CS:520 Repeated A* in Grid World

Project 1: Voyage into the unkown

Karan Ashokkumar Pardasani(kp955)
Naishal Chiragbhai Patel(np781)
Himaniben Hareshkumar Patel(hhp46)

September 28, 2021

[1]karan.pardasani@rutgers.edu
naishal.patel@rutgers.edu
himani.h.patel@rutgers.edu

Contents

1

Glossary

- Agent Grid : Grid updated by the agent drawing on the knowledge gained using its field of view during Repeated Froward A*. Initially, it is assumed that all cells are unblocked in this grid.
- Known Grid : Grid containing full knowledge of the environment.
- Final Discovered Grid : Grid formed after Repeated Forward A*, containing the knowledge of all cells exposed to agent's field of view. Also, all unexposed cells are considered blocked.
- Heuristic :
  - g(n) : shortest path length discovered from the *start cell* to *current cell.*
  - h(n) : heuristic value which estimates the remaining distance from *current cell* to *goal cell.*
  - f(n) : g(n) + h(n), i.e. total path length estimate from *start cell* to *goal cell*
- Manhattan Distance : d((x1, y1),(x2, y2)) = |x1 x2| + |y1 y2|
- Euclidean Distance : d((x1, y1),(x2, y2)) =

$$\sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

- Chebyshev Distance : d((x1, y1),(x2, y2)) = max(|x1 x2|, |y1 y2|).
- Traversed Path : Cells explored by agent while traversing planned path in all Repeated A* cycles.
- Tragectory Length : Length of traversed path.

2

Introduction

2.1 Problem Statement

For this problem we design a Grid World consisting of blocked and unblocked cells. Initially, the agent is not fully aware of the knowledge of the environment. Knowledge of environment (blocked and unblocked cells) must be learned by the agent as it moves through the grid world. As per our problem constraint agent will start from the top leftmost node and goal is to reach the bottom rightmost node of the grid. Field of view of the agent is limited to four directions: North, South, East, West. So as the agent moves along the path it can sense the knowledge of the environment based on its field of view and use it for future path planning.

2.2 Grid Environment

The grid environment is defined by constraint of knowledge. Agent gains knowledge of grid as it traverses through the grid and encounters blocks in its field of view which is one cell in main four directions(north, south, east, west).

2.2.1 Cell States

- Blocked cell : Cells agent can not occupy.
- Unblocked cell : Cells agent can occupy.

2.2.2 Parameters

The following parameters regulates the formation of the Grid World.

- Grid Dimension : It determines the size of the Grid World.
- Blocked cell density(p) : Its the probability that a given cell will be blocked.
- Start cell : Initial starting point in the Grid World for the agent.
- Goal cell : End Target of the agent.

3

Strategy

We implement Repeated Forward A* for traversing the Grid World to reach the end goal. Initially, the agent does not have any knowledge of the Grid Environment and assumes that there are no blockages and plans its path based on following two metrics:

- g(n) - shortest path length discovered from the *start cell* to *current cell*.
- h(n) - heuristic value which estimates the remaining distance from *current cell* to *goal cell*.
- f(n) - g(n) + h(n), i.e. total path length estimate from *start cell* to *goal cell*

As it traverses the environment, it updates the its knowledge acquired through its field of view. If encountered with a blockage in its planned path, path planning is done with the newly acquired knowledge of the environment taken into consideration. This strategy is applied until the agent reaches the goal node or the paths are exhausted which means a path from *start* to *goal* does not exist.

3.1 Question 1

Why does re-planning only occur when blocks are discovered on the current path? Why not whenever knowledge of the environment is updated?

The agent will traverse in real grid on the planned path derived from a* search algorithm on agent's grid(grid with partial knowledge of environment); so for every unblocked node traversed by the agent on this path will be an optimal choice so far,as A* is optimal(proof in next paragraph), and the blocked nodes encountered by agent which are not in the planned path will not affect our decision. When block is encountered in the planned path, only then the agent needs to re-plan the path based on the updated knowledge and again traverse on the new planned path from the current position.
A* algorithm implements a fringe as a priority queue and pops from the queue the node which has the least value of $f(n)$. Now, if the goal node is popped then all other paths will have total estimated path length, i.e.$f(n)$ more than

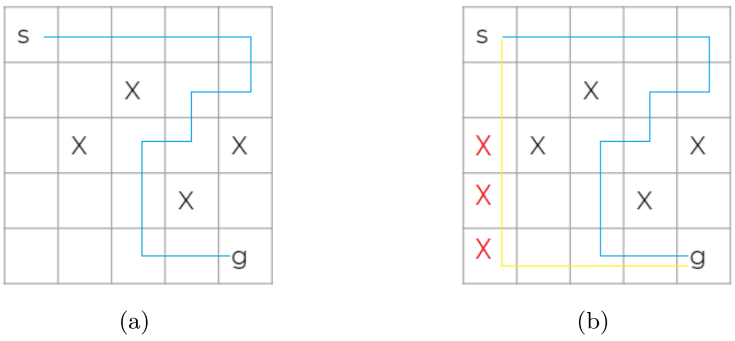


Figure 3.1: Question 3: Counter Example showing A* does not always give optimal path

the actual path the algorithm found from start to goal. Also, as the heuristic is admissible, the actual path length would be grater than or equal to the $f(n)$. Thus, the path length obtained when goal first popped from the fringe would be the most optimal for given knowledge and start position.

