2019

Semester 1

Assignment 3: Dynamic Programming

Connor McDowall
cmcd398
530913386

June 7, 2019

Listings

Contents

List of Figures

List of Tables

1 Coin Counting (30 Marks)

1.1 Stages, States, Actions, Costs

$Stages$: The denominations of coins, $D_n = \{D_1, D_2, ..., D_N\}$
$States$: Current change owed (x), assume $x \geq 0$
$Actions$: The number of coins used in the transaction as per the denominations, $a\epsilon A_n(x) = \{0, 1, 2, ..., \dfrac{x}{D_n}\}$
$Costs$: The number of coins exchanged, $c_n(a) = a$ The costs are independent of x

1.2 Value Function

$$V_N(x) = \begin{cases} \frac{x}{D_N}, & \text{if } \frac{x}{D_N}\epsilon\mathbb{Z}. \\ \infty, & \text{otherwise.} \end{cases} \tag{1}$$

The problem is infeasible if $\frac{x}{D_N}$ is not integer. Change must be given as an integer number of coins Since D_N is the smallest denomination, if the output of $\frac{x}{D_N}$ doesnt give an integer number of coins, then there is there is no combination of coins that can be given as change to form a feasible solution for the value of x in the transaction. This is model by giving this non integer combination a value function of infinity since we are minimising. This is so they are heavily penalised.

1.3 Dynamic Programming Recursion

1.3.1 Equation

$$V_n(x) = \begin{cases} \min\limits_{a\epsilon A_n(x)} \{c_n(a) + V_{n+1}(x - D_n c_n(a))\}, & \text{if x > 0.} \\ 0, & \text{if x = 0.} \end{cases} \tag{2}$$

1.3.2 Explaination

$V_n(x)$: The minimum number of coins with denomination given at stage n as change.
$A_n(x)$: The set of actions, the denominations of coin available for change at stage n.
$c_n(a)$: The cost of given change at stage n.
$V_{n+1}(x - D_n c_n(a))$: The cost to go based on subsequent decisions.

1.4 Optimal substructure and overlapping subproblems explanations

1.4.1 Optimal substructure

Optimal substructure means a combination of locally optimal subproblems find a globally optimal solution. This is the case for most recursive problems. In the context of coin coin tossing, take the following example. If the optimal solution to give change given the set of denominations for x money is c coins, you can break up the problem into two subproblems where the optimal solutions to give change given the same set of coin denominations for y and z money is d and e coins respectively. The combination of the optimal solutions to the two subproblems form the global optimal solution as x money = y + z money and c coins = d + e coins.

1.4.2 Overlapping subproblems

Overlapping subproblems means the optimal solution to subproblem(s) is reused in constructing the optimal solution to the main problem. In the context of this problem, the minimum number of coins to give change for x will be repeatedly used to find the minimum number of coins to give change for y where x < y.

1.5 Natural Ordering

New Action Set : The denominations of coins, $A(x) = \{D_1, D_2, ..., D_N\}$

1.5.1 New recursion

$$V(x) = \begin{cases} \min\limits_{a \epsilon A(x), 0 < a \leq x} \{1 + V(x - a)\}, & \text{if otherwise.} \\ 0, & \text{if x = 0.} \end{cases} \tag{3}$$

1.5.2 Natural ordering

A bottom up ordering is the natural ordering of the subproblems. Firstly, solve for n = 0. Secondly, solve upwards to from n = 0 to n = x. No explicit recurson is required as all solutions to the subproblems are solved and saved by the time you reach n = 0, creating a more efficient formulation.

