Intro to AI - Assignment 1

Alan Tong, Aidan O'Gorman

February 25, 2024

1 Introduction

This is our submission for the Assignment 1, the Fast Trajectory Replanning project. We implemented the simulation of the gridworld in the Rust programming language, and the source code is attached in the zip file. You can refer to the Rust Documentation for instructions on how to install the Rust compiler and how to compile and run the source code.

2 Simulation

2.1 Setup

The simulation generates n number of mazes of a certain size using a randomized depth-first search. This works by splitting the map into node tiles that each have 1 tile between each other. Since randomized DFS produces mazes that are are one giant connected component without cycles, we also applied a second stage where random cells between nodes are chosen to be closed or opened. This produces different connected components, which allows for unreachable goals as well as cycles in the maze.

The maps aren't stored as files, however the map generation uses a fixed random seed. This allows for the same maps to be generated for a given map size in different runs of the simulation, making the runs of each test reproducible.

A comprehensive documentation on how to use our simulation executable from the command line can be found in the `README.md` file in the zip folder.

2.2 Results

Here are the following results of running of the simulation on various pathfinding algorithms for different amounts of 101 x 101 grids with a random seed of 1234. The executable was also built in release mode to ensure maximum code optimization and performance. The trials were done on a device with the following specifications:

```
OS:              Windows 11 Pro
Processor:       Intel(R) Core(TM) i9-10885H CPU @ 2.40GHz   2.40 GHz
Installed RAM:   64.0 GB (63.8 GB usable)
Manufacturer:    DELL
Device Model:    Precision 5550
```

Listing 1: Device Specifications

```
cargo run --release auto 1234 50 101 101
    Finished release [optimized] target(s) in 0.12s
     Running `target\release\assignment_1.exe auto 1234 50 101 101`
Auto Tests
DFS Rand Map, Trials: 50, Map Size: Vec2(101, 101), RNG Seed: 1234
A* Forward:
  Average Time:           6.38 ms
  Average Expanded Cells: 11,350.6 cells
A* Backward:
  Average Time:           26.3 ms
  Average Expanded Cells: 43,398.1 cells
A* Forward Lower G:
  Average Time:           33.02 ms
  Average Expanded Cells: 55,769.7 cells
A* Forward Higher G:
  Average Time:           6.1 ms
  Average Expanded Cells: 11,350.6 cells
Adaptive A*:
  Average Time:           6.06 ms
  Average Expanded Cells: 8,876.26 cells
```

Listing 2: 50 map run

```
cargo run --release auto 1234 100 101 101
    Finished release [optimized] target(s) in 0.09s
     Running `target\release\assignment_1.exe auto 1234 100 101 101`
Auto Tests
DFS Rand Map, Trials: 100, Map Size: Vec2(101, 101), RNG Seed: 1234
A* Forward:
  Average Time:           6.96 ms
  Average Expanded Cells: 11,629.69 cells
A* Backward:
  Average Time:           24.37 ms
  Average Expanded Cells: 40,736.48 cells
A* Forward Lower G:
  Average Time:           30.27 ms
  Average Expanded Cells: 52,467.68 cells
A* Forward Higher G:
  Average Time:           6.46 ms
  Average Expanded Cells: 11,629.69 cells
Adaptive A*:
  Average Time:           5.45 ms
  Average Expanded Cells: 8,980.98 cells
```

Listing 3: 100 map run

3 Questions

3.1 Part 1 - Understanding the methods [10 points]

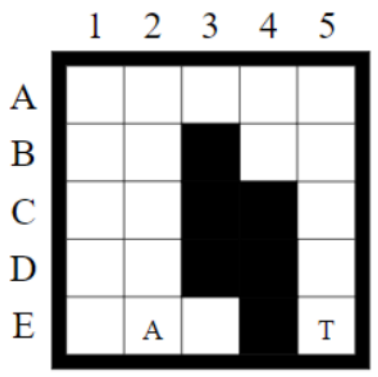


Figure 8: Second Example Search Problem

a) Given that the agent does not know which cells are blocked and that the agent is using a Manhattan distance heuristic, the agent would first choose to explore the east cell. Note that the g-cost of all adjacent cells is 1, therefore the heuristic becomes the determining factor in which cell to explore first. Since the east cell has the lowest heuristic cost of 2 compared to the north and west cells' heuristic costs of 4, the agent will choose to explore the east cell first.

b) We define a cell "J" in our maze where cell J is a "junction" connecting paths in all four cardinal directions.

Case 1: The first path that the agent goes down allows it to not have to return to cell J. Cell J is passed through once.

Case 2: The first path that the agent goes down is a dead end, but the second path it goes down allows it to not have to return to cell J. Cell J is passed through twice.

Case 3: The first two paths that the agent goes down are dead ends, but the third path it goes down allows it to not have to return to cell J. Cell J is passed through three times.

