CPSC 490 – Problem Solving in Computer Science

Lecture 26: Min Cost Flow

Jason Chiu and Raunak Kumar
Based on TopCoder Tutorial

2017/03/27

University of British Columbia

Problem 1 - Easy Warm-Up

Given a directed graph, find a set of vertex disjoint cycles that cover all the vertices, or say that it doesn't exist.

Cycles with 2 vertices are fine.

Problem 1 - Solution

Use Bipartite Matching!

- For every vertex in the original graph, add a vertex on the left and right.
- For edge $i \to j$ in the original graph, add the edge in the bipartite graph.
- Find a maximum matching in this graph.
- If size of the matching is n, found a cycle cover.
- Matching i with j: vertex j comes after vertex i in a cycle.

Min Cost Max Flow Problem

Given:

- Directed graph $G = (V, E)$
- Capacity u_{ij} for each edge

Min Cost Max Flow Problem

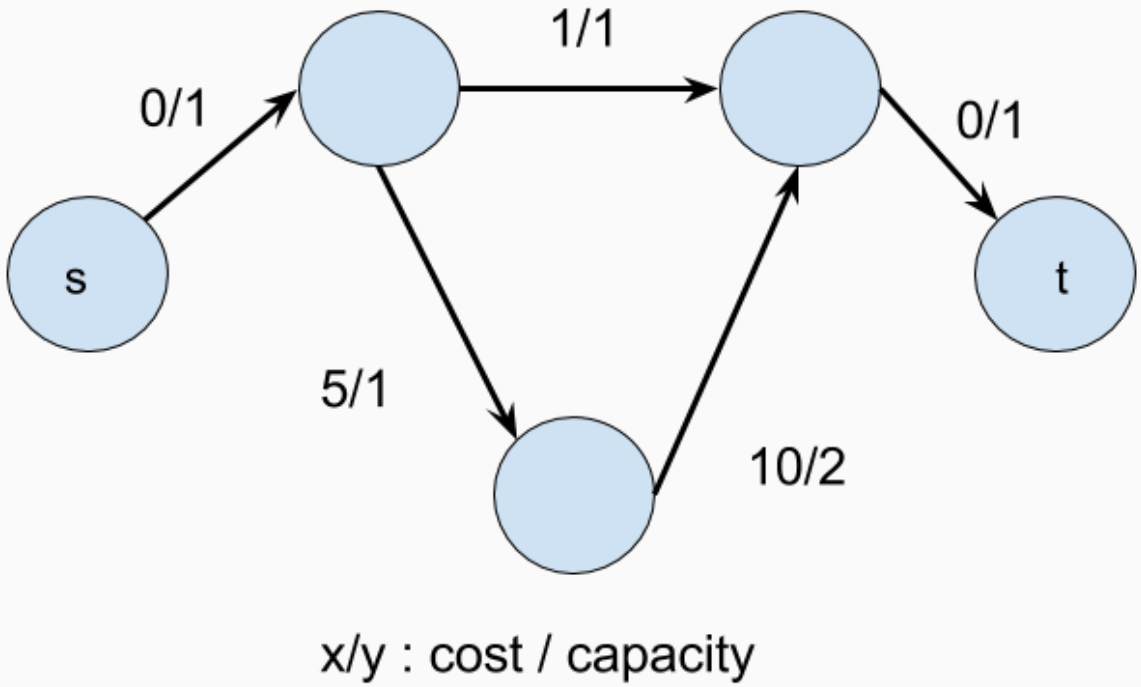
Given:

- Directed graph $G = (V, E)$
- Capacity u_{ij} for each edge
- Cost c_{ij} for each edge: **the cost of sending one unit of flow**
- Source s and sink t

