Comparing Performance of MHA* and A*

Yashasvi Baweja
2015116

Mohit Agarwal
2015060

***Abstract*—Heuristic selection is an important step in search algorithms. Performance of an algorithm can vary substantially depending on what heuristic is selected. While one heuristic may solve the problem efficiently (in terms of time, objective), other might cater to only one of these aspects or maybe none. Therefore choice of the heuristic function is one of the most major decision. In such a case, comparison of Multi heuristic A* (MHA*) and A* seems an interesting one. We have compared the results of this algorithm with A* on a large uniform cost graph, tile-sliding puzzle and taxi-pathfinding problem.**

Keywords*—A*, MHA*, Heuristic, graph, tile-sliding, path-finding

I. Introduction

Motivated by our intention to compare the combination of different heuristic functions and see their importance in different scenarios, we found two interesting papers on some searching which we have studied and implemented:

- Aine et al 2016
- A multi heuristic approach 2015

With these as an aid, we have been able to compare the performances of A* and MHA* on various problems like the tile-sliding puzzle, taxi path finding etc. We have tried to visualize the working of both the algorithms. We have tried to compare the the path, time, operations taken to get a good idea about the working of both of these algorithms.

II. Background

A. Heuristic

A heuristic function $h(n)$ is the expected cost to reach the goal state. Heuristic function is used as an aid to solve the actual problem at hand, as it provides an approximate solution to the goal and does not take much time to compute. At every step of the algorithm, it provides information on expected cost to reach goal from different branches and thus in turn help us decide which branch to follow. Some of the used heuristic functions are Euclidean Distance (euclidean distance from a node to the goal node), Manhattan Distance (summation of difference of x_i's of current node and goal node) etc. These heuristics come in handy in variety of problems ranging from simple directed-undirected graph, tile-sliding puzzles, path finding problems (taxi, TSP) etc.

B. A* Search

Here the search is performed using only a single heuristic function $h(n)$. $h(n)$ in combination with $g(n)$ [cost to reach a node from the current node] is used to implement A* search algorithm. At each step, A* picks the node with the lowest f-value (sum of cost to reach that node and estimated cost to reach goal from that node $f(n) = g(n) + h(n)$.

C. Multi-Heuristic A* Search

The search is performed in a round robin fashion with multiple heuristics. These heuristics can be arbitrarily inconsistent. To ensure the optimality of the result of MHA* the algorithm parallely compares the heuristic value with a consistent heuristic(usually Euclidean distance). If at any step the heuristic exceeds the correct value, we dont expand that search instead we expand the consistent heuristic search.

III. Use Cases

A. Large undirected graph ($n = 400$)

For our initial test of the algorithms, we tested our implementations on a fairly big graph with $n(= 400)$ nodes with uniform edge costs in different cases - (1) no blocks: no obstructions to reach the goal on the grid , (2) blocks in the grid: fairly big obstructions to reach the goal (however ensuring there exists a path to goal) (3) no path to goal: grid having no possibility of reaching the goal state. Here we have used A* search with both Manhattan distance and euclidean distance as heuristic. For MHA* search, we have taken the combination of Manhattan Distance, Euclidean Distance and Euclidean Distance divided by total time taken until now (speed like heuristic) to compare the performances.

B. Tile Sliding Puzzle

In an $n \times n$ tile puzzle, we have numbers from 1 to $(n**21)$ displaced randomly across the grid and we have to sliding operations to arrange these numbers back in order 1 to $(n * *21)$. Here we have used the following heuristics:

- Manhattan Distance (MD) - The sum of absolute difference of x-coordinates and y-coordinates between two tiles.
- Linear Conflicts (LC) - Two tiles t_j and t_k are in a linear conflict if t_j and t_k are in the same line, the goal positions of t_j and t_k are both in that line, t_j is to the right of t_k and goal position of t_j is to the left of the goal position of t_k, thereby adding a cost of 2 moves per conflict.
- Misplaced Tiles (MT) - Number of misplaced tiles in the current board configuration.

