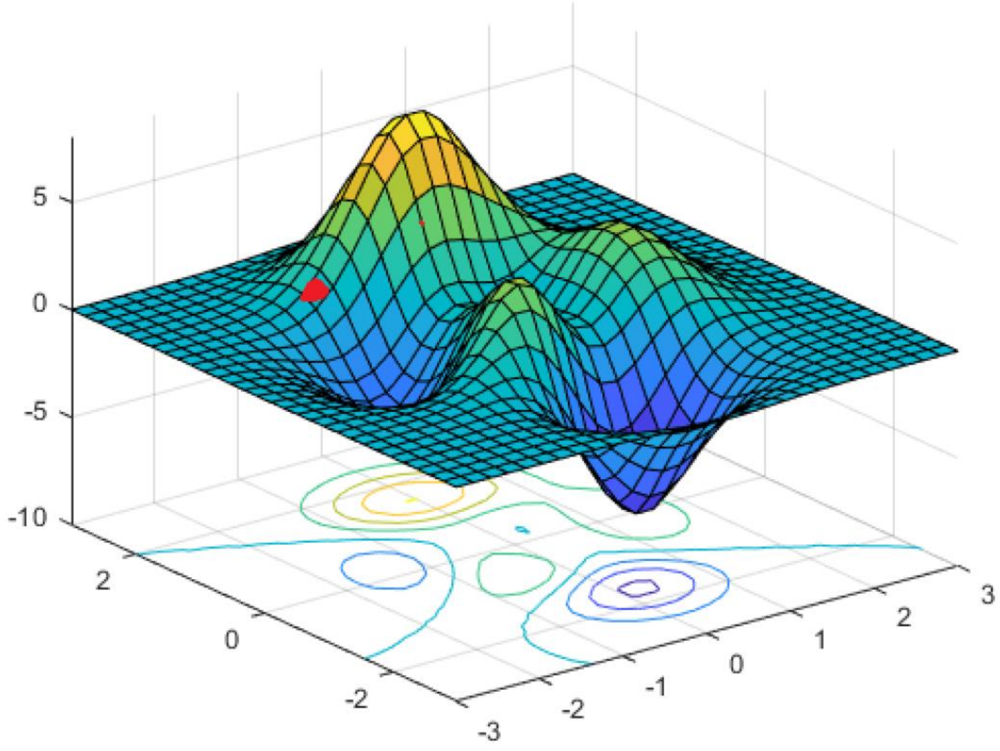
CSEP 521: Optimization

Richard Anderson, March 4, 2021



Announcements

- Remaining lectures on Optimization
  - Combinatorial Optimization
  - Linear Programming

- Readings
  - Skim textbook chapters on Matching/Network Flow (CLRS or KT)
  - Linear programming readings should be available Friday

- Last homework is due Thursday, March 11
  - Notify instructor if any homework is going to be turned in after March 14

Algorithmic Ideas

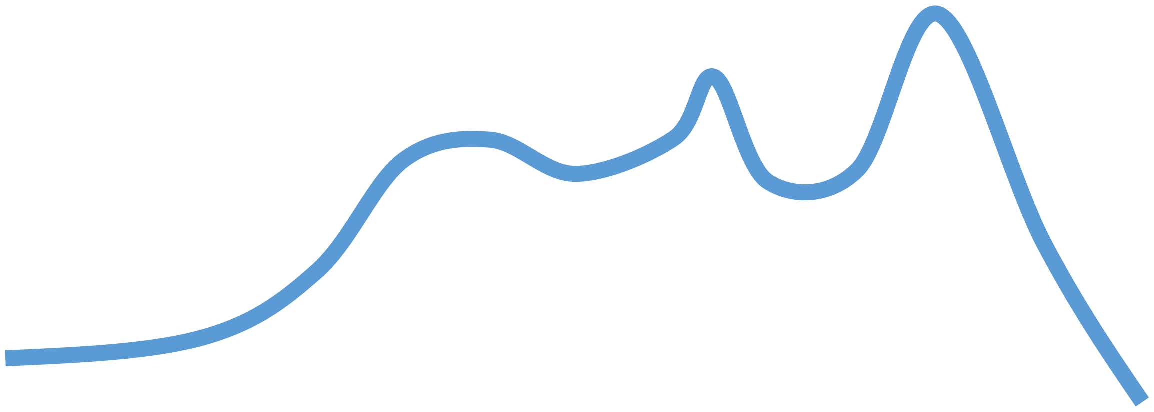
- Randomized algorithms and expected case analysis
- Hashing techniques
- Sketches and ultra low-space algorithms
- Geometry and high dimensional data

Optimization

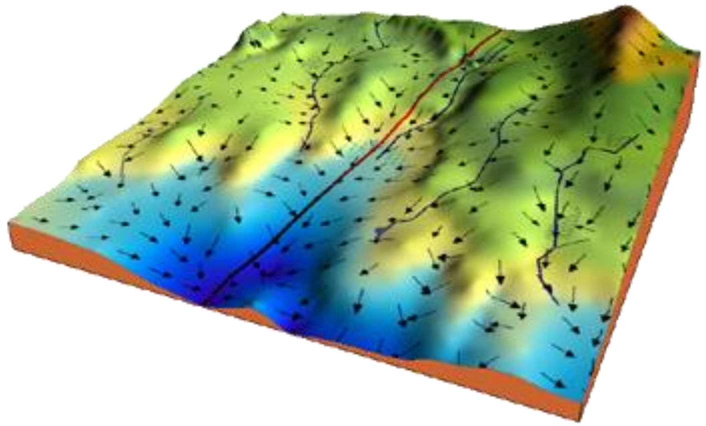
- Express a problem as a mathematical function, and then find a solution that minimizes or maximizes the objective function
  - Domain
  - Instance
  - Solution
  - Optimization Function

