Dynamic Programming: Bottum-up Problem Solving

Remington Greko, Tyler Gutowski, and Spencer Hirsch

February 13, 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.

Bellman Summary:

Richard Bellman was an applied mathematician who was the first to introduce the concept of dynamic programming to the field. One of his first major contributions was the Bellman Equation. This equation is also known as dynamic programming and outlines how to optimize the solution of a given problem. Uses of this equation were first introduced to the engineering and mathematics fields, and later made its way into the economic field as well.

A second major contribution Bellman made was the Hamilton-Jacobi-Bellman Equation. It is a partial differential equation which is related to his previously created Bellman Equation. This new equation provides the "optimal cost-to-go for a given dynamical system" (Bellman Wikipedia). This equation is closesly related to the Bellman equation and was another pioneering push for the development of the theory of dynamic programming.

A third major contribution is the Curse of Dimensionality which describes the issues with the "exponential increase in volume" (Bellman Wikipedia) when adding dimensions to a space. The curse demonstrated how some problems may grow exponentially when introducing "more state variables to the value function" (Bellman Wikipedia). This means that the volume of the mathematicl space increases so fast that the data becomes sparse and therefore requires much more data to obtain a result (Curse of Dimensionality Wikipedia).

The final major contribution discussed on Bellman's Wikipedia page is the Bellman-Ford algorithm, also known as the Label Correcting Algorithm. This algorithm computes "single-source shortest paths" (Bellman Wikipedia) in weighted digraphs where some edges may have negative values. This algorithm is very similar to Dijkstra's algorithm but, unlike Dijskta's, works with negative edge values.

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)

Coin Change Problem:

One problem that can utilize dynamic programming to optimize the solution is the coin change problem. I chose this problem because it reminds me of a problem that we had in CSE 1002 and is something that I often think about because I typically handle cash at work.

The problem goes as follows:

Given an unlimited supply of coins as well as the denominations of coins as input. Find all possible ways the desired change, also given as input, can be returned. There are always different solutions to returning change, however, we typically default to the most convient, least number of coins.

This problem can he solved using recursion, which would most likely be the go-to solution for most programmers. However, using dynamic programming you can optimize this solution bringing down the time, space and memory complexity of the problem. This being said, using dynamic programming would be the best way to solve this particular problem.

Example Two: (Remi)

Longest Common Subsequence:

A second example of a problem which can be solved using dynamic programming is the longest common subsequence problem. This problem is one that I remember learning in data structures & algorithms, so I chose it for my example.

The problem goes as follows:

Given two strings, find the longest subsequence of characters that is shared between the two strings. The dynamic approach to solving this problem is performed in O(x * y) time where x and y are the lengths of the two strings being compared. This is a far better complexity than an approach where every possible subsequence is generated for each string and then compared. This would be an example of a brute-force approach, and it's time complexity is far worse.

Example Three: ()

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.

The Solution:

Using the iterative dynamic programming approach you would define a 2d array, where the rows corresponds to the index of the items and the weights are defined on the columns. For every weight you can either choose to ignore it or use it when constructing the 2d array. Thus, you can calculate the maximum weight that can be held in the knapsack based on the items and their corresponding weights.

By iterating through the 2d array and constructing all possibilites, comparing the maximum for each to the absolute maximum of the knapsack will give you the best solution to the given problem. The use of dynamic programming gives the most optimal space and time complexity as opposed to the brute force approach that could also be used for this problem.

Longest Common subsequence: (Remi)

The problem goes as follows:

The shortest path problem is a programming problem where the goal is to find the quickest route between an origin and goal point. In programming the points travelled would be represented by a graph where, for example, cities are denoted by nodes and roads are denoted by edges. Each edge is assigned a weight which represents the length of the road between any two nodes. The combination of edges to arrive at the goal with the lowest total value represents the shortest path.

The Solution:

One very popular algorithm used to solve this problem is Dijkstra's algorithm, which utilizes a dynamic approach in a graph. Firstly the start node is assigned a distance of 0 and every other node is assigned infinity. First the source node is added to the queue. Each step pops the node with the shortest distance to the source node and checks the distance of all of its neighboring nodes. If the new distance is shorter than the previously assigned one, it is updated. The neighboring nodes are then added to the queue and the process repeats until all nodes have either been assigned a value, or the goal node has been reached.

Dijkstra's algoirithm garuntees the optimal path between any two nodes, meaning that it will not waste time producing paths which are not the shortest between the two input nodes. By updating the shortest path to every node at every step, the algorithm will never have to backtrack and reevaluate a distance which has alreaddy been assigned. No unnecessary calculations are perfomed, meaning the time and space complexity is satisfactory.

Name	**Section**
Remington Greko	Second example of Dynamic Programming & Shortest Path
Tyler Gutowski	
Spencer Hirsch	One Example use of Dynamic Programming & 0-1 Knapsack Problem algorithm example