CMSC 27200 Project Reflection

Jamar Sullivan, Jason Jia

May 2022

1 The Algorithm

1.1 How does your final algorithm work?

Our final algorithm is the greedy algorithm with two forms of partial brute force, $Greedier(n, \alpha, \beta, m)$.

- n: The brute force portion gets all feasible partial sequences of up to $n + 1$ nodes. Greedy is then run on each of these feasible sequences.
- α: The extra weight on time taken to go from current node to next node, $d_{i,j}$.
- β: The weight on time taken to go from next node back to the origin, $d_{j,O}$.
- m: The top m next nodes based on the greedy heuristic (or the number of feasible next nodes, whichever is lower) are all considered for evaluation.

Our pre-processing is as follows:

- We remove nodes with 0 utility.
- We also create a graph with the weights of each edge as the time taken to travel between the two nodes (rounded up to the nearest integer).
- Initialize a list of feasible nodes F containing all nodes.
- Initialize a queue containing all partial feasible sequences of length 1.
- Initialize best score = 0 and best sequence = [ ].

Our main loop for the brute force portion is as follows:

- While the queue is non-empty, pop the left-most element, to get a partial feasible sequence that ends at i, a list of potential feasible nodes, current time and current score. If the partial feasible sequence has length $> m$, add the popped element back to the queue and break out from the loop.
- We remove infeasible nodes from F_t, defined as nodes where $S_{i,j,t}$, the sum of traveling time $d_{i,j}$ (from Jonathan's current location i), waiting time $w_{j,t}$, time spent at the attraction t_j, and the traveling time $d_{j,O}$ (from the node j to the origin) exceeds the remaining time available (at the current time t).

- If there are no feasible nodes left, the sequence ends here. If this sequence has a higher score than the current best score, the best score and sequence is updated.
- If there is only 1 feasible node left, then the sequence can only go on to that node, and then end there. So if this updated sequence has a higher score than the current best score, the best score and sequence is updated.
- If there are at least 2 feasible nodes left:
  - For every feasible node j, we update the sequence, remaining candidate feasible nodes (without j), score and time accordingly, and insert an entry into the queue.

Our main loop for greedy is as follows:

- While the queue is non-empty, pop the right-most element, to get a partial feasible sequence that ends at i, a list of potential feasible nodes, current time and current score.
- We remove infeasible nodes from F_t.
- If there are no feasible nodes left, the sequence ends here. If this sequence has a higher score than the current best score, the best score and sequence is updated.
- If there is only 1 feasible node left, then the sequence can only go on to that node, and then end there. So if this updated sequence has a higher score than the current best score, the best score and sequence is updated.
- If there are at least 2 feasible nodes left, where n_f is the number of feasible nodes, we compute the heuristic to choose the best $m' = \min(m, n_f)$ next node(s):
  - We define time $T_{i,j,t} = (1+\alpha)d_{i,j} + w_{j,t} + t_j + \beta d_{j,O}$.
  - Heuristic: Given that Jonathan is at point i at time t, the score for choosing node j as the next feasible node is: $S(i,j,t) = \alpha \frac{u_j}{T_{i,j,t}} + \frac{1}{|H|}\sum_{h \in H} \frac{u_h}{T_{j,h,(t+T_{i,j,t})}}$, where H is the set of all feasible nodes given that Jonathan is at node j at time $t + T_{i,j,t}$, and has already visited a certain sequence of nodes.
  - We choose the m' best nodes. For each feasible next node j, we update the sequence, remaining candidate feasible nodes (without j), score and time accordingly, and insert an entry into the queue.

1.2 Why did you settle on it, and how well do you think it performed?

- We settled on the greedy algorithm because it provided reasonably good outputs to all inputs within a short amount of time. It captured the idea that a node is more likely to be the optimal node if it has high average utility over time, where time includes the time required to travel from Jonathan's current location, wait for the attraction to open (if needed), finish the attraction, and go back to the origin.
- It also captured the idea that a node j is more likely to be the optimal node if choosing this node now still allows Jonathan to choose other nodes with high average utility over time. We do this by averaging the weighted utilities of nodes that are further accessible from node j.

- Another major reason is that the greedy algorithm can be tweaked easily in response to challenging inputs, and this ability to generate iterative improvement makes it attractive. This proved to be a good decision: our addition of α, β, and most importantly m, significantly improved performance compared to the original $Greedy(n)$.
- This is in contrast to brute force which works perfectly, but only if it finished its computation - and many did not. Partial results from brute force didn't really help either, as they tend to be significantly worse than outputs from greedy.
- We couldn't use the genetic algorithm because it didn't converge to a solution in the end.
- When we made the decision, we found it very attractive that the greedy algorithm is able to incrementally brute-force our way to better solutions, provided we had sufficient time.
- Looking back, this is only somewhat true. There were many inputs with improved scores, but these were also the ones that brute-force could solve relatively quickly. The ones where brute-force takes a long time, also tend to be the ones where (a) greedy's scores don't improve quickly with n, and where (b) greedy takes infeasibly long even for modest values of n, such as 2. However, this was mitigated with a choice of $n = 0, m \geq 2$.
- On balance, we therefore think that it performed slightly below expectations, but the performance was definitely not bad. We made up for it with manual search and brute force.