- Vaccine allocation – determine what order people get vaccines
  - Output: Partial order on the set of all people in Washington
  - Objective function: Minimize the DALYs (Disability-adjusted life years) due to Covid

One big, obvious idea



- Local improvement algorithms
  - Start with a valid solution
  - Keep modifying the solution as long as it improves
  - When done, hope you have a maximum



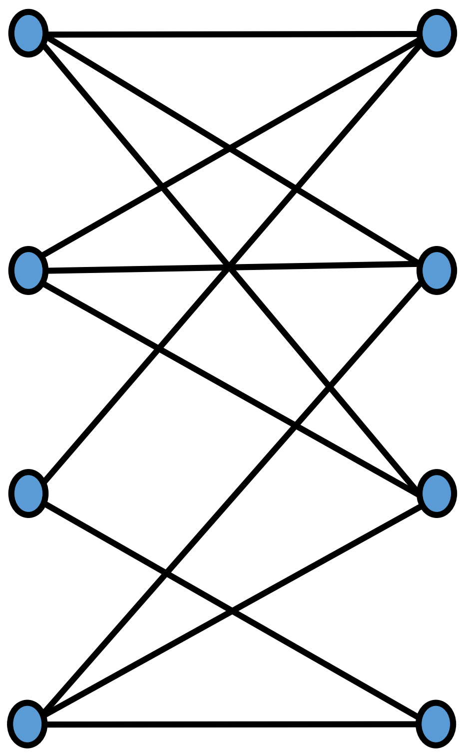
- Geometry of solution space
  - When is a local solution guaranteed to be a global solution?

Another big, less obvious idea

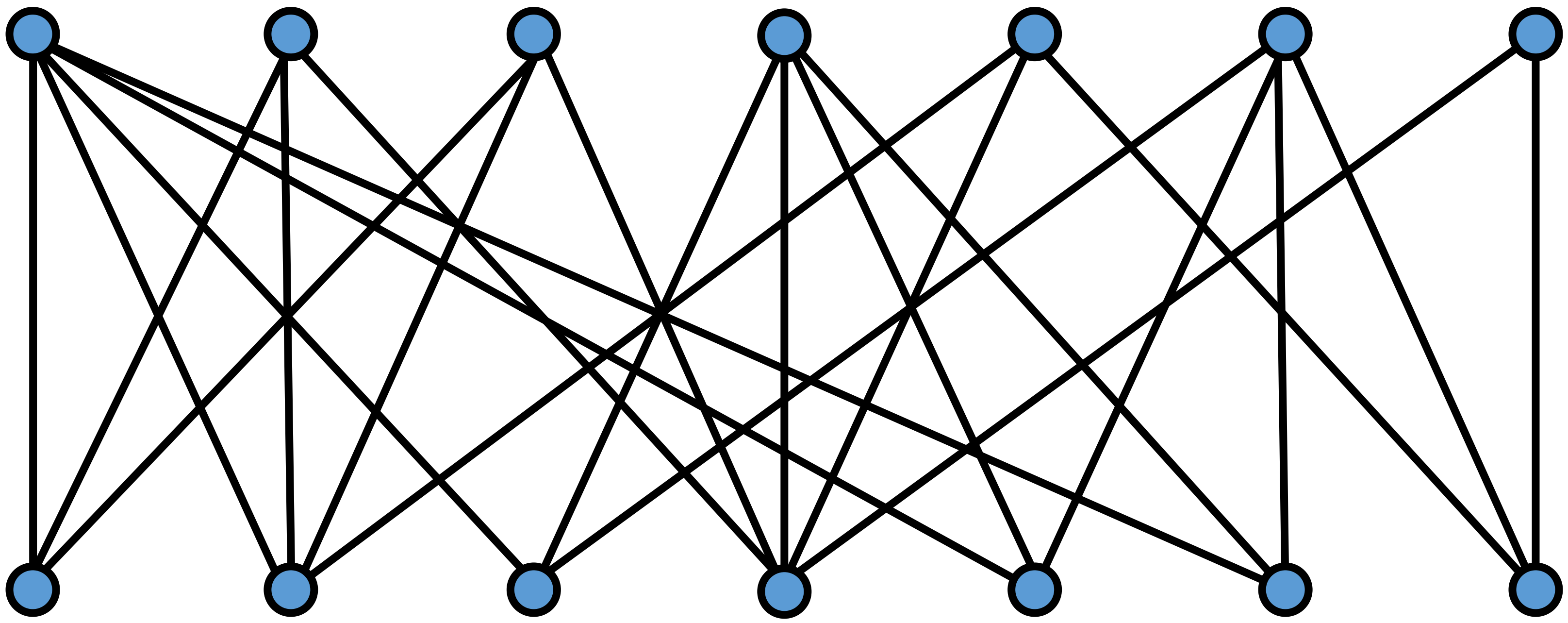
- Duality – pairing a maximization problem with a corresponding minimization problem
- Pairs of problems over a domain D
  - P_1: Solution S_1, Optimization F_1
  - P_2: Solution S_2, Optimization F_2
  - $I \in D$, $s \in S_1(I)$, $t \in S_2(I)$, $F_1(s) \leq F_2(t)$
  - $I \in D$, $\text{Max} \{ x \in S_1(I) \mid F_1(x) \} = \text{Min} \{ y \in S_2(I) \mid F_2(y) \}$