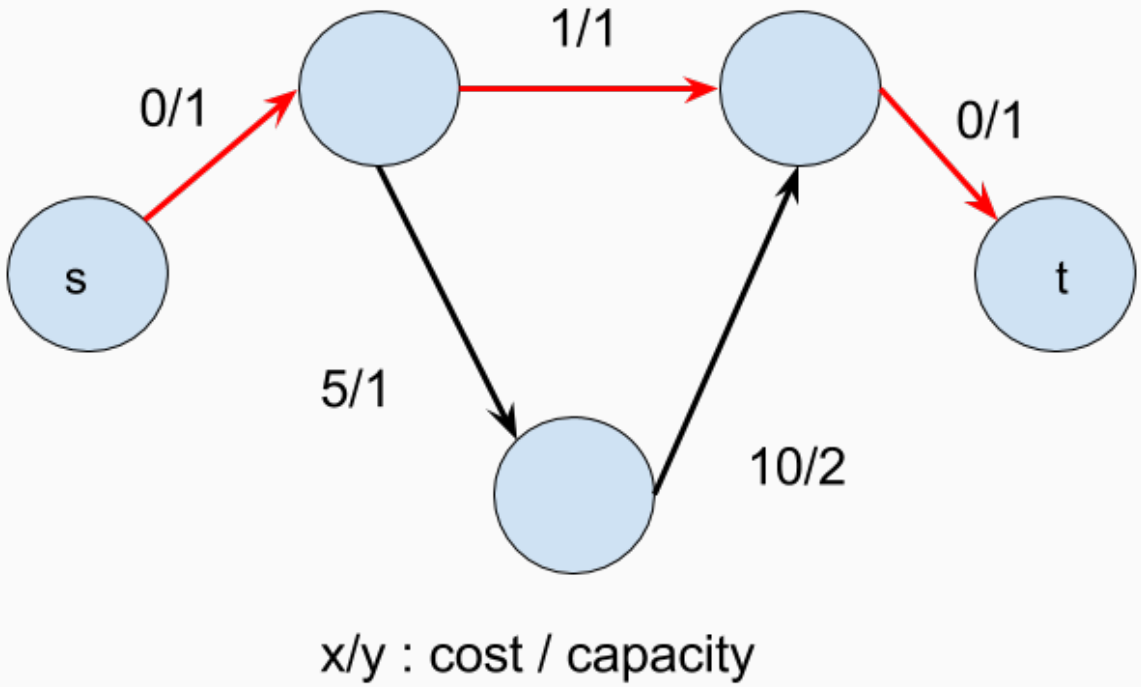
Min Cost Max Flow Problem

Goal: Find the minimum cost maximum flow, i.e. out of all possible maximum flows, what's the cheapest?

