Grace Henke

CS 4300

Fall 2022

How To Efficiently Move a Hockey Team

Introduction

Problem Description: There is a total of 32 different hockey teams in the NHL, and per the official rules of hockey they are required to at least play each team in their league three to four times throughout the season. To simplify the problem, I determined that I wanted to create a route that would allow each team to play two home games, and then the rest of the other teams in their division. In each division there are 8 teams, meaning the total number of possible routes is 362,880. While this seems like a small number, to search for the most efficient route in all four divisions, one must look at 1,451,520 routes. So, since their tons of possibilities, it would be reasonably to introduce some different search techniques to find the most efficient route efficiently.

Searching Methods: To search for a solution to this problem, I modeled it after a classic computer programming problem which is the Traveling Salesman Problem. Each of the cities were nodes, and the edges were weighted using the distance between the cities. I compiled all the distances into an excel spreadsheets that would be loaded into the program allowing the user to control which path is generated. I also split each league into four different networks, to generate an optimal path for each division. I leveraged the nx network python module to make storing the nodes and weights a lot simpler.

The algorithms I used to search for the most efficient path were simple hill climbing and random restart with hill climbing. The simple hill climbing algorithm would generate a set of "neighbor" nodes to a randomly generated path. Then it would iterate through all the neighbors, determine which one had the best possible cost. After it went through all the neighbors, it would spit out the path with the best possible path. This algorithm is built on the principle that the best possible path exists in a neighbor node close to the randomly generated path. For the random restart with hill climbing, it works like the hill climbing algorithm, except the path is randomly generated before each pass of the simple hill climbing. This allows the search agent to be more exposed to different combinations of the possible paths. I wanted to compare the efficiency of the algorithms against each other.

Program Instances: A variety of instances were generated. For each division, 500 simple hill climbing and 500 random restart instances were generated. On top of that, each city was manual set as the starting point, allowing for more diverse coverage of the paths the algorithm attempted to generate. There was a total of 8,000 data points for each division created (32,000)

PEAS ASSESSMENT

Summary: The AI will find the best path to take for a hockey tour. It will find optimal paths for all leagues in the NHL. (a schedule will be outlined, and then optimal paths will be created)

Percepts: current location of the agent, the current path cost, and the whole network, node neighbors

Potential Actions: Pick a neighbor to add to the path (the design is cyclic, so the agent can pick any city to "move" to. (This was implemented through the generate neighbors' function, picking random cities to change/manipulate)

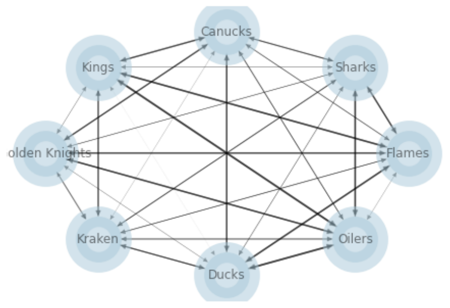
The Environment:

> **Observability**: Fully observable. The agent knows the locations of all the nodes/cities in the network. The agent also knows the starting and ending cities. Lastly, the agent also knows the distance between each city.
>
> **Uncertainty**: Deterministic. Movements are exact.
>
> **Duration**: Sequential. Actions need to happen in order. Since the algorithm is trying to find the lowest cost path, it needs to know which actions led to the destination.
>
> **Stability**: Static. While the league information can be changed so that it can compute different shortest paths, the information between runs will stay the same.
>
> **Granularity**: Discrete. All values are discrete. The distances between the cities are exact.
>
> **Participants**: Single Agent.
>
> **Knowledge**: Known. All distances have been added to an excel sheet and added to the   program.

Performance Measures:

The path will be evaluated with the associated cost. The highest cost (every city visited) will be compared to the path that was found.

Pacific Results:



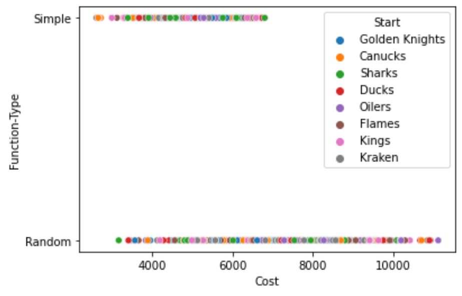
	Simple Hill Climbing	Random Restart
Mean	4965	7329
STD	790	1423
MIN	1931	2878
MAX	7194	7194

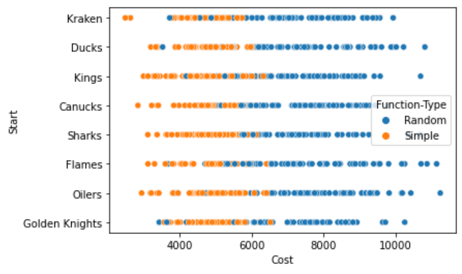
(*The numbers are the cost (distance) of the path, rounded to the nearest whole number)

Simple Best Path: ['Flames', 'Oilers', 'Flames', 'Canucks', 'Kraken', 'Golden Knights', 'Kings', 'Ducks', 'Sharks']