Bipartite Matching

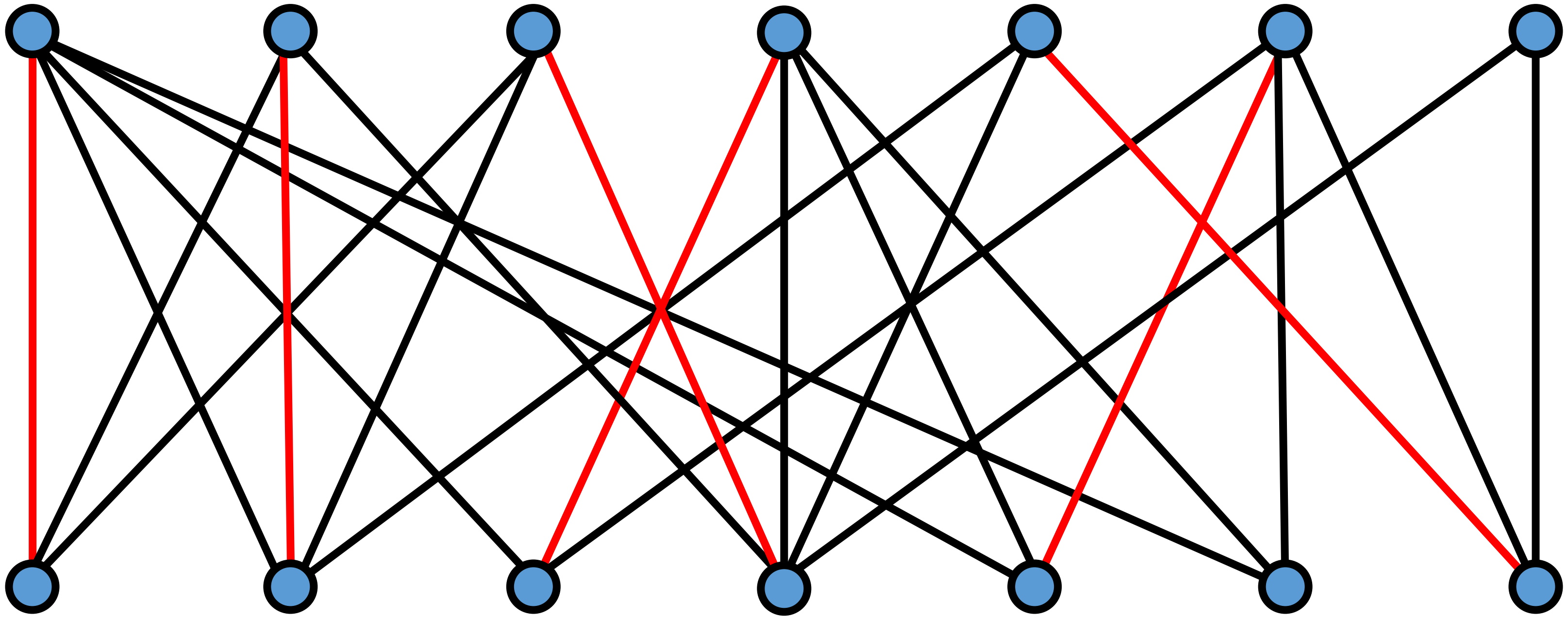
- Given a bipartite graph G=(U,V,E), find a subset of the edges M of maximum size with no common endpoints.
- Application:
  - U: Professors
  - V: Courses
  - (u,v) in E if Prof. u can teach course v



Find a maximum matching



Augmenting Path Algorithm



Bipartite Matching

```
M = ∅;
while ((P = AugmentingPath(G, M)) != null)
    M = M ⊕ P


AugmentingPath(G=(U,V,E), M){
    Orient edges in M from U to V
    Orient edges not in M from V to U
    Find a path P from an unmatched vertex in U to
        an unmatched vertex in V
    return P
```

$\oplus$ Is the symmetric difference (XOR) operator on sets

Path can be found with a standard path finding algorithm

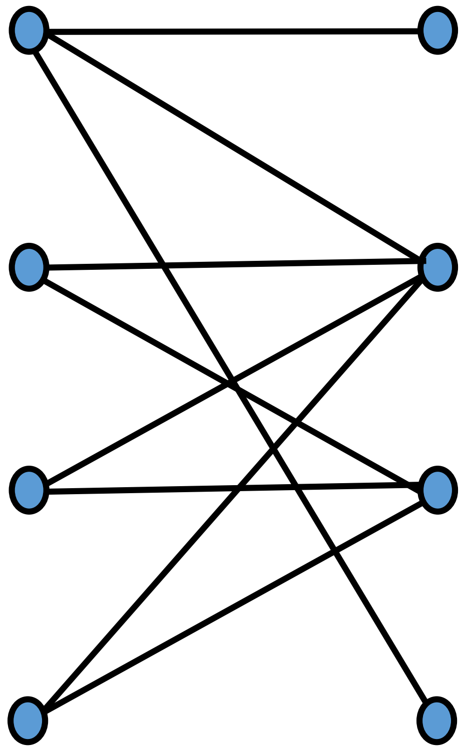
Each time a path is found the number of edges in the matching increases