Following are the different states in the puzzle:

- Board - the placement of tiles in the present state.

- Start State achieved after performing random shuffling of tiles.
- Goal state - placement of tiles in the order of natural numbers with the $n \times n$ tile being free.

We have also implemented the GUI for the sliding problem which initiates by shuffling the board and then on clicking on solve, goes through the solution. This was implemented using pygame (python) including the sliding animations, button clicks. We have made a generalized version of the problem for different n's. We have presented the analysis of both the algorithms on the basis of Iteration vs No of moves, N vs time, N vs # of queue operations.

C. Taxi Pathfinding Problem

To compare the performances further, we define a new problem, which is more real time and automated. There is a city tour cab, which has to carry its passengers to various tourist locations and its ultimate goal is to bring the passengers to IIITD. This is different from the Travelling Salesman Problem(TSP) as we dońt aim at covering all of the tourist spots instead we try to maximize the number of tourist spots that we can cover with the ultimate aim of reaching IIITD(i.e., goal). We use Google Maps API to show the route taken by the algorithm. Using the names of locations we get the coordinates and then these coordinates are converted into graph for our implementation that generates the sequence of states with respect to the above objective. The cost is the distance between the two locations and this cost (i.e., the actual path cost $g(n)$) is calculated from API. Here we define 2 heuristics, firstly the consistent heuristic, that is the haversine distance used to calculate the distance between two coordinates. The other heuristic that we have used is the expected time to reach the goal. This time takes into consideration various factors like traffic, time of the day etc. while calculating the time.

IV. Results

A. Large undirected graph (n = 400)

The heuristics used here are euclidean distance(consistent) and manhattan distance and speed. As we can see A* algorithm is optimal as it is travelling very much close to the diagonal. Here optimality is defined as the as the path taken by the algorithm using admissible heuristic of euclidian distance in graphs. MHA* on the other hand deviates a bit from the diagonal as it is also incorporating other inadmissible heuristics, but tries to converge to the optimal solution after few iterations. Refer Figure 1, 2

B. Tile Sliding Puzzle

Iteration vs Number of Moves- For most of the times, A* remains below the MHA* line curve, and can be thus seen to perform marginally better in terms of number of moves.

N vs Queue Operations- A* takes lot of queue operations, and the queue operations increase substantially when the randomness in the grid size is high. (same amount of randomness results in greatest shuffles in grid size = 4)

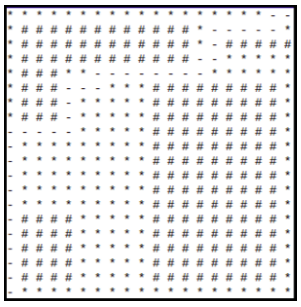


Fig. 1. MHA*

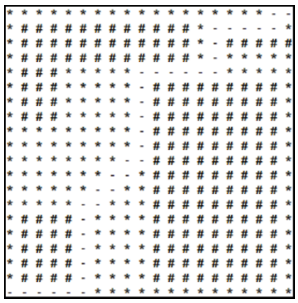
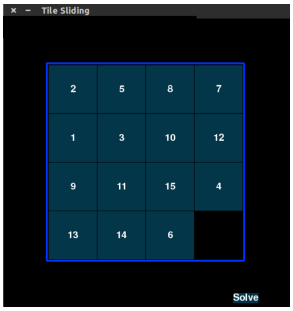
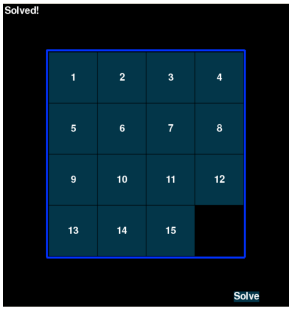


