Fast Trajectory Re-planning

CS 520 - Assignment 1

Sarthak Dalal (NetID - spd115)
Harish Balasubramaniam (NetID - hb425)
Shlok Kothari (NetID - sk2677)

October 17, 2022

Professor

Prof. Abdeslam Boularias

Part 0 - Setting up the Environment

We took inspiration from the algorithm suggested *here*[1] to implement maze generation using DFS, with a few modifications. We generated a random number between 0 and 1 and blocked a cell if it was less than 0.3, else kept it unblocked. We created 50 grids and used the *pickle*[2] library in python to store them in the 'Grids' folder. These are a couple of examples of 5x5 grids created using the algorithm ('.' represent unblocked cell and '#' represent blocked cells):

Figure 1: Maze 1

Figure 2: Maze 2

Please refer to the following file in the zip file included for the python implementation of the maze generator:

1. `CreateGrid.py`

Part 1 - Understanding the Methods

(a) *Explain in your report why the first move of the agent for the example search problem from Figure 8 is to the east rather than the north given that the agent does not know initially which cells are blocked:*
Ans) The cost of A moving to the blocks D2, E1 or E3 is 1, and assuming that the heuristic is the Manhattan distance from A to T.
At D2 (North of A):
G value is 1, and H value is 4. Hence the F value is 5.
At E3 (East of A):
G value is 1, and H value is 2. Hence F value is 3.
We can see that the F value at E3 (East of A) is smaller than the F value at D2 (North of A).
Hence, the first move of the agent is to the east rather than the north, given that the agent does not know initially which cells are blocked.

(b) *This project argues that the agent is guaranteed to reach the target if it is not separated from it by blocked cells. Give a convincing argument that the agent in finite grid worlds indeed either reaches the target or discovers that this is impossible in finite time:*
Ans) When there is a path to the target, the agent will find the path, even if it encounters a blocked cell, because the algorithm makes sure it does, as it is complete. Now, even in the case that there is no path to the target, since the agent traverses the grid world and moves at least one step towards an unblocked cell before encountering a blocked cell, it will eventually know the whole grid world in finite time as the grid world itself is finite.
Prove that the number of moves of the agent until it reaches the target or discovers that this is impossible is bounded from above by the number of unblocked cells squared:
Ans) In each search, the agent discovers at least 1 unblocked cell (or it realizes that there is no path). Hence, it is fair to say that the total number of searches is bounded by the total number of unblocked cells.
Now, in each search, the agent can move to an unblocked cell only once at most (or it doesn't go to the unblocked cell, depending upon it's f-value). Hence, it is fair to say that the total number of searches is also bounded by the total number of unblocked cells.
We know that,
Total number of moves the agent makes = Total number of searches * Total number of moves in each search
Since both, the total number of searches and the total number of moves in each search are bounded by the number of unblocked cells, we have:
Total number of moves <= Number of unblocked cells * Number of unblocked cells. **Hence,**
Total number of moves <= Number of unblocked cells squared.

Implementation

Attached below are two images of the grids where the shortest paths are as discovered by the algorithms. The white cells represent unblocked cells, while the black cells represent blocked cells. The agent is represented by a neon cell and the target is represented by a violet cell. After clicking 'run', the algorithm first calculates the shortest path from the agent to the target and then the animation starts moving the agent to the target. Each loop, while moving the agent has a delay of 50 milliseconds so that the animation is seen smoothly. We used the *pygame*[3] module to generate the animation of the grids and the paths.

