NAME: **PARTH KALPESH KHARKAR**
NET ID: **pk674**

16: 332:573 – DATA STRUCTURES AND ALGORITHMS

FINAL PROJECT REPORT

COMPARISON BETWEEN A* ALGORITHMS

INTRODUCTION

Search Algorithm is a method to find solutions to problems by exploring various states of the problem domain. Search algorithm starts with an initial state in search of the solution to the problem while searching for solution it uses various data structures such as breadth-first-search, depth-first-search, or a heuristic search. There are various types of search algorithms and A* algorithm is one of them.

A* is a type of search algorithm also known as a Heuristic Search Algorithm. A* is one of the popular path-finding algorithms which are used to find the shortest distance between the source node and destination node. A heuristic function is used to calculate the estimate distance between the source node and the destination node. The simple A* algorithm is a deterministic algorithm, so which means that it knows very well about its environment, but in this case, we are finding a path in an indeterministic state. A* is one of the smartest algorithms compared to other conventional algorithms.

There are various parameters which are involved while A* algorithms is implemented such as the heuristic function which is used to calculate the distance between the source node and destination node. F, G and H are the functions which are used to evaluate the cost of a path from the source node to destination node in the grid that we are going to evaluate on. Here, we also need an open list which will store the nodes that are visited by A* but are not yet expanded as well as we will need closed list of nodes that we have been visited and expanded. For the source node to reach the destination node we will need path which would be a list of nodes representing a route between both the nodes and finally we will need function that will evaluate each node based on its cost and heuristic value calculated using the formula. With all these parameters, we can search for the optimal path to travel from the source node to the destination in the grid. The algorithm is designed in a way wherein it calculates the node with the lowest code and then proceeds to the next node. This evaluation is performed until it has reached the destination node.

PROCEDURE TO PERFOM A* ALGORITHM:

STEP - 1: INITIALIZING AN OPEN SET AND ADDING THE SOURCE NODE TO IT

Here, we will be creating an open list of nodes that have been found but are not evaluated. We will start by initializing the open list and adding the source node to it. The source node is the starting point of our algorithm.

STEP – 2: INITIALIZING A CLOSED SET AS AN EMPTY SET

Here, we will be initializing a closed set of nodes which have been already established. It is an empty set at this moment because we haven't really evaluated any nodes.

STEP – 3: FUNCTION USED TO EVALUATE COST PATH

f(n) – represents the cost of the cheapest solution path. It is represented as f_score or 'f'.
g(n) – represents the actual cost of the path from source node. It is represented as g_score or'g'.
h(n) – heuristic function used to estimate the cost of the cheapest path from node 'n' to the destination node

Initialize the g_score to 0 and f_score to the heuristic value of the source node. The g_score here is the cost of the path from the source node to the destination node. The reason behind setting the g_score to 0 is that we haven't moved from the start node yet. The f_score is the calculated as the sum of the g_score and the heuristic function.

STEP – 4: ITERATING THE A* ALGORITHM

As the open list is still not empty, we will keep on iterating over the list until we find the destination node or the optimal path to the destination node. We will start by accounting the node which has the lowest f_score because it is the node that will lead us to our destination node. In case the node with lowest f_score is our destination node that means we have successfully implemented our A* algorithm and will return the path which can be used to get the destination node. By the help of this node, we can now construct the path which we needed using the parent pointers of the node from the destination node to the source node.

We will now move the selected node from the open list to the closed list as we have completed the search. The whole process follows a simple structure wherein we evaluate each adjacent node to the selected node to know if the adjacent node is worth exploring as an option to get to the destination node. If in case the adjacent node is already in the closed list, we can ignore it as it is already evaluated, or we can calculate its tentative g_score and update only if it is necessary. The tentative g_score is calculated as the sum of the g_score of the selected node and the cost of the edge between them.

STEP – 5: RETURN FAILURE IF OPEN LIST EMPTY

If in case we have exhausted all the possible nodes which could have been used to find the optimal path for the source node to reach to the destination node, we return failure.

In this whole procedure, we are exploring the grid trying to evaluate an optimal path between the source node and the destination node. Here, by optimal path it means that we are searching for a path with the lowest cost. The two lists which we are maintaining (open list and closed list) are used to keep a track of the nodes which are explored and yet to be explored. The important factor to implement A* algorithm is the heuristic function which used to calculate the estimate between the given node and the destination node. The A* algorithm will be more efficient only if the heuristic function is accurate which will then help us to find the optimal path. Basically, heuristic function guides the A* algorithm in its searching process.

There are various types of A* algorithm amongst which in this project we are going to implement the Repetitive Forward A* Algorithm and the Adaptive A* algorithm. In this project we will be implementing both the types and compare them to know which one more efficient and why.

REPETITIVE FORWARD A* ALGORITHM

Forward A* is one of the potential variants of the A* algorithm that is uses to find the optimal path between the source node and the destination node by the exploring the grid. Repetitive Forward A* algorithm is an extension of the Forward A* algorithm. It is particularly designed to handle the dynamically changing environment of the grid.