Fig. 2. A*



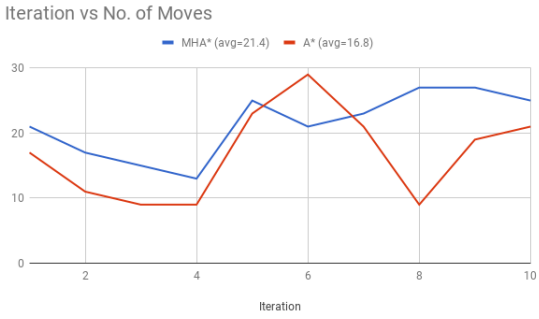


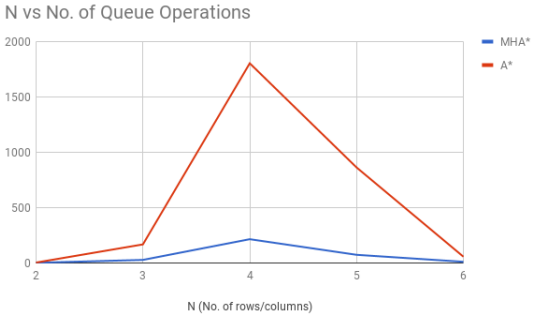
C. Taxi Pathfinding Problem

Fig 3 and Fig 4 in the document. Set of places in Delhi taken as a part of Delhi tour with source as Rashtrapati Bhawan and destination culminating in IIIT Delhi. This can be seen in the figure 3 with blue as source and green as our destination. In Figure 4, the green path highlights the current path chosen and blue path is the already taken path. The path taken for both MHA* and A* came out to be the same upon testing with different heuristics also.

V. Conclusion and Further Work

We can see that MHA* doesn't differentiate from A* in terms of path planning. Both of the algorithm have similar paths as in the case of taxi driver problem. The variation comes in the case of queue operations(here iterations are defined as the queue operations). The reason A* is taking more iterations is that A* looks at all possible states in a particular path, even if at some point in future, the algorithm will exclude that path. Thus in A* there is a lot of backtracking. MHA* on the other hand is better as it compares with other heuristics while expanding a particular, and if it is deviating too much from the consitent heuristic, then it doesn't choose that path.





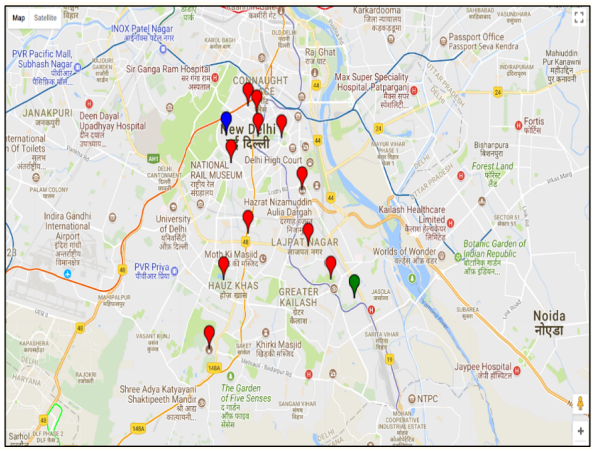


Fig. 3. All locations with source and destination

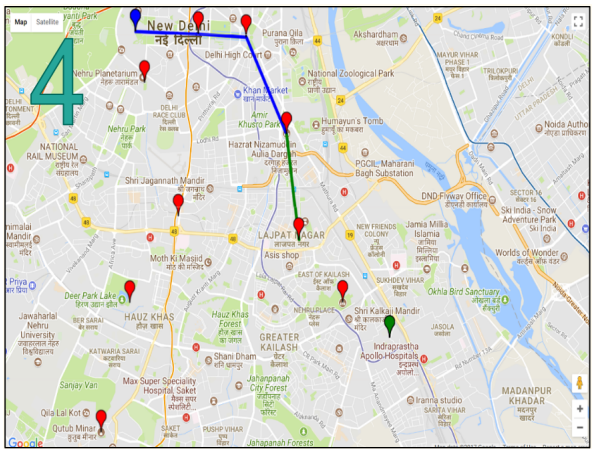


Fig. 4. 4th ieration of the solution

REFERENCES

[1] Aine et al 2016

[2] A multi heuristic approach 2015