CSE 421
Introduction to Algorithms

Winter 2024

Lecture 21

Mincut Applications: Task Selection

Today's topics

- Task Selection
- Network Flow Extensions
  - Reading: 7.5, 7.6, 7.10-7.12
- Starting Wednesday – NP Completeness
  - Reading: 8.1-8.10

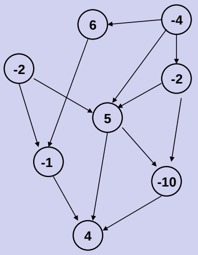
Application of Min-cut

- Task Selection Problem
- Reduction to Min Cut problem

S, T is a cut if S, T is a partition of the vertices with
s in S and t in T
The capacity of an S, T cut is the sum of the capacities of
all edges going from S to T

Task Selection

- Precedence graph G=(V,E)
- Each v in V has a profit p(v)
- A set F is *feasible* if when w in F, and (v,w) in E, then v in F.
- Find a feasible set to maximize the profit

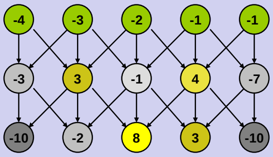


Open Pit Mining

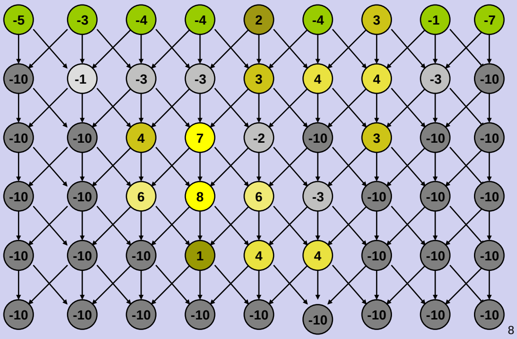
Open Pit Mining

- Each unit of earth has a profit (possibly negative)
- Getting to the ore below the surface requires removing the dirt above
- Test drilling gives reasonable estimates of costs
- Plan an optimal mining operation

Mine Graph



Determine an optimal mine

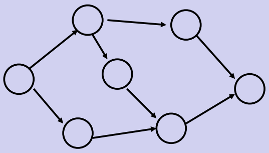


Min cut algorithm for profit maximization

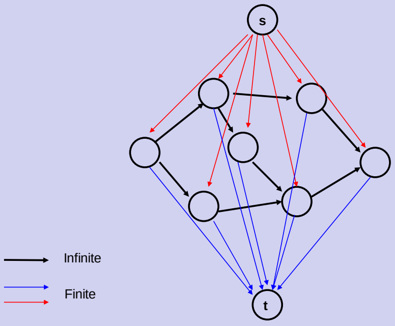
- Construct a flow graph where the minimum cut identifies a feasible set that maximizes profit

Precedence graph construction

- Precedence graph G=(V,E)
- Each edge in E has infinite capacity
- Add vertices s, t
- Each vertex in V is attached to s and t with finite capacity edges

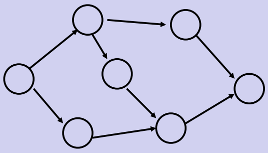


Find a finite value cut with at least two vertices on each side of the cut



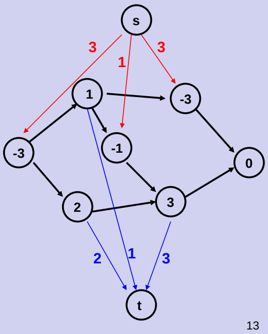
The sink side of a finite cut is a feasible set

- No edges permitted from S to T
- If a vertex is in T, all of its ancestors are in T

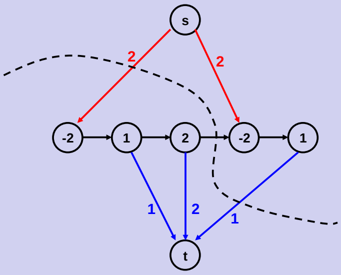


Setting the costs

- If p(v) > 0,
  - cap(v,t) = p(v)
  - cap(s,v) = 0
- If p(v) < 0
  - cap(s,v) = -p(v)
  - cap(v,t) = 0
- If p(v) = 0
  - cap(s,v) = 0
  - cap(v,t) = 0



Minimum cut gives optimal solution Why?