The idea behind implementing a repetitive forward A* algorithm is that it runs the forward A* algorithm repeatedly from the source node to the destination by always finding a new path when the previous path is found to be incorrect. Once the path is found to be incorrect, new A* algorithm search is started from its current node. This process is continued until it reaches the destination node.

Initially, the simple A* algorithm is used to find the path without having any knowledge about the environment. Once, the path is found the grid agent tries to traverse this path to the destination node. During the process, it updates its knowledge about the environment and if the current path which is traversing is found to be blocked by an obstacle or incorrect, it again implements the A* algorithm from the current node. The key difference we can find while implementing this algorithm is that it considers the uncertainty in the environment. Here, the agent does not know about the environment and figures it out while traversing the path.

PROCEDURE TO PERFORM REPETITIVE FORWARD A* ALGORITHM

STEP – 1:
Grid Agent does not have any idea about the environment except its source node and destination node.

STEP – 2:
Grid Agent uses the A* Algorithm to find the path from source node to destination node with limited information.

STEP – 3:
The Grid agent starts traversing the path and, at the same time, learning about the environment and updating its information about the environment. If the agent reaches the destination, we terminate the algorithm with a return Success.

STEP - 4:
If the path is blocked, we again use the A* algorithm to trace a path from the current node to the destination node.

STEP – 5:
If there is no node to trudge upon by the grid agent, we terminate the algorithm with return failure. Else we again go back to step 3

ADAPTIVE A* ALGORITHM

Adaptive A* algorithm is another type of the A* algorithm that uses adaptive heuristic implementation to improve the search efficiency. The idea behind using Adaptive A* algorithm is to use the information of its previous search and refine the heuristic estimate of the remaining cost from a node to the destination node.

The reason behind using Adaptive A* algorithm is that the heuristic function implemented in the ordinary A* algorithm can sometimes overestimate or underestimate the respective cost leading to find an inefficient path. The use of Adaptive A* algorithm can actually improve the accuracy of the heuristic function implemented and can help us achieve a better performance.

In Adaptive A*, the heuristic function is updated after every search iteration which is based on the actual cost of the node to the destination node. This implemented based on the previous search experience. The actual cost is used as a corrective factor for the heuristic estimate which can help the next search to be more efficient and accurate.

PROCEDURE TO IMPLEMENT ADAPTIVE A* ALGORITHM:

STEP -1:
Grid Agent does not know about the environment except its source node and destination nodes.

STEP – 2:
Grid Agent uses the A* Algorithm to find the path from source to destination with limited information.

STEP – 3:
The Grid Agent starts traversing the path and, at the same time, learning about the environment and updating its information about the environment. If the agent reaches the destination, we terminate the algorithm with a return Success.

STEP - 4:
If the path is blocked, we use the expression h(expanded_node) = g(destination_node) − g(expanded_node) to update our heuristics. Then we again use the A* algorithm to trace a path from the current node to the destination node.

STEP – 5:

If there is no node to trudge upon by the agent, we terminate the algorithm with return failure. Else we again go back to step 3

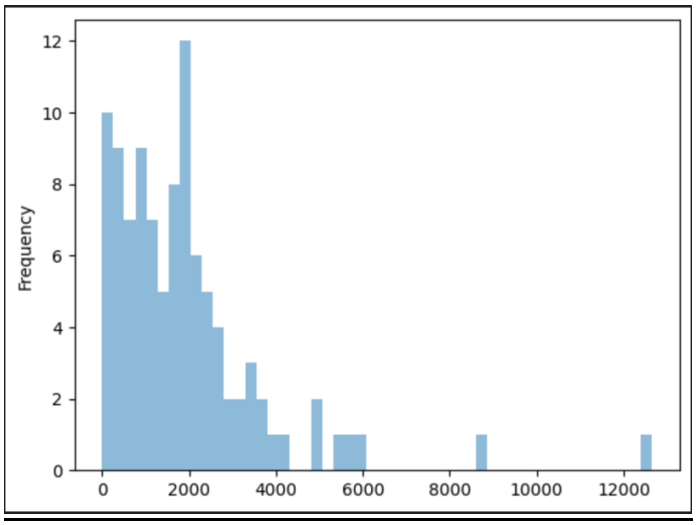
DIFFERENCE BETWEEN REPETITIVE FORWARD A* AND THE ADAPTIVE A* ALGORITHMS

- REPETIVE FORWARD A* ALGORITHM:
  - The Repetitive Forward A* Algorithm uses the information from the previous search iteration to increase the speed of the search.
  - The previous search information is used as a guiding tool for the present search iteration of the algorithm.
  - Using this method decreases the sequence of weights and increases the search space in order to find the optimal path between the source node and the destination node.