Simple runtime is O(nm), can be improved to $O(n^{1/2}m)$

Vertex Cover (for bipartite graphs)

- A vertex cover for a graph G=(V,E) is a set of vertices C, such that every edge has at least on endpoint in C

- If C is a vertex cover and M is a matching $|C| \geq |M|$



Weighted Matching

- Let G=(U,V,E) be a bipartite graph with weights assigned to the edges
- For simplicity, we are going to assume
  - $|U|=|V|=n$
  - This is a complete graph
  - $w(u,v) \geq 0$ for $u \in U$ and $v \in V$
- We want to find a complete matching ($|M| = n$) of minimum weight

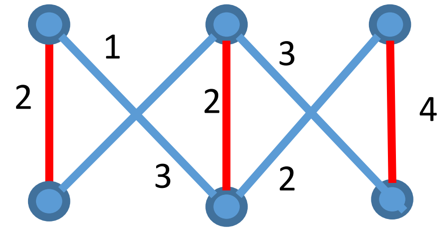
Weighted Matching Algorithm

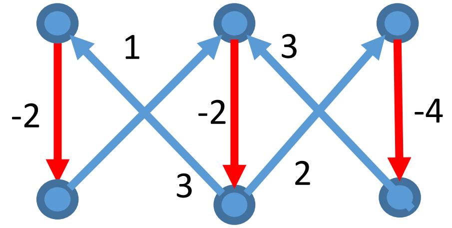
- Given a graph with a complete matching M, find a matching M’ with w(M’) < w(M)
- Iterate until you can no longer improve the solution
- Show that when this process stops you are at a global minimum
- To establish runtime, derive a bound on the number of iterations

Augmenting cycles for matching M

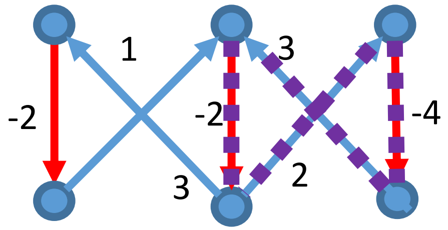
Construct a directed graph:

- If $(u,v) \in M$
  - $(u,v) \in E$ with $w'(u,v) = -w(u,v)$
- If $(u,v) \notin M$
  - $(v,u) \in E$ with $w'(v,u) = w(u,v)$

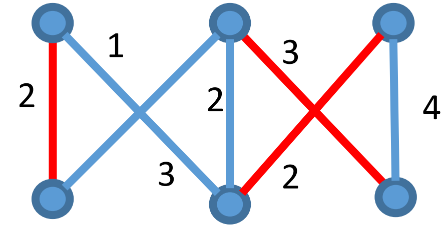




Find a negative cost cycle C with the Bellman-Ford Algorithm



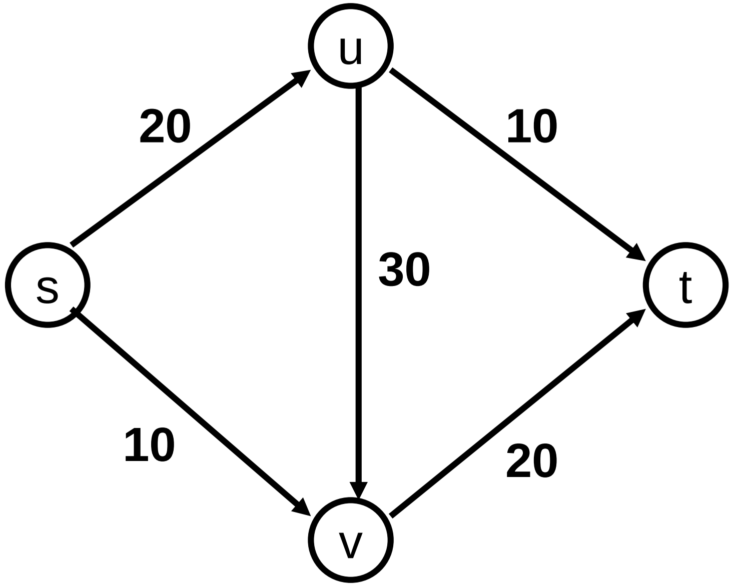
Update the matching to $M' = M \oplus C$



Weighted Matching

- Also called the Assignment Problem
- Standard Algorithm is the Hungarian Algorithm
  - Runtime is $O(n^3)$ or $O(nm + n^2 \log n)$
- Implementation as an $n \times n$ matrix
- Same ideas work for finding maximum weight matching

