Class 28

MAXIMUM FLOW PROBLEM

Maximum Flow Problem

Maximizing the flow through a network is an important problem:

- Traffic flow
- Network flow
- Flow of electricity

Let us assume that we can represent such a problem by a connected weighted digraph with *n* vertices.

Maximum Flow Problem

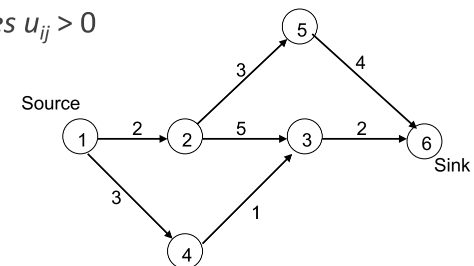
Maximum flow problem: maximize flow from source to sink while staying under edge capacities

A single *source* vertex v_1 with no entering edges.

A single *sink* vertex v_n with no leaving edges.

Edges (i,j) have *capacities* $u_{ij} > 0$

Example:



Flow-Conservation Requirement

Flow-conservation requirement: For all intermediate vertices:
total inflow = total outflow

$$\sum_{j:(j,i)\in E} x_{ji} = \sum_{j:(i,j)\in E} x_{ij} \quad \text{for } i = 2, 3, \ldots, n-1,$$

Nothing gets added, nothing gets removed.

This implies that the total going into the network at the source must end up at the sink:

$$\sum_{j:(1,j)\in E} x_{1j} = \sum_{j:(j,n)\in E} x_{jn}.$$

Formal Problem Statement

Let the *capacity constraints* be such that:

$$0 \le x_{ij} \le u_{ij} \quad \text{for every edge } (i,\ j) \in E.$$

A *flow* is an assignment of flow values x_{ij} to edges (*i*,*j*) of a given network that satisfy the flow-conservation requirements and the capacity constraints.

Maximum Flow Problem

$$\text{maximize} \sum_{j:(1,j)\in E} x_{1j}$$

subject to

$$\sum_{j:(j,i)\in E} x_{ji} - \sum_{j:(i,j)\in E} x_{ij} = 0 \quad \text{for } i = 2, \ldots, n-1$$

$$x_{ij} \le u_{ij} \quad \text{for every edge } (i,j) \in E$$

$$x_{ij} \ge 0 \quad \text{for every edge } (i,j) \in E$$

Can be solved using simplex method or other linear programming solver.

However, the special structure allows a more efficient problem-specific algorithm.

Augmenting Path Method

Also called the Ford-Fulkerson method.

Start with the zero flow (x_{ij} = 0 for every edge)

On each iteration, try to find a *flow-augmenting path* from source to sink, i.e. path that can handle some additional flow

- If one is found, adjust the flow along the edges of this path to get a flow of increased value and try again
- If none found, the current flow is maximum. We'll prove correctness of this later…

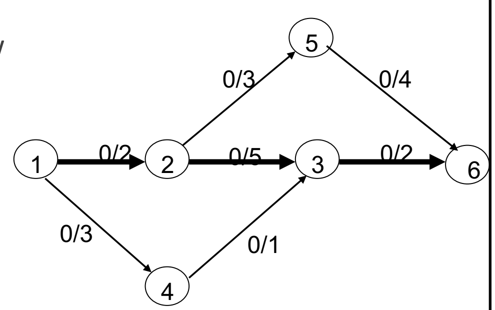
Method, not algorithm, since how to find flow-augmenting path is not specified.

Ford-Fulkerson Example

Consider the earlier example.

The edges are annotated with assigned_flow/capacity.

At first, we have zero flow. Suppose we identify the flow augmenting path: 1→2 →3 →6
Since the smallest capacity along that path is 2, we update edges (1,2), (2,3) and (3, 6) to 2. (next slide)



Ford-Fulkerson Example

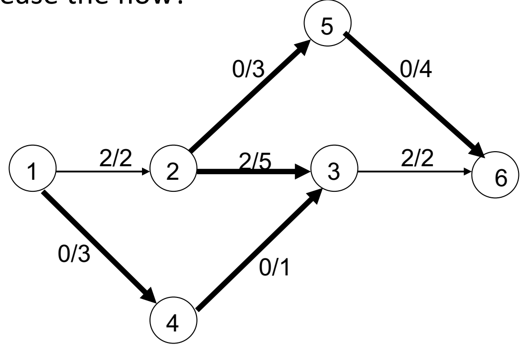
Since the smallest capacity along that path is 2, we update edges (1,2), (2,3) and (3, 6) to 2.

