

Dijkstra's and Dynamic Programming

Lecture 7

Timothy Kim
GWU CSCI 6212

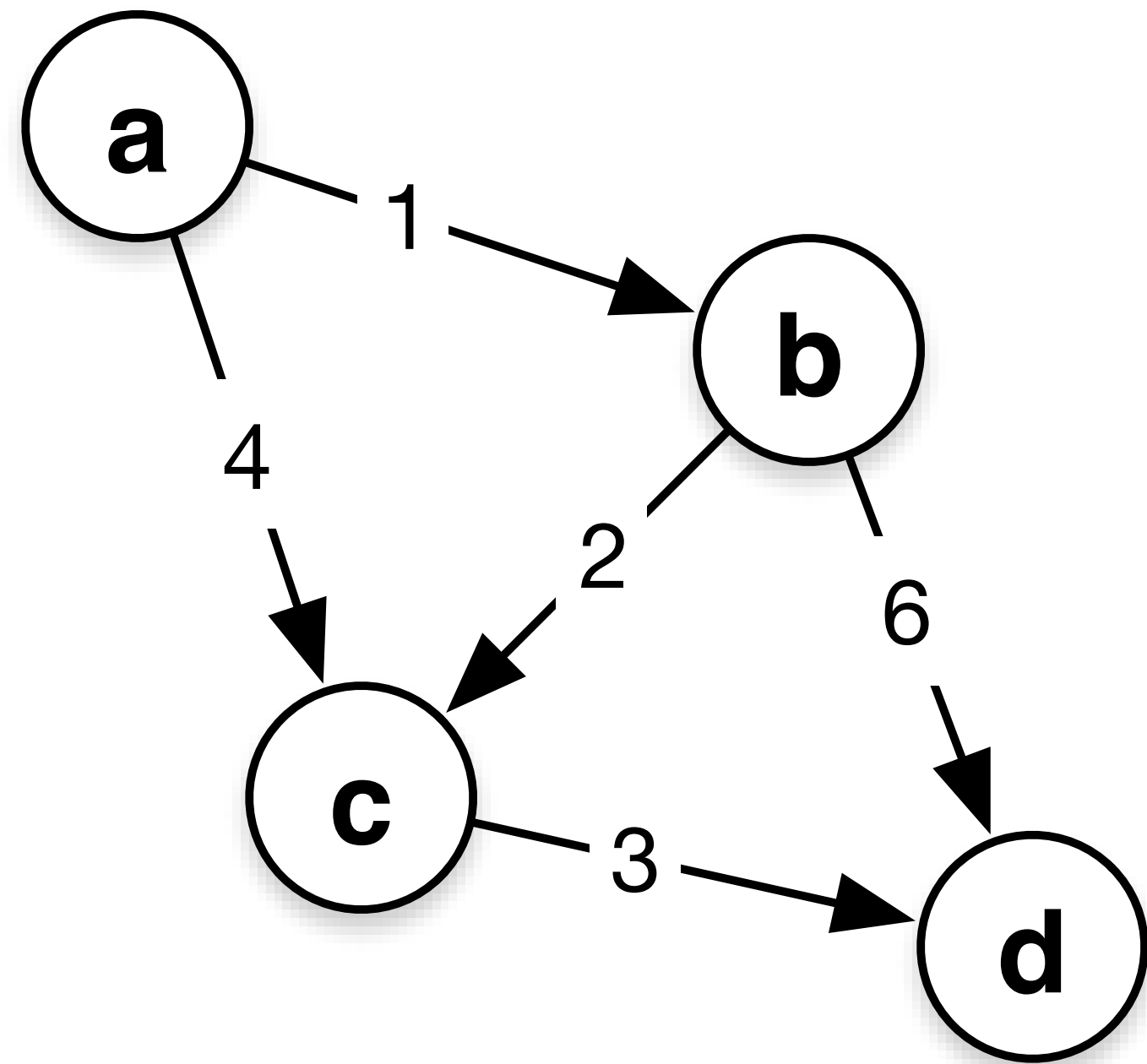
Dijkstra's Shortest Path

- **Problem:** Find the shortest distance between a source vertex to every other vertices.
- **Input:** directed graph $G=(V,E)$, starting vertex, S , and edge lengths, where all edge lengths are non-negative
- **Output:**
 - Lengths of shortest paths from S to all vertices

Algorithm

```
1. Dijkstra:
2.    $X = \{S\}$ 
3.    $A[S] = 0$ 
4.   while  $X \neq V$ :
5.     for all edges crossing the  $\text{Cut}(X, V-X) \rightarrow (u,v)$ :
6.        $(u^*, v^*) = \min(A[u] + \text{length}(u,v))$ 
7.        $A[v^*] = A[u] + \text{length}(u^*, v^*)$ 
8.        $X += v^*$ 
9.   return  $A$ 
```

Example



v	a	b	c	d
{a}	0	∞	∞	∞
{a,b}	0	1	∞	∞
{a,b,c}	0	1	3	∞
{a,b,c,d}	0	1	3	6

Proof of Correctness

- **Lemma:**
 - For every directed graph, w/ non negative edge length, Dijkstra's algorithm correctly computes all shortest-path distances. In other words, let $A[V]$ be the output of Dijkstra's, and $L[V]$ be the shortest-path distances, then $A[V] = L[V]$
- **Proof by Induction** (on the number of iteration)
- **Base Case:** $A[s] = L[s] = 0$

Proof of correctness

- Inductive Hypothesis:
 - All previous iterations are correct:
 - for all v in X , $A[v] = L[v]$
- Inductive Step:
 - The algorithm at this point picks an edge (u^*, v^*) and adds v^* to X .
 - $A[v^*] = A[u^*] + \text{LENGTH}(u^*, v^*) \stackrel{?}{=} L[v^*]$
 - We know that $A[u^*] == L[u^*]$ from inductive hypothesis
 - In other words, does all paths $s \rightarrow v^*$ has length $\geq L[u^*] + \text{LENGTH}(u^*, v^*)$?

