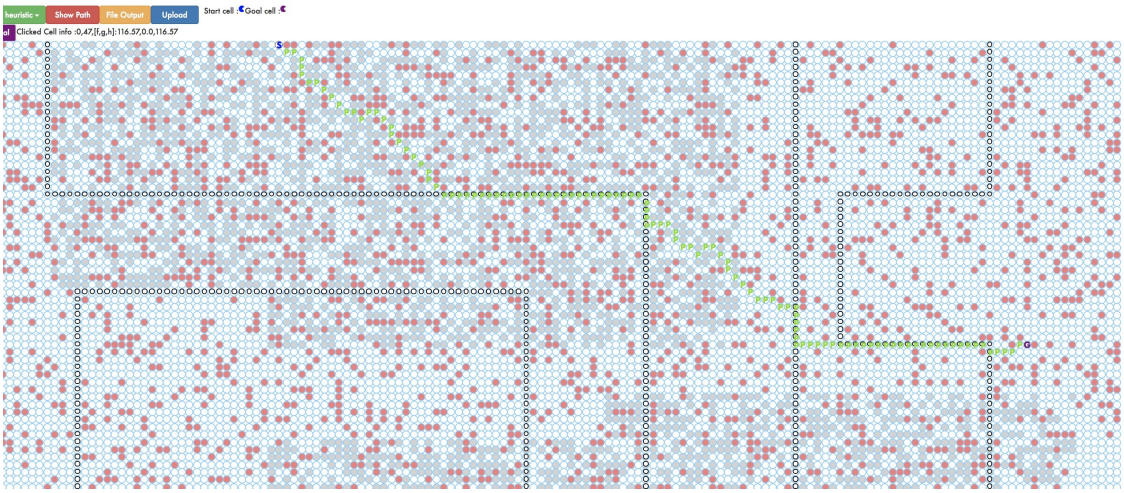
Assignment 3 Heuristic Search

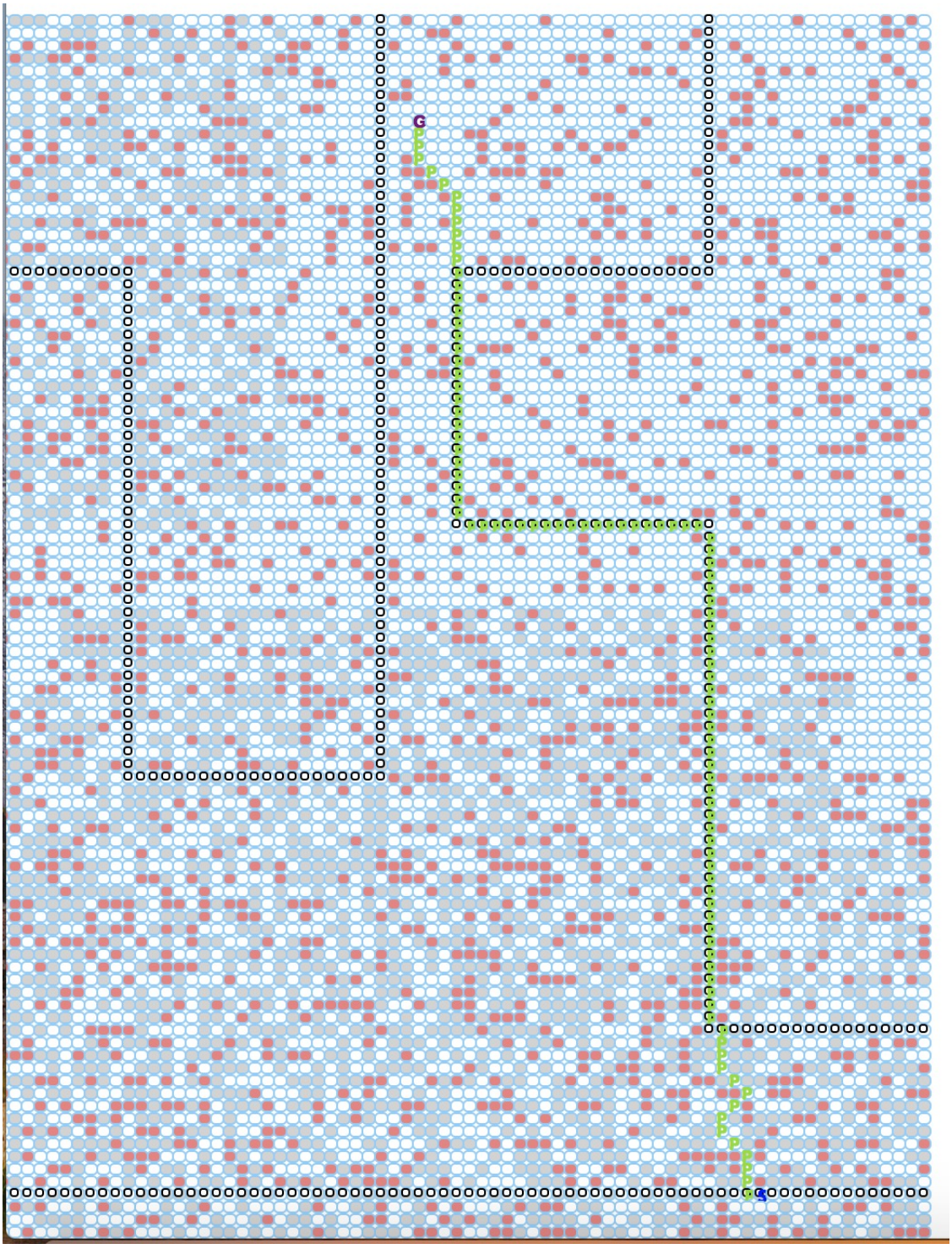
By Brandon Young and Ruicheng Wu

A. Interface

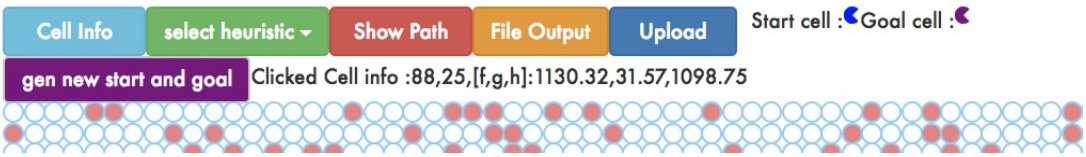
Here are two examples of the GUI. The first image shows the overall GUI. The buttons on the top-left are used to interact with the grid. The red "Show Path" button is used to upload a path file to show a green path on the grid. The yellow "File Output" is used to output a file representing the current grid. The blue "Upload" button is used to upload and show a grid from an existing file. Clicking on a cell on the grid shows its h, g and f values below the buttons and above the grid.