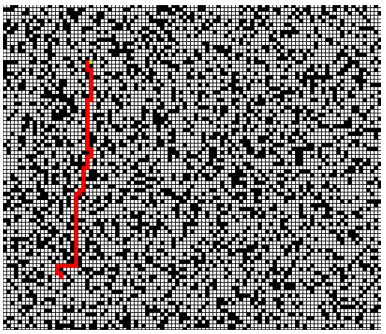

Figure 3: Implementation on Grid 1

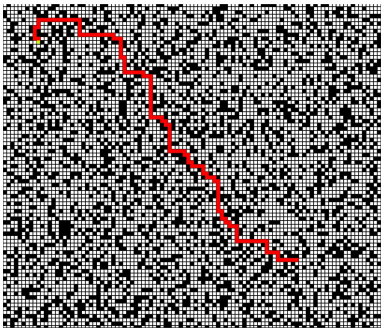

Figure 4: Implementation on Grid 22

Part 2 - The Effects of Ties

Repeated Forward A needs to break ties to decide which cell to expand next if several cells have the same smallest f-value. It can either break ties in favor of cells with smaller g-values or in favor of cells with larger g-values. Implement and compare both versions of Repeated Forward A* with respect to their runtime or, equivalently, number of expanded cells. Explain your observations in detail, that is, explain what you observed and give a reason for the observation.*
Ans) Please refer to the next page for a table of comparison between Repeated Forward A* breaking ties in favor of cells with larger G-values and Repeated Forward A* breaking ties in favor of cells with smaller G-values.
Please refer to the following files in the zip file included for the python implementation of the 2 aforementioned algorithms:

1. `All_times_ForwardAstar_LargerG.py`
2. `All_times_ForwardAstar_SmallerG.py`

Grid No.	Larger G Time(s)	Smaller G Time(s)
1	0.0	0.005000114440917969
2	0.0	0.029999971389770508
3	0.03129911422729492	0.029999971389770508
4	0.0	0.004999876022338867
5	0.015622377395629883	0.0019998550415039062
6	0.0	0.0004837512969970703
7	0.24999761581420898	0.2674989700317383
8	0.0	0.0065000057220458984
9	0.0	0.0
10	0.458575963973999	0.5539822578430176
11	0.019500732421875	0.01801776885986328
12	0.06147575378417969	0.05501675605773926
13	0.038019657135009766	0.0425260066986084
14	0.0019991397857666016	0.007000923156738281
15	0.011996984481811523	0.018500566482543945
16	0.005500078201293945	0.006514310836791992
17	0.00500178337097168	0.005516529083251953
18	0.011500120162963867	0.01399993896484375
19	0.0014810562133789062	0.001983165740966797
20	0.0010018348693847656	0.001516580581665039
21	0.05948305130004883	0.04999995231628418
22	0.5415201187133789	0.6034958362579346
23	0.12397956848144531	0.15300202369689941
24	0.00999945068359375	0.0019998550415039062
25	0.0005009174346923828	0.0004994869232177734
26	0.09600138664245605	0.11299443244934082
27	0.005503654479980469	0.005498647689819336
28	0.008500814437866211	0.04149961471557617
29	0.005999565124511719	0.008499622344970703
30	0.015505075454711914	0.014499902725219727
31	0.0	0.0004990100860595703
32	0.08599972724914551	0.09149956703186035
33	0.006014347076416016	0.00899958610534668
34	0.009501934051513672	0.011999845504760742
35	0.03399968147277832	0.04600048065185547
36	0.08999943733215332	0.12150025367736816
37	0.021019697189331055	0.059563636779785156
38	0.0	0.0010008811950683594
39	0.0004994869232177734	0.0
40	0.010999679565429688	0.013002157211303711
41	0.0559999942779541	0.07200026512145996
42	0.046979427337646484	0.052483558654785156
43	0.005998849868774414	0.010998964309692383
44	0.000499725341796875	0.0
45	0.0074994564056396484	0.009001016616821289
46	0.01450037956237793	0.018999814987182617
47	0.020999670028686523	0.023000001907348633
48	0.0019998550415039062	0.002000093460083008
49	0.00700068473815918	0.009999513626098633
50	0.17399907112121582	0.19150018692016602

From the above table, we compute the average run-time of the algorithm breaking ties in favor of Larger G-values to be **0.047289 seconds** and the average run-time for the algorithm breaking ties in favor of Smaller G-values to be **0.055581 seconds**.
Hence, we observe that, implementing Repeated Forward A* by breaking the ties in favor of the Larger G-values, gives a better running time as compared to Smaller G-values with respect to the grids that we implement them on.
We observe that the ratio of the running times is **1.175**. [Smaller/Larger]
This variation in the running time between the 2 algorithms may be explained by the fact that the Smaller G-value algorithm chooses a cell with a Smaller G-value rather than a cell with a Larger G-value, which means that the algorithm favors the cell with the higher H-value. Now, since we know that the H-value of a cell is the lower bound for the path from that cell to the target cell, choosing a cell with a higher H-value means that the expanded cell would be farther from the target, as compared to the cell with a lower H-value. As this process continues, the algorithm would traverse all the neighboring cells that might be father away from the target, resulting in a delayed run-time.