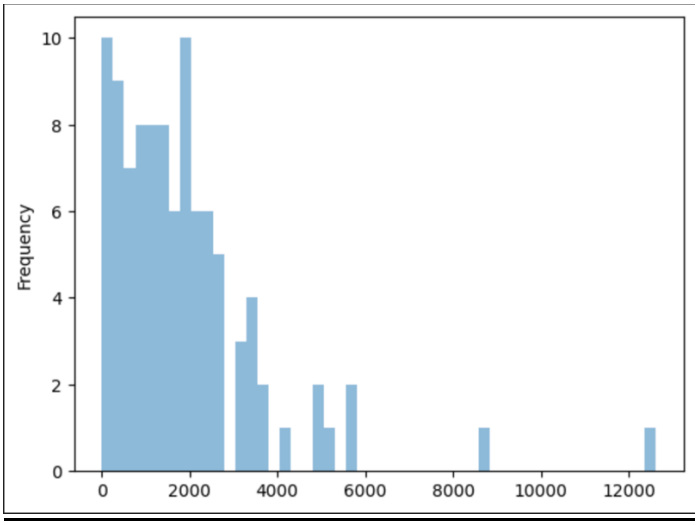
- ADAPTIVE A* ALGORITHM:
  - The Adaptive A* Algorithm uses the result of the previous search iteration to adapt the heuristic estimate of the remaining cost to improve the efficiency of the search.
  - The heuristic is being updated on the actual cost from the node to the destination node which is found during the previous search iteration.
  - Here, the learning rate parameter is used to control the balance between the previous search heuristic estimation and the new information which becomes available to the algorithm.

- MAIN DIFFERENCE BETWEEN BOTH THE ALGORITHMS:
  - The key difference between the Repetitive Forward A* algorithm and the Adaptive A* algorithm is that the repetitive forward algorithm reuses the information from the previous search iteration, whereas the Adaptive algorithm is implemented to focus on the heuristic estimate of the remaining cost of the destination.
  - The Repetitive forward algorithm does use any previous search iteration information to update the heuristic estimate but focuses on implementing a series of forward searches with each starting from the beginning. It may be slower than the Adaptive A* algorithm.
  - The unique factor about using the Repetitive Forward A* algorithm is that it can be memory efficient as it stores information from the current node and the pervious search iterations.
  - The Adaptive A* Algorithm requires an additional memory which is used to store the expanded nodes and the updated heuristic values.

COMPARISON BETWEEN REPETITIVE FORWARD A* AND ADAPTIVE A* ALGORITHMS:



REPETITIVE FORWARD A* ALGORITHM



ADAPTIVE A* ALGORITHM

- As seen from the graphs above, we can discern the expanded cells covered in each computation by both algorithms. However, it is difficult to draw and exact comparison among the two, much less determine which of the two are more efficient.
- This means that we need to develop a specific metric that analyzes the graphs to give us a more astute value. For this, the method that would be most appropriate would be a t-test, followed by a p-value generation to denote our hypotheses.
- In the case of our following calculations, our hypothesis would be that the means of the two representations above are statistically and significantly different. We value this hypothesis with a 0.95 confidence, which means we need to garner a p-value that is less than 0.05 to achieve our hypothesis correctly.
- Based on our computations, we found the following results –

```
T-statistic: 0.09014738195527473
P-value: 0.46482865030341813
```

  While our t-test score is significant in value, our p-value score fails to reject the null hypothesis. This means that through this calculation, we were unable to discern any significant difference between the two means. While there could be a myriad of factors affecting this p-value, we assume that the lack of comprehensive features might be a good cause. Therefore, this method does not provide us with a good conclusion.
- Given that our initial test did not achieve a good result, we go for a simpler assumption. Both computations can be tallied and checked for a larger sum. Whichever algorithm has more expanded cells in its comparison is the weaker algorithm.

- Based on our computations, our results are listed below –

```
THE NUMBER OF TIMES ADAPTIVE IS BETTER THAN FORWARD:  67
THE NUMBER OF TIMES FORWARD AND ADAPTIVE HAD A TIE:  22
THE NUMBER OF TIME FORWARD IS BETTER THAN ADAPTIVE:  11
```

  As seen above, the adaptive A* algorithm took fewer guesses while computing its expanded cells. While repetitive forward A* algorithm did perform better in certain cases, the difference between the two algorithm's efficiency is stark.
- Through these computations, we can safely conclude that Adaptive A* algorithm is more efficient than Repetitive Forward A* algorithm.

CONCLUSION

In this report, I have experimented with two main algorithms – Repetitive Forward A* algorithm and Adaptive A* algorithm. Through our computations, we found that both algorithms can generate optimal paths between the source node and destination node, but we wanted to prove which one was more efficient. We tried two means. We started by generating p-value to determine statistical difference in the means of both algorithms. This method did not signify towards our hypothesis; hence we went with a simpler assumption to calculate the difference in expanded cell computation of each algorithm to check which is more efficient. We conclude through this report that Adaptive A* algorithm is a more efficient algorithm at pathfinding.