This second image shows a closer look to the grid representation. Red squares represent blocked cells, gray for hard-to-traverse cells and white for unblocked cells. A black circle within a square shows that the cell has a highway on it. Green P's show the path. The start (in blue) and goal (in purple) are either represented by S and G respectively or pie-shaped symbols.


G
S

Control Panel

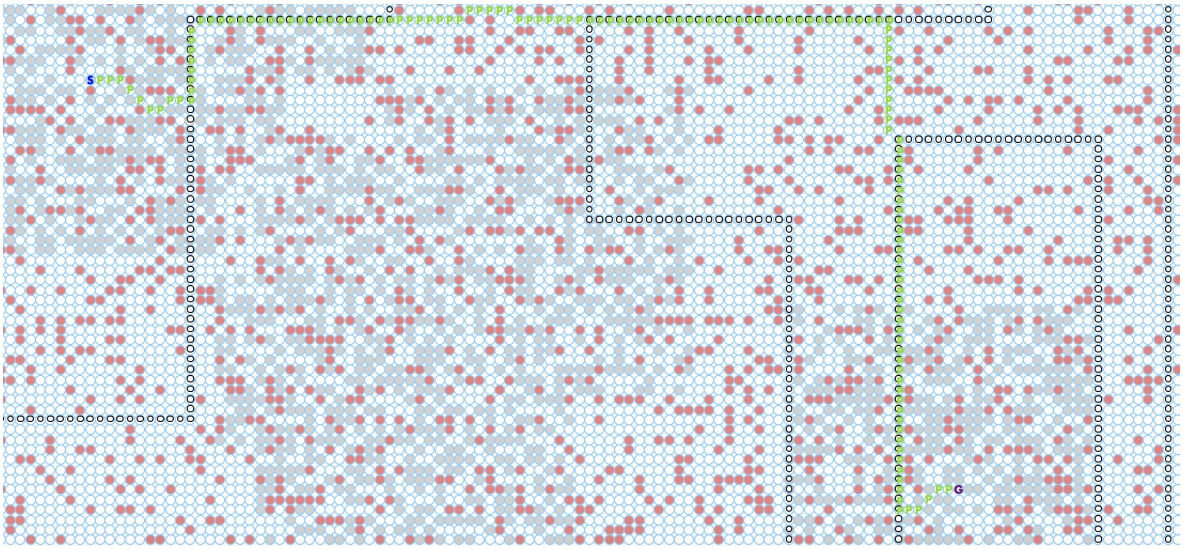


B. UCS, A* and Weighted A* Implementations

There are three examples shown below, running uniform-cost search (UCS), A* and weighted A* respectively on the same map(map5-1).

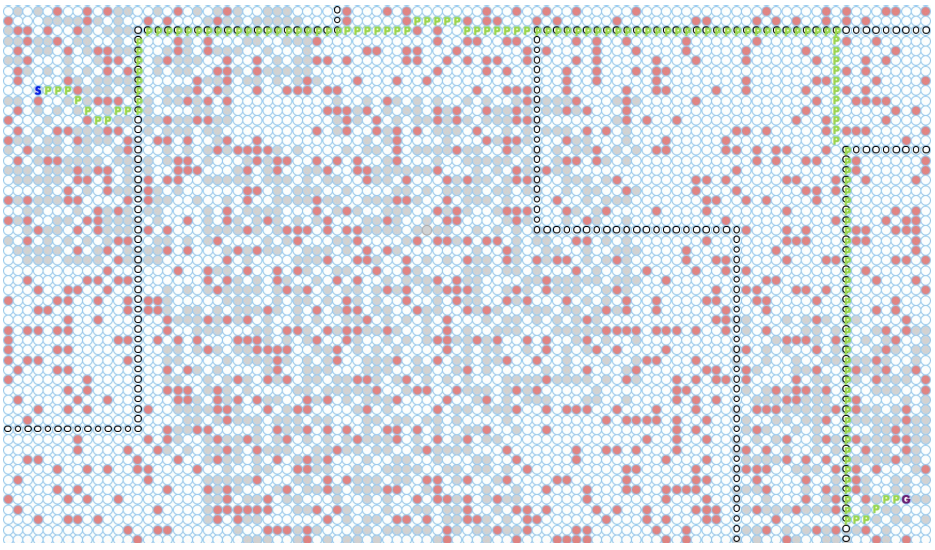
1. Uniform-cost search:

Path length:81.08, Node Expanded:11595, Run Time: 0.384272



