CS 577 # Assignment 7 – Dynamic Programming Fall 2022

Answer the questions in the boxes provided on the question sheets. If you run out of room for an answer, add a page to the end of the document.

Name: Kayley Scow Wisc id: kscow

Dynamic Programming

Do **NOT** write pseudocode when describing your dynamic programs. Rather give the Bellman Equation, describe the matrix, its axis and how to derive the desired solution from it.

1. *Kleinberg, Jon. Algorithm Design (p.313 q.2).*

   Suppose you are managing a consulting team and each week you have to choose one of two jobs for your team to undertake. The two jobs available to you each week are a low-stress job and a high-stress job.

   For week i, if you choose the low-stress job, you get paid ℓ_i dollars and, if you choose the high-stress job, you get paid h_i dollars. The difference with a high-stress job is that you can only schedule a high-stress job in week i if you have no job scheduled in week $i-1$.

   Given a sequence of n weeks, determine the schedule of maximum profit. The input is two sequences: $L := \langle \ell_1, \ell_2, \ldots, \ell_n \rangle$ and $H := \langle h_1, h_2, \ldots, h_n \rangle$ containing the (positive) value of the low and high jobs for each week. For Week 1, assume that you are able to schedule a high-stress job.

   (a) Show that the following algorithm does not correctly solve this problem.

Algorithm: JOBSEQUENCE

Input : The low (L) and high (H) stress jobs.
Output: The jobs to schedule for the n weeks
for *Each week i* **do**
  if $h_{i+1} > \ell_i + \ell_{i+1}$ **then**
    Output "Week i: no job"
    Output "Week i+1: high-stress job"
    Continue with week i+2
  else
    Output "Week i: low-stress job"
    Continue with week i+1
  end
end

2

L: <1, 6, 100>
H: <1, 10, 20>
pick 10

$L = \langle 1, 1, 1, 1, 1, 1 \rangle$

$H = \langle 5, 10, 15, 20, 25, 30 \rangle$ · $20+25 = 45$, instead of 60, which is less

$10+20+30 = 60$

In the case where Low is all lower than High, then we would go with the high option based on this algorithm. However, because this takes the first high job and skips over the other ones, we end up with a total of 45 instead of 60, which is not the highest value possible.

(b) Give an efficient algorithm that takes in the sequences L and H and outputs the greatest possible profit.

max Profit from week 1 to i denoted as m[i]

$$m[i] = \begin{cases} 0, \text{ if } i = 0 \text{ or } i = -1 \\ opt[i] = \max(\ell_i + opt[i-1], h_i + opt[i-2]) \end{cases}$$

Matrix is one dimension, as we have one degree of freedom for the profit and each cell represents the cumulative best value up to that point.

Solution will be in cell n, where n is the total number of days. The cells will be populated from 1 to n.

(c) Prove that your algorithm in part (c) is correct.

Base Case: m[0]. At m[0], or m[-1], we return 0 because with no weeks or negative weeks of jobs, which we only reach because we are decrementing by 2, we will return 0 as the value to the base case. We catch all the base cases, so we know that our algorithm terminates.

Inductive Step:

→ Inductive Hypothesis: When we get to week i, it is the optimal schedule for that i to k weeks

For the k+1 week, $\max(M[k] + \ell_{k+1}, M[k-1] + h_{i+1})$ will be selected

→ By strong induction, we know M[k] and M[k-1] are guaranteed to be maximums.

We only have two options, giving us ℓ_{k+1} or h_{k+1}. If we have $h_{k+1} > \ell_{k+1}$, $M[k-1] + h_{k+1} > M[k]$, and vice versa.

→ Therefore, our algorithm chooses the maximum between ℓ_{k+1} and h_{k+1}, giving us the best possible value.

2. *Kleinberg, Jon. Algorithm Design (p.315 q.4).*

Suppose you're running a small consulting company. You have clients in New York and clients in San Francisco. Each month you can be physically located in either New York or San Francisco, and the overall operating costs depend on the demands of your clients in a given month.

Given a sequence of n months, determine the work schedule that minimizes the operating costs, knowing that moving between locations from month i to month $i+1$ incurs a fixed moving cost of M. The input consists of two sequences N and S consisting of the operating costs when based in New York and San Francisco, respectively. For month 1, you can start in either city without a moving cost.