3.2 Question 2

Will the agent ever get stuck in a solvable maze? Why or why not?

In general the cycle of re-planing and calling A* algorithm is repeated until either the agent reaches the target or all reachable unblocked nodes are traversed. If the grid is solvable then the goal node will also be reachable; since all reachable unblocked nodes will be traversed eventually, the agent will be able to reach the target. Hence agent will never get stuck in a solvable gird.
Also, implementation of repeated A* search algorithm maintains a priority queue of children nodes based on h and g. When a node is popped from the priority queue we add that node in closed list.
Lets assume a case when agent comes across a long hallway(dead end), so all the nodes popped along this hallway are added to closed list. So for re-planing A* will avoid traversing through the nodes already present in closed list and hence avoiding infinite loop. In this way all the reachable unblocked nodes will be eventually traversed and agent will reach the goal given that the grid is solvable.

3.3 Question 3

Once the agent reaches the target, consider re-solving the now discovered Grid World for the shortest path (eliminating any backtracking that may have occurred). Will this be an optimal path in the complete Grid World? Argue for, or give a counter example.

Figure 3.1(a) represents a possible outcome of path from repeated A* search algorithm on grid 1. The cells marked in red in Figure 3.1(b) were never in the field of view of the agent during the repeated A* and thus never discovered. Therefore, they are marked as blocked during the re-solving phase. Now, when we apply A* on final discovered grid, we get the shortest path(blue) as shown in figure 3.1(b) when more optimal path(yellow) is available.

4

Implementation

4.1 Data Structures Used

4.1.1 Sorted Set

Open List,i.e. fringe, stores the visited nodes and pops it out according to it's priority and we implemented it using a sorted set as it gives us the best possible time complexity for push, pop and update operations simultaneously.
T(push) : O(logn)
T(pop) : O(logn)
T(update) : O(logn)

4.1.2 Set

Closed List contains the nodes that have been popped from priority queue and Visited List contains the nodes that have already been added in priority queue. We have used Set to implement and maintain both closed list and visited list.
Amortized add: O(1)
Amortized remove: O(1)

4.1.3 Deque

BFS(Breadth First Search) is implemented using Deque.
add: O(1)
popLeft: O(1)

4.2 A*

The A* function is passed two parameters:

- Start position
- $f(n)$

Initially, as the priority of the start node would be the highest, it would be explored first and its children will be generated. Among the children, the one with the lowest value of $f(n)$ will be explored first as it optimally guides the agent towards the goal. The algorithm stops if goal node is reached or all the reachable nodes are visited. If goal node is reached we will return the path found by backtracking from goal to source node otherwise in case if all reachable nodes are visited we can say that grid is unsolvable or no path is found from source to goal node.

4.3 Repeated Forward A*

The first step of Repeated Forward A* is to compute our planned path by performing A* on current node to the target based on agent's knowledge of environment(*Agent Grid*). After we get our planned path agent tries to traverse the planned path in *Known Grid* until:

- The goal is reached
  -> We return the path traversed by the agent from start to goal.
- Block is encountered
  ->We perform A* from the last unblocked node as our current node on the basis of the agent's updated knowledge.
- Fringe becomes empty
  ->It implies that all reachable unblocked nodes are traversed by the agent. Hence, a path from source to goal does not exist and the grid is unsolvable.

.

If goal is reached, we return the path followed from start to goal and if a block is encountered, we will perform A* from the last unblocked node as current node on the basis of the agent's updated knowledge.

5

Analysis

5.1 Question 4: Solvability

A gridworld is solvable if it has a clear path from start to goal nodes. How does solvability depend on p? Given dim= 101, how does solvability depend on p? For a range of p values, estimate the probability that a maze will be solvable by generating multiple environments and checking them for solvability. Plot density vs solvability, and try to identify as accurately as you can the threshold p0 where for p < p0, most mazes are solvable, but p > p0, most mazes are not solvable. Is A the best search algorithm to use here, to test for solvability? Note for this problem you may assume that the entire gridworld is known, and hence only needs to be searched once each.*

Here we will analysis how solvability depends on density(p).

1000 solvable grids of dimension 101*101 are analyzed using A* algorithm, assuming the grid environment is known to the agent, for p = 0.02 to p = 0.98 with 0.02 step size.

From Figure 5.1, we can observe that as Density of blocked cell increases, number of solvable grids decrease. We also observe that rate of decrease of solvable grids also decreases till density reaches 0.38 after which number of solvable grids quickly approaches zero.
Also, after density reaches 0.3, majority(more than 50%) of the grids become unsolvable.

5.2 Question 5: Heuristics

Among environments that are solvable, is one heuristic uniformly better than the other for running A? Consider the following heuristics:– Euclidean Distance, Manhattan Distance, Chebyshev Distance. How can they be compared? Plot the relevant data and justify your conclusions.

Here we will analysis how average success changes as we change the obstacle density.

