CS 440 : Intro to Artificial Intelligence

Assignment 1 : Fast Trajectory Replanning

Ayush Joshi, Dan Snyder, Nikhil Mathur

Feburary 26, 2018

Abstract

We used Python as our primary language to write the code for this project. To generate the Environment we are using Pygame framework. We have implement threading to efficiently build and update the Environment in real time as the algorithms are running for visual effect.

Each algorithm implements an abstract class methods and the secondary thread is responsible for their execution. We have used Node structure for every cell in the grid which has following attributes: x coordinate, y coordinate, Parent, search, g value, h value, and flags.

We have also implement our version of Heap. Libraries required: Pygame, threading, random, sys, argparse, and re.

Usage

For Instructions : python Fast_Trajectory_Replanning.py -h

Part 1

a The first move of the agent is to the east because the first step of A* is to proceed along the "shortest presumed unblocked path". The agent "knows" that the cell is unblocked, because it is included in the original knowledge about the environment available to the agent.

b Since the grid is a finite number, 101 by 101 cells, the target will ultimately be reached. In the worst case scenario, the agent will visit all unblocked nodes and the target would have been visited. There are two different possibilities that can occur: one of them being that the target has been reached and added to the open list or the target hasn't been reached and the open list is empty. Regarding the latter, if this is the case, then the algorithm stops running and the target has not been visited meaning it hasn't been found.

Part 2

We implemented both versions of Repeated Forward A*, one with smaller g value and another with larger g value. On comparison based on the number of expanded cells and runtime, we found that the algorithm with smaller g values expand way more nodes because it always expands nodes closer to the target and found shorter paths as it evaluated more options. On the other hand, the algorithm with larger g values expanded fewer nodes but couldn't always find the most optimal path.

In algorithm with ties broken by smaller g values the average number of nodes visited by the agent was 231 and took an average of 0.74 seconds to complete the search. As compared to the algorithm in which ties broken by larger g values the average number of nodes visited by the agent was 295 and took an average of 7.6 seconds to complete the search. (Note: generated time without displaying visual effect)

Part 3

We implemented both Repeated Forward A* as well as Repeated Backward A* and compared them with respect to runtime and number of nodes expanded. Although the distance is the same for Forward and Backward A*, the paths that agents of both algorithm encountered were different. From our observations, we see that Backward A* is quicker than Forward A* on average because Backward A* took 0.68 seconds compared to the 0.74 seconds of Forward A*.

The average number of nodes visited by Forward A* is 295 to complete the search as compared to 280 nodes expanded for Backward A* which again shows that Backward A* is a little quicker compared to Forward A*. (Note: generated time without displaying visual effect)

Part 4

a The Manhattan distance is the distance between two points on a grid based on a strictly horizontal and/or vertical path. It is considered consistent because it is computed summing up the shortest possible horizontal and vertical distances, especially since the agent will not be moving diagonally in any direction. If the agent were to move diagonally, it wouldn't be considered consistent as the heuristic would overestimate the target.

b To show the stronger condition of consistency, we will use the form of the general triangle inequality: h(n) <= c(n,a,n') + h(n'). A heuristic h(n) is consistent if, for every node n and every successor n' of n generated by any action a, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n' plus the estimated cost of reaching the goal from n'. Using this inequality, if there is an action cost increase, c would be the action cost before increase and c' would be the action cost after increase. The hnew(n) <= hnew(n') + c(n,a,n') <= hnew(n') + c'(n,a,n'). Looking at this, we can see the heuristic is left consistent.

Part 5

We implemented both Repeated Forward A* and Adaptive A*. On average, their runtimes are almost similar, but in the best scenario, Forward A* speed was 0.74 seconds and Adaptive A* speed was 0.65 seconds. And, the nodes expanded by the agent for Forward A* was 295 as compared to 290 nodes expanded for Adaptive A*. So Adaptive A* is ideally the better.

Through our experiments, we observed that the speed of both algorithms are almost the same but Adaptive A* is slightly faster in most case because in most cases the heuristic doesn't stay constant, the Adaptive A* algorithm adapts to a changing environment which makes the algorithm slightly more efficient. (Note: generated time without displaying visual effect)

Part 6

We could improve memory consumption by not using python because it uses 28 bytes for a single integer and 24 bytes for booleans. We could pack all the data into one integer per cell which will end up to be 40 byte integer per cell instead of 132 bytes. Another suggestions would be to use lists and dictionaries over the node class because each node object has a 56 byte overhead. If we get rid of that 56 byte overhead and pack all the values into one integer, it can be slimmed down to 64 bytes per cell. This would then lead to usage of 62 MB for a grid of size 1001x1001.

The largest grid within 4 MB will be of size 178x178. Running grid of size1001x1001 would take 126 MB.