Random Restart: ['Kraken', 'Flames', 'Oilers', 'Canucks', 'Kraken', 'Golden Knights', 'Kings', 'Sharks', 'Ducks']

Graphs:

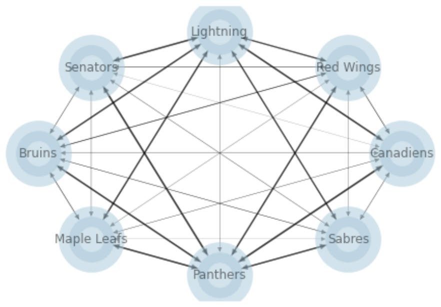




(These graphs were generated using a sample from each, so that the graphs weren't overpopulated with data)

Discussion: Based on the results, it looks like the simple hill climbing generated the best cost path for the pacific division. It looks like the simple hill climbing also generated the most paths with a lower cost than the random restart algorithm.

Atlantic Results:



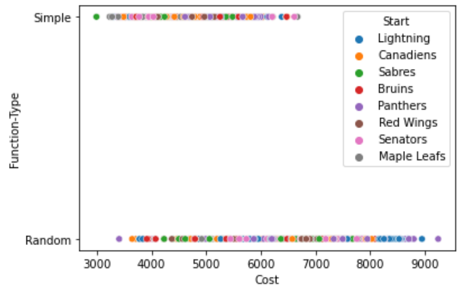
	Simple Hill Climbing	Random Restart
Mean	4679	6526
STD	676	1153
MIN	2931	3076
MAX	6791	9244

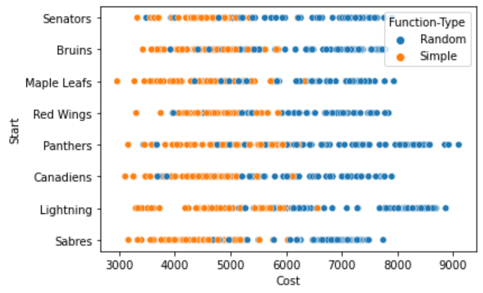
(*The numbers are the cost (distance) of the path, rounded to the nearest whole number)

Simple Best Path: ['Maple Leafs', 'Red Wings', 'Sabres', 'Maple Leafs', 'Senators', 'Canadiens', 'Bruins', 'Lightning', 'Panthers']

Random Restart: ['Lightning', 'Panthers', 'Lightning', 'Red Wings', 'Maple Leafs', 'Sabres', 'Bruins', 'Senators', 'Canadiens']

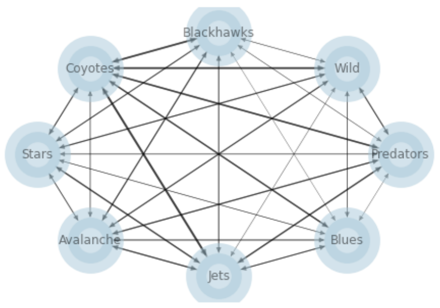
Graphs:





Discussion: Again, the simple hill climbing generated the best path. This however does a great job of showing the diversity of the data that a random restart algorithm generates.

Central Results:



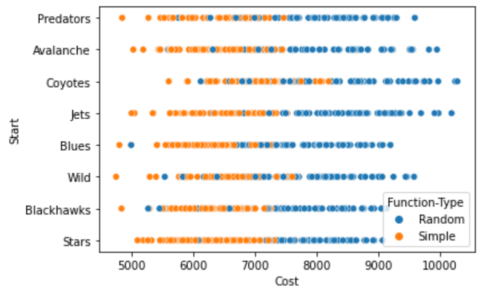
	Simple Hill Climbing	**Random Restart**
Mean	6433	7918
STD	629	936
MIN	4295	4673
MAX	8291	10800

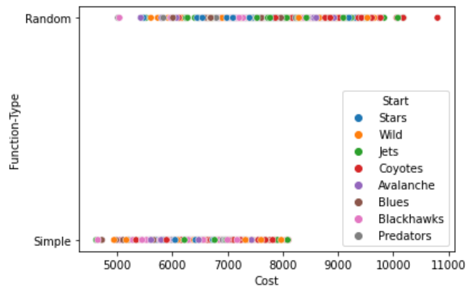
(*The numbers are the cost (distance) of the path, rounded to the nearest whole number)

Simple Best Path: ['Wild', 'Jets', 'Wild', 'Blackhawks', 'Predators', 'Blues', 'Stars', 'Coyotes', 'Avalanche']

Random Restart: ['Blues', 'Predators', 'Blues', 'Blackhawks', 'Wild', 'Jets', 'Avalanche', 'Coyotes', 'Stars']

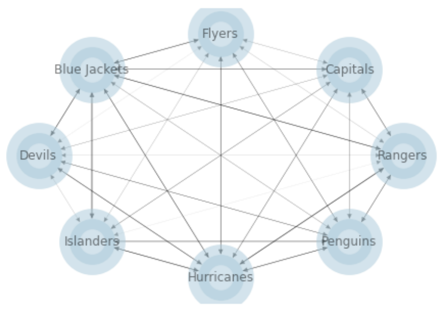
Graphs:





