CS440 Assignment 1

Fast Trajectory Replanning

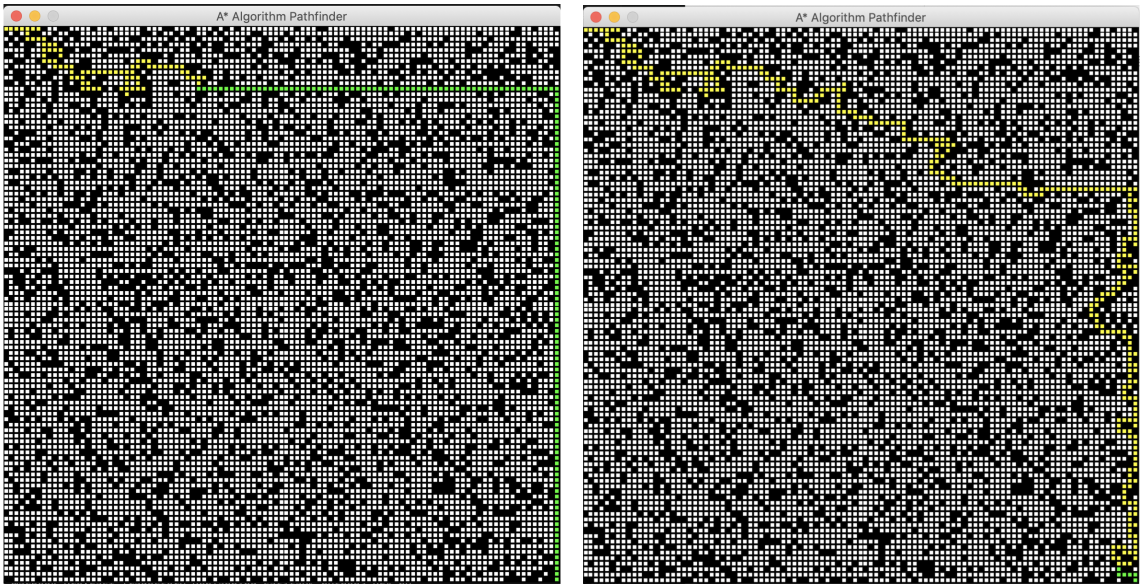
Liqin Long
Xin Lin
Satita Vittayaareekul

October 13, 2019

Contents

Part 0 - Setup your Environments

In order to generate a maze/corridor-like structure with a depth-first search approach by using random tie breaking to test all the experiments in the same 50 gridworlds of size 101x101. We estimated runtime by calculating total expanded cells



(a) From start

(b) To end

Figure 1: RFA - from starting to ending /path generating/

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.

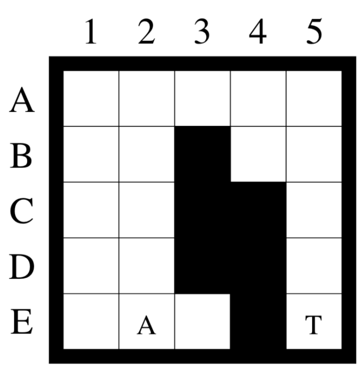


Figure 2: "Figure 8: Second Example Search Problem" from Assignment Description

By the given figure, A is at cell E2, which is the start point. And T assumed to be the goal point, where the agent is at point A and does not know the blocked cell initially. By using Manhattan distance as heuristics, we run A* method from A to T by the given knowledge that its neighbors are unblocked cells, E1, D2, and E3. We first push A to the open list and by expanding A, we add those three unblocked cells to the open list as well.
Thus, in order to decide which cell should the agent move to, A* method will calculate the g-value, h-value, and f-value for cells in the open list. And it performs:

cell E1:	g-value = 1	h-value = 4	f-value = 5
cell D2:	g-value = 1	h-value = 4	f-value = 5
cell E3:	g-value = 1	h-value = 2	f-value = 3

Since cell E3 has the smallest f-value, the agent will expand from point A to cell E3.

b.1). 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 gridworlds indeed either reaches the target or discovers that this is impossible in finite time.
This project has set the rules for the gridworld to be:
1). It is a finite gridworld, and the gridworld is bounded by the boundries of the gridworld.
2). There is a give start point (A) and a target point (T), where A has to find a way to reach T.
3). There are randomly set blocked cells to increase the difficulty for A to reach T.
Therefore after the agent searches all the unblocked cells it can reach by following its neighbor pointers, it either found the target T, or all the cells it has reached is not its goal state. And in this case, it would determine this is impossible to reach the target T.