1.3 What ideas did you have after the design doc that you ended up abandoning, and why?

- We wanted to loop over many different values of (α, β) in the algorithm, and also add parameters to the other components of $T_{i,j,t}$ in the heuristic, but we didn't have enough time.

2 Running your code

2.1 What steps would we have to take to run your code on the inputs and get the outputs you've submitted? If different inputs were run on different algorithms, please describe how and why.

- The solutions were initialized and replaced with better solutions after each run. As we didn't precisely keep track of which configurations produced which result, and there were many errors that cancelled the program (e.g. exceeded memory, took up too much time), we provide the steps we took to the best of our knowledge.
- For a first run, if any algorithm doesn't return a solution after 15 minutes, we cancel and run on the next input. Subsequent runs depend on which outputs underperformed relative to the leaderboards.
- If a solution had the same score as the max score in the leaderboards, add it to a list of solved inputs. Future code runs will skip all solved inputs to save time.
- Greedy: Run *greedier* on all the inputs.

- The main configurations (m, α, β, n) run (or at least attempted) were:
  - $(0,0,0,1), (1,0,0,1), (2,0,0,1)$
  - $(0,0.3,0,1), (0,0.5,0,1), (0,0.7,0,1)$
  - $(0,0,0.3,1), (0,0,0.5,1), (0,0,0.7,1)$
  - $(0,0.3,0.3,1), (0,0.5,0.5,1)$
  - $(0,0,0,2), (0,0.3,0.3,2), (0,0.5,0.5,2)$
- Brute force: Run $brute\_force$ on all the inputs, from small to medium to large.
- Manual: Manually solve the rest.
- We did this to improve our total score.

2.2 Did you do anything outside of running the code (like manually solving inputs or tweaking outputs)? If so, what did you do, which outputs are affected, and why did you choose to do so?

- We tried to manually solve inputs if the outputs significantly underperformed, which we defined as scoring > 20% lower than the max score in the leaderboards.
- This happened after we wrote $Greedy(n)$, but before we wrote $Greedier(n, \alpha, \beta, m)$. We did not test these inputs against *Greedier* since time was tight and we already had the best scores.
- The inputs we solved manually are:
  - Dijkstra'sDisciples: large1
  - RMG: medium2, medium3, large3
  - thecoolguys: medium2, medium3, large2
  - [noname]: all 9 inputs (because we created these)
  - <the group with the chinese name>: small2, large1
  - SuperSickFireDubDestroyers: large3
- We did this to improve our total score.

3 External Resource Usage

3.1 What outside computational tools/services/resources did you use, and to what extent did you use them?

- Greedy algorithm: Python
- Genetic algorithm: Javascript
- Resources: Local machines

4 Team Reflection

4.1 How was work distributed among the team?

- Both: Inputs, Design document, Reflection
- Both: Brute force algorithm, Manually solve inputs
- Jamar: Genetic algorithm
- Jason: Greedy algorithm

4.2 What do you think you did well as a team?

- We communicated regularly to reduce procrastination.
- Both of us were involved in every stage of the project, and took time to come to a consensus before moving on to the next stage.
- Jamar dropped out of the course half-way through the project, but we still worked to complete the project together till the end.

4.3 What surprised you as you worked on the project?

- We expected brute-force to completely fail and implemented it only for the sake of checking our code, but it worked better than expected. This is likely because a considerable number of inputs have rather strict start and end times for nodes, so the total number of possibilities was not exponential in N, and brute-force was able to generate a solution. In a similar vein, we were surprised that it was possible and reasonably fast to solve inputs manually, especially for those where our algorithm significantly underperformed.
- We were surprised when our initial idea of greedy with partial brute force didn't work quite as well as expected, because of the infeasible runtime. It was only after a while that we realized we could instead choose multiple next nodes based on the greedy heuristic, which was actually also a form of greedy with partial brute force, but faster and hence performed better.
- We were surprised when other teams submitted a higher score than what we had conceived when coming up with our own inputs. As a result, we had to manually re-solve a few of them.

4.4 What would you have done differently if you could do the same project again?

- We would have started sooner to give us more time to work on different versions of the greedy algorithm. We would also have prioritized implementing and executing *Greedier*, instead of spending time on brute force or *Greedy*.

4.5 What would you change about the project itself, if you could?

- It would be great if there was a row at the top/bottom of the leaderboards and points, showing the max score and points, respectively, for each input.