Part 3 - Forward vs Backward

Implement and compare Repeated Forward A and Repeated Backward A* with respect to their run-time or, equivalently, number of expanded cells. Explain your observations in detail, that is, explain what you observed and give a reason for the observation. Both versions of Repeated A* should break ties among cells with the same f-value in favor of cells with larger g-values and remaining ties in an identical way, for example randomly:*
Ans) Please refer the next page for a table of comparison between Repeated Forward A* and Repeated Backward A*
Please refer to the following files in the zip file included for the python implementation of the 2 aforementioned algorithms:

1. `All_times_ForwardAstar_LargerG.py`
2. `All_times_BackwardAstar_LargerG.py`

Grid No.	Repeated Forward A* Time(s)	Repeated Backward A* Time(s)
1	0.0	0.011500120162963867
2	0.0	0.0009999275207519531
3	0.03129911422729492	0.04851984977722168
4	0.0	0.03647446632385254
5	0.015622377395629883	0.0029997825622558594
6	0.0	0.0005035400390625
7	0.24999761581420898	0.09786057472229004
8	0.0	0.014500617980957031
9	0.0	0.0
10	0.458575963973999	0.027999401092529297
11	0.019500732421875	0.035500288009643555
12	0.06147575378417969	0.05199909210205078
13	0.038019657135009766	0.04901742935180664
14	0.0019991397857666016	0.0019998550415039062
15	0.011996984481811523	0.041500091552734375
16	0.005500078201293945	0.008499622344970703
17	0.00500178337097168	0.004499197006225586
18	0.011500120162963867	0.01250004768371582
19	0.0014810562133789062	0.002000093460083008
20	0.0010018348693847656	0.0014996528625488281
21	0.05948305130004883	0.00850057601928711
22	0.5415201187133789	0.019498348236083984
23	0.12397956848144531	0.06598258018493652
24	0.00099945068359375	0.0025000572204589844
25	0.0005009174346923828	0.0015001296997070312
26	0.09600138664245605	0.08700418472290039
27	0.005503654479980469	0.0024983882904052734
28	0.008500814437866211	0.02500295639038086
29	0.005999565124511719	0.00799870491027832
30	0.015505075454711914	0.016000032424926758
31	0.0	0.0
32	0.08599972724914551	0.13201284408569336
33	0.006014347076416016	0.002160787582397461
34	0.009501934051513672	0.010954618453979492
35	0.03399968147277832	0.016008853912353516
36	0.08999943733215332	0.11002111434936523
37	0.021019697189331055	0.015993595123291016
38	0.0	0.0009765625
39	0.0004994869232177734	0.0
40	0.010999679565429688	0.011000394821166992
41	0.0559999942779541	0.0492711067199707
42	0.046979427337646484	0.07722043991088867
43	0.005998849868774414	0.014089107513427734
44	0.000499725341796875	0.0
45	0.0074994564056396484	0.007995128631591797
46	0.01450037956237793	0.012986898422241211
47	0.020999670028686523	0.013998746871948242
48	0.0019998550415039062	0.0
49	0.00700068473815918	0.008989572525024414
50	0.17399907112121582	0.025284290313720703

From the above table, we compute the average time for Repeated Forward A* to be **0.047289 seconds** and the average time for Repeated Backward A* to be **0.023916 seconds**. Hence, implementing Repeated Backward A* gives a better running time as compared to Repeated Forward A* with respect to the grids that we implement them on.
We observe that the ratio of the running times is **1.8149**. [Forward/Backward] This variation in the running time between the 2 algorithms may be explained by the fact that the number of cells being explored during each search of the algorithms is more in Repeated Forward A* than in Repeated Backward A* for the random grids that we generated. The reason for this might be that, with respect to our grid worlds, there are more blocked cells near the agent as compared to the target. Hence, each search of our algorithm explores more number of cells from from the agent to the target (Forward) as opposed to the number of cells from the target to the agent (Backward), resulting in a delayed run-time.