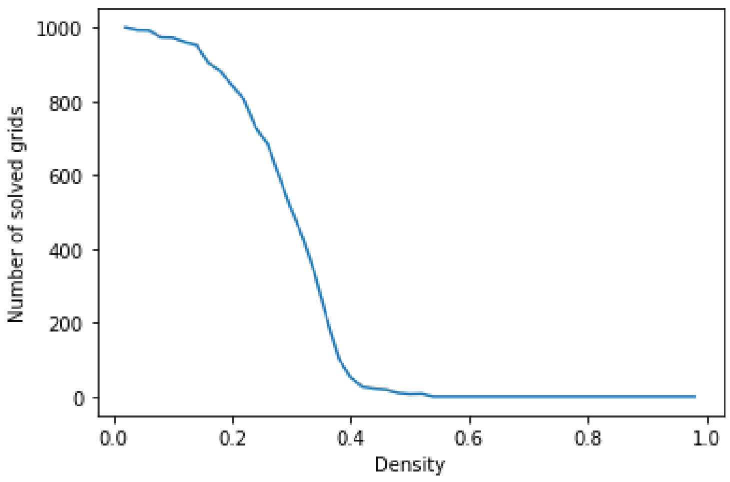


Figure 5.1: Question 4: Number of solved grids vs Density of Blocked cells

If we do not take into consideration the information provided to the agent through the heuristics, it essentially behaves like Dijkstra's algorithm. Using information about the goal sets A* apart from uninformed search algorithms and thus it can make better decisions for further traversing the search space.

Intuitively, we feel that using Manhattan Distance would give us less traversal time and less average number of cells as our problem constraint is consistent with the way distances are geometrically calculated. Additionally, Euclidean and chebyshev distances are too far an underestimate and hence do not provide more insightful information to the agent(given our problem constraint).

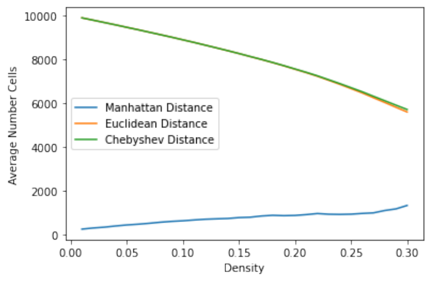
For example, take two triangles as shown in Figure 5.5. Here, a1 + b1 <= a2 + b2 but the hypotenuse of both the triangles are same. Now, to traverse from A to B, Euclidean Distance would give us an estimate c where Manhattan Distance would give us an estimate a1 + b1 and a2 + b2 which is how we actually move in the grid.

Also, as c > a1 + b1 and c > a2 + b1, Manhattan Distance is less relaxed and thus tighter bounded heuristic which implies it would give more optimal results.

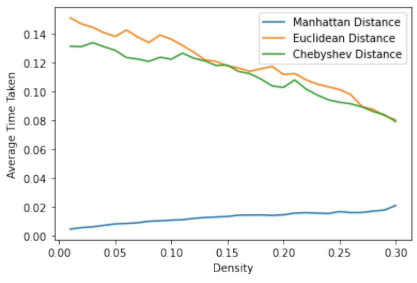
Here we test the given heuristics on following 3 metrics:

- Number of cells traversed to reach the *goal* node from *start* node.
- Time taken to reach the *goal* node from *start* node.
- Length of trajectory followed by the agent to reach the *goal* node from *start* node.

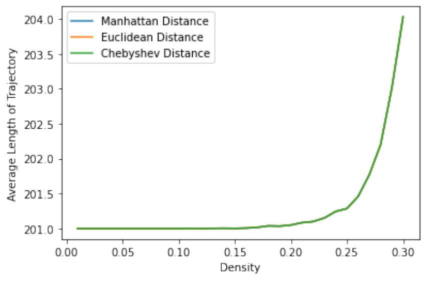
From Figures 5.2 and 5.3 we observe that the agent takes least time and traverses least number of cells when Manhattan distance is used as a heuristic for a solvable grid while Average length of trajectory is the same for all three heuristics.
Thus we conclude that Manhattan Distance is uniformly better than the other heuristics for running A* algorithm confirming our initial intuition.



(a)



(b)



(c)

Figure 5.2: Question 5: Result Graphs: (a) Average Number cells processed vs Density (b) Average Time taken vs Density (c) Average Length of Trajectory vs Density

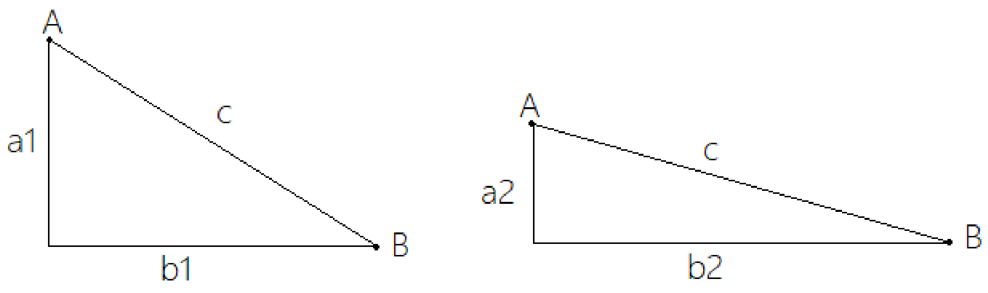


Figure 5.3: Question 5: Geometrical Illustration of Euclidean Distance vs Manhattan Distance

5.3 Question 6: Performance

Taking dim= 101, for a range of density values from 0 to min(p0,0.33),and the heuristic chosen as best in Q5, repeatedly generate gridworlds and solve them using Repeated Forward A. Use as the field of view each immediately adjacent cell in the compass directions. Generate plots of the following data: 1. Density vs Average Trajectory Length, 2. Density vs Average (Length of Trajectory / Length of Shortest Path in Final Discovered Gridworld), 3. Density vs Average (Length of Shortest Path in Final Discovered Gridworld / Length of Shortest Path in Full Gridworld), 4. Density vs Average Number of Cells Processed by Repeated A*. Discuss your results. Are they as you expected? Explain.*

1. As density increases the probability of getting not only a more direct path decreases but also the probability of getting multiple or any path to the goal at all also decreases(as seen in figure 5.4(a)). Also, the probability of bumping into a block increases and as a result, backtracking increases which implies the increase in average trajectory length.