How can we further increase the flow?

Consider adding 1 on path 1→4 →3.

Problem: Flow from 3→6 has capacity of 2

Solution: Add 1 unit of flow to 2→5 →6 syphoning it from 2→3

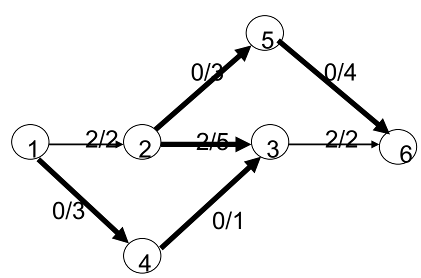


Some Notation

Forward edge. (i,j) with positive unused capacity $r_{ij} = u_{ij} - x_{ij}$

Backward edge. (i,j) where there is positive flow x_{ji}

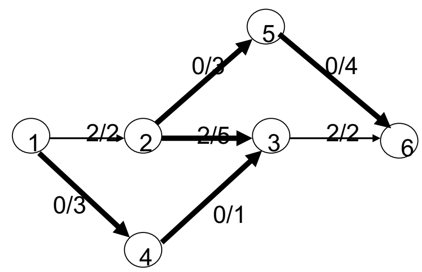
Example: 1→ 4 → 3 ← 2 → 5 → 6



Goals

If a flow-augmenting path is found, the current flow can be increased by r units by increasing x_{ij} by r on each forward edge and decreasing x_{ji} by r on each backward edge, where

r = min {r_{ij} on forward edges, x_{ji} on backward edges}



Finding a Flow-Augmenting Path

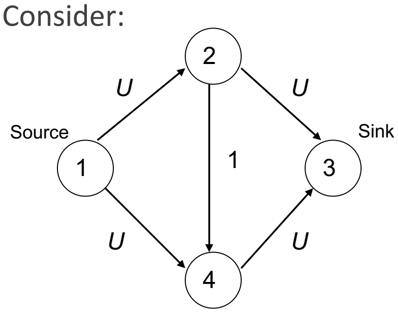
Will we make finitely many augmentations?

- Assuming the edge capacities are integers, r is a positive integer
- On each iteration, the flow value increases by at least 1
- Maximum value is bounded by the sum of the capacities of the edges leaving the source; hence the augmenting-path method has to stop after a finite number of iterations

Possible Performance Degradation

Selecting a bad sequence of augmenting paths could impact the method's efficiency.

Consider:

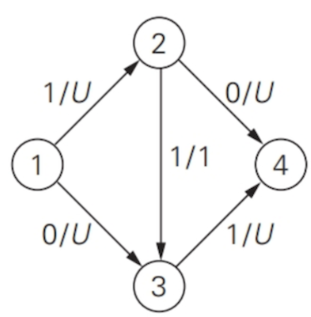


U = large positive integer

Possible Performance Degradation

Suppose we select 1 → 2 → 3 → 4.

Maximum flow is: 1

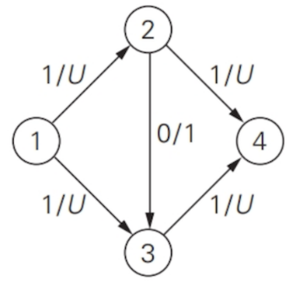


Possible Performance Degradation

Now, suppose we select 1 → 3 → 2 → 4.

Notice that we in essence undo the 2 → 3 flow.

Maximum flow is: 2

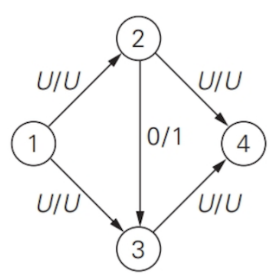


Possible Performance Degradation

Notice that the maximum flow through this network is 2U.

Using the method of increasing the flow by 1 each time, then arriving at this step would take approximately 2U + 1 steps.

If we had selected 1 → 2 → 3 and then 1 → 3 → 4, we would have found the max outright, i.e. in 2 steps.

