MSSC 6000 - Homework 6

Henri Medeiros Dos Reis

May 5, 2023

1.

Start by sorting the jobs by deadline. (Let's keep the number of original job inside parenthesis)

job#	1(3)	2(5)	3(4)	4(1)	5(2)
Duration	1	2	1	2	5
Deadline	2	2	3	5	8
Profit	1	5	3	3	2

Let sol[i][j] denote solution with i jobs and j units of time (latest deadline available).

Then the recurrence is

$$\text{sol[i][j]} = \max(\text{sol[i-1][j]}, \text{sol[i-1][j - duration of job i]})$$

This recurrence states that the solution with i items in j units of time is the maximum solution between taking the item and reducing the number of available time slots, and not taking the item and keeping the units of time. Keeping in mind the the maximum number of time slots (units of time) is the latest deadline available from the jobs we have left.

In order to derive this recurrence, I first started thinking about how to construct the partial solutions, which at each job we are thinking about weather to take it or not take it. So each partial solution can either take the job or not take it. From there I tried to see what were the consequences of taking that job, which is to decrease the remaining available time. Also, we should always check if the amount of time left is the same as the latest deadline from the jobs we have not analyzed yet.

The solution by hand:

latest dead line available = 8 = j

sol[5][8] = max(sol[4][8], 2 + sol[4][3])

sol[4][8] => sol[4][5] since the latest dead line of the available jobs is 5

sol[4][5] = max(sol[3][5], 3 + sol[3][4])

sol[3][5] => sol[3][3] latest available deadline = 3

sol[3][3] = max(sol[2][3], 3 + sol[2][2])

sol[2][3] => sol[2][2] latest available deadline = 2

sol[2][2] = max(sol[1][2], 5 + sol[1][0])

sol[1][2] = max(sol[0][2], 1 + sol[0][1]) = 1

sol[2][2] = max(1, 5) = 5 ← take job 2

sol[3][3] = max(2, 3 + 5) = 8 ← take job 3

sol[4][5] = max(8, 3 + sol[3][4])

sol[3][4] => sol[3][3] latest available deadline = 2

sol[4][5] = max(8, 11) = 11 ← take job 4

sol[5][8] = max(11, 2 + sol[4][3])

sol[4][3] = max(sol[3][3], 3 + sol[3][1])

sol[3][1] = max(sol[2][1], 3 + sol[2][0])

sol[2][1] = max(sol[1][1], 5 + sol[1][-1]) = sol[1][1] ← cant

sol[1][1] = max(sol[0][1], 1 + sol[0][0]) = 1

sol[2][1] = 1

sol[3][1] = max(1, 3) = 3

sol[4][3] = max(8, 3 + 3) = 8

sol[5][8] = max(11, 10) = 11 ← don't take job 5

Then, we can see that the optimal solution consists in taking jobs 2,3, and 4 from the sorted jobs. Those correspond to taking jobs 1,4, and 5 from the unsorted list of jobs. With a score of 11.

2.

In order to handle the boundary conditions, every time a particle is going to move, right before it moves I check weather that move is allowed or not. In order to check it, I am using two functions, one that checks if it is inside the bounds, and one to check if it meets all the constraints. If that move meets all the requirements, then the particle can move there, if not, then we do not move and let the inertia take care of the boundary problem, since next time it times to move its speed is going to be smaller.

For the choice of parameters, I chose α just to be the standard 0.9, β to be one, so that the particle is always moving to its personal best, and γ to be 1.2, so that the particle prioritizes the global best. The number of particle chosen was 1000, I tested with more particles, but the results did not improve by a lot, while the run time increased significantly.