Case 4: All three paths are dead ends, so the agent goes back the way it came after exploring all three paths. Cell J is passed through four times.

It is impossible for all unblocked cells of a maze to be Case 4 junctions, but even if they were, each of the n unblocked cells would be passed through four times, for a total of $4n$ moves. Since $n < n^2$, the number of moves the agent makes until it reaches the target or discovers that this is impossible is bounded from above by n^2.

3.2 Part 2 - The Effects of Ties

We found that the Repeated Forward A* that chose cells with greater g-values consistently ran faster and expanded fewer cells than Repeated Forward A* that chose cells with smaller g-values.

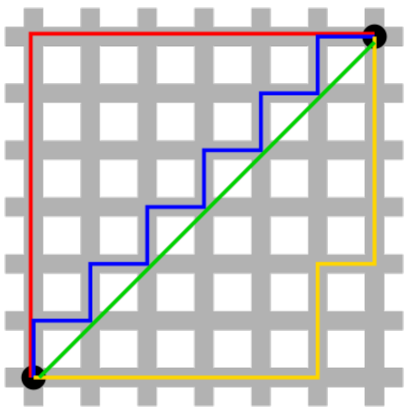
Given f(n) = g(n) + h(n), we know g(n) is the path cost from the start node to cell n, h(n) is the estimated cost of the cheapest path from n to the goal, and h(n) is the estimated cost of the cheapest solution through n. If two cells share the same f value, we would rather choose the cell with the larger g value because the cell with the larger g value will have a smaller h value. The h value always underestimates the true distance to the goal, which is to say h is an optimistic and generous estimate of the distance to the goal. If we choose the cell with the greater h value and smaller g value, we are risking using a longer distance optimistic estimate, which leaves more physical room for undiscovered obstacles to be in our way, which would force us to explore more cells than if we had just chosen the cell with the greater g value.

3.3 Part 3 - Forward vs. Backward

We found that Repeated Forward A* consistently ran faster and expanded fewer cells than Repeated Backward A*.

This is because Forward A* takes advantage of the fact that the agent has more information about the cells around it. Backward A* does not do this, instead opting to pathfind starting from the goal, where we have little to no information about the positions of unblocked and blocked cells. This causes Backward A* to more or less make a heuristic "beeline" for the agent, making it more likely to encounter obstacles that Forward A* may not have hit.

3.4 Part 4 - Heuristics in the Adaptive A*



Why Manhattan Distances are Consistent A heuristic function $h(n)$ is consistent if it underestimates (or accurately estimates) the true cost to reach the goal, and if $h(n) = 0$ at any node that has the goal state.

If an agent can only move in the four cardinal directions (north, east, south, west), that means the agent can only move along one axes at a time — either in the x-axis for the y-axis. Furthermore, the agent can only move in fixed-step intervals along each axis at a time. This means the shortest path between the any tile and a goal tile is the number of steps it takes to move along the y-axis from the tile to the goal + the number of steps it takes to move along the x-axis from the tile to the goal.

The Manhattan distance is defined as the number of fixed-interval steps it takes to move along the horizontal axis to a goal + the number of steps it takes to move along the vertical axis to a goal. The Manhattan distance from the goal to the goal itself = 0. Therefore, the Manhattan distance is a consistent heuristic.

Note that the order in which you take y-axis or x-axis moves does not matter. As shown in the diagram above, the Manhattan distance between an agent on the bottom left corner to a goal on the top right is 12 blocks. The agent must take 12 steps up and 12 steps right to get to the goal. Both the red, blue, and yellow paths represent possible paths the agent could take, which would all have Manhattan distances of 12 blocks.

Why Adaptive A* Leaves Consistent h-values Adaptive A* calculates an h-value using the following formula.

$$h(n) = g_{\text{prev}}(n) - g_{\text{prev}}(n_{\text{goal}})$$

n is a given tile being inspected in the current A* search, and $g_{\text{prev}}(n)$ is the actual cost of the getting to that tile from the start of the previous A* search, while $g_{\text{prev}}(n_{\text{goal}})$ is the actual cost of getting to the goal from the start of the previous A* search. $g_{\text{prev}}(n) - g_{\text{prev}}(n_{\text{goal}})$ therefore represents the cost to get from the given tile to the goal based on the pathfinding from the previous A* search.

Since $h(n)$ in Adaptive A* is the cost to get from the given tile to the goal using the knowledge from the **previous** A* search, this $h(n)$ cost is an optimistic estimate of the actual cost from the the given tile to the goal. This is because in the steps after the previous A* search, the agent has discovered more parts of the map and has updated it's mental model of the map to include more obstacles that were previously unseen. Therefore the actual cost from a given tile to the goal can only increase as the agent uncovers more of the map and finds more obstacles. Therefore the actual cost $g(n) \geq h(n) = g_{\text{prev}}(n) - g_{\text{prev}}(n_{\text{goal}})$.