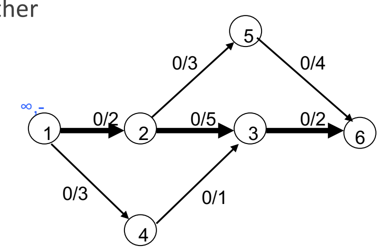


Edmonds-Karp Algorithm

Generate augmenting path with the least number of edges:
BFS from the source, marking unlabeled vertices with labels amount, previous:

- amount of additional flow that can be sent from the source to this vertex
- previous vertex in the path that allows the additional flow, with "+" or "–" added to indicate whether via a forward or backward edge

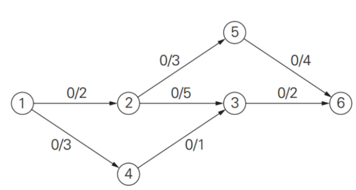
Initialization: label source with $\infty, -$ and enqueue it.



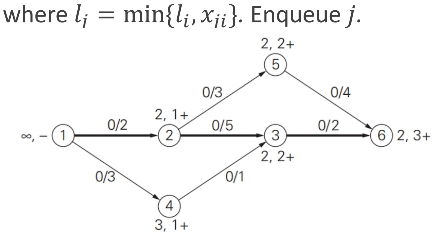
Edmonds-Karp Algorithm

Loop:

- Dequeue vertex i.
- For all unlabeled successors j of i with positive unused capacity $r_{ij} = u_{ij} - x_{ij}$,
  Label j with l_j, i^+ where $l_j = \min\{l_i, r_{ij}\}$. Enqueue j.
- For all unlabeled predecessors j of i with positive flow x_{ji},
  Label j with l_j, i^- where $l_j = \min\{l_i, x_{ji}\}$. Enqueue j.



Queue: 1 2 4 3 5 6
↑ ↑ ↑ ↑

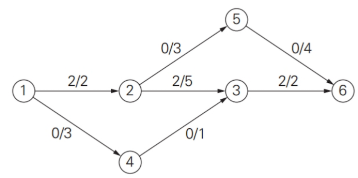


Augment the flow by 2 (the sink's first label) along the path $1 \to 2 \to 3 \to 6$.

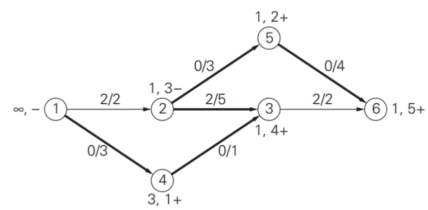
Edmonds-Karp Algorithm

Loop:

- Dequeue vertex i.
- For all unlabeled successors of i with positive unused capacity $r_{ij} = u_{ij} - x_{ij}$,
  Label j with l_j, i^+ where $l_j = \min\{l_i, r_{ij}\}$. Enqueue j.
- For all unlabeled predecessors of i with positive flow x_{ji},
  Label j with l_j, i^- where $l_j = \min\{l_i, x_{ji}\}$. Enqueue j.



Queue: 1 4 3 2 5 6
↑ ↑ ↑ ↑ ↑

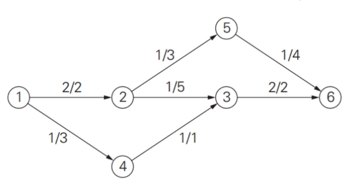


Augment the flow by 1 (the sink's first label) along the path 1 → 4 → 3 ← 2 → 5 → 6.

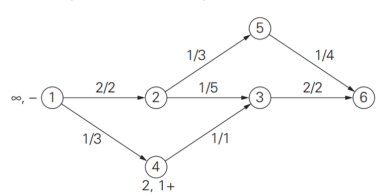
Edmonds-Karp Algorithm

Loop:

- Dequeue vertex i.
- For all unlabeled successors of i with positive unused capacity $r_{ij} = u_{ij} - x_{ij}$,
  Label j with l_j, i^+ where $l_j = \min\{l_i, r_{ij}\}$. Enqueue j.
- For all unlabeled predecessors of i with positive flow x_{ji},
  Label j with l_j, i^- where $l_j = \min\{l_i, x_{ji}\}$. Enqueue j.



Queue: 1 4
↑ ↑



No augmenting path (the sink is unlabeled); the current flow is maximal.

Edmonds-Karp Efficiency

Let n = #vertices, m = #edges

Claim: # of augmenting paths is $O(nm)$.

- Whenever one of the m edges becomes saturated (brought up to capacity), the distance from the saturated edge to the source along the augmenting path must be longer than last time; also, this length is at most n.

For adjacency lists:

- time to find shortest augmenting path by BFS is in $O(n+m)=O(m)$
- Overall time efficiency: $O(nm^2)$

More efficient algorithms have been found that can run in close to $O(nm)$ time, but these algorithms aren't iterative-improvement