Colleen Lemak
Professor Martinez
CSCI 4740
28 February 2022

A* Search Applications

A-Star Search is very advantageous as it uses actual cost, as well as heuristic cost to estimate the time it will take you to travel distance from point A to point B. In the below model, I use the satellite Google Maps figure to map from the Guzman el Bueno metro station to the San Ignacio building of Saint Louis University, marked in yellow.

I decided to divide up the distance by using circles to represent walking time, with different colors representing different paths available. Each circle is about 18 seconds for the average person to walk.

distance between each circle is approxamately 18 seconds walking.

17 circles

28 circles

30 circles

Each circle represents about 18 seconds of walking, and there are three paths one could take. I am going to predict the path in pink will be most efficient as it looks the most direct. When walking to classes from the metro, I tend to take the green path, but I will rethink how I walk to school since there may be shorter paths available.

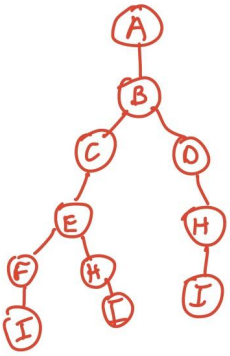
Below is a representation of how I assigned nodes and their labels to make the most efficient map.

distance between each circle is approxamately 18 seconds walking.

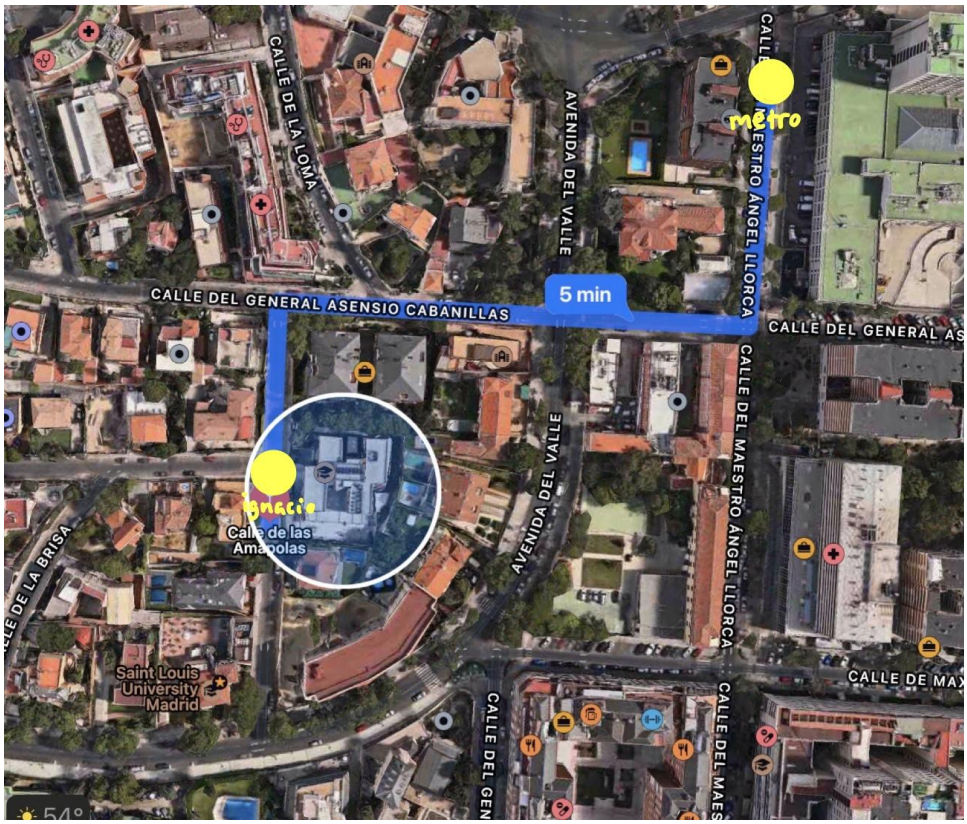
17 circles

28 circles

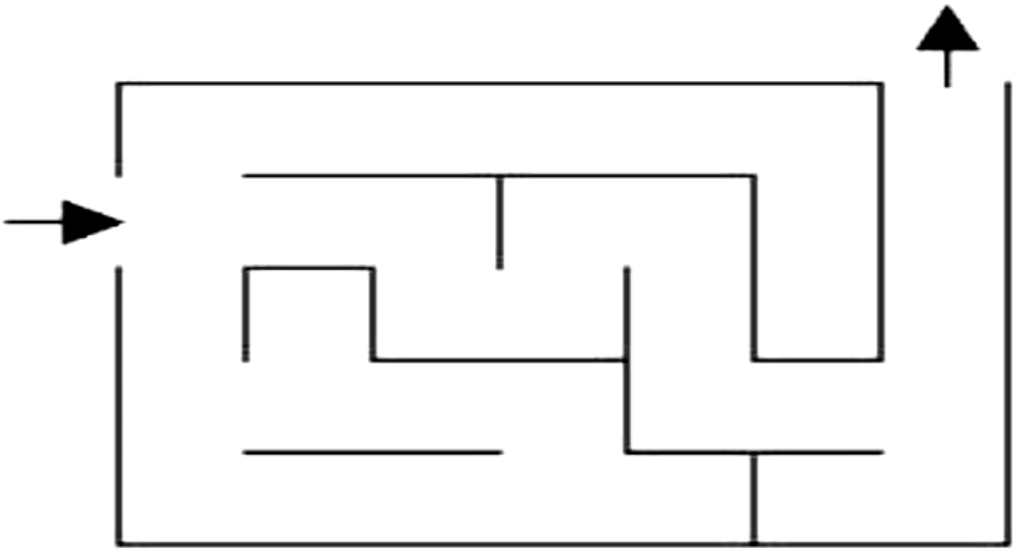
30 circles



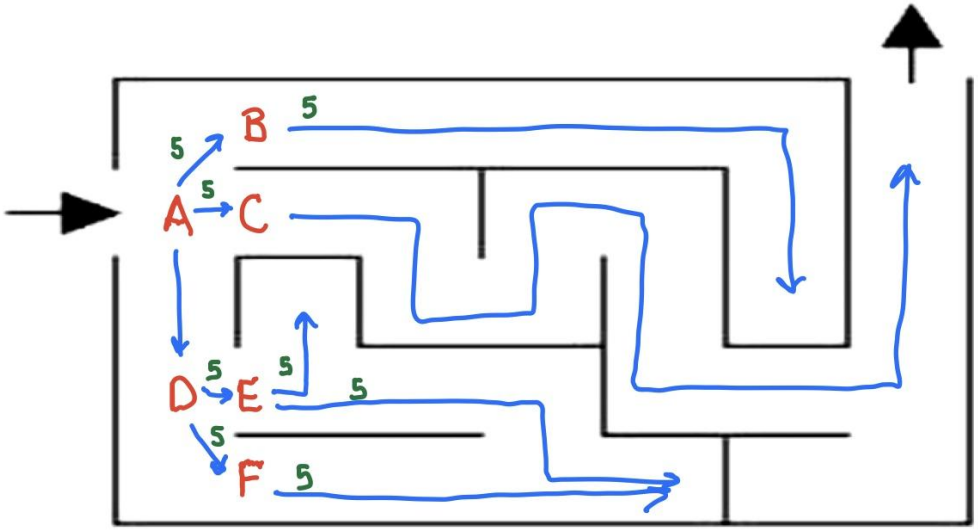
Then I coded the nodes in Python, and the output was the pink path like I predicted: A* search f(n)=g(n)+h(n) A-B-C-E-F-I with cost 252 after exploring 7 nodes. Cost 252 is the amount of seconds it will take to walk there, which is about 4.5-5 minutes. This is the ideal walking path to take. However, using Greedy-Search, you can also take another path. A* search f(n)=h(n) A-B-D-H-I with cost 540 after exploring 7 