Discussion: The simple hill climbing, and random restart generated good paths. While they are close to each other, the simple hill climbing algorithm again generated the best path

Metropolitan Results:



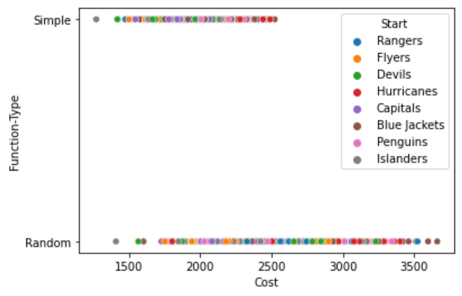
	Simple Hill Climbing	Random Restart
Mean	1985	2572
STD	226	375
MIN	1272	1392
MAX	2727	3754

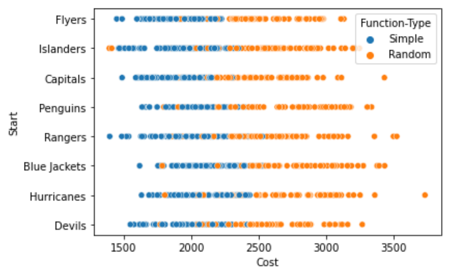
(*The numbers are the cost (distance) of the path, rounded to the nearest whole number)

Simple Best Path: ['Islanders', 'Rangers', 'Islanders', 'Flyers', 'Devils', 'Capitals', 'Penguins', 'Blue Jackets', 'Hurricanes']

Random Restart: ['Islanders', 'Rangers', 'Islanders', 'Devils', 'Flyers', 'Capitals', 'Blue Jackets', 'Penguins', 'Hurricanes']

Graphs:

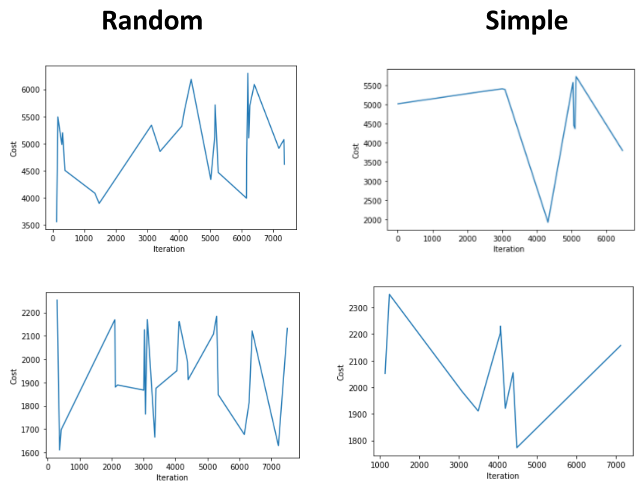




Discussion: Simple hill climbing, wins again.

Overall Results

Graphs:



(X: Cost Y: # of iteration)

(This is comparing the costs of paths through iterations of the running searches on the pacific and metropolitan division)

This does a good job of showing the difference between the simple hill climbing algorithm and the random restart algorithm. As we can see, the random restart introduces a lot of randomness into the paths, allowing paths of all costs to be generated. The simple hill climbing on the other hand, shows less randomness and more order.

Conclusion: Overall, the simple hill climbing algorithm produced a better cost path for this problem. I think that the simple hill climbing did better than the random restart because the problem size was small. I think that random restart algorithms work better on problems that are larger than this problem. It might be better to use random restart in situations where the dataset is so large that it is hard to use simple hill climbing due to size.

Real Life Applications: While it might seem trivial to figure out the best path for a hockey team to take, there is an important real-life application to this problem. Traveling in planes causes a lot of CO2 emissions, so if a hockey team takes a more efficient path, they will emit less CO2. To show the difference between taking an efficient path vs an inefficient path, lets analyze the pacific divisions CO2 emissions. The most efficient path (times four, to simulate a complete tour) would emit 4.9 tons of CO2 (we are assuming that they are traveling via an SUV, because I couldn't find a website that calculated the CO2 emissions of a plane) The most inefficient path would generate 18.2 tons of CO2 emissions. This means that by taking a more efficient path, the pacific division teams could save 13.3 tons of CO2 emissions.

Learning Outcomes: I have learned several things through this project. First, the importance of data collection. I had to collect all the data I wanted to use by hand. This was probably the most difficult thing I had to do. It was tedious and annoying, because at one point I realized I made a lot of mistakes, so I had to restart the whole process. Second, I learned the importance of understanding what data you are working with. During the charting process, I realized I didn't collect all the search data I needed. It ended up being okay, but had I really taken the time to understand what my data was, I wouldn't have had issues when I while I was trying to wrap up my project. Lastly, I feel like I made several errors along the way, so I wish I spent more time researching these algorithms. While I believe that I implemented the search algorithms correctly, I still feel like there were better implementations I could've done.