Min Cost Flow Problem



Min Cost Flow Problem

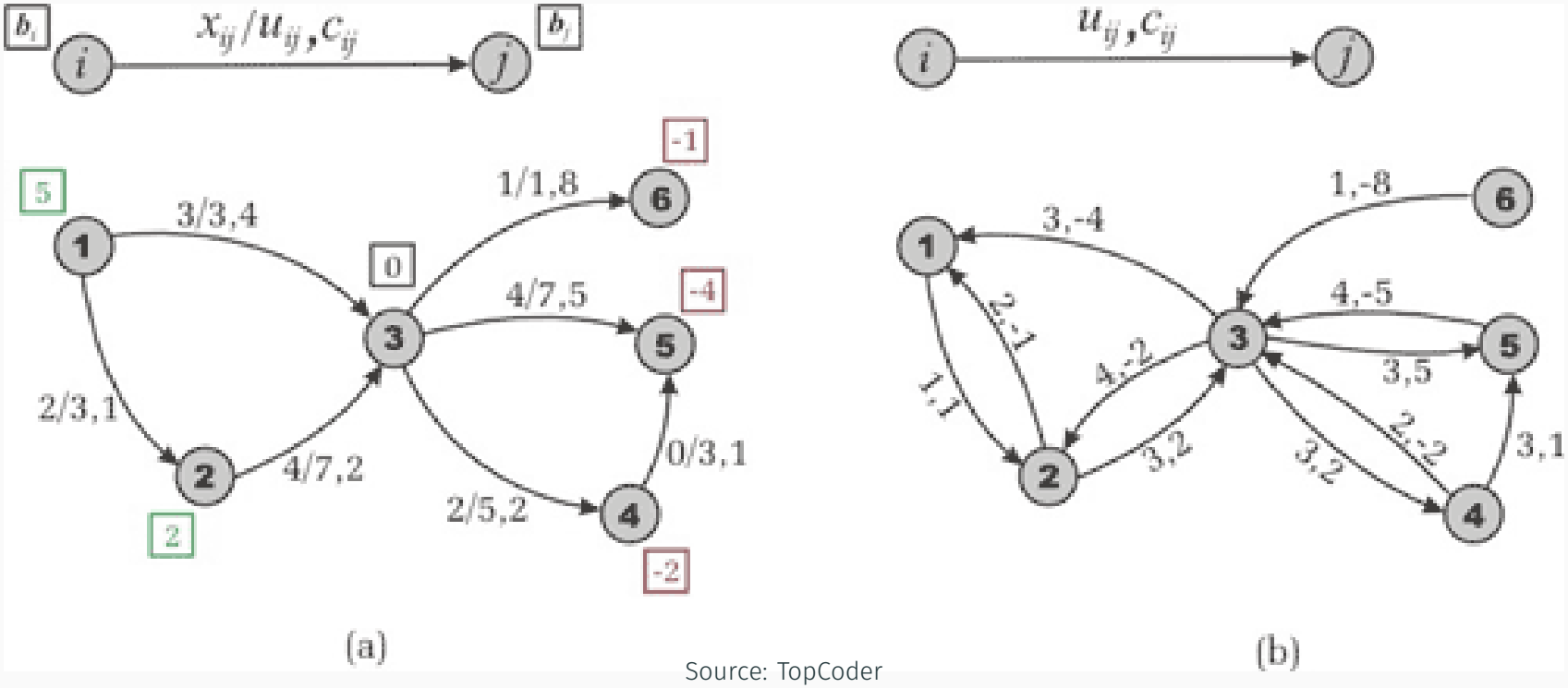


Optimality Condition

What does the residual network look like?

- If (i, j) is an edge with
  - flow x_{ij}
  - residual capacity $r_{ij} = u_{ij} - x_{ij}$
  - cost per flow c_{ij}
- The corresponding backward edge (j, i) has
  - residual capacity $r_{ji} = x_{ij}$
  - cost per flow $c_{ji} = -c_{ij}$.
    This is because sending a unit of flow on the reverse edge decreases the objective by c_{ij}

Optimality Condition



Source: TopCoder

Optimality Condition

Notice that we cannot have negative cost cycles in residual graph if our flow has optimal cost!

Optimality Condition

Theorem. A maximum flow has minimum cost if and only if residual graph has no negative cost cycles

Successive Shortest Path Algorithm

Idea: find minimum cost augmenting path

Successive Shortest Path Algorithm

Idea: find minimum cost augmenting path

Why does this work? If we start with no negative cost cycle, then shortest augmenting path will never introduce negative cost cycles!

Successive Shortest Path Algorithm

Idea: find minimum cost augmenting path

Why does this work? If we start with no negative cost cycle, then shortest augmenting path will never introduce negative cost cycles!

$\Rightarrow$ Algorithm will NOT work if original graph has negative cost cycles (we can have flow "circulation", in addition to $s \rightarrow t$ flow)

Successive Shortest Path Algorithm

Just run Ford-Fulkerson, but use a shortest path algorithm to find a minimum cost $s - t$ path on the residual graph.

Successive Shortest Path Algorithm

Just run Ford-Fulkerson, but use a shortest path algorithm to find a minimum cost $s - t$ path on the residual graph.

Residual graph has negative costs $\Rightarrow$ need Bellman-Ford

Successive Shortest Path Algorithm

Just run Ford-Fulkerson, but use a shortest path algorithm to find a minimum cost $s - t$ path on the residual graph.

Residual graph has negative costs $\Rightarrow$ need Bellman-Ford

Suppose B is the largest capacity of an edge leaving s, then possibly augment nB times, with $O(nm)$ Bellman-Ford per augmenting path

$\Rightarrow$ Time complexity: $O(n^2mB)$

Can we do better?