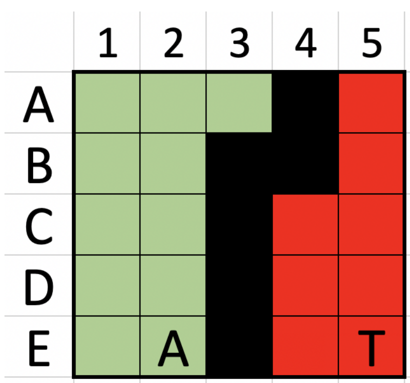


Figure 3: Scenario 1

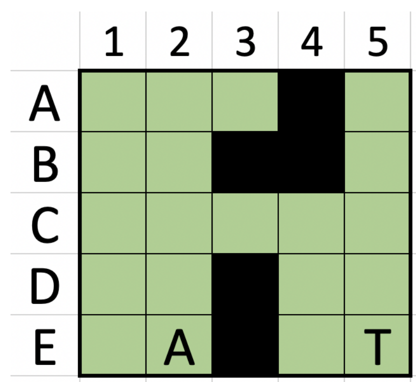


Figure 4: Scenario 2

In Figure 2 and 3, the green cells are the cells that the agent can start from A and reaches through its neighbor pointers. And cells in color red means the cells are not reachable starting from point A. So in Scenario 1, the unblocked regions that the agent can reach does not contain point T, and thus, after the agent went through all the cells in the green area, it will discover that it is impossible to reach the target and terminate the program. And as of Scenario 2, the agent could go through the path:

$$E2->D2->C2->C3->C4->C5->D5->T$$

and find the goal and return the path as the final result.

b.2). 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.

In short, we need to prove: $\quad number\ of\ moves \leqslant\ (number of unblocked cells)^2$

The Repeated Forward/Backward Algorithm is set to execute the A* algorithm for every decision the agent has to make. For each A* algorithm, it is aim to find a path from its current cell to the target cell within its current knowledge of the blocked/unblocked cells. In case of a dead-end, where the agent at that current cell found that its surrounded by either blocked cells or visited cells or gridworld boundaries, then algorithm will backtrack to the parent nodes on the search tree until it reaches a cell with an unvisited neighbor, and continue to find the path by visiting that new/unvisited cell. However, in case the algorithm backtrack all its parent and didn't discover any unvisited cell, then the algorithm will determine that this is impossible to reach the target point. In other words, whenever the agent moves, it will need to implement the A* method to find a possible path. If there is indeed a path from A to T, then the number of moves should be less then the number of unblocked cells. However, if T is not reachable from A, then the agent will need to backtrack to parent nodes at some point when it meets a dead-end, then the number of moves will be less than number of unblocked cells squared.

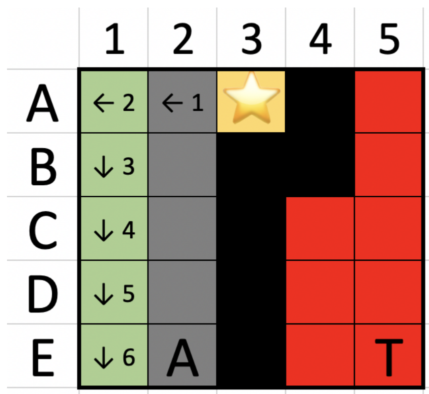


Figure 5: Sample 5x5 Gridworld

For instance, as in Figure 4, the agent still start from point A and set its target as point T. In this case, we assume that the agent has moves itself from A to cell A3 already and it is currently at cell A3. At cell A3, it found that it is a dead-end, so that it needs to track-back to its parent node, which is A2, and at cell A2 it found that the only new and invested cell is A1, so it moves to A1. Then by A* method, the agent will move through

$$A1->B1->C1->D1->E1$$

and found that it once again got into a dead-end. At this point, the agent found that D1 has been visited and E2 has been visted as well. Therefore, it will determine this maze is impossible to reach from point A to point T. More precisely, the number of unblocked cells in this case is 19, and the number of moves is 12. And $12 \leqslant\ 19^2$.