(a) Give an example of an instance where it is optimal to move at least 3 times. Explain where and why the optimal must move.

NY: <1000, 100, 1000, 100>

SF: <100, 1000, 100, 1000>

You would be moving four times, as the moving cost alternates depending on the month.

move when M + operating cost of another city is higher than current.

(b) Show that the following algorithm does not correctly solve this problem.

Algorithm: WorkLocSeq

Input : The NY (N) and SF (S) operating costs.
Output: The locations to work the n months
for *Each month i* **do**
  if $N_i < S_i$ **then**
    Output "Month i: NY"
  else
    Output "Month i: SF"
  end
end

This algorithm does not correctly solve the problem, as it does not account for M, which is the moving cost. If the costs of SF and NY are lower than the moving cost, it would make sense to not move, even if the costs in S or N change. For example, with M = 200, we have an example with our two arrays, where staying in N or S would be cheaper than having to move three times, which incurs the cost of moving twice, $400.

N: <1, 2, 1>

S: <2, 1, 2>

(c) Give an efficient algorithm that takes in the sequences N and S and outputs the value of the optimal solution.

- Matrix is one dimension, one df as we need to account for month
- overall operating cost from month 2 to i denoted as C[i]

$$C[i] \begin{cases} C[0] = 0 \\ C[1] = \min(N_i, S_i) \\ C[i] = \min(N_i + M, S_i) + C[i-1] \text{ if in SF} \\ C[i] = \min(N_i, S_i + M) + C[i-1] \text{ if in NY} \end{cases}$$

- Solution will be in cell n, where n is the total number of months. The cells would be populated from 2 to n.
- For schedule, just backtrack among Bellman equation

(d) Prove that your algorithm in part (c) is correct.

Base Case: C[0] = 0, returns 0 because at month 0, there is no cost needed to operate, as there are no operations going on. In the case of C[1], we choose between the minimum of N_i and S_i, since there is no cost incurred

Inductive Step:

→ Inductive Hypothesis: The cost at month i for operations is the cumulative optimal least cost for all i up to k months.

→ for the $k+1^{th}$ month, $\min(\min(N_{k+1}+M, S_{k+1})+C[k+0], \min(N_{k+1}, S_{k+1}+M)+C[k+0])$ will be selected

→ By Strong induction, we know that C[k] is guaranteed to be a minimum, as well as $C[N_k+M]$, $C[S_k]$, $C[N_k]$, and $C[S_k+M]$.

With these options, we are choosing between two options for minimums, which is the min of NY and SF, and in the case there is no operating cost or no month, we favor that in two.

→ Therefore, our algo chooses the minimum of all possible choices, giving us the optimal least cost.

3. *Kleinberg, Jon. Algorithm Design (p.333, q.26).*

Consider the following inventory problem. You are running a company that sells trucks and predictions tell you the quantity of sales to expect over the next n months. Let d_i denote the number of sales you expect in month i. We'll assume that all sales happen at the beginning of the month, and trucks that are not sold are stored until the beginning of the next month. You can store at most s trucks, and it costs c to store a single truck for a month. You receive shipments of trucks by placing orders for them, and there is a fixed ordering fee k each time you place an order (regardless of the number of trucks you order). You start out with no trucks. The problem is to design an algorithm that decides how to place orders so that you satisfy all the demands $\{d_i\}$, and minimize the costs. In summary:

- There are two parts to the cost: (1) storage cost of c for every truck on hand; and (2) ordering fees of k for every order placed.
- In each month, you need enough trucks to satisfy the demand d_i, but the number left over after satisfying the demand for the month should not exceed the inventory limit s.

(a) Give a recursive algorithm that takes in s, c, k, and the sequence $\{d_i\}$, and outputs the minimum cost. (The algorithm does not need to be efficient.)

It costs k to order trucks, and ct to store t trucks. if I have to order the trucks. I will order k+ct where ct=0, or t=0 0. either we will pay k or we pay ct, t_i is the trucks stored at mon n, and we store 0 trucks TS(n,0). We end up either storing t_i or 0 trucks depending on dc

(b) Give an algorithm in time that is polynomial in n and s for the same problem.

- Matrix is two dimensional, as we have to store [months, trucks]
- We have 2D array, first index being month, 2nd index being stored trucks.