Part 4 - Heuristics in the Adaptive A*

The project argues that "the Manhattan distances are consistent in grid worlds in which the agent can move only in the four main compass directions." Prove that this is indeed the case:
Ans) In grid worlds where the agent can move only in the four main compass directions, that is, north, south, east and west, the Manhattan Distance would never overestimate the distance between two cells, which is the definition of a consistent heuristic. Since the agent moves once, in a single direction, it wouldn't be possible for the Manhattan distance between the agent and the target to be an overestimate. **Hence, Manhattan Distances are consistent in such grid worlds**.
Furthermore, it is argued that "The h-values hnew(s) ... are not only admissible but also consistent." Prove that Adaptive A leaves initially consistent h-values consistent even if action costs can increase :*
Ans) We know that,
$h(n) <= c(n, a, n') + h(n')$
where, h is the heuristic function, and *c(n,a,n')* is the action cost of going from cell n to a neighboring cell *n'* through action *a*.
A* search basically starts by determining the shortest path from the agent to the target, following which it increases the h values of the expanded cells so that future A* searches have a more focused path. The consistency of the heuristic function can be explained with the help of the Triangle Inequality. Triangle Inequality states that the length of one side of a triangle is always less than or equal to the sum of the lengths of the other two sides. Now, taking the equation mentioned above and increasing the action cost from *c(n,a,n')* to *c'(n,a,n')*, we have a new equation,
$h'(n) <= c'(n, a, n') + h'(n')$
Here we see that, on increasing the action cost, the heuristic is still consistent

(as it is still an underestimate) because of the Triangle Inequality property.

Part 5 - Heuristics in the Adaptive A*

Implement and compare Repeated Forward A and Adaptive A* with respect to their run-time. Explain your observations in detail, that is, explain what you observed and give a reason for the observation. Both search algorithms should break ties among cells with the same f-value in favor of cells with larger g-values and remaining ties in an identical way, for example randomly:*
Ans) Please refer the next page for a table of comparison between Repeated Forward A* and Adaptive A*
We took inspiration from the pseudo code in the paper - *Generalized Adaptive A*[4] to create our code. Please refer to the following files in the zip file included for the python implementation of the 2 aforementioned algorithms:

1. `All_times_ForwardAstar_LargerG.py`
2. `All_times_AdaptiveAstar.py`

Grid No.	Repeated Forward A* Time(s)	Adaptive A* Time(s)
1	0.0	0.005498409271240234
2	0.0	0.0014989376068115234
3	0.03129911422729492	0.030499935150146484
4	0.0	0.0039997100830078125
5	0.015622377395629883	0.0025005340576171875
6	0.0	0.0005002021789550781
7	0.24999761581420898	0.1959993839263916
8	0.0	0.005498170852661133
9	0.0	0.0004980564117431641
10	0.458575963973999	0.3495159149169922
11	0.019500732421875	0.019000768661499023
12	0.06147575378417969	0.05299949645996094
13	0.038019657135009766	0.039499759674072266
14	0.0019991397857666016	0.006001949310302734
15	0.011996984481811523	0.01449894905090332
16	0.005500078201293945	0.004499912261962891
17	0.00500178337097168	0.010500431060791016
18	0.011500120162963867	0.01349949836730957
19	0.0014810562133789062	0.0015006065368652344
20	0.0010018348693847656	0.0015001296997070312
21	0.05948305130004883	0.043000221252441406
22	0.5415201187133789	0.5540163516998291
23	0.12397956848144531	0.14600038528442383
24	0.00099945068359375	0.0020003318786621094
25	0.0005009174346923828	0.0009999275207519531
26	0.09600138664245605	0.10551595687866211
27	0.005503654479980469	0.005500078201293945
28	0.008500814437866211	0.017499923706054688
29	0.005999565124511719	0.00800013542175293
30	0.015505075454711914	0.015999794006347656
31	0.0	0.0
32	0.08599972724914551	0.0950002670288086
33	0.006014347076416016	0.010483741760253906
34	0.009501934051513672	0.013098955154418945
35	0.03399968147277832	0.040407657623291016
36	0.08999943733215332	0.09874939918518066
37	0.021019697189331055	0.06108236312866211
38	0.0	0.001001596450805664
39	0.0004994869232177734	0.0
40	0.010999679565429688	0.011897563934326172
41	0.0559999942779541	0.06699776649475098
42	0.046979427337646484	0.04399681091308594
43	0.005998849868774414	0.010998010635375977
44	0.000499725341796875	0.0
45	0.0074994564056396484	0.008998394012451172
46	0.01450037956237793	0.014263391494750977
47	0.020999670028686523	0.023300886154174805
48	0.0019998550415039062	0.000995635986328125
49	0.00700068473815918	0.007004261016845703
50	0.17399907112121582	0.18103599548339844

