Dynamic Programming

Pramesh Kumar

IIT Delhi

February 5, 2024

Outline

Motivation

Solving SP using DP

General framework

Solving Knapsack using DP

Motivation: Recursion

- Suppose you want to calculate $f(n) = n!$
- You know that if you were given the value of $f(n-1)$, then you can easily compute the value of $f(n)$. How?
  $f(n) = n \times f(n-1) = n \times (n-1)!$.
- Similarly, to compute the value of $f(n-1)$, you need to evaluate $f(n-2)$.
- At last, you know that $f(1) = 1$.
- In general, one can write

$$f(n) = \begin{cases} 1 & n = 1 \\ nf(n-1) & n > 1 \end{cases}$$

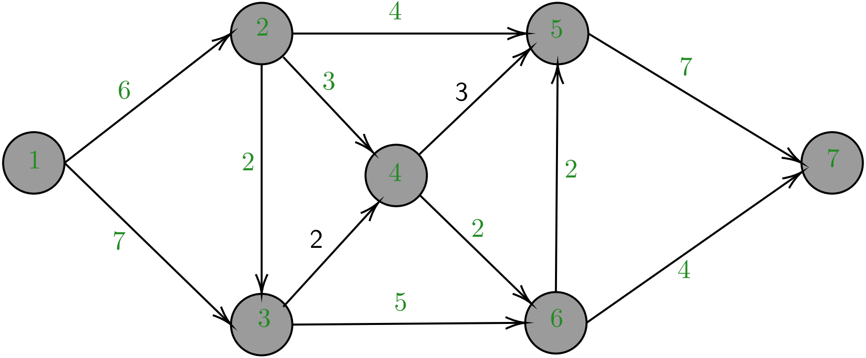
Dynamic Programming

- Suppose we have a large optimization problem at hand, which cannot be solved easily.
- However, we realize that we can solve this problem by solving similar smaller subproblems.
- Similarly, we can those subproblems by solving even smaller subproblems.
- Continuing in this fashion, we will encounter the subproblems that can be trivially solved.
- Typically, the number of subproblems to solve should be polynomial in input size.

Outline

Shortest path problem

Find the shortest path from 1 to 7. The link costs are shown over the links.

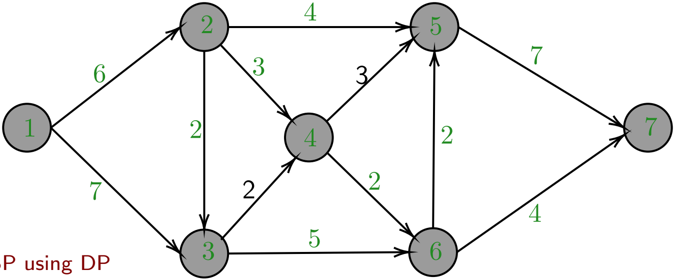


Shortest path problem

- Let V_i be the cost of shortest path from node i to 7 and c_{ij} be the cost of traversing link (i,j).
- Assume that if you are given the V_2 (cost of shortest path from 2 to 7) and V_3 (cost of shortest path from 3 to 7), you can easily evaluate the cost of shortest path from 1 to 7.

$$V_1 = \min\{6 + V_2, 7 + V_3\}$$

- Similarly, you can compute V_2 if you are given V_5 and V_4.
- Continuing in the same fashion, we will encounter V_7, i.e., the cost of shortest path from 7 to 7, which we know is equal to 0.



Shortest path problem

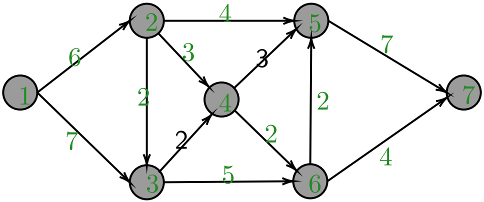
- In general, we have

$$V_i = \begin{cases} 0 & \text{if } i = t \\ \min_{j \in FS(i)}\{c_{ij} + V_j\} & \text{if } i \neq t \end{cases}$$

- The following observation can be made: If the shortest path from s to t passes through node k, then the subpaths $(s \rightsquigarrow t)$ and $(k \rightsquigarrow t)$ must be shortest paths from s to k and k to t respectively.
- If this were not true, then you can construct a shorter path from s to t, which is a contradiction.
- To solve the above problem, we'll use the fact that $V_7 = 0$, which can help evaluate the values of V_5 and V_6. Such a technique is called backward recursion.
- To trace the shortest path, let's assume

$$\mu(i) = \text{argmin}_{j \in FS(i)}\{c_{ij} + V_j\}, \forall i \neq t$$

Shortest path problem



- $V_5 = \min\{7 + V_7\} = 7, \mu(5) = 7$,
- $V_6 = \min\{2 + V_5, 4 + V_7\} = \min\{9, 4\} = 4, \mu(6) = 7$,
- $V_4 = \min\{3 + V_5, 2 + V_6\} = \{10, 6\} = 6, \mu(4) = 6$,
- $V_3 = \min\{2 + V_4, 5 + V_6\} = \min\{8, 9\} = 8, \mu(3) = 4$
- $V_2 = \min\{4 + V_5, 3 + V_4, 2 + V_3\} = \min\{11, 9, 10\} = 9, \mu(2) = 4$
- $V_1 = \min\{6 + V_2, 7 + V_3\} = \min\{15, 15\} = 15$, $\mu(1) = 2, 3$

There are two shortest paths from 1 to 7, both having cost $= 15$. The shortest paths are given as: $1 - 2 - 4 - 6 - 7$ and $1 - 3 - 4 - 6 - 7$.

Outline

General framework of Dynamic Programming

- Dynamic Programming is quite helpful in formulating problems which involve sequential decision making.
- For deterministic problems, we attempt to find the following components:
  - State: The state is the information about the system that is enough to summarize present condition. Let us denote the state space by S indexed by s.
  - Actions: At each state, there are only few actions one can take. Let us denote the set of actions by $A(s)$.
  - Reward/Cost: For each action $a \in A(s)$, there is an immediate reward or cost $c(s, a)$.
  - Transitioning state: Once one takes the action $a \in A(s)$ at state s, the system transitions to a new state denotes by $s'(x, a)$.
  - Value function: It denotes the optimal value if we choose the optimal action in this state and onward. Let us denote it using $V(s)$.
  - Our aim is to minimize the total cost from an initial state.

$$\boxed{V(s) = \min_{a \in A(s)} \{c(x, a) + V(s^{'}(x, a))\}}$$

- Above equation is called Bellman's principle of optimality.

Outline

Knapsack problem

Given a set of n items, each with a weight w_i and a value a_i, determine which items to include in the Knapsack so that the total weight is less than or equal to a given limit W and the total value is as large as possible.

$$Z = \underset{\mathbf{x}}{\text{maximize}} \quad \sum_{i=1}^{n} a_i x_i \tag{1}$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_i x_i \leq W \tag{2}$$

$$x_i \in \{0, 1\}, \forall i = 1, ..., n \tag{3}$$

Can we solve the above problem using DP?

DP formulation of Knapsack Problem

- State: (i, b) represent the items from 1 to i to pick from and available knapsack capacity b.
- Action: Whether to pick item i or not. In case, $w_i > b$, we do not have choice to pick item i.
- Reward: If we pick item i, we get the value a_i.
- Value function: $Z(i, b)$ represent the maximum value of the selected items if we restrict our selection to the items 1 through i and available knapsack capacity b.
- Principle of optimality:

$$Z[i, b] = \max\{Z[i-1, b], a_i + Z[i-1, b-w_i]\}$$

  We need to compare the value of picking or not picking the item i. If we pick the item i, we get the value of a_i but our next state will be $(i-1, b-w_i)$. If we do not pick the item i, we do not get any new value, and our next state will be $i-1, b$.
- Our goal is to find $Z[n, W]$.

Origins of Dynamic Programming

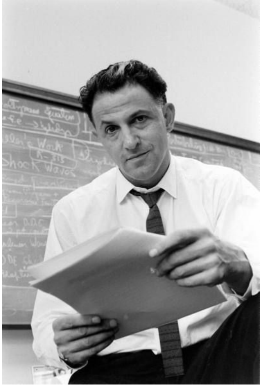


Figure: Richard Ernest Bellman (Source: Pinterest)

Richard Ernest Bellman was an American applied mathematician who first developed dynamic programming in 1953. (Source: AMO)

Final thoughts

- DP is an important tool for solving complex problems by breaking down it into smaller and easier subproblems.
- The key is to find the state and recursion formula.
- Under uncertainty, there are some amazing results. Please refer to material on Sequential Decision Making under uncertainty.

Thank you!