Additionally, for the goal tile, the Manhattan distance = 0, therefore the initial $h(n_{\text{goal}} = 0$. When Adaptive A* recalculates the h-value,

$$h(n_{\text{goal}}) = g_{\text{prev}}(n_{\text{goal}}) - g_{\text{prev}}(n_{\text{goal}}) = 0$$

A heuristic function $h(n)$ is consistent if it underestimates (or accurately estimates) the true cost to reach the goal, and if $h(n) = 0$ at any node that has the goal state.

Therefore by the definition of consistency, Adaptive A* leaves consistent h-values even if action costs increase (such as from discovering new obstacles).

3.5 Part 5 - Heuristics in the Adaptive A*

We found that Adaptive A* consistently ran faster and expanded fewer cells than Repeated Forward A*.

This is because Adaptive A* does not always use the Manhattan distance for the heuristic in each cell like Repeated Forward A* does. Instead, Adaptive A* uses the cost from a given cell to the goal based on the pathfinding from the previous A* search.

This means the heuristic cost in Adaptive A* accounts for obstacles that were seen by the time the previous A* search was ran, and is $\geq$ the Manhattan distance from the tile to the goal, since the Manhattan distance is a very optimistic estimate of the cost from the tile to the goal that assumes no obstacles are blocking the way. Adaptive A* ultimately allows for more focused searches, by assigning higher h-costs to nodes near dead ends to deter exploration there. However in Repeated A*, the agent will often search dead ends because it relies on Manhattan distances which don't account for the actual obstacles on the map. Ultimately, this means the Adaptive A* heuristic dominates the simplistic Manhattan distance heuristic, since

$$h_{\text{Adaptive A*}}(n) \geq h_{\text{Manhattan Distance}}(n), \forall n$$

Therefore, Adaptive A* is invariably equal to or faster than Repeated Forward A*.

3.6 Part 6 - Statistical Significance

Here are the steps you'd have to take to make an experiment that tests whether Forward A* and Backward A* have the same running time.

1. Assume Forward A* = Backward A*, where the two algorithms being "equal" means they have equivalent running times, not necessarily that the two algorithms are identical. This is our null hypothesis (H_0). The alternative hypothesis (H_1) is that Forward A* $\neq$ Backward A*.
   (a) Let running time distribution of Forward A* = distribution 1.
   (b) Let running time distribution of Backward A* = distribution 2.

$$H_0 : \mu_1 = \mu_2$$
$$H_1 : \mu_1 \neq \mu_2$$

2. Determine the α value of the t-test which will serve as the cutoff for whether we accept or reject the null hypothesis. Typically $\alpha = 0.05$.
3. Run each algorithm on the same set of 100 randomly generated gridworld maps of size 101×101. Then calculate the mean running time in seconds of each algorithm across all $n = 100$ maps.
   (a) Given that we used samples of size 100 which is > 30 and is considered a large sample size, under the Central Limit Theorem we can assume the sampling distribution of the mean of each algorithm follows a normal distribution.
   (b) Since we don't know the standard deviation of the actual runtime distribution for the two algorithms, we will use a t-test for a difference between the mean runtime of the two algorithm.
4. Find the sampling distribution of the difference between the algorithms' mean running time in seconds (Δ) under the assumption that Forward A* = Backward A*. This can be created using the mean and standard deviation of the samples for the two algorithms.

$$\mu_{\bar{x}} = \bar{x}_1 - \bar{x}_2$$
$$\rho_{\bar{x}} = \sqrt{\frac{s_1^2}{n} + \frac{s_2^2}{n}}$$

5. Use this distribution, perform a t-test to calculate the probability p that Δ equals what you found it to equal given Forward A* = Backward A*. The probability p is your residual uncertainty that Forward A* *might* equal Backward A*.
   (a) First calculate the t-statistic for the difference between the two mean

$$t_{\text{stat}} = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n} + \frac{s_2^2}{n}}}$$

   (b) This should give you a t-statistic for one tail of the t-distribution. Since $H_0 : \mu_1 \neq \mu_2$, we are performing a two-tailed test. Since t-distributions are symmetrical, multiply t_{stat} by -1 to find the opposite t-statistic for the other tail.
   (c) Then using a t-distribution with $df = n + n - 2$, calculate p = the area to the left of the negative t_{stat} + the area to the right of the positive t_{stat}.
6. Draw a conclusion based on the probability p and your cutoff α.
   (a) If $p < \alpha$, then you reject the null hypothesis. There is sufficient evidence that Forward A* $\neq$ Backward A* in terms of runtime.
   (b) If p $p \geq \alpha$, then you fail to reject the null hypothesis. There is insufficient evidence that Forward A* $\neq$ Backward A* in terms of runtime.

3.7 Extra Credit

- This document is written in LaTeX.
- We implemented our own array-based binary heap for this project, which can be found in the `heap.rs` source file.