$TS(i,t_i) = \min\{k + TS(i-1,0),\ c(t_i+d_i) + TS(i-1, t_i+d_i)\}$

→ where d_i is the cumulative cost per truck

Memoize

$$M[\text{mx}]\begin{cases} M[0,0] = 0 \\ M[i,t_i] = \min\{k + M[i-1,0],\ c(t_i+d_i) + M[i-1, t_i+d_i]\} \end{cases}$$

- Solution in M[n,0] where n is number of months, and 0 trucks are stored.
- For schedule, backtrack along Bellman equation
- Runtime: $n \cdot s$ cells, time/cell $= S \rightarrow O(ns^2)$ runtime

(c) Prove that your algorithm in part (b) is correct.

Base Case: $M[0,0]=0$, returns 0 because with 0 months and 0 trucks left, we incur no costs at all.

Inductive step

*Inductive Hypothesis: The cost at month i with t_i trucks left is the optimal least cost for the current i and t_i, up to k months.

→ for the $k+1$ month, $\min\{B+M[k,0], c(t_{k+1}+d_{k+1})+M[k,t_{k+1}+d_{k+1}]\}$ will be our cost

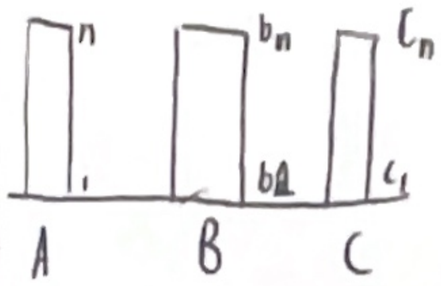
→ By strong induction, we know that $[k,0]$ gives us the minimum cost ($M[k,t_{k+1}+d_{k+1}$ as well) at month k, with the designated amount of trucks left over. We know that they will be minimums.

We only have two options, to either decide between ordering, with the cost of K, in addition to the 0 trucks left over, or if we have trucks left over, the price to store them. With these two options only, we have crossed over all possibilities.

→ Therefore, our algo chooses the most optimal, minimum cost of the two possible choices, giving us the overall least, minimum, cost possible.

4. Alice and Bob are playing another coin game. This time, there are three stacks of n coins: A, B, C. Starting with Alice, each player takes turns taking a coin from the top of a stack – they may choose any nonempty stack, but they must only take the top coin in that stack. The coins have different values. From bottom to top, the coins in stack A have values $a_1, \ldots, a_n$. Similarly, the coins in stack B have values $b_1, \ldots, b_n$, and the coins in stack C have values $c_1, \ldots, c_n$. Both players try to play optimally in order to maximize the total value of their coins.

   (a) Give an algorithm that takes the sequences $a_1, \ldots, a_n$, $b_1, \ldots, b_n$, $c_1, \ldots, c_n$, and outputs the maximum total value of coins that Alice can take. The runtime should be polynomial in n.



$$OPT_A(i,j,k) = \begin{cases} c_i & i=j=k \\ \max \begin{cases} c_i + OPT_B(i-1,j,k) \\ c_j + OPT_B(i,j-1,k) \\ c_k + OPT_B(i,j,k-1) \end{cases} & i \neq j \neq k \end{cases}$$

$$OPT_B(i,j,k) = \max \begin{cases} 0 & i=j=k \\ \min \begin{cases} c_i + OPT_A(i-1,j,k) \\ c_j + OPT_A(i,j-1,k) \\ c_k + OPT_A(i,j,k-1) \end{cases} & i \neq j \neq k \end{cases}$$

Matrix is three dimensions, as we have three degrees of freedom for the three stacks of coins. each cell represents the options on the kth term.

Populating starting from n to 1.

Solution in [1,n,n]

Runtime is $O(n^3)$, as we have a 3D matrix and 3df in this problem.

   (b) Prove the correctness of your algorithm in part (a).

Base case: $OPT[i,i,i] = c[i]$ for all I. as this is the base case and when $i=j=k$ it means there are no more coins left.

Inductive step

→ IH: Whenever get to round $i/j/k$, it is optimal move for Alice to take as it maximizes her coins and minimizes her opponents

→ By strong induction, we know that any call to OPT_A maximizes chances for Alice, and OPT_B minimizes the chances for Bob.

→ Therefore, our algo chooses the best optimal option for Alice at a given i, j, and k.