From the above table, we compute the average time for Repeated Forward A* to be **0.047289 seconds** and the average time for Adaptive A* to be **0.046947 seconds**. Hence, implementing Adaptive A* gives a better running time as compared to Repeated Forward A*.
We observe that the ratio of the running times is **1.0072**. [Forward/Adaptive]
This variation in the running time between the 2 algorithms may be explained by the fact that Adaptive A* uses information from its previous searches to update the H-value of the cells in the path from the current state to the final state. This ensures that the next A* search and the other future A* searches have a more informed search. This increase in H-value makes the Adaptive A* algorithm reach the target much faster, as it doesn't explore as many cells as Repeated Forward A* does, resulting in a time better than Repeated Forward A*.

Part 6 - Statistical Significance

Read up on statistical hypothesis tests and then describe for one of the experimental questions above exactly how a statistical hypothesis test could be performed. You do not need to implement anything for this question, you should only precisely describe how such a test could be performed.
Ans)
We will test whether the grid size is a systematic error for our problem. To determine the occurrence of this error we will use hypothesis testing in the following way:

1. Null hypothesis: Backward A* has lower average run time than Forward A* for a grid of size of 101×101.
   Ho = μruntime forward A* > μruntime backward A*

2. Alternate Hypothesis: Backward A* will have higher average run time than Forward A* for a grid size larger than 101×101.
   H1 = μruntime forward A* < μruntime backward A*

3. Test statistic: The sample size is 50, and we can calculate the standard deviation of our data. We will therefore use the Z- statistic for this particular hypothesis testing. We will use one tailed Z test for this scenario.

4. Alpha and confidence interval: We will keep the confidence level to be 95%. Since this is a one tailed test, alpha = 0.05.

5. Computing the test statistic: For the grid size 101 × 101, we will calculate the mean of difference between run time of Forward A* and Backward A* for all 50 grids. We also calculate the standard for the same.

$$\mu_{\text{difference in runtime } 101 \times 101} = \sum_{g=1}^{50} (\text{runtime}_{\text{g backward A*}} - \text{runtime}_{\text{g forward A*}}) \Big/ 50$$

Now, in order to compute the test statistics, we will randomly generate grid sizes of G×G where G > 101. Without changing the GPU and heuristic function we will compute the run time of Forward A* and Backward A* for each grid. We will compute the difference for run-time for each grid and calculate their mean and standard deviation. In the next step, we calculate the Z-score for our data using the formula.

$$Z = \frac{M - \mu_{\text{difference in runtime } 101 \times 101}}{\sigma_M}$$

Results: We will compare the Z score to the critical Z score. If the Z score is greater than the critical Z score then we reject the null hypothesis. On the other hand if the Z-score is less than the critical Z score, then we do not reject the null hypothesis.

References

[1] https://www.geeksforgeeks.org/depth-first-traversal-dfs-on-a-2d-array/

[2] https://docs.python.org/3/library/pickle.html

[3] https://www.pygame.org/news

[4] https://www.researchgate.net/publication/221455192_Generalized_Adaptive_A