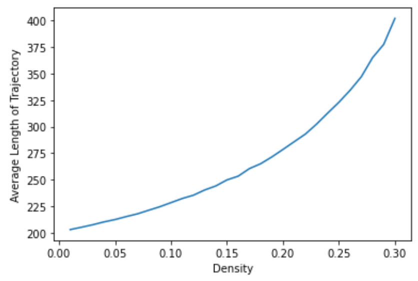
2. The length of shortest path in the final grid world can be found using backtracking on the trajectory path and eliminating all the extra nodes traversed(loops). So the ratio of length of trajectory and the length of shortest path in the final discovered grid world will always be greater than or equal to one. And with the increase in density of blocks both the length of trajectory and the length of shortest path in the final discovered grid will increase; since the rate of increase in the length of trajectory is more than the rate of increase in the length of the shortest path in the final discovered grid, the overall ratio will increase (as seen in figure 5.4(b)).

3. As the knowledge of unblocked nodes in the final discovered grid will be less than that of fully discovered grid; the length of shortest path in final discovered grid will always be more than or equal to the length of shortest path in fully discovered grid. With the increase in the density of the blocks both length of shortest path in the final discovered grid and the length of shortest path in fully discovered grid will increase; but the rate of increase of the former is more than the latter and so the overall ratio will increase (as shown in figure 5.4(c)).

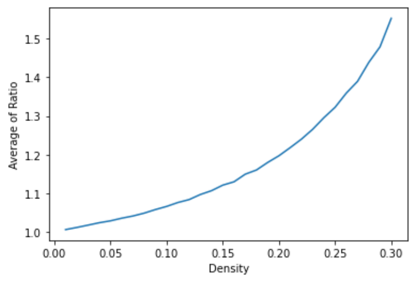
4. As the density increases, number of blocked cells increase and therefore the probability of bumping into a block increases. Thus, re-planning needs to be done more often and this increases the number of cells processed (as shown in figure 5.4(d)).

5.4 Question 7: Performance Part 2

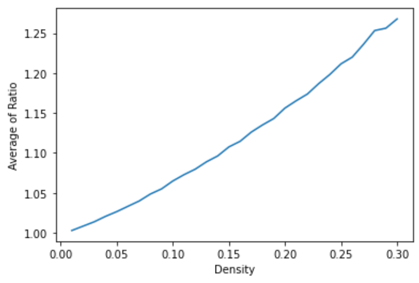
Generate and analyze the same data as in Q6, except using only the cell in the direction of attempted motion as the field of view. In other words, the agent may attempt to move in a given direction, and only discovers obstacles by bumping into them. How does the reduced field of view impact the performance of the



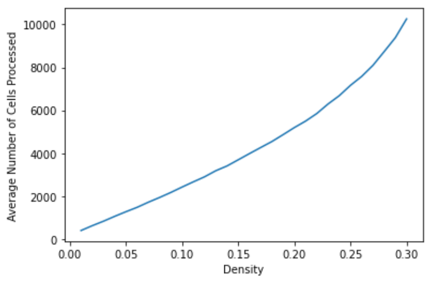
(a)



(b)



(c)



(d)

Figure 5.4: Question 6: Result Graphs Repeated A* 4-Directional field of view: (a) Average trajectory length (b) Ratio of trajectory by shortest path in discovered gridworld (c) Ratio of shortest path in discovered gridworld by shortest path in complete gridworld (d) Average cells processed

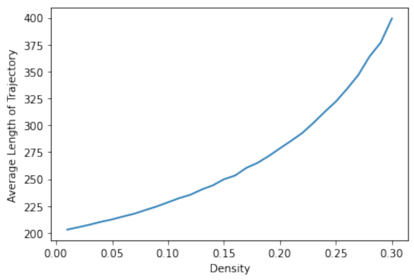
algorithm?

1. With the change in field of view of agent in the direction of path, the knowledge of environment updated will be lesser as compared to 4-direction field of view. This may cause agent to call repeated A* more number of times (due to lesser knowledge). But eventually in both the cases agent will choose same path only; just in case of field of view in path direction the agent may have to call repeated A* multiple times before traversing on the same path as in case of 4-direction field of view. So there will be no change in the average length of trajectory with the change in field of view.

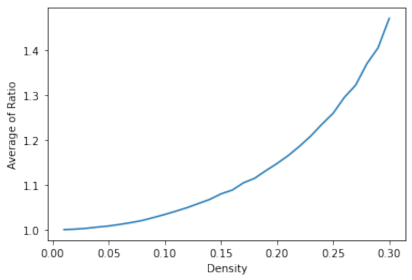
2. The knowledge of environment in the final discovered grid in case of field of view in direction path will be less than that with 4-direction field of view. As we can see in figure 5.6, if agent has field of view in 4-direction, the knowledge of nodes highlighted with yellow will be updated in final discovered grid but in case of field of view in the direction of path, these nodes will not be updated in final discovered grid. Thus the length of the shortest path in the final discovered grid in case of 4-direction world will be less(shown by green line in figure 5.6) and with the case of field of view in direction of path the length of shortest path will be more(shown by red line in figure 5.6). Therefore, the length of shortest path in this case might be greater than or equal to the case with 4-direction field of view. So the overall ratio of length of trajectory and the length of shortest path in the final discovered grid will be less than or equal to the case of 4-direction field of view.