Part 2 - The Effects of Ties

We implement and compare Repeated Forward A* with larger g values and smaller g values using 50 samples with 101 x 101 grids. We use the number of expanded cells as the way to see which one is faster. We found that the algorithm with larger g values is much faster than the algorithm with smaller g values. During implementation, we found the path of larger g value is more straight forward, while path of smaller g value is indirect and takes more steps. The reason is that when we choose a bigger g value step, it means that we are closer to the target. A smaller g value means it is closer to the start rather than to the target. Expanding a smaller g value point is useless and take it back to several steps before.

Below is the number of expanded cells for two algorithms for 50 different grids as well as the line plot:

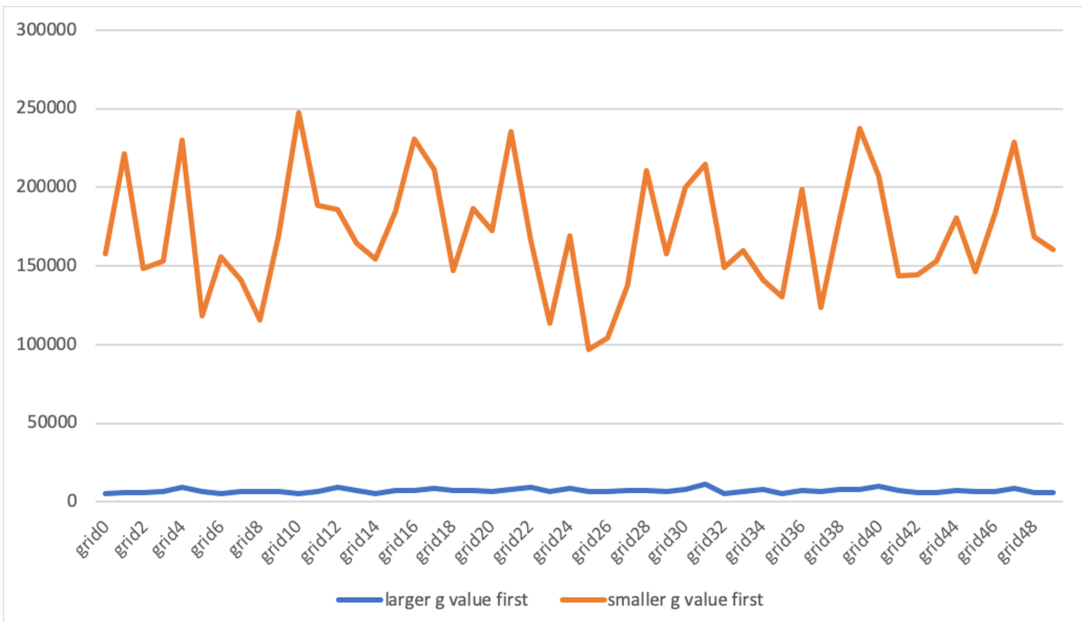


Figure 6: Larger g-value First VS. Smaller g-value First

	Larger g Value First	Smaller g Value First
Grid0	5143	158026
Grid1	5699	221645
Grid2	5658	148686
Grid3	6372	152865
Grid4	8990	230182
Grid5	6444	118485
Grid6	5288	155648
Grid7	6739	140752
Grid8	6686	115871
Grid9	6809	169949
Grid10	5211	247493
Grid11	6297	188744
Grid12	9288	186032
Grid13	7562	164466
Grid14	5402	154367
Grid15	7128	184198
Grid16	7323	230977
Grid17	8881	211361
Grid18	7502	147156
Grid19	7082	186820
Grid20	6876	172318
Grid21	8166	235267
Grid22	9499	166070
Grid23	6675	113657
Grid24	8536	169010
Grid25	6429	96871
Grid26	6229	104243
Grid27	7450	137528
Grid28	7088	210393
Grid29	6555	158057
Grid30	7992	199927
Grid31	11043	214862
Grid32	5338	149363
Grid33	6697	159772
Grid34	7790	141272
Grid35	5363	130260
Grid36	6907	198729
Grid37	6792	123928
Grid38	7893	180669
Grid39	7591	237231
Grid40	10087	207058
Grid41	7166	143943
Grid42	5803	144217
Grid43	5581	152945
Grid44	7290	180473
Grid45	6448	146368
Grid46	6509	183968
Grid47	8782	228515
Grid48	5955	168574
Grid49	6026	160548

Table 1: 50 experiments that comparing between set larger g-value first and smaller g-value first

Part 3 - Forward vs. Backward

Implement and compare Repeated Forward A* and Repeated Backward A* with respect to their runtime or, equivalently, number of expanded cells

We implemented 50 experiments on Repeated Forward A* and Repeated Backward A* within 50 different 101x101 gridworld and the environment is set following the instruction in Part 0.
From Table 2, we can clearly observe that the number of expanded cells on Repeated Forward A* is much smaller than the number of expanded cells on Repeated Backward A* by roughly look through the table. Which lead to the result that Repeated Forward A* is much faster than the Repeated Backward A*.