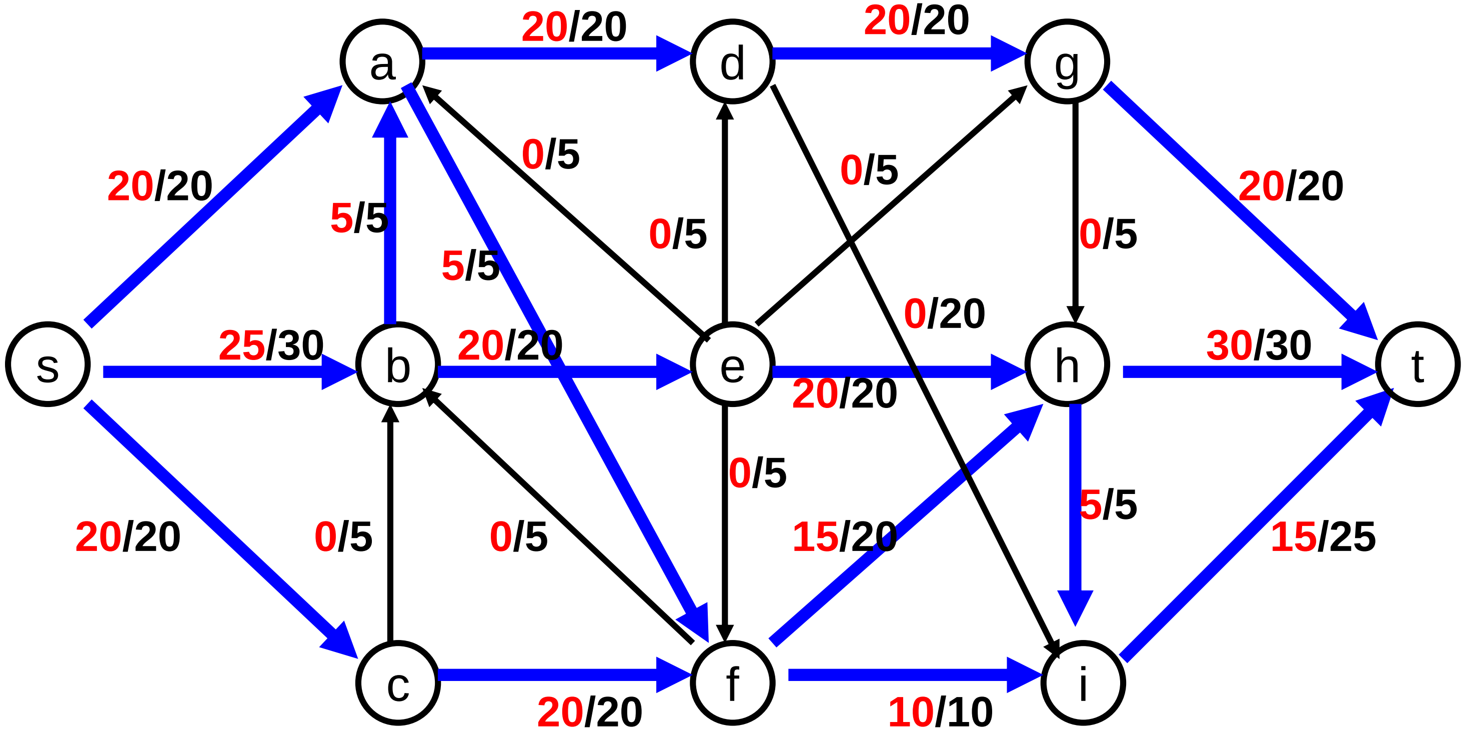
Network Flow



Network Flow Definitions

- Flowgraph: Directed graph with distinguished vertices s (source) and t (sink)
- Capacities on the edges, c(e) >= 0
- Problem, assign flows f(e) to the edges such that:
  - 0 <= f(e) <= c(e)
  - Flow is conserved at vertices other than s and t
    - Flow conservation: flow going into a vertex equals the flow going out
  - The flow leaving the source is a large as possible