3. The length of shortest path in fully discovered grid will be same in this case and the case with 4-direction field of view. But as stated in point 2; the length of shortest path in final discovered grid might be lesser in this case; hence the overall ratio of the length of shortest path in final discovered grid and the length of shortest path in fully discovered grid will increase in this case as compared with the case of 4-direction field of view.

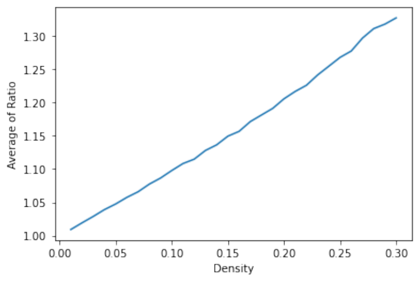
4. As mentioned in point 1; with the reduction in field of view agent might have to call repeated A* more number of times as compared to 4-direction field of view. Hence, with the increase in number of calls of repeated A*, the number of cells processed will also increase.



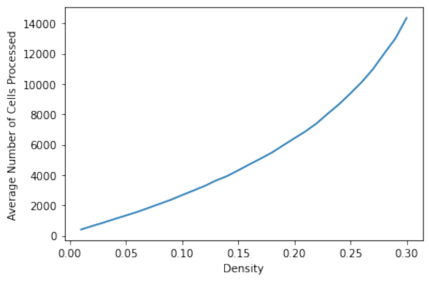
(a)



(b)



(c)



(d)

Figure 5.5: Question 7: Result Graphs Repeated A* field of view in direction of path (a) Average trajectory length (b) Ratio of trajectory by shortest path in discovered gridworld (c) Ratio of shortest path in discovered gridworld by shortest path in complete gridworld (d) Average cells processed

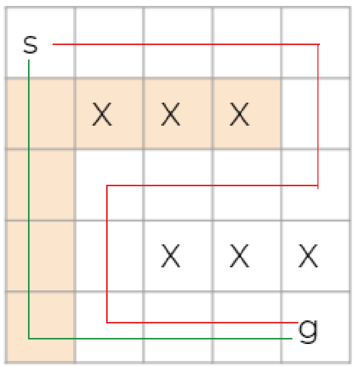


Figure 5.6: Question 7: Comparison of length of shortest path for different field of view

5.5 Question 8: Improvements

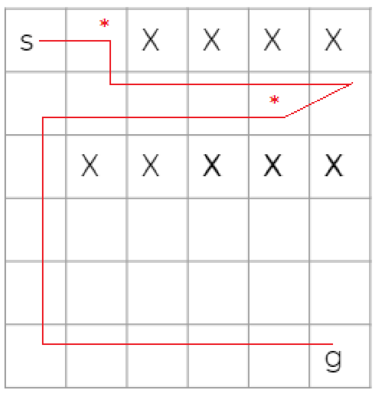
Repeated A may suffer in that the best place to re-start A* from may not be where you currently are - for instance if you are at the dead end of a long hallway, you can save some effort by backtracking to the end of the hallway (recycling information you already have)before restarting the A* search. By changing where you restart the search process, can you cut down the overall runtime? What effect does this have on the overall trajectory (given that you have to travel between the current position and the new initial search position)?*

In usual scenario we start re-planning from the last unblocked cell but in case the agent bumps into a block in the end of a long hallway, we will have to re-plan for all the nodes in the entire hallway(as shown in figure 5.7). In order to overcome unnecessary traversal to the start of the hallway; the search process can be restarted from the entry point of the long hallway. This way agent will have to traverse less nodes and still might get the same planned path. The length of trajectory in both the cases will remain the same since by backtracking the path will be same in both the cases. Since search is restarted from the entry point of the long hallway the number of nodes processed will be less in this case as compared to the usual scenario.

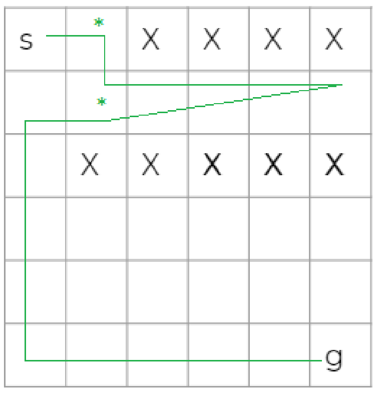
When the block density is less, even the change in restart position of search will not greatly affect time taken as long hallway formation probability would be very less at low block density. But as the density of blocks increases this improvement in algorithm will give us better results as compared to usual scenario(as shown in figure 5.7).