Successive Shortest Path Algorithm

Bellman-Ford is slow $\Rightarrow$ can try Dijkstra after re-weighing costs

- Idea is same as Johnson's algorithm
- Use Bellman-Ford once to compute distance to each node, $p(u)$
  $\Rightarrow$ Call $p(u)$ the potential of node u
- Re-weigh edge cost $c(u, v) \mapsto c(u, v) + p(u) - p(v)$
- New costs non-negative because $p(v) \leq p(u) + c(u, v)$

Successive Shortest Path Algorithm

Bellman-Ford is slow $\Rightarrow$ can try Dijkstra after re-weighing costs

- Idea is same as Johnson's algorithm
- Use Bellman-Ford once to compute distance to each node, $p(u)$
  $\Rightarrow$ Call $p(u)$ the potential of node u
- Re-weigh edge cost $c(u, v) \mapsto c(u, v) + p(u) - p(v)$
- New costs non-negative because $p(v) \leq p(u) + c(u, v)$

Key: can update $p(u)$ "for free" after augmenting path!

- When finding path, run Dijkstra from source to all nodes
- Get distance from source and use it as new $p(u)$
- Why does this work? New reverse edges have re-weighed cost 0.

Time complexity: $O(n^3 B)$, using $O(n^2)$ Dijkstra

Other Algorithms

- In Max Flow, Ford-Fulkerson / Edmonds-Karp augment along a single path in every iteration.
- Dinic's algorithm augments all shortest paths at once $\Rightarrow$ faster!

Other Algorithms

- Similarly, the previous algorithm augments the flow along a single path in every iteration.
- The Primal-Dual algorithm augments all shortest paths at once.

Min Cost Matching

- We know how to find maximum matching in a bipartite graph using maximum flow
- Now, we can find minimum/maximum cost maximum matching in a bipartite graph using min cost flow!

Problem 2

You are the CEO of company that sells hats.

Your company owns y warehouses and z shops.

- The warehouse at location i stocks b_i hats $(1 \leq i \leq y)$
- The shop at location j wants to sell b_j hats $(y+1 \leq j \leq y+z)$

There are roads between locations in the city and it costs c_{ij} to move 1 hat on a road from location i to j. You also cannot move more than u_{ij} hats on that road.

What is the minimum amount of money you need to spend so that your company can still run?

Problem 2 - Solution

Construct a graph:

- Vertices are locations in the city
- Each road in the city: add same edge with capacity u_{ij}, cost c_{ij}
- Source to each warehouse with capacity b_i and cost 0
- Each shop to sink with capacity b_j and cost 0
- Find the minimum cost flow on this network.

Problem 3

There are n boys and m girls on a dating app. All boys like all the girls and vice versa.

We want to match as many couples as possible. However, the we get c_{ij} dollars if we match boy i with girl j.

What's the maximum amount of money we can make while matching as many couples as possible?

Problem 3 - Solution

- We want to want find the maximum weight maximum matching.
- We can negate the edge costs and find the minimum cost maximum matching instead.
- Flow graph has no cycles so obviously no negative cost cycles.

Problem 4

A cities contains intersections (vertices) and roads (edges).
It takes a certain time to run through each road.

You and your partner in crime are trying to escape the police.

- Both of you are at intersection 0 initially.
- There is a supercar at intersection $n - 1$, which you want to use to escape.

You and your partner both need to get to intersection $n - 1$, but must go on paths sharing no roads to avoid capture by police.

Minimize combined time it takes for both of you escape!

Problem 4 - Solution

This is just finding 2 edge disjoint paths in the city at minimum cost.

- Add a capacity of 1 and cost = time to each road.
- Add source to intersection 0 with capacity 2 and cost 0.
- Add intersection n-1 to sink with capacity 2 and cost 0.
- Check if the flow value is 2, and obtain the minimum cost.