The reason for this is mostly because A* method relies on a heuristics function that usually states that the closer, the smaller the f-value is, the better and thus evaluate that cell first. As for the Repeated Backtrack A* method, it tried to generate the path from the goal the target point to the start point, however, it does not have the knowledge of whether the cells near its "start point" is blocked or not. And thus, it will generate a lot of unnecessary cells where the cells might be blocked and it does not know. Then, it decide the path based on all the cells' f-value and choose the smallest one regardless of whether the cell is blocked or not.

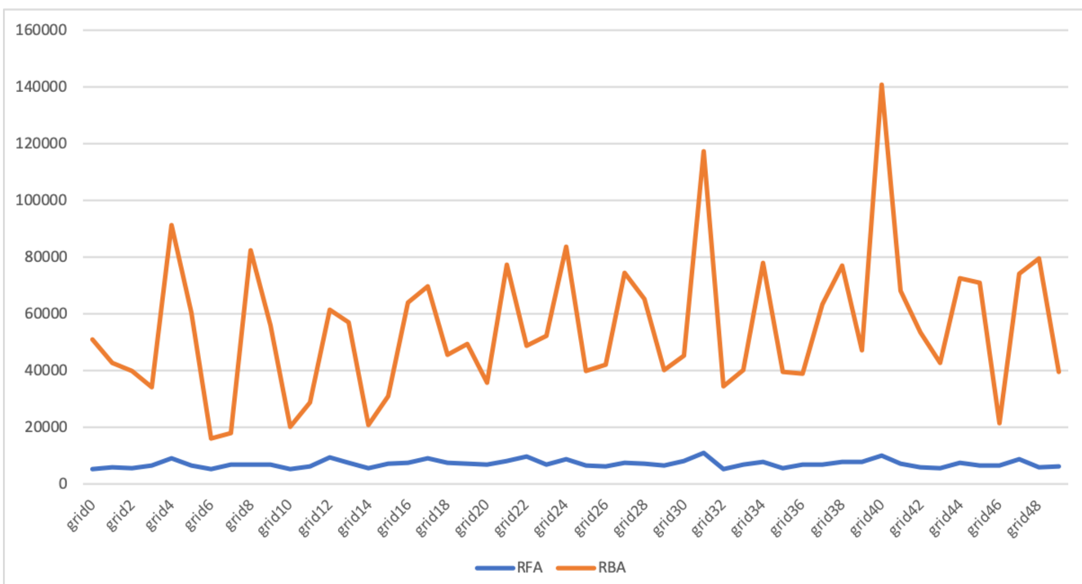


Figure 7: RFA VS. RBA

	RFA	RBA
Grid0	5143	50939
Grid1	5699	42795
Grid2	5658	39736
Grid3	6372	34004
Grid4	8990	91321
Grid5	6444	60508
Grid6	5288	15891
Grid7	6739	17828
Grid8	6686	82452
Grid9	6809	56048
Grid10	5211	20184
Grid11	6297	28706
Grid12	9288	61593
Grid13	7562	57143
Grid14	5402	20807
Grid15	7128	31020
Grid16	7323	64135
Grid17	8881	69592
Grid18	7502	45397
Grid19	7082	49284
Grid20	6876	35718
Grid21	8166	77215
Grid22	9499	48827
Grid23	6675	52114
Grid24	8536	83709
Grid25	6429	39786
Grid26	6229	42208
Grid27	7450	74587
Grid28	7088	65333
Grid29	6555	40031
Grid30	7992	45109
Grid31	11043	117450
Grid32	5338	34346
Grid33	6697	40283
Grid34	7790	78079
Grid35	5363	39416
Grid36	6907	38734
Grid37	6792	63370
Grid38	7893	77051
Grid39	7591	47055
Grid40	10087	140904
Grid41	7166	68124
Grid42	5803	53552
Grid43	5581	42731
Grid44	7290	72457
Grid45	6448	70893
Grid46	6509	21386
Grid47	8782	74002
Grid48	5955	79529
Grid49	6026	39656