Proof of correctness

- Let $P = \text{any } s \rightarrow v^* \text{ path.}$
- P must cross $(X, V-X)$
- Let (y, z) be the first edge that is on P and crosses the cut $(X, V-X)$, then:
 - $\text{Length}(s \rightarrow y) \geq L[y]$
 - $\text{Length}(z \rightarrow v^*) \geq 0$ (we don't allow negative edges)
- $\text{Length}(P) \geq L[y] + \text{LENGTH}(y, z)$
- By our algorithm $A[u^*] + \text{LENGTH}(u^*, v^*) \leq L[y] + \text{LENGTH}(y, z) \leq \text{Length}(P)$
- In other words, all paths $s \rightarrow v^*$ has length $\geq L[u^*] + \text{LENGTH}(u^*, v^*)$ is true.
- QED

Running Time

- Naive implementation: $O(n^2)$
- Using Heap: $O(m \log n)$

Dynamic Programming

Maximum Weight Independent Set

- Input: Path graph $G=(V,E)$ with non-negative vertices
- Output: An independent set, subset of non-adjacent vertices, with maximum weight
- Brute force solution: exponential running time
- Greedy?
- Dynamic programming?

Principle of Optimality

- "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision." (See Bellman, 1957, Chap. III.3.)
- In simpler terms, optimal solution must come from "smaller" sub-problems that are optimal.

MWIS Problem

- Assume that we have a solution and work "backwards"
- Let S be the MWIS(G), and $v_n = \text{last vertex in } S$.
- Case 1: $v_n \notin S$,
 - Let $G' = G - v_n$, then $\text{MWIS}(G') = S$
 - Proof by contradiction (exercise)
- Case 2: $v_n \in S$,
 - Let $G'' = G - v_n - v_{n-1}$, then $\text{MWIS}(G'') = S - v_n$
 - Proof by contradiction (exercise)

Line of thought

```
1. MWIS(G):  
2.   if (|G| == 1): return G[1]  
3.   S1 = MWIS(G - V[n])  
4.   S2 = MWIS(G - V[n] - V[n-1])  
5.   return max(S1, S2)
```

- Only if we knew if $v_n \in S$ is true or not, then we can recursively compute G' or G'' .
- Since we do not, why not try both and pick the better one?
- But this is same as brute force!

Question

```
1. MWIS(G):  
2.   if (|G| == 1): return G[1]  
3.   S1 = MWIS(G - V[n])  
4.   S2 = MWIS(G - V[n] - V[n-1])  
5.   return max(S1, S2)
```

- How many distinct calls to MWIS are there?
- $O(n)$, one for each of the distinct prefix of G .
- So let's compute things bottom up!

Algorithm

1. $\text{MWIS}(G):$
2. $A[0] = 0$
3. $A[1] = V[1]$
4. for $i = 2$ to n :
5. $A[i] = \max\{A[i-1], A[i-2] + V[i]\}$

Reconstruction

```
1. S = []
2. while i ≥ 1:
3.     if A[i-1] ≥ A[i-2] + V[i]:
4.         i--
5.     else:
6.         S += V[i]
7.         i -= 2
```


Dynamic Programming

1. Find a sub-problem that preserves the Principle of Optimality (requires proof)
2. Derive recurrence from 1
3. Use the recurrence to formulate the algorithm
4. Write a reconstruction algorithm to generate the desired output

Integral Knapsack

- Input:
 - n items with their
 - v_i , values (non-negative)
 - w_i , sizes (non-negative and integral)
 - W , capacity (non-negative and integral)
- Output:
 - Find $S \subseteq \{1, 2, \dots, n\}$ such that
 - Maximizes $\sum v_i$
 - Subject to $\sum w_i \leq W$

Principle of Optimality

- Let S be the max-value solution, and impose arbitrary ordering to our items, therefore n is the last item.
- Case 1: $n \notin S$
 - S must be optimal with the first $n-1$ items with the capacity W .
- Case 2: $n \in S$
 - $S-n$ must be optimal with the first $n-1$ items with capacity $W-w_n$
 - Proof by contradiction (exercise)

Recurrence

- Let $V_{i,x}$ = value of the best solution for $\{1, 2, \dots, i\}$ with capacity X .
- $$V_{i,x} = \begin{cases} V_{i-1,x} & \text{if } w_i > x \\ \max\{v_{i-1,x}, v_i + V_{i-1, x-w[i]}\} & \text{otherwise} \end{cases}$$

Subproblem Identification

- We need to apply the objective function (sum of v_i) to
 - All possible prefix of item $\{1, 2, \dots, n\}$
 - All possible residual capacity of $\{0, 1, \dots, W\}$

Algorithm

1. Let A = 2d-Array
2. Init $A[0,x] = 0$ for all $x = 0, 1, \dots, W$
3. for $i = 1$ to n :
4. for $x = 0$ to W :
5. $A[i,x] = \max\{A[i-1,x], A[i-1,x-w[i]] + v[i]\}$
6. return $A[n,W]$

Example

- $n = 4$
- $W = 6$
- $v = \{3, 2, 4, 4\}$
- $w = \{4, 3, 2, 3\}$

6	0	3	3	7	8
5	0	3	3	6	8
4	0	3	3	4	4
3	0	0	2	4	4
2	0	0	0	4	4
1	0	0	0	0	0
0	0	0	0	0	0
	0	1	2	3	4

Reconstruction

- Homework