Solving Dynamic Programming questions using Integer Optimisation

Sachit gaudi

Jan 5, 2024

Introduction

Dynamic programming are a set of algorithms that are applied to reduce the time complexity with memoization. We will pick 3 problems and walk step by step to improve the time complexity. The common aspect of the below problems is that the problem are explicitly to optimize some parameter either number of coins in coin change or profit in knapsack. If the problems are explicitly asking for optimisation why are not using optimisation techniques to solve them. There are few challenges to it. One is that the variables to be optimised take integer values, meaning the domain is discontinuous. The second challenge is constraints-equality constraint in coin change and inequality constraint in knapsack. Also other constraints include the variable can take only positive values, number of coins, stock either bought or not can take 0-1 . So this leads us to study integer optimisation techniques, with constraints. We will try to understand generic integer optimisation and try to fit both the problems to it. All the problem descripts which are in grey are taken directly from leetcode[1].

Convex Integer Optimisation with constraints

$$\begin{aligned} \min_{x} \quad & c^T x \\ \text{s.t.} \quad & b_l \leq Ax \leq b_u \\ & l \leq x \leq u \\ & x \in \mathcal{Z}^n \end{aligned} \tag{1}$$

The first part of the problem is to prove that this is a convex optimisation,

Coin Change

The problem is taken from leetcode[2]

You are given an integer array coins representing coins of different denominations and an integer amount representing a total amount of money.

Return the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return -1.

You may assume that you have an infinite number of each kind of coin.

Example 1:

Input: coins = [1,2,5], amount = 11
Output: 3
Explanation: 11 = 5 + 5 + 1

Example 2:

Input: coins = [2], amount = 3
Output: -1

[1] https://leetcode.com/
[2] https://leetcode.com/problems/coin-change/description/

Example 3:

Input: coins = [1], amount = 0

Output: 0

Knapsack

In this sample we will consider a new wording of the knapsack problem, the problem is called **Maximum Profit From Trading Stocks**

You are given two 0-indexed integer arrays of the same length present and future where present[i] is the current price of the ith stock and future[i] is the price of the ith stock a year in the future. You may buy each stock at most once. You are also given an integer budget representing the amount of money you currently have.

Return the maximum amount of profit you can make.

Example 1:

Input: present = [5,4,6,2,3], future = [8,5,4,3,5], budget = 10

Output: 6

Explanation: One possible way to maximize your profit is to: Buy the 0th, 3rd, and 4th stocks for a total of 5 + 2 + 3 = 10. Next year, sell all three stocks for a total of 8 + 3 + 5 = 16. The profit you made is 16 - 10 = 6. It can be shown that the maximum profit you can make is 6.

Example 2:

Input: present = [2,2,5], future = [3,4,10], budget = 6

Output: 5

Explanation: The only possible way to maximize your profit is to: Buy the 2nd stock, and make a profit of 10 - 5 = 5. It can be shown that the maximum profit you can make is 5.

Example 3:

Input: present = [3,3,12], future = [0,3,15], budget = 10

Output: 0

Explanation: One possible way to maximize your profit is to: Buy the 1st stock, and make a profit of 3 - 3 = 0. It can be shown that the maximum profit you can make is 0.

```
1
2  import numpy as np
3  from scipy.optimize import milp
4  from scipy.optimize import LinearConstraint,Bounds
5
6  class Solution:
7      def maximumProfit(self, present, future, budget):
8
9
10         present = np.array(present)
11         profit = np.array([f-c for c,f in zip(present,future)])
12         c = -1*profit
13         l = np.zeros(present.shape[0])
14         u = np.ones(present.shape[0])
15         b_u = budget
16         A = present
17         b_l = np.zeros_like(b_u)
18
19         constraints = [LinearConstraint(A, b_l,b_u)]
20
21         res = milp(c=c, constraints=constraints,
22                    integrality =np.ones_like(c),bounds=Bounds(l, u))
```

```
        if res.status ==0:
            v = res.fun

            if v - int(v) >0.5:
                return -1*math.ceil(v)
            else:
                return -1*int(v)
        else:
            return -1

    def maximumProfitDp(self, present , future, budget)
        dp = [0] * (budget+1)
        for p, f in zip(present, future):
            for j in range(budget, p-1, -1):
                dp[j] = max(dp[j], dp[j-p] + f-p)
        return dp[-1]

    def maximumProfitRec(self, present, future, budget):
        profit = [f-c for c,f in zip(present,future)]
        return self.maximumProfitRec(present,profit,budget,0)

    def _maximumProfitRec(self,present,profit,budget,idx):
        if budget <0:
            return -10**5
        if idx == len(present):
            return 0
        return max(self.maximumProfitRec(present,profit,budget-present[idx],idx+1)
                +profit[idx],self.maximumProfitRec(present,profit,budget,idx+1))
```