Problem 5

Consider a factory and warehouse for k consecutive time periods

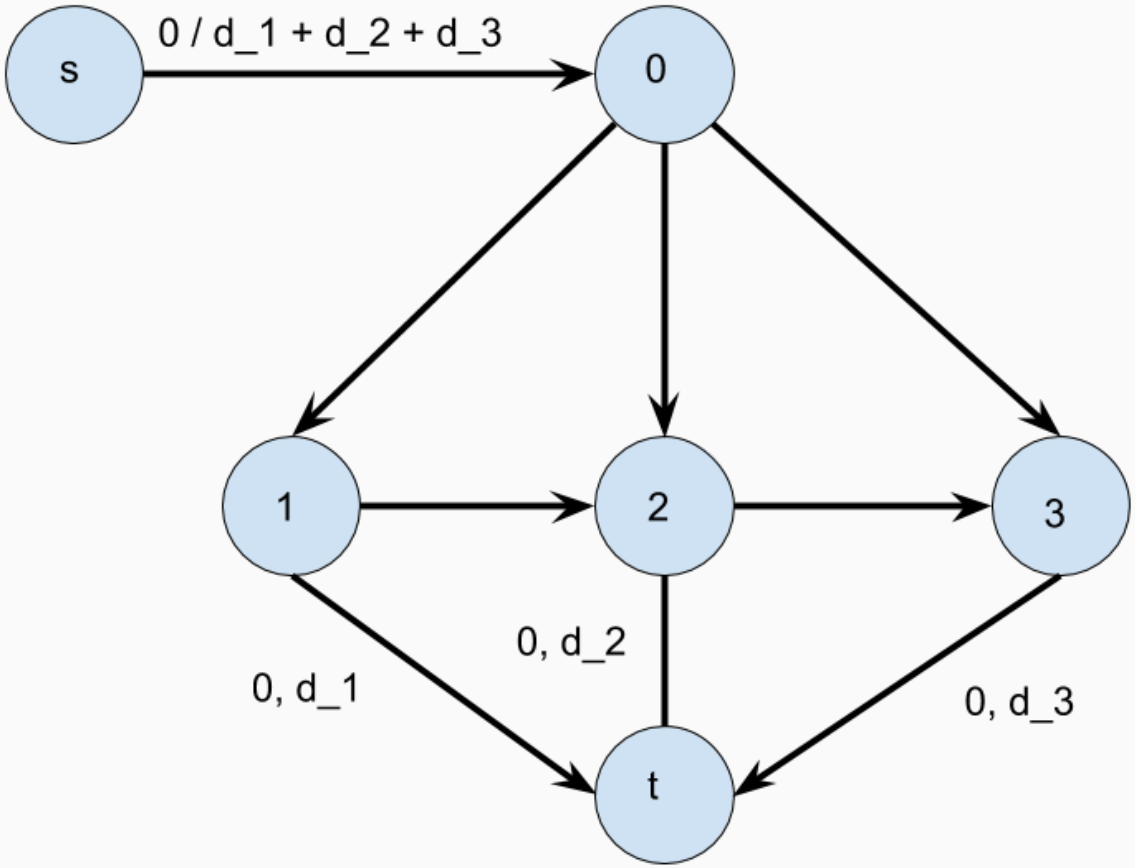
During time period j there is a demand d_j widgets, which can be satisfied by doing one of the following

- Producing widgets during time j for a cost of c_j per widget
- Take surplus widget in warehouse from time $j-1$

The warehouse can store at most u_j widgets from time $j-1$ to j and this costs c'_j per widget.

What's the optimal cost of satisfying all requirements?

Problem 5 - Solution



Remaining edge have the obvious capacities and costs.

References

TopCoder articles:

- https://www.topcoder.com/community/data-science/data-science-tutorials/minimum-cost-flow-part-one-key-concepts/
- https://www.topcoder.com/community/data-science/data-science-tutorials/minimum-cost-flow-part-two-algorithms/
- https://www.topcoder.com/community/data-science/data-science-tutorials/minimum-cost-flow-part-three-applications/

Next Class

Presentations!