CMSC 451: Shortest Paths with Negative Weights

Slides By: Carl Kingsford

Department of Computer Science
University of Maryland, College Park

Based on Section 6.8 of *Algorithm Design* by Kleinberg & Tardos.

Dynamic Programming Principles, Reviewed

Dynamic Programming Pattern:

1. Decompose the problem into subproblems.
2. Recursively define the value of a solution of a subproblem by the value of solutions of smaller subproblems.
3. Compute the value of the solutions for subproblems from smaller to larger.
4. Use the choices made (arrows) to reconstruct an actual solution.

DP Principles, 2

Principle of Optimality: A problem obeys this principle if an optimal solution to the problem contains within it optimal solutions to subproblems.

- Solution to the problem requires making a choice e.g. include the last interval or not?
- This choice leaves 1 or more subproblems unsolved.
- Assuming you're given the optimal solution to these subproblems, you show how to construct the optimal solution to the larger subproblem.

Shortest Path Problem

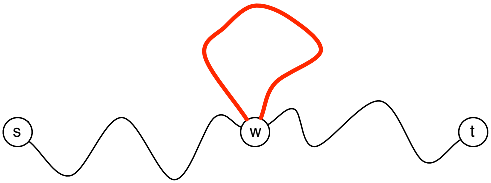
Shortest Path with Negative Weights

Given directed graph G with weighted edges (weights may be positive or negative), find the shortest path from s to t.

Complication of Negative Weights

Negative cycles: If some cycle has a negative total cost, we can make the $s - t$ path as low cost as we want:

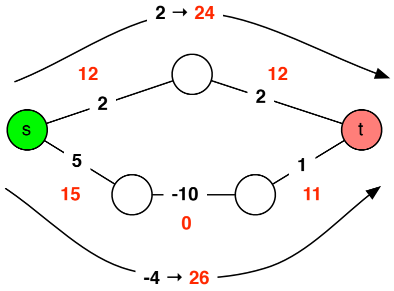
Go from s to some node on the cycle, and then travel around the cycle many times, eventually leaving to go to t.



Assume, therefore, that G has no negative cycles.

Let's just add a big number!

- Adding a large number M to each edge doesn't work!
- The cost of a path P will become $M \times \mathrm{length}(P) + \mathrm{cost}(P)$.
- If M is big, the number of hops (length) will dominate.



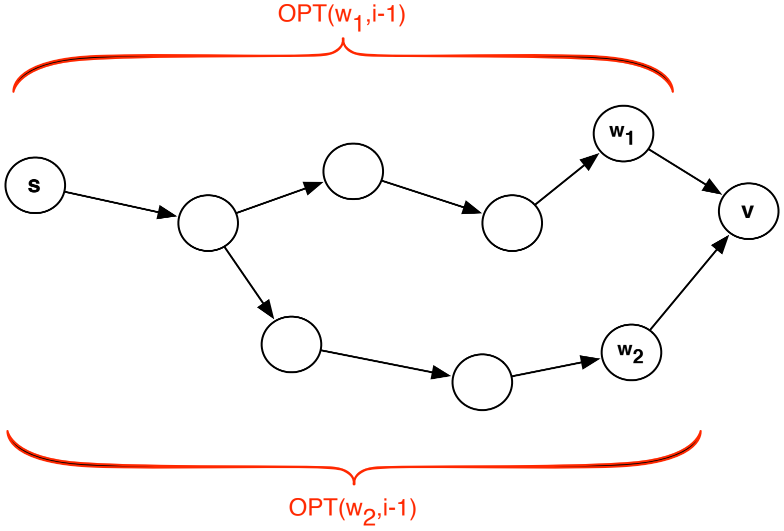
Subproblems

Definition

$OPT(v, i)$ is minimum cost of a path from s to v that uses at most i edges.

1. If best $s - v$ path uses at most $i - 1$ edges, then $OPT(v, i) = OPT(v, i - 1)$.
2. If best $s - v$ uses i edges, and the last edge is (w, v), then $OPT(v, i) = c_{wv} + OPT(w, i - 1)$.

Subproblems, picture



Recurrence

Let $N(w)$ be the neighbors of w.

$OPT(v, i) =$ cost of best path from s to v using at most i edges.

Recurrence:

$$OPT(v, i) = \min \begin{cases} OPT(v, i-1) \\ \min_{w \in N(v)} OPT(w, i-1) + c_{wv} \end{cases}$$

Goal: Compute $OPT(t, n-1)$.

What do we need i?

Why do we introduce the variable i?

What do we need i?

Why do we introduce the variable i?

- i gives us a natural ordering on the problems from larger to smaller.
- To solve $OPT(v, i)$ we need to know only $OPT(w, k)$ for $k < i$.

$\Longrightarrow$ by expanding our class of subproblems, ordering them can become simpler.

Code

```
ShortestPath(G, s, t):
 For i = 1,...,n-1:
   For v in V:
     // try all possible w's:
     best_w = None
     for w in N(v):
        best_w = min(best_w, OPT[i-1,w] + c[w,v])

     M[v,i] = max(best_w, OPT[v, i-1])
   EndFor
 EndFor
 Return M[t, n-1]
```

Running Time

Simple Analysis:

- $O(n^2)$ subproblems
- $O(n)$ time to compute each entry in the table (have to search over all possible neighbors w).
- Therefore, runs in $O(n^3)$ time.

A better analysis:

- Let n_v be the number of edges entering v.
- Filling in each entry actually only takes $O(n_v)$ time.
- Total time $= O\left(n \sum_{v \in V} n_v\right) = O(nm)$.