Professor Turner

Ryan Handlon

Due : March 2nd, 2021

# **Agents and Search Programming Assignment**

## **Background**

This assignment involves of different types of agents and search algorithms in a grid-based robot world. Students are given code for a simulation world and must build 6 different types of agents. They include a simple reflex agent, a model-based agent, a hill climbing agent, a agent that utilized uniform cost search, and two agents utilizing A* search, each with a different heuristic function. Students then had to build a new world that was based on a simple puzzle problem and implement the uniform cost and A* agents in that world as well. After all agents we're implemented in their worlds, students must analyze the effectiveness of each agent and compare them

## **Methods**

### Implementation of each agent:

The Implementation of the Reflex Agent was very simple. Given the nature of a reflex agent, it only has its percepts to rely on and can't store information. To find the corner, it continuously moves forward. Should it ever encounter something in front of it, it moves right. This implementation is very simple. In fact, it's so simple that because it can't save information, once it reaches a corner it continuously bumps into it, never stopping.

A Model Based Agent can save information. This allows for a little more sophisticated decision making. For this agent I decided to implement depth first search. Coming from the reflex agent, the implementation of the depth first search didn't add much to the agent's capability to find a corner. It added complexity for minimal return. It did however allow the agent to keep track of its location relative to it's starting position. Although this wasn't super helpful on its own.

A Hill Climbing Agent can know where it starts in the simulation as well as it's goal. The model-based agent was able to remember it's position, but this new information allows it to make decisions to progress in a direction that is towards its goal rather than searching blindly.

For this agent I copied my depth first search code from the model-based agent. From there I was able to easily update it so that rather than just choosing a random branch to go down, it followed a Manhattan distance heuristic choosing the direction going towards the goal. One thing of note in my implementation was that because of poor planning there were a couple of quirks in my implementation of the Hill Climbing Agent and the future Agents. Given the fact that the agents are not supposed to access the internals of the robot, any time an agent needed a variable I just made one for it. This comes off as weird when you have to create the agent because you sometimes have to input the same information (such as start location) multiple times. Once for the robot internals and once for the agent decision making. On a similar note, and because at the time I had a limited understanding of the simulation coordinate system, I implemented them backwards for my hill climbing agent. This means that if you were inputting the starting location of (x y) for the internals of the robot, when you inputted that same information for the agent decision making, you'd have to input (y x). This is all noted in the comments of the code.

A* Search is a very sophisticated algorithm and very hard to implement. It can know what the whole simulation looks like at the beginning of the search. It utilizes this information by creating a priority list and expanding nodes in the priority list by lowest cost given a that nodes distance from the start and a heuristic of its distance to the goal. I had to implement two different heuristics. I chose one to be the Manhattan distance from the node to the goal and my other to be the distance as the crow flies from the node to the goal (c = sqrt(a^2 + b^2)).

Uniform Cost Search is tricky but actually very similar to A*, but doesn't use a heuristic to estimate the distance from the node to the goal. After seeing a suggestion on discord from the professor, I decided the best way to implement Uniform Cost Search was by implementing A* first. From there I copy-pasted my A* code and replaced the heuristic function so it just returned Zero.

For my second domain I decided to implement the 8-puzzle. I was successful in implementing it but could not build the agents in time. Due to this I won't have any discussion in my write up on it, however my in-progress code is still there and hasn't been deleted.

**In case you read this first, you must to ("load "agents.lisp") after you do (load "run-assignment.lisp")**

<u>Statistical techniques:</u>

For my statistical analysis of my agents I conducted paired t-tests using Excel on all the information I gathered. For the Reflex and Model Based Agents I tracked whether they hit the corner and how many moves it took to get there in 12 simulations. 4 without obstacles, 4 with obstacles but no fake corners, and 4 with obstacles and fake corners. For my Hill Climb Agent, I recorded whether it hit its goal, and how many moves it took to stop. This happened in 60 simulations. 20 without obstacles, 20 with obstacles but no fake corners, and 20 with obstacles and fake corners. For my Uniform Cost and to A* Searches I tracked how many nodes were opened and the max size of the open-list. These also happened in 60 simulations. 20 without obstacles, 20 with obstacles but no fake corners, and 20 with obstacles and fake corners. In all of my simulations the robot started at (1 1), and if it could search for a specific spot on the grid, searched for a randomly specified one.