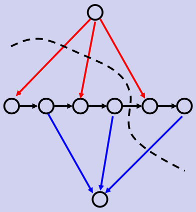


Computing the Profit

- Cost(W) = $\Sigma_{\{w \text{ in } W;\ p(w) < 0\}}$-p(w)
- Benefit(W) = $\Sigma_{\{w \text{ in } W;\ p(w) > 0\}}$ p(w)
- Profit(W) = Benefit(W) – Cost(W)

- Maximum cost and benefit
  - C = Cost(V)
  - B = Benefit(V)

Express Cap(S,T) in terms of B, C, Cost(T), Benefit(T), and Profit(T)



Cap(S,T) = Cost(T) + Ben(S) = Cost(T) + Ben(S) + Ben(T) – Ben(T)

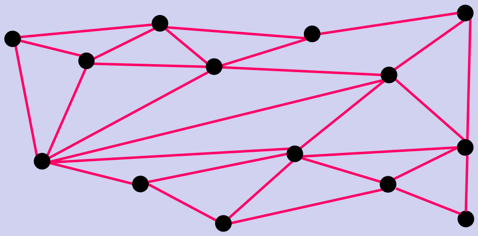
= B + Cost(T) – Ben(T) = B – Profit(T)

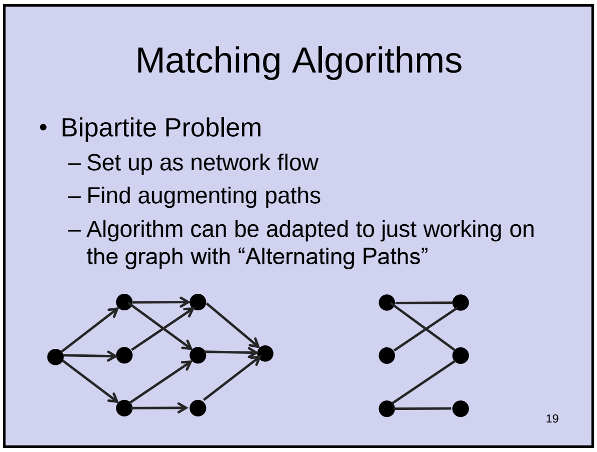
Beyond Network Flow

- Related Optimization Problems
  - Matching
  - Minimum Cost Flow
  - Multicommodity Flow
  - Linear Programming

Matching

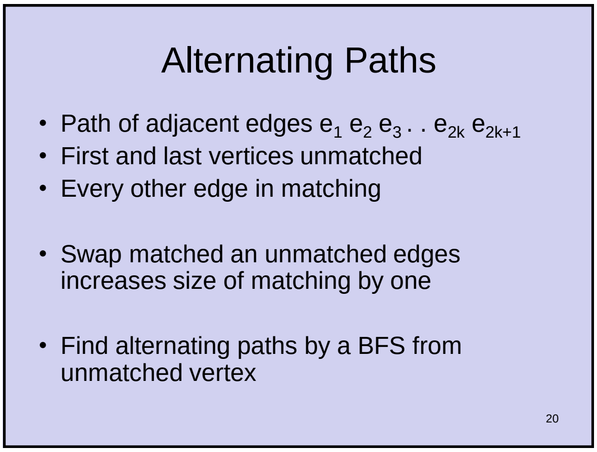
- Undirected Graph G=(V,E), find a maximum cardinality matching





Matching Algorithms

- Bipartite Problem
  - – Set up as network flow
  - – Find augmenting paths
  - – Algorithm can be adapted to just working on the graph with “Alternating Paths”

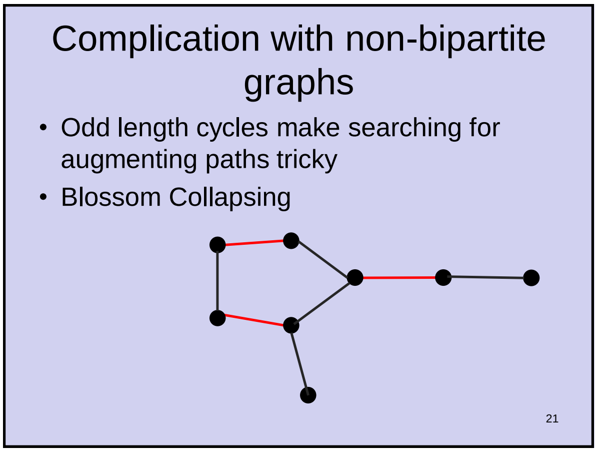


Alternating Paths

- Path of adjacent edges $e_1\ e_2\ e_3\ .\ .\ e_{2k}\ e_{2k+1}$
- First and last vertices unmatched
- Every other edge in matching

