Jordan Deagan

Joint Questions

a) List of nodes expanded by depth-first:
s
h
k
c
a
b
d
m
e
n
g

b)

i) h is an admissible heuristic, as at any node, the heuristic assigned is always less than the shortest path to g.

ii) The nodes expanded by the greedy best first search is as follows:
s
h
k
c
a
b
d
m
g

iii) Given the A* algorithm with the Manhattan Distance heuristic, the nodes would be expanded in the following order:
s
h
k
f
c
p
q
r
t
g

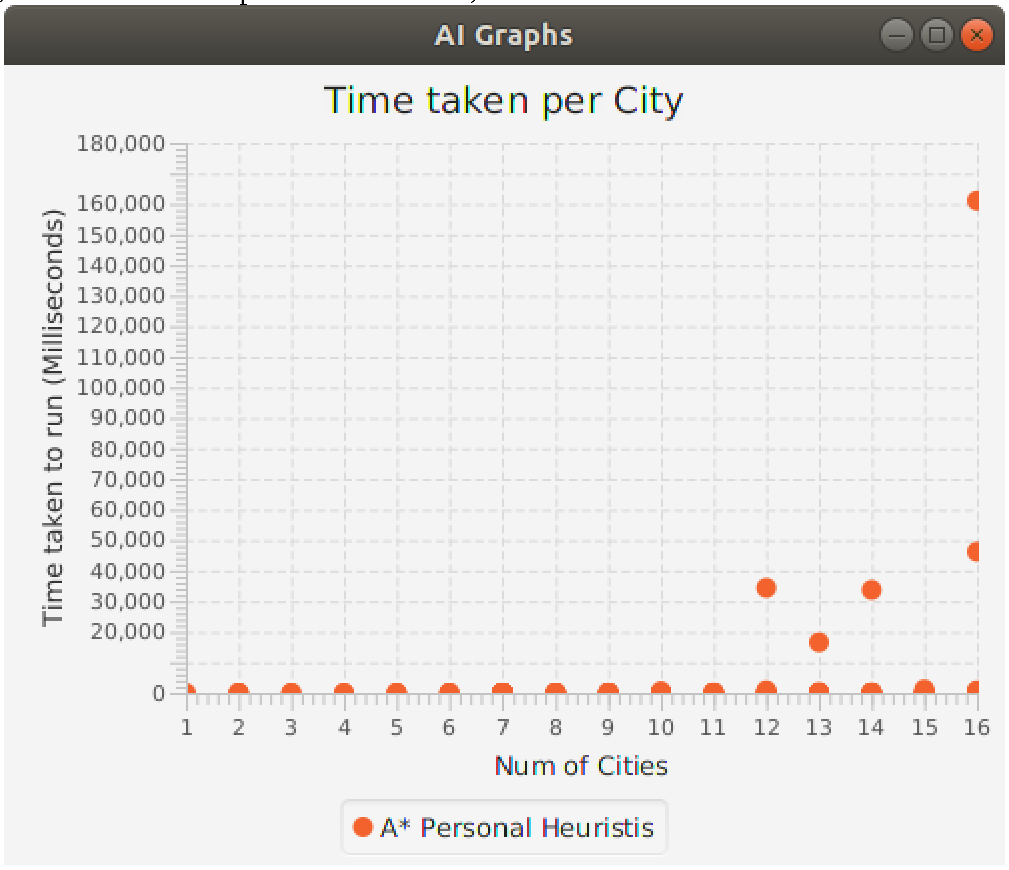
Jordan Deagan

A) A* Search

a) To represent the TSP problem for A*, I created a Vector2 class to track x and y coordinates, a City class to keep track of a city's name and coordinates in the form of a Vector2, and a Node class that holds a pointer to it's parent, a reference to all cities in the scenario, the city it represents, the cost to get there, references to it's children, and it's heuristic and total value. In my A* class, titled A_star, I read the instance file, creating a list of cities for reference, then created an initial city A Node. From there, I had a method to check the frontier for the best cheapest Node, and another method that moved to it, expanded it, and checked if it had traveled through all cities and returned to A. In my TSP class, I called all of the appropriate methods from the main method.