1.6 Algorithm

Listing 1: Bottom Up Optimal Coin Change

```
function numCoins = optimalCoinChange(x, denoms)
% Function finds the minimum number of coins required to change a
    monetary
% amount.
% Inputs:
% x = amount of money to be given in coins, given as an INTEGER, in
    cents.
%            e.g. $1.35 is input as 135
% denoms = denominations of coins available, in INTEGER cents,
%            given as a ROW VECTOR.
%
% Output:
% numCoins = optimal number of coins used to find x
%
% Connor McDowall, cmcd398, 530913386

% This code implements a bottom up approach. This approach was adapted
    from
% https://github.com/bephrem1/backtobackswe/blob/master/Dynamic...
%20Programming,%20Recursion,%20&%20Backtracking/changeMakingProblem.
    java
% as this is a very common problem.

% Set a parameter to create the appropriately sized subproblem storage
% array and set the values of the matrix to the largest number of coins
% possible plus one (The value of change required). The array includes
    one
% additional element for the base case.
```

```
dpsols = ones(1,x+1)*(x+1);

% Set the base case for the problem
dpsols(1) = 0;

% Sets the number of coin denominations passed into the function
n_denoms = size(denoms,2);

% Solves all the subproblems
% Iterates through all subproblems
for donIdx = 2:x+1

    %Iterates through all coin denominations
    for coinIdx = 1:n_denoms

        % Compares the coins values to the subproblem value
        if denoms(coinIdx) <= donIdx - 1
            % Performs a form of recursion test for an improved
                solution
            % and sets the new sub problem.
            dpsols(donIdx) = min(dpsols(donIdx), ...
                dpsols(donIdx-denoms(coinIdx)) + 1);
        end
    end
end
% Returns the minimum number of coins to use as change. The minimum
    value
% will be the last value.
 numCoins = dpsols(end);
end
```

2 Conducting Interviews (10 Marks)

2.1 Mathematical expression

$$\hat{V}_N = \mathbb{E}(R) = 1 \tag{4}$$

The derivation is show below

$$\begin{aligned}\hat{V}_N &= \mathbb{E}(R)\\ &= \int_0^\infty f(r)dr\\ &= \int_0^\infty e^{-r}dr\\ &= [-e^{-r}]_0^\infty\\ &= [-e^{-\infty}] - [-e^0]\\ &= 0 - (-1)\\ &= 1\end{aligned}$$

2.2 Policy

$$\begin{aligned}\hat{V}_{N-1} &= \max\{r, \hat{V}_N\}\\ &= \max\{r, 1\}\end{aligned}$$

Therefore, hire the applicant and stop interviewing if $r \geq 1$. Otherwise, reject the candidate. Therefore, the policy is:

$$\pi \geq 1 \tag{5}$$

2.3 Proof

$$\hat{V}_{N-1} = \int_0^\infty \max\{\hat{V}_N, r\} f(r) dr \tag{6}$$

Use the accept or reject policy criteria to split it up then solve

$$\begin{aligned}
\hat{V}_{N-1} &= \int_0^\pi \hat{V}_N f(r) dr + \int_\pi^\infty r f(r) dr \\
&= \int_0^1 e^{-r} dr + \int_1^\infty r e^{-r} dr \\
&= \int_0^1 e^{-r} dr - [re^{-r}]_1^\infty - \int_1^\infty -e^{-r} dr \\
&= [-e^{-r}]_0^1 - [re^{-r}]_1^\infty - [e^{-r}]_1^\infty \\
&= -e^{-1} + 1 - 0 + e^{-1} - 0 + e^{-1} \\
&= e^{-1} + 1 \text{ as required}
\end{aligned}$$

2.4 Dynamic programming recursion

Similar working as the previous question, just replace the policy in the integrals with $\hat{V}_{N+1}$

$$\begin{aligned}
\hat{V}_N &= \int_0^{\hat{V}_{N+1}} \hat{V}_{N+1} f(r) dr + \int_{\hat{V}_{N+1}}^\infty r f(r) dr \\
&= \hat{V}_{N+1} \int_0^{\hat{V}_{N+1}} e^{-r} dr + \int_{\hat{V}_{N+1}}^\infty r e^{-r} dr \\
&= \hat{V}_{N+1} \int_0^{\hat{V}_{N+1}} e^{-r} dr - [re^{-r}]_{\hat{V}_{N+1}}^\infty - \int_{\hat{V}_{N+1}}^\infty -e^{-r} dr \\
&= \hat{V}_{N+1} [-e^{-r}]_0^{\hat{V}_{N+1}} - [re^{-r}]_{\hat{V}_{N+1}}^\infty - [e^{-r}]_{\hat{V}_{N+1}}^\infty \\
&= -\hat{V}_{N+1}(e^{-\hat{V}_{N+1}}) + \hat{V}_{N+1} - 0 + \hat{V}_{N+1}(e^{-\hat{V}_{N+1}}) - 0 + e^{-\hat{V}_{N+1}} \\
&= e^{-\hat{V}_{N+1}} + \hat{V}_{N+1}
\end{aligned}$$