5.6 Question 9: Heuristics

A can frequently be sped up by the use of inadmissible heuristics - for instance weighted heuristics or combinations of heuristics. These can cut down on runtime potentially at the cost of path length. Can this be applied here? What is the effect of weighted heuristics on runtime and overall trajectory? Try to reduce*

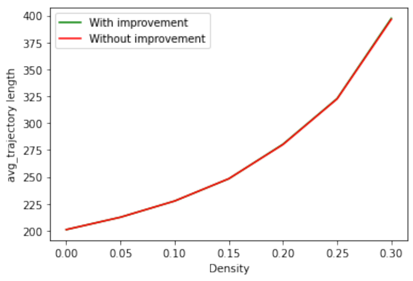


(a)

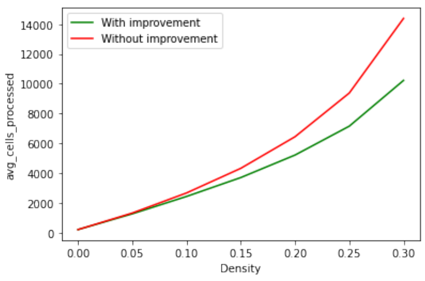


(b)

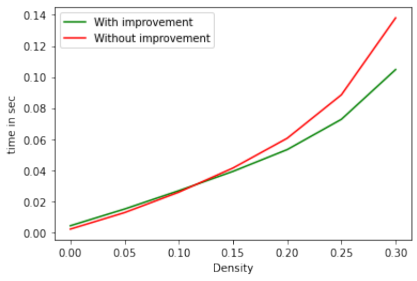
Figure 5.7: Question 8: Illustrations: (a)Repeat A* from last unblocked cell. (b)Repeat A* from hallway entry point



(a)

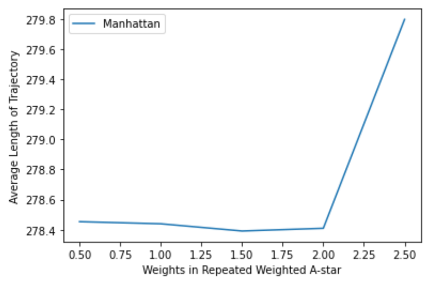


(b)

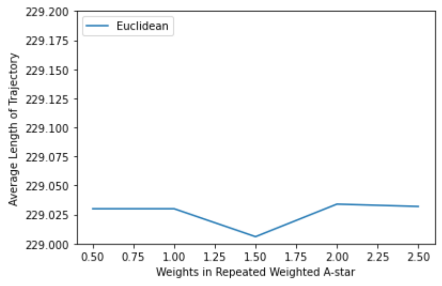


(c)

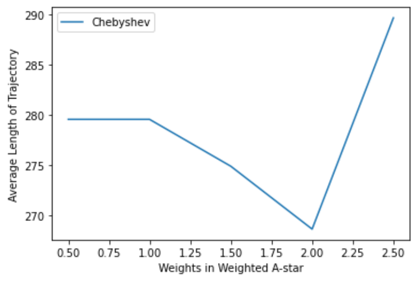
Figure 5.8: Question 8: Result Graphs: (a) Average Number cells processed vs Density (b) Average Time taken vs Density (c) Average Length of Trajectory vs Density



(a)



(b)

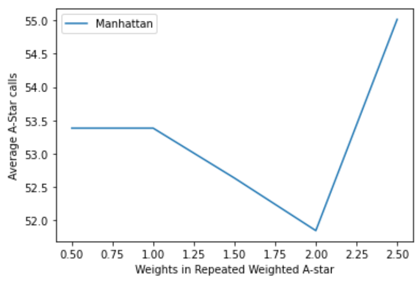


(c)

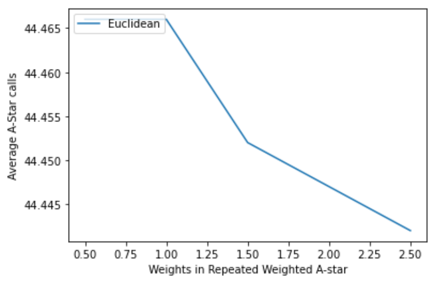
Figure 5.9: Question 9: Repeated A* : Average length of trajectory for different weighted heuristics

the runtime as much as possible without too much cost to trajectory length.

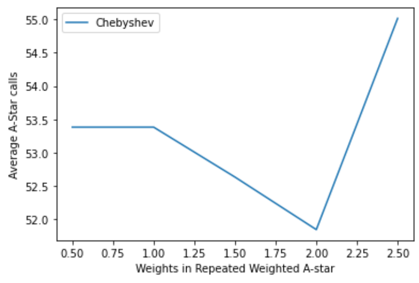
An inadmissible heuristic might overestimate, at certain nodes, the cost of reaching the goal. Thus, the agent might ignore a more optimal path and end up reaching the goal from a longer path.
We can confirm our intuition from the graphs that with the change in weights of heuristics; the average length of trajectory in case of Manhattan remains almost same initially and then increases after w1:w2($f(n) = w1g(n) + w2h(n)$) reaches 1/2. In case of Euclidean, we can observe a slight decrease in average length of trajectory for some weights(as shown in figure 5.9). In case of Chebyshev, the average length of trajectory decreases for some weights and then increases rapidly. Refer to figure 5.10
Average number of A* calls will decrease slightly and then increases rapidly after w1:w2 reaches 1/2. In case of Euclidean the average number of A* calls decreases with the increase in weights. For Chebyshev, the average number of A* calls decreases for some weights and the again increases. Refer to figure 5.11
Average time taken will increase with the increase of weights in case of Manhattan. For euclidean, the average time taken will decrease with the increase in weights and remains constant after w1:w2 is 1/1.5. In case of Chebyshev, average time taken decreases mostly with the increase in weights.