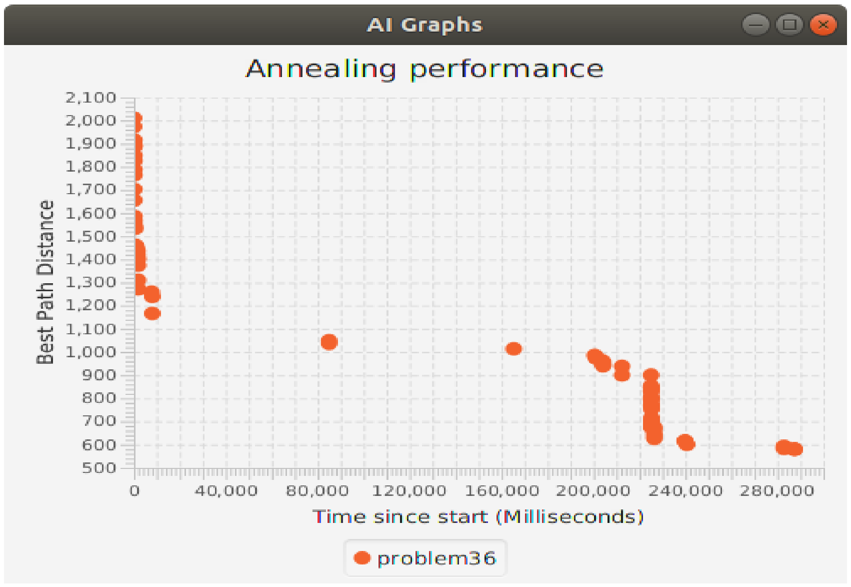
b) My heuristic goes to every city not passed through, starting at the initial city represented by the node and progressing through all of their successors as generated to discover the shortest path from each city to any other. When it finds the shortest path for a city, it records the distance and lets the other city on the path that it was visited by the first city. When the next city checks, it will look at all remaining cities that have not alerted it to having visited it. This means if the shortest path for city F is to H, when H looks for it's shortest path, it will not look towards F as a possible path, as that path was already used.

c)Based on the scatter plot of my Heuristic's performance, I would assume that it would either run immediately, or it would take upwards of an hour, at least.



Jordan Deagan

B) Local Search
a) For my local search program, I have a method that creates a random path out of Node objects starting with A, randomly placing all other cities, and returning to A. After this, until a set amount of time has passed or it has been 1000 cycles since the path has improved in distance, the method will randomly path two cities that are not A and swap them. If the new path has a shorter distance than the current path, it is accepted, otherwise it is left to a random chance based on the temp value, which gets smaller after 1/3, 1/2, 2/3, and 3/4 of the allotted time has passed. The method remember order and distance of the shortest path, and the nodes of the current path.

b) I did not alter the timing for lowering the temp, but I altered the initial, final temp, as well as how much the temp is lowered by at the given times. My initial temp settings started at .75, lowered to .5 at 1/3 time, .25 at 1/2 time, .10 at 2/3 time, and .01 at 3/4 time. To create different schedules, I create one that started at a higher temp and lowered to a lower temp more gradually, and one that was closer in range, but lowered quicker. My second schedule had the temp values at 1.00, .66, .33, .15, and 0 respectively, and my third schedule had values .8, .4, .2, .1, and .5 respectively. When ran on the same instances, with the same starting path, my initial schedule had the best performance. I believe this is because my second schedule, while changing at a similar rate, allows for too much freedom at the beginning, and the third schedule reduces the temp in a more linear style, which is not ideal for trapping the agent. For future research, a lower starting point, between .8 and .7, with a more exponential change in temp would be best. Higher starting temps will not find a good path in time, and a linear shift will not create a rapid descent agent in a timely manner, and might let the agent escape from the global minima.
c)