Table 2: 50 experiments on RFA and RBA

Part 4 - Heuristics in the Adaptive A*

The project argues that "the Manhattan distances are consistent in gridworlds in which the agent can move only in the four main compass directions." Prove that this is indeed the case.
Suppose we are at the state S(x, y), since we have four main directions, the cost of moving to each direction is {[0, 1], [1, 0], [-1, 0], [0, -1]} and we can mark the next four state as S1, S2, S3, S4. The Manhattan distance for each is:

$$S1 - S = |(x + 0) – x| + |(y + 1) – y| = 1$$

$$S2 - S = |(x + 1) – x| + |(y + 0) – y| = 1$$

$$S3 - S = |(x + 0) – x| + |(y - 1) – y| = 1$$

$$S4 - S = |(x - 1) – x| + |(y + 0) – y| = 1$$

If Manhattan distance is not consistent, there must be a path which can be get smaller than 1. However, we have constraints in the direction so it could not happen. Remember it is the shortest path from S to S1, S2, S3, S4, meaning that there is no indirect path between them. And there is no diagonal ways or shortcuts can happen to shorten the cost of path. Assume we have another state T(a, b). The Manhattan distance

$$S - T = |x – a| + |y – b|$$

and always consistent in gridworld.

Prove that Adaptive A* leaves initially consistent h-values consistent even if action costs can increase:

For each compute path function in Repeated Forward A*, it computes the shortest path from current state to the target state under current knowledge of blocking cell in a gridworld.

hnew = f(target) − g(state)

Since we are exploring the world, the number of blocking cell can only increase, which means there is no other shorter way than the current computed path. Thus hnew is consistent.

Part 5 - Heuristics in the Adaptive A*

Implement and compare Repeated Forward A* and Adaptive A* with respect to their runtime

From Table 3, there is no obvious difference between the runtime of Repeated Forward A* with A* and the runtime of Repeated Forward A* with Adaptive A*, but if we take a closer look and compare between each Grid, we found that Repeated Forward A* with Adaptive A* is only slightly faster than the Repeated Forward A* with A*. (With only a few times where Repeated Forward A* with A is faster than the Repeated Forward A* with Adaptive A*)

And the reason for most of the time RFA with Adaptive A* is faster than A* is because Adaptive A* is more accurate since it updates the h-value of expended cells after every path generation. Moreover, the h-value is based on the current information about the blocked cells. The actual cost from the current state to the target is much closer to the hnew-value than Manhattan distance. It can help us better determine the order of f-value in the open list.

Thus, this is why Repeated Forward A* with Adaptive A* is sometimes faster than Repeated Forward A* with A*.