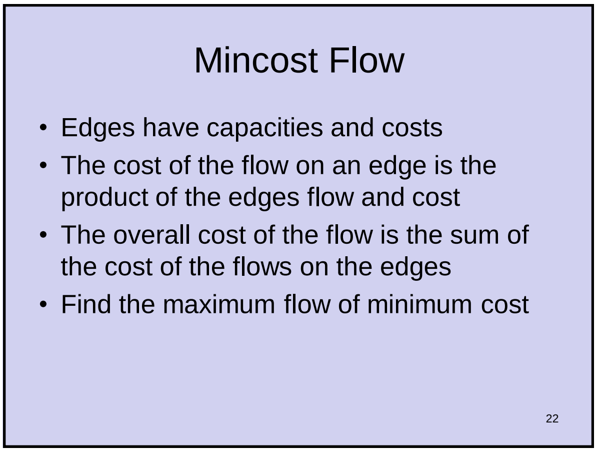
- Swap matched an unmatched edges increases size of matching by one

- Find alternating paths by a BFS from unmatched vertex



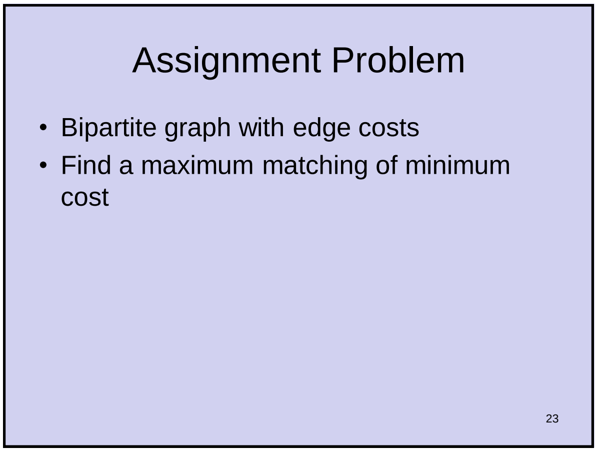
Complication with non-bipartite graphs

- Odd length cycles make searching for augmenting paths tricky
- Blossom Collapsing



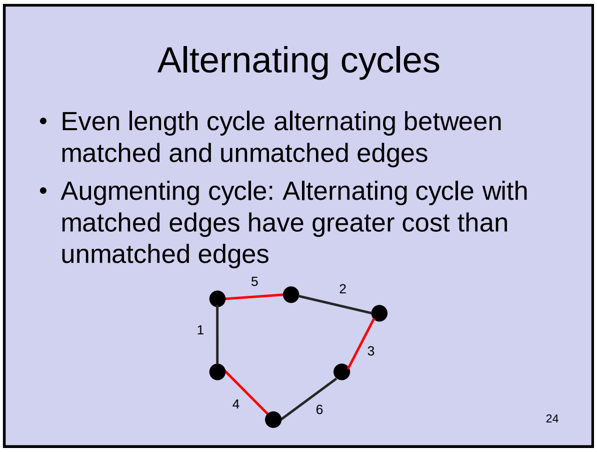
Mincost Flow

- Edges have capacities and costs
- The cost of the flow on an edge is the product of the edges flow and cost
- The overall cost of the flow is the sum of the cost of the flows on the edges
- Find the maximum flow of minimum cost



Assignment Problem

- Bipartite graph with edge costs
- Find a maximum matching of minimum cost



Alternating cycles

- Even length cycle alternating between matched and unmatched edges
- Augmenting cycle: Alternating cycle with matched edges have greater cost than unmatched edges

Mincost matching

```
Find maximum cardinality matching
While there exists an augmenting cycle C
    Swap matched edges along C
```

Multicommodity Flow

- Two types of flows, f_1 and f_2
- Sources and sinks, s_1, s_2, t_1, t_2
- f_1 routed from s_1 to t_1, f_2 from s_2 to t_2
- $0 \le f_1(e) + f_2(e) \le cap(e)$
- f_1 and f_2 conserved at vertices

Linear Programming

- Maximize a linear function subject to linear constraints