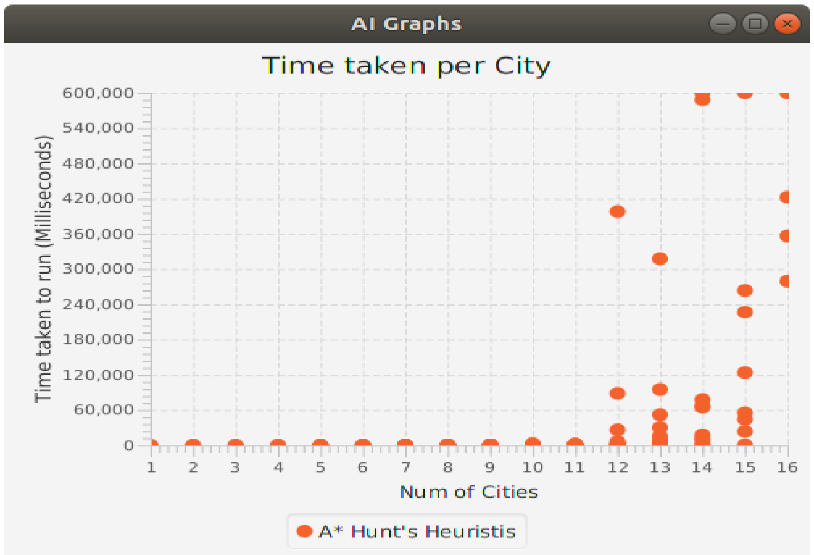
d) Simulated annealing is complete in that it starts with a potential solution, meaning that at any point, the process could be stopped and the returned solution could be used.
e) It is optimal if given a good annealing schedule. If the schedule is not good, there is no guarantee that the agent will get close to or stay near the global max or min. With a good annealing schedule, the agent will find the global min or max and then get trapped there, only capable of moving towards the optimal solution.

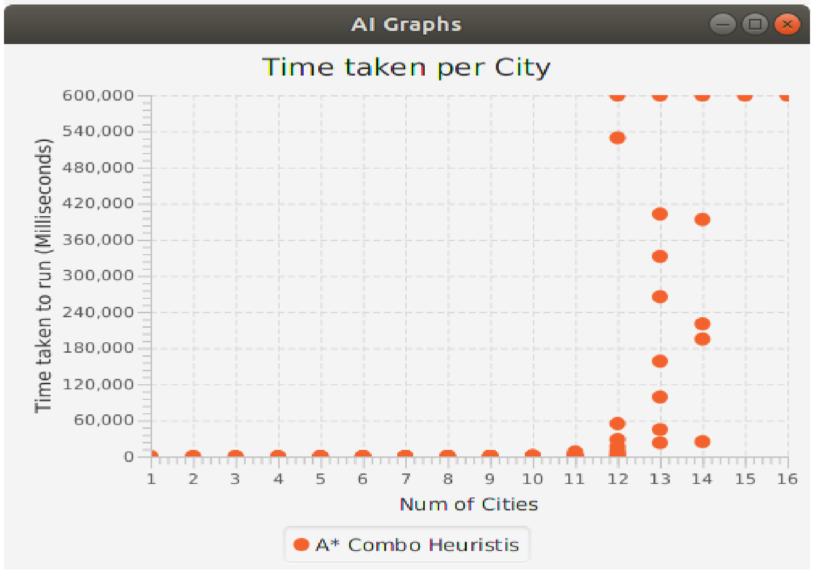
Jordan Deagan

C) Heuristic Hunt

I worked with Hunt Sparra on this section

The heuristic we came up with together is to find the furthest unexplored city from the current city, and the closest unexplored city to A, and add those distances together. This heuristic is admissible, but was less accurate than my original heuristic, which meant it took longer to find the optimal solution than mine took. On the other hand, it is much simpler to implement and the actual heuristic itself is significantly quicker, as it only runs through n cities, where n is the number of unexplored cities, compared to my heuristic that runs through n^n cities. At a significantly higher number of cities, this simplicity might prove more advantageous than my accuracy, as my heuristic will start to take too long to feasibly run, but at the number of cities presented, my accuracy won out. In running both of these heuristics, I stopped the program if it started to run longer than 10 minutes, so actual time may vary more than shown.


AI Graphs
Time taken per City
Time taken to run (Milliseconds)
Num of Cities
● A* Hunt's Heuristis


AI Graphs
Time taken per City
Time taken to run (Milliseconds)
Num of Cities
● A* Combo Heuristis