane schedule with the minimum number of planes

Example







Time complexity

Assuming there is a $O(1)$ time algorithm for testing compatibility of a pair of flights, building G' takes $O(n^2)$ time

Finding the maximum matching M takes $O(n^3)$ time using FF

Extracting the schedule from M takes $O(n)$ time

Thm. The airline scheduling problem can be solved in $O(n^3)$ time

Open-pit mining

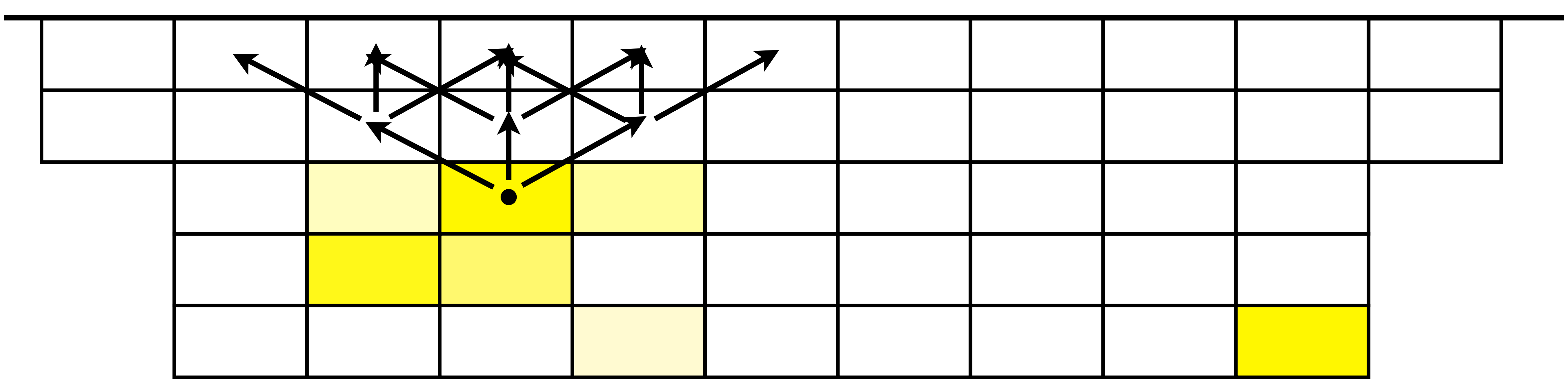
THE UNIVERSITY OF SYDNEY

Mining operation planning

Motivation:

- Open-pit mining is the standard for modern mining operations
- The underground is divided into uniform blocks
- Each block has an individual revenue and cost of extraction
- There are precedence constraints that govern the extraction of blocks

Which blocks should we extract to maximize profit?



Project selection problem

Input:

- directed graph $G=(V, E)$
- revenue function $\text{rev}: V \rightarrow Z$
- cost function $\text{cost}: V \rightarrow Z$

Task:

- find closure* S of V maximizing total profit $\text{rev}(S) - \text{cost}(S)$

* S is a closure in G if there are no edge from S into $V \setminus S$;

in other words, if $u \in S$ and $(u,v) \in E$ then $v \in S$ as well

Auxiliary flow network

Create a new graph $G'=(V', E')$ where

- $V' = V \cup \{ s, t \}$
- $E' = E \cup \{ (s, u) : u \text{ in } V \} \cup \{ (u,t) : u \text{ in } V \}$

The capacity function is defined as

- $c(u,v) = +\infty$ for (u,v) in E
- $c(s,u) = rev(u)$
- $c(u,t) = cost(u)$

We need to argue that a cut (A, B) is minimum in G' if and only if $A \setminus \{s\}$ is a maximum profit closure in G

Time complexity

Building the auxiliary graph takes O(m) time, assuming there are no isolated vertices in G. If there are those can be dealt with independently in O(n) time.

Finding the minimum cut takes O(m rev(V)) using FF

Thm. The project section problem can be solved in O(m rev(V)) time

Other applications

The maximum flow and minimum cut problem have many applications beyond the few we have seen here:

- Survey design [7.8]
- Image segmentation [7.10]
- Baseball elimination [7.12]
- Densest subgraph
- Graph orientation problems
- ⋮