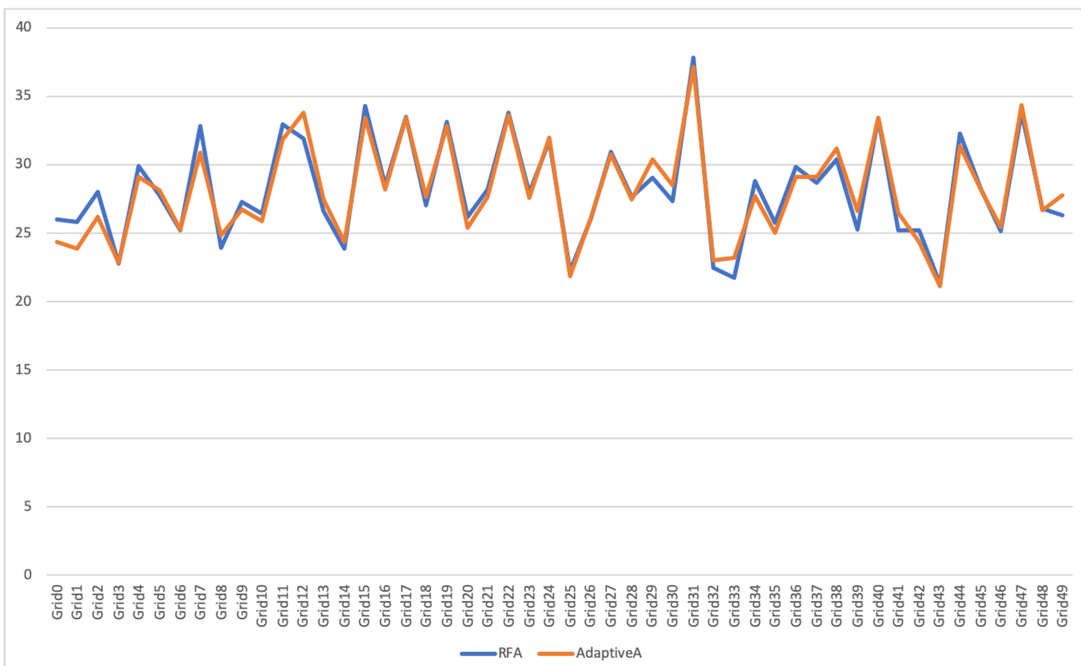


Figure 8: RFA vs. Adaptive A

	RFA	**AdaptiveA**
Grid0	25.96773815	24.32995224
Grid1	25.81692958	23.87904382
Grid2	28.02603531	26.19863009
Grid3	22.74367356	22.79548192
Grid4	29.86446452	29.10989237
Grid5	27.78339839	28.14819503
Grid6	25.20602465	25.23994613
Grid7	32.78997374	30.87176323
Grid8	23.92296624	24.81746745
Grid9	27.27751946	26.71721196
Grid10	26.42414951	25.85394955
Grid11	32.95952177	31.8677752
Grid12	31.90341735	33.78737974
Grid13	26.59002233	27.46237636
Grid14	23.84556365	24.32597709
Grid15	34.2613399	33.40574002
Grid16	28.47283769	28.20689321
Grid17	33.48170424	33.40335798
Grid18	27.04863429	27.64181757
Grid19	33.09418225	32.83549595
Grid20	26.17461896	25.41119003
Grid21	28.1649828	27.63770223
Grid22	33.77553725	33.58089733
Grid23	27.96880102	27.59864998
Grid24	31.72165155	31.93809843
Grid25	22.19373512	21.86650681
Grid26	26.07821918	26.10690641
Grid27	30.89845324	30.75308728
Grid28	27.6109798	27.42579794
Grid29	29.04140782	30.3666451
Grid30	27.34814548	28.50892591
Grid31	37.81713986	37.13845992
Grid32	22.45202327	23.016783
Grid33	21.71478462	23.19156408
Grid34	28.76521015	27.67972827
Grid35	25.76935482	25.03672981
Grid36	29.83836269	29.07770514
Grid37	28.65647769	29.10694313
Grid38	30.37919903	31.19070816
Grid39	25.27616191	26.60934019
Grid40	33.35931492	33.40206504
Grid41	25.21035123	26.47132397
Grid42	25.22128057	24.34904742
Grid43	21.31319666	21.10589504
Grid44	32.2693491	31.36138487
Grid45	28.22173405	28.21191669
Grid46	25.15070868	25.41393661
Grid47	33.77624011	34.34416461
Grid48	26.78401756	26.67752695
Grid49	26.27680802	27.76953053

Table 3: 50 experiments on RFA and Adaptive A

Part 6 - Memory Issues

Suggest additional ways to reduce the memory consumption of your implementations further.

In our current implementation of the gridworld, each cell contains five int values (g-value, h-value, f-value, x and y coordinates) and one pointer (pointer to parent). To improve the memory usage, we can get rid of the h-value and compute the h-value at runtime. This can save us 4 bytes of memory. Moreover, we can save x and y value into a single int, and we can extract the x and y value by using bit shift. This can also save us 4 bytes of the memory. Additionally, we can get rid of the pointer to the parent and save parent's positional coordinate. This will allow us to save 4 bytes of the memory.

Calculate the amount of memory that they need to operate on gridworlds of size 1001 1001 and the largest gridworld that they can operate on within a memory limit of 4 MBytes

Each of our cell contains 5 int values and 1 pointer, therefore, the memory usage for each cell will be:
5 * 4 + 8 = 28 Bytes. Total memory usage for grid size of 1001 * 1001 will be: 1001 * 1001 * 28 = 28.06 Mbytes.
The largest gridworld that can run with memory restriction of 4 Bytes:

$$(4 * 1024)/28 = 146.28$$

Hence, the largest gridworld that can run with memory restriction of 4 Bytes will be a 12 * 12 size grid.