Describe the algorithm you used to generate your outputs. Why do you think it is a good approach?

We first sort the tasks using a heuristic and then try to insert each task at the latest time step where the task would not be late, shifting each subsequent task earlier in the schedule if there was not enough space. Our heuristic was a linear function of the deadline and benefit rate where the weights were randomly chosen. Randomly choosing the weights 1000 times, we selected the schedule with the highest total benefit.

What other approaches did you try? How did they perform?

We started initially with creating buckets of tasks and brute forcing each bucket, then brute forcing the buckets together. This worked alright, but it relied on brute force to solve and thus was quite slow. Next, we investigated certain reductions to graphs, but there was no clear way to reduce a task schedule to a graph. The idea was to connect tasks that could be done after another, and look for a longest path, but the reduction proved inefficient. We also tried starting with a random sequence of tasks then using a heuristic to swap out tasks to attempt to increase profit, but this was too inconsistent.

This left us to try a greedy approach. We started with a basic greedy approach based on the task deadline in order to get our solver going. Although it yielded a valid correct solution, it was far from optimal. We then decided to stick with a greedy approach, but change our greedy algorithm to implement a linear heuristic model that would take in factors such as deadline, priority, profit, etc. We tried various weights for the factors by hand to check for which would yield the best profit on a subset of inputs.

What computational resources did you use? (e.g. AWS, instructional machines, etc.)

Microsoft Azure with free student credit