## <u>**Results**</u>

All of my results are in an attached excel file titled "simulation-results.xlsx". It includes two sheets, one title "Paired T-Tests" and another titled "Raw Data".

## <u>**Discussion**</u>

Would you think your agents would scale up?

Looking at my results I think it's clear that my A* agent using a Manhattan distance would probably scale up on a larger grid the best in terms of speed. Of my agents, it had the best mean nodes opened among my three search agents in all simulation types. I believe this is due to the accuracy of the heuristic, allowing it to travel more accurately and visit less. Comparatively, the Uniform Cost search which has no heuristic had a mean number of nodes opened of 272.6 in simulations with no obstacles. I think it would take up too mush storage. Looking at my paired t-tests where $p < 0.5$, mean max open-list size was almost always biggest for A* Manhattan. This logically makes sense though because as it travels in the right direction it opens more nodes, leaving others untouched. This is unlike Uniform Cost which was always the lowest, simply because it visits all nodes put in the open-list in the order they're added.

How did the results compare?

The mean nodes opened for each search agent and simulation type were as follows:

Mean Nodes Opened

| | No Obstacles | No Fake Corners | Fake Corners |
|---|---|---|---|
| Uniform Cost | 272.6 | 326.7 | 351.85 |
| Manhattan | 22.95 | 30.8 | 27.2 |
| Crow | 130.1 | 106.9 | 107.7 |

All of these when compared in a paired t-test had one-tail p values $< 0.05$ as can be seen in the Excel sheet. From this I think it's pretty evident across the board that the Manhattan distance heuristic is by far and away the best option when it comes to number of nodes opened. "As the Crow Flies" heuristic coming in second with Uniform Cost being the worst by a long shot. It is worth noting that "As the Crow Flies" had very variable results. Under the no obstacle simulations, it's lowest number of nodes opened was 3 while its highest was 386. Compare this to Manhattan whose lowest was 3 and highest was 41.

The mean Open-list size for each search agent and simulation type were as follows:

Mean Open-List Size

| | No Obstacles | No Fake Corners | Fake Corners |
|---|---|---|---|
| Uniform Cost | 20.05 | 26 | 31.5 |
| Manhattan | 32.8 | 30.7 | 32.35 |
| Crow | 22.9 | 33.05 | 33.85 |

All of these when compared in a paired t-test also had one-tail p values $< 0.05$ as can be seen in the Excel sheet. In this table the values are a lot closer together, but Uniform cost search just barely edges out the two A* searches for least small open list size. This makes sense to me given the results of the previous table and my discussion about the agents scaling. It's like a tradeoff between your speed in how many nodes you visit vs your storage in how many nodes you have open at once.

The difference between Manhattan and Crow is very small, almost negligible, except for in the No Obstacles category where Manhattan is behind Crow by almost 10 nodes. I'm not sure why this is the case.

If I had had enough time, I would've liked to compare search/run time with the number of nodes opened and max open-list size. My guess would be that A* would've been the fastest to run while Uniform Cost was the slowest, but it would be nice to reaffirm that hypothesis.

How did your search agents compare to your reflex agents?

My reflex agents and their results were lackluster compared to my search agents. I believe this is partly since I started all my simulations with my nodes at (1 1). This allowed both my agents to just travel straight up and then right till they hit a corner in the no fake corner and no obstacle simulations. This resulted in them always having the same number of total moves for those search categories. For the fake corners simulations, they got stuck every single time. I feel this is mostly in part due to the agents and he amount of information they possess. As my agents had more information, they were able to utilize it better. The fact that the reflex and model-based agents had so little information made them perform so poorly compared to search agents like A* Manhattan.

I think the biggest thing I've learned and demonstrated about implementing AI systems in the real world has to do with information utilization. Basically, the more information you can feed your AI and the better it can utilize that information, the better off it's going to be.