(a)

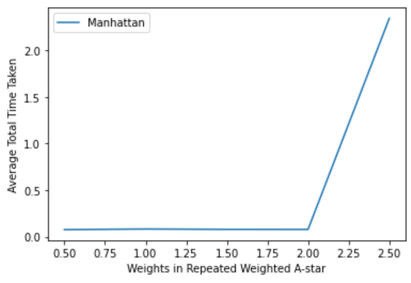


(b)

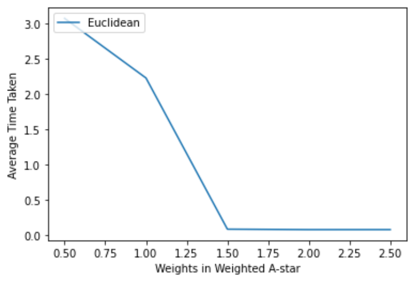


(c)

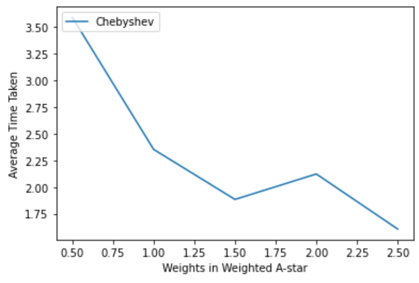
Figure 5.10: Question 9: Repeated A* : Average number of A* calls for different weighted heuristics



(a)

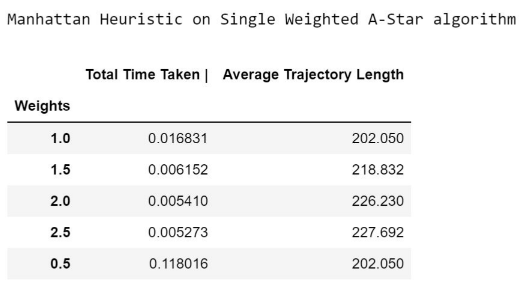


(b)



(c)

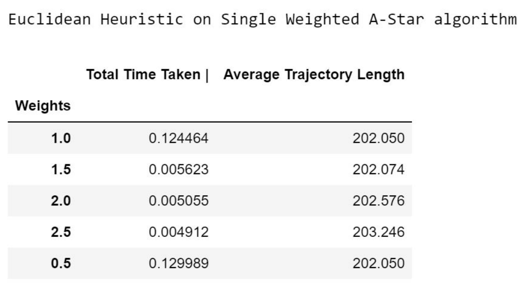
Figure 5.11: Question 9: Repeated A* : Average time taken for different weighted heuristics



Manhattan Heuristic on Single Weighted A-Star algorithm

Weights	Total Time Taken |	Average Trajectory Length
1.0	0.016831	202.050
1.5	0.006152	218.832
2.0	0.005410	226.230
2.5	0.005273	227.692
0.5	0.118016	202.050

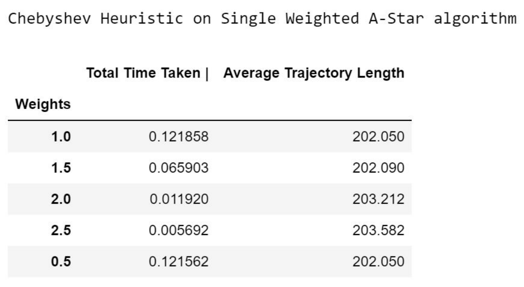
(a)



Euclidean Heuristic on Single Weighted A-Star algorithm

Weights	Total Time Taken |	Average Trajectory Length
1.0	0.124464	202.050
1.5	0.005623	202.074
2.0	0.005055	202.576
2.5	0.004912	203.246
0.5	0.129989	202.050

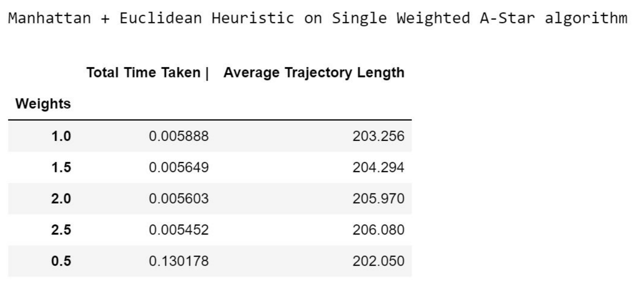
(b)



Chebyshev Heuristic on Single Weighted A-Star algorithm

Weights	Total Time Taken |	Average Trajectory Length
1.0	0.121858	202.050
1.5	0.065903	202.090
2.0	0.011920	203.212
2.5	0.005692	203.582
0.5	0.121562	202.050

(c)



Manhattan + Euclidean Heuristic on Single Weighted A-Star algorithm

Weights	Total Time Taken |	Average Trajectory Length
1.0	0.005888	203.256
1.5	0.005649	204.294
2.0	0.005603	205.970
2.5	0.005452	206.080
0.5	0.130178	202.050

(d)

Figure 5.12: Question 9: Single Weighted A-Star Results: (a) Manhattan (b) Euclidean (c) Chebyshev (d) Manhattan + Euclidean