nodes. This method explores the same amount of nodes, but it estimates a walking time much longer than our A* search path. It will be 288 more seconds, or about 5 more minutes than the A* path. This does not seem like a huge difference, but if we were to apply this to a driving road trip and this method added on an hour to the trip, it would not be efficient time or money-wise. A* search performs much better as a search algorithm in maps.



My second application of A* will be using a maze instead of a map. Below is the maze to solve.



To the human eye, this seems like a very simple maze, but there are many paths the computer can calculate as I have drawn below.



I set the cost of each move to be an arbitrary 5 cost; the rules are simple: if you run into a wall, you will have to backtrack which costs more than directly reaching the goal, and you want to get the lowest score to get out of the maze efficiently. In this case, I will set 'P' to be the goal state, or when the maze ends, represented as the second arrow on the right in the above figure.

The A* solution path is A* search f(n)=g(n)+h(n) A-C-P with cost 10 after exploring 6 nodes. I set up this path by creating nodes and inputting each one's cost, with backtracking involved for nodes that would not directly improve progress toward the goal state. Using Greedy-Search, everything is much less effective. A* search f(n)=h(n) A-C-P with cost 10 after exploring 9 nodes. This method explores 9 nodes instead of 6, which may take more time, although it has the same cost overall. When navigating mazes, there may be an important time

factor to consider because taking an indirect path may leave someone stuck for a while, whereas the A* search helps us take a direct path and limit the amount of times we visit various places in the maze so it is more useful and helpful to getting to the goal state.