Maximum flow example

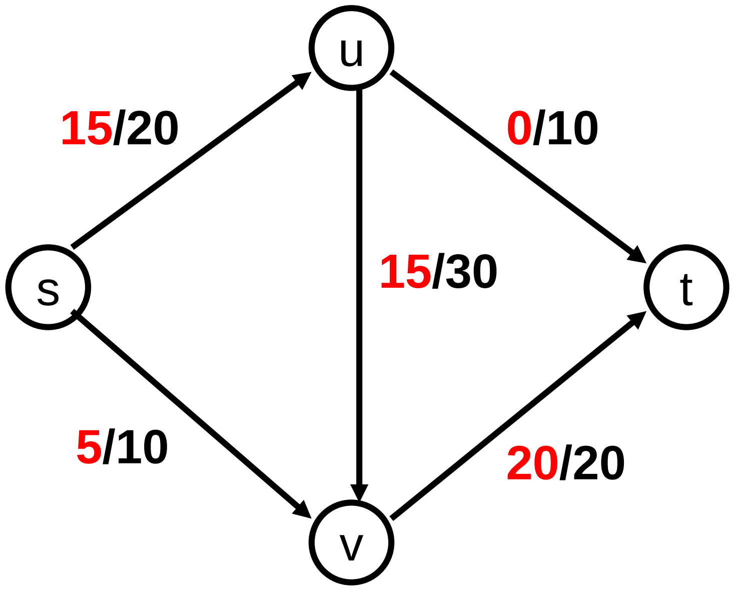


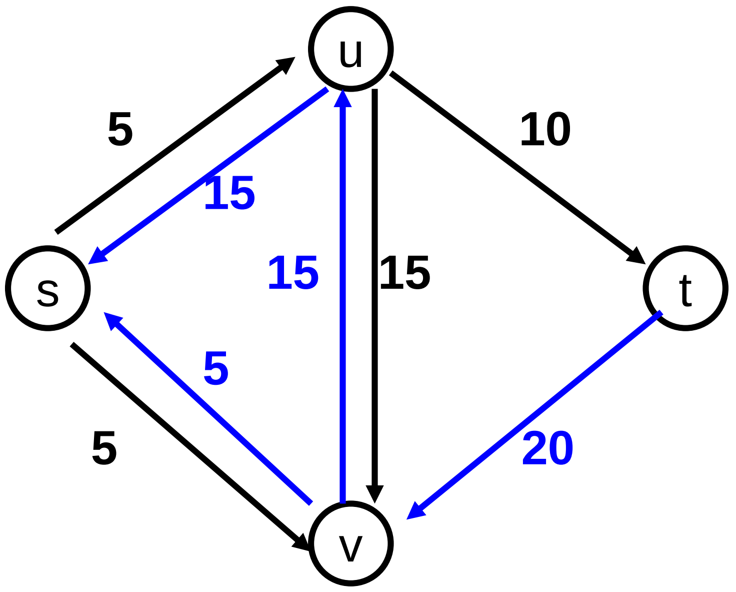
Residual Graph

- Flow graph showing the remaining capacity
- Flow graph G, Residual Graph G_R
  - G: edge e from u to v with capacity c and flow f
  - G_R: edge e' from u to v with capacity c – f
  - G_R: edge e'' from v to u with capacity f

- Find a path from s to t in G_R with minimum edge capacity (in G_R) of $b > 0$
- Add flow b to the path from s to t in G

Flow assignment and the residual graph





Ford-Fulkerson Algorithm (1956)

while not done

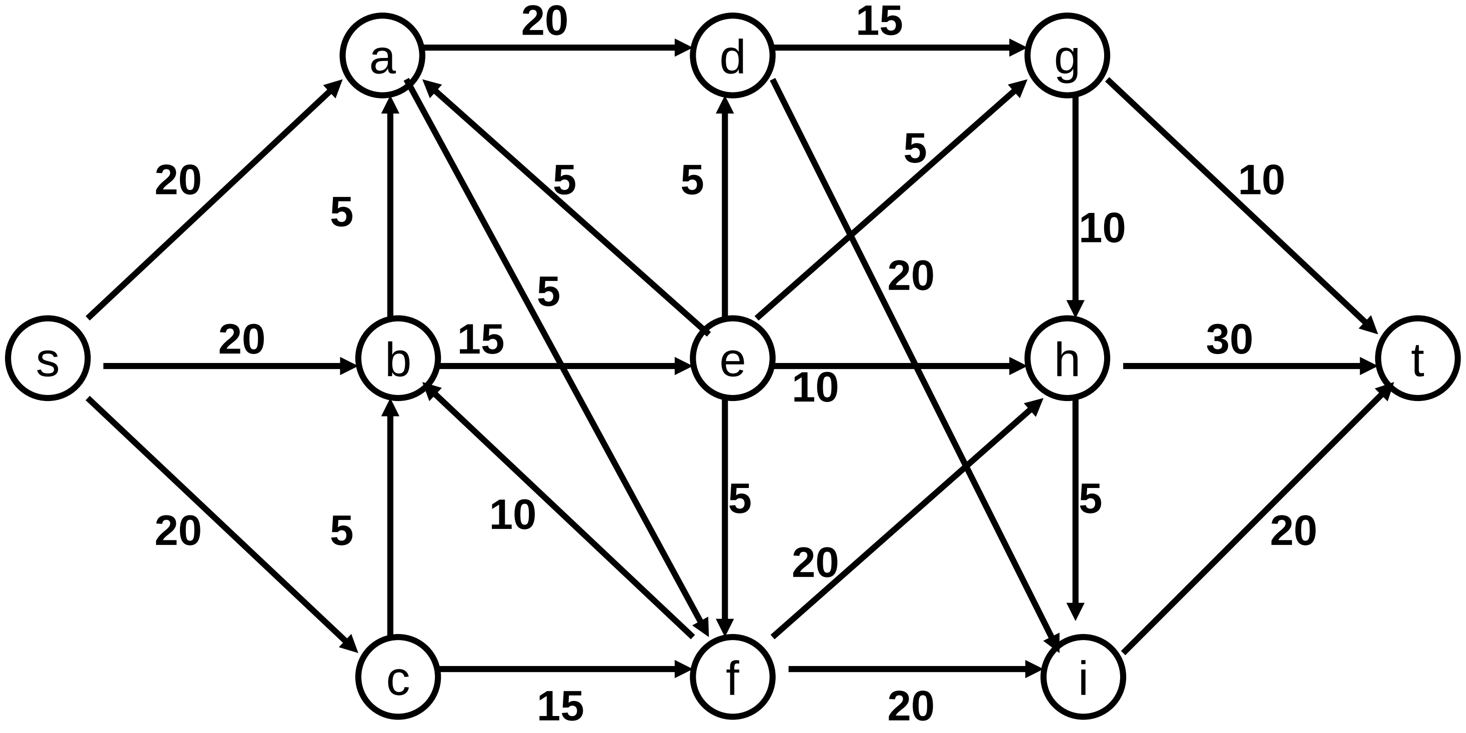
Construct residual graph G_R

Find an s-t path P in G_R with capacity $b > 0$

Add b units along in G

If the sum of the capacities of edges leaving S is at most C, then the algorithm takes at most C iterations