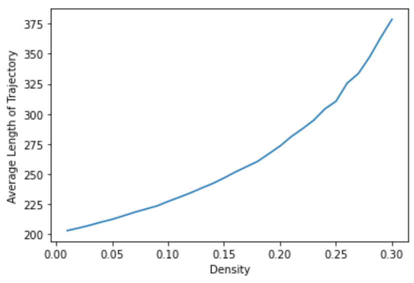
6

Extra Credit

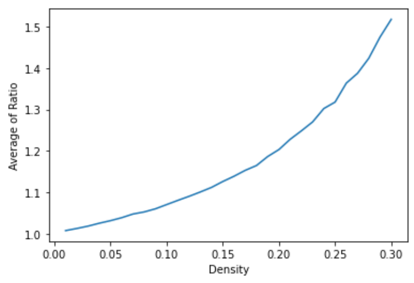
Repeat Q6, Q7, using Repeated BFS instead of Repeated A. Compare the two approaches. Is Repeated BFS ever preferable? Why or why not? Be as thorough and as explicit as possible.*

After implementing repeated BFS on the same dataset used for Q6 and Q7, we observed that there was no significant change in the average length of trajectory, ratio of length of trajectory to length of the shortest path on final discovered grid and ratio of length of shortest path in the final discovered grid to length of shortest path in the fully discovered grid.

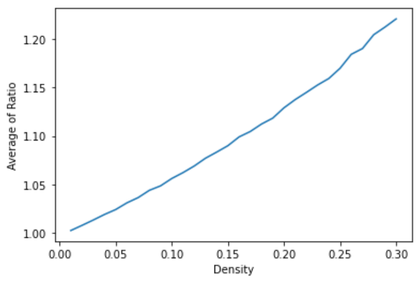
The only metric that differs majorly between repeated A* and repeated BFS is the Average number of cells processed. The average number of cells processed in BFS are significantly higher and hence repeated BFS is not preferable over repeated A* in any scenario since their performance is almost similar; in fact the average number of cells processed and the time taken in case of repeated BFS are significantly higher. In other words we can say that A* behaves as BFS in worst case scenario and so average case and best case of A* will be significantly better and hence A* will always be preferred over BFS.



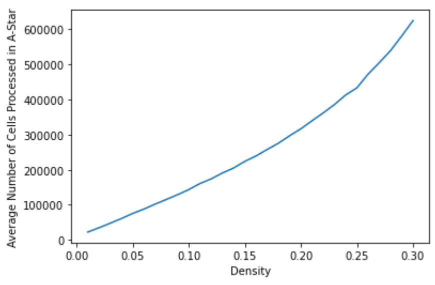
(a)



(b)

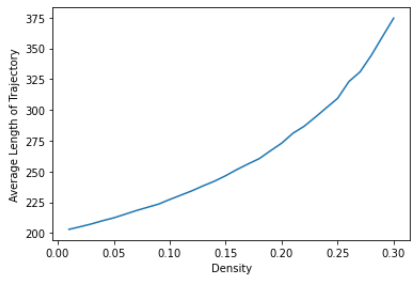


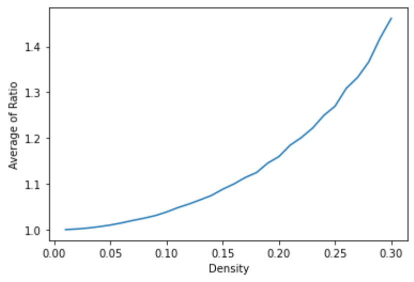
(c)



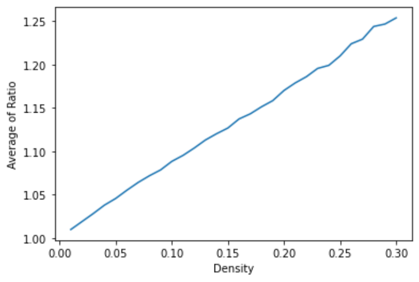
(d)

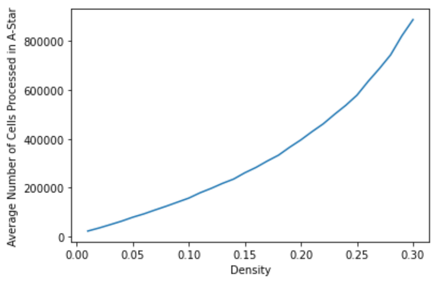
Figure 6.1: Extra Credit Result Graphs BFS 4-Directional field of view: (a) Average trajectory length (b) Ratio of trajectory by shortest path in discovered gridworld (c) Ratio of shortest path in discovered gridworld by shortest path in complete gridworld (d) Average cells processed





(a) (b)





(c) (d)

Figure 6.2: Extra Credit Result Graphs BFS field of view in direction of path: (a) Average trajectory length (b) Ratio of trajectory by shortest path in discovered gridworld (c) Ratio of shortest path in discovered gridworld by shortest path in complete gridworld (d) Average cells processed

7

Applications

Different heuristics and their combinations can be used according to the problem statement and the requirements.

1. One such application can be a house robot. The placing of things like furniture does not change frequently in a house and thus finding an optimal path once, even if it takes more time is more efficient in the long run. One can train the robot initially for finding optimal path for constant obstacles(example: walls, furniture, etc.) which remains in the memory of the robot and then can use heuristics that take less time for dynamic obstacles(example: humans, pets, etc.).

2. Another application of the algorithm would be in rescue operations, where rescuing someone in shorter time is more important than finding optimal path.