Homework 4

MSSC 6000 — Scientific Computing — Spring 2023

Due Monday, April 3, on D2L, by 11:59pm

Read this first! Some of these questions instruct you to devise an algorithm to solve a problem and explain it to me, but not to write any code. In this assignment, approach describing your algorithm like writing a report to a boss or an advisor. Your goal is for me to read your answer and have a full understanding of how it works. That doesn't mean it should be long! It just means you should think carefully about how to write a clear explanation. If you think examples or pictures help, use them! If I don't understand your algorithm, I can't give full credit. (But let me repeat once more: this does not imply that your answers need to be really long.)

The emphasis of this assignment is not just on devising algorithms, but also describing them, testing them, validating them, possibly visualizing them, etc.

1. In Homework 2, we considered the Job Scheduling Problem. First, I will restate it.

As a graduate student, you have a very busy day ahead of you. You have n tasks that you could do. Each one has: an amount of time it takes to do (duration, in minutes), a deadline (given in number of minutes from the start of the day, which we'll call 0), and an amount of profit you get for completing it (in dollars, or gold stars, or slices of pizza, or whatever we pay graduate students with these days). You can't do jobs that would finish after the deadline. Your goal is to maximize profit.

To make sure you understand the problem, here is an example:

Job #	1	2	3	4	5
Duration	2	5	1	1	2
Deadline	5	8	2	3	2
Profit	3	2	1	3	5

Job #1, for example, takes 2 minutes to complete, and its deadline is 5 minutes after the start of the day. So you can start it right away (at minute 0), or at minute 1, 2, or 3, but you can't start it at minute 4, because then you would miss the deadline.

One (non-optimal) solution for this example would be to do jobs 3, 4, and 2, in that order. In this solution, job 3 runs from time 0 to time 1, job 4 runs from time 1 to time 2, and job 2 runs from time 2 to time 7. The total profit is $1 + 3 + 2 = 6$. The optimal solution in this case is to do jobs 5, 4, and 1, in that order. (At least, I think so, I only did it in my head.)

On this assignment, you will design a branch-and-bound algorithm to solve it. **I have provided code, attached as separate file on D2L, that does all parts of it, including the recursion, except the upper bound.**

- First, read through the code carefully so you understand it. I have used an object-oriented approach, so there are Job objects and Schedule objects.
- I have provided a function to compute the brute force solution, so you can compare your answers (for small numbers of jobs, at least) to make sure they are correct. Try running this function a few times for practice.

- I have provided a list called "all_jobs" with 50 jobs that will be used for testing the speed of your algorithm.
- You will try to think of a useful upper bound, and then you will implement it in the appropriate spot of the code.

Submit a written document with answers to (a), (b), and (d) and submit your code to (c).

(a) Based on the code I provided, describe the branching procedure.

(b) Come up with an upper bound for a partially-determined schedule, and explain it clearly. Providing an example may help me to understand your bound.

(c) Implement your upper bound into the code I provided, and submit it along with your written document. See below for information about how I will evaluate your bound.

(d) There is one thing we discussed in class that helps branch-and-bound algorithms get a head start that I am not doing in my code. What is it?

Information about evaluation. Better upper bounds will receive better scores on this assignment. Part (b) will be worth four points total. In the table below, I show you speeds for brute force, backtracking alone (no upper bound), three different upper bounds I came up with, and a technique called dynamic programming that we may cover later[1]. Runtimes can vary tremendously for different sets of jobs, even of the same size. In this case n means I ran with the first n jobs of the "all_jobs" list defined in the code I provided (except for $n = 2000$ which was randomly chosen). Entries that are blank mean the run took so long I gave up.

n	brute force	backtracking	UB1	UB2	UB3	dyn. prog.
10	10.9	0	0	0	0	0
20		0.5	0.2	0.08	0.1	0.0009
25		10.7	5.3	1.7	1.3	0.001
27		76	32	9	6	0.001
30			260	17	6	0.0015
35				42	22	0.002
40				72	23	0.0024
45				87	26	0.003
50				101	29	0.0032
2000						5

- If your upper bound is equivalent to backtracking (i.e., never prunes anything), you will receive one point for (b).
- If your upper bound is roughly as good as my UB1 (can do the first 27 jobs of "all_jobs" in about a minute on my computer), you will received three points for (b).
- If your upper bound is roughly as good as my UB2 (can do the first 50 jobs in under 3 minutes), you will receive a for full points for (b).
- If your upper bound is as good as my UB3 (can do 50 jobs in under a minute), then you will receive bonus points for (b).

If any of this is confusing, don't hesitate to email me and ask about it!

[1]It does not apply to many problems, but when it does it can be very fast, like here.