Flow Example



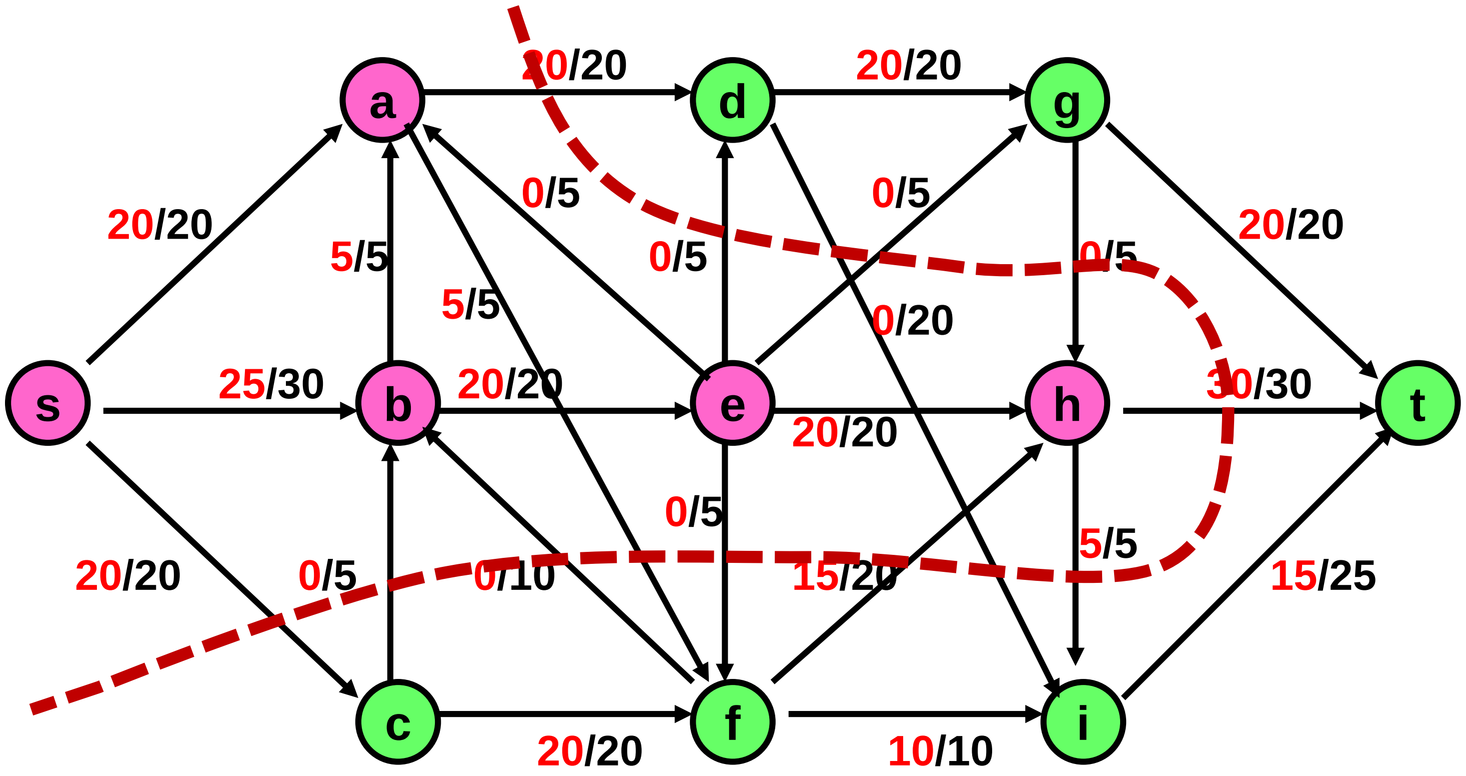
Cuts in a graph

- Cut: Partition of V into disjoint sets S, T with s in S and t in T.
- Cap(S,T): sum of the capacities of edges from   S to T
- Flow(S,T): net flow out of S
    - Sum of flows out of S minus sum of flows into S

- Flow(S,T) <= Cap(S,T)

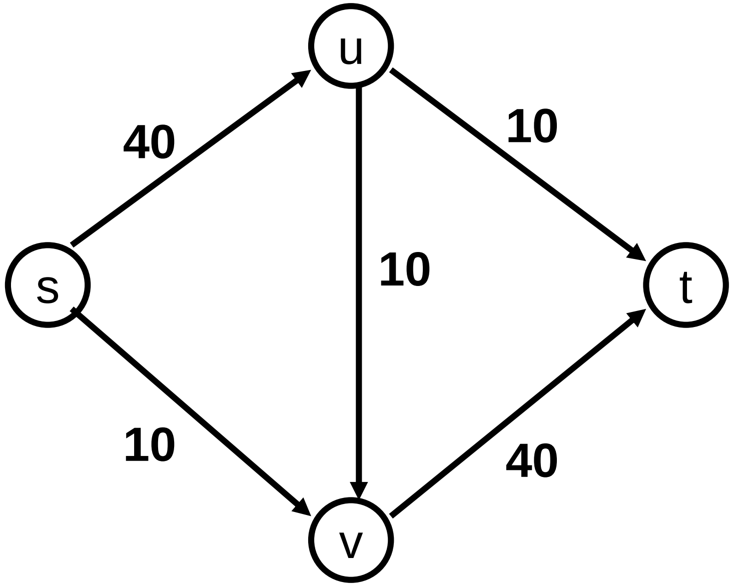
Cap(S,T) and Flow(S,T)

S={s, a, b, e, h},   T = {c, f, i, d, g, t}



Cap(S,T) = 95,   Flow(S,T) = 80 – 15 = 65

Minimum value cut

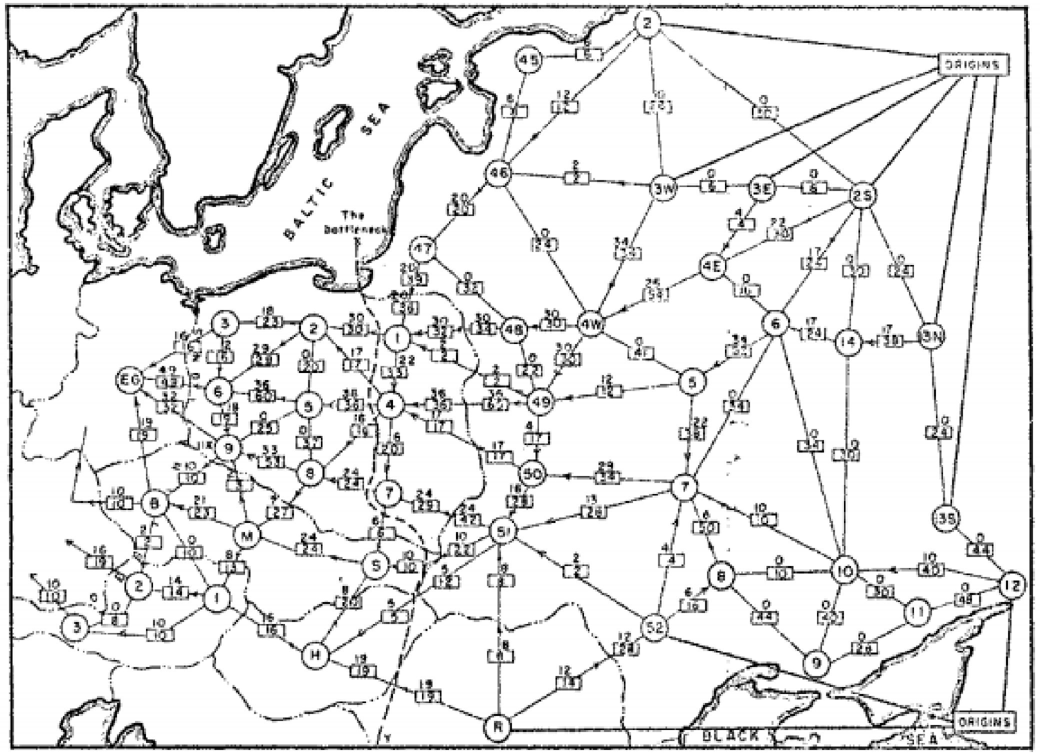


MaxFlow – MinCut Theorem

- There exists a flow which has the same value as the minimum cut
- This shows that both the flow is maximum and the cut is minimum since the flow is always less than or equal to the cut
- The proof is to run the Ford-Fulkerson algorithm until it completes and look at the residual graph
- This is a ``duality theorem'' as the MaxFlow and MinCut problems are duals

History

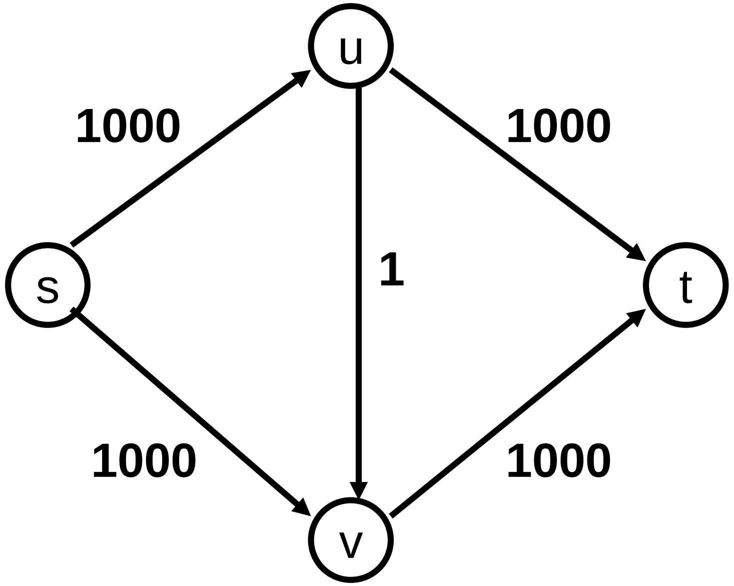
- Ford / Fulkerson studied network flow in the context of the Soviet Rail Network



Ford Fulkerson Runtime

- Cost per phase times number of phases

- Phases
  - Capacity leaving source: C
  - Add at least one unit per phase
- Cost per phase
  - Build residual graph: O(m)
  - Find s-t path in residual: O(m)



Better methods of finding augmenting paths

- Find the maximum capacity augmenting path
  - $O(m^2 \log(C))$ time algorithm for network flow
- Find the shortest augmenting path
  - $O(m^2 n)$ time algorithm for network flow
- Find a blocking flow in the residual graph
  - $O(mn \log n)$ time algorithm for network flow