Dynamic Programming: Bottum-up Problem Solving

Remington Greko, Tyler Gutowski, and Spencer Hirsch

February 12, 2023

Bottom-up Problem Solving

Richard Bellman described this problem solving technique in the 1950's and called it Dynamic Programming to impress his sponsors. Dynamic Programming is an important problem solving paradigm. Read about Bellman on Wikipedia and summarize the importance of his contributions.

Describe how dynamic programming works: Solve and memoriz- ing solutions to small problems and use these solutions as building blocks to construct a solution to a larger problem from the bottom- up.

Example uses of Dynamic Programming

Give at least three examples problems that use the dynamic program- ming paradigm to solve problems. (You should be able to search finding examples and summarize these in your group's words.)

Example One: (Spencer)

Good approximations to hard problems

There are Hard Problems: Problems that do not (seem) to have effi- cient solutions. You will explore some of these later in future quests.

There are dynamic programming algorithms that provide good ap- proximate solutions to these hard problems. My memory tells me the 0–1 Knapsack Problem is an example of a hard problem with a good dynamic programming approximation. Report on this example and at least 2 additional hard problems with good approximate solutions.

0-1 Knapsack Problem: (Spencer)

The problem goes as follows:

The 0-1 Knapsack problem is a programming problem where as input you are given, an array of items, [1, 2, ..., n - 1, n], and their respective weights given as an array, where each index in the array corresponds to it's respective item. You are also given a maximum capacity of the the sack that is going to be used to hold the items. You must find the maximum number of items you can fit into the knapsack without exceeding the capacity of the knapsack. However, the quantity of the items that can be put into the bag is either zero or one of each specific item.

Solution:

- A DP[][] table will contain all of the possible weights that the backpack could contain.
- The state DP[i][j] will denote a maximum value of the weight considering all values. So you can consider the weight of the row and fill in the columns that have the total possible weight $\geq$ the weight of the row.
  - Fill in weight of row
  - Don't fill in weight of the row
- Find the maximum weight of the filled in columns that have been compared to the maximum weight of the knapsack.

+

Name	**Section**
Remington Greko	
Tyler Gutowski	
Spencer Hirsch	One Example use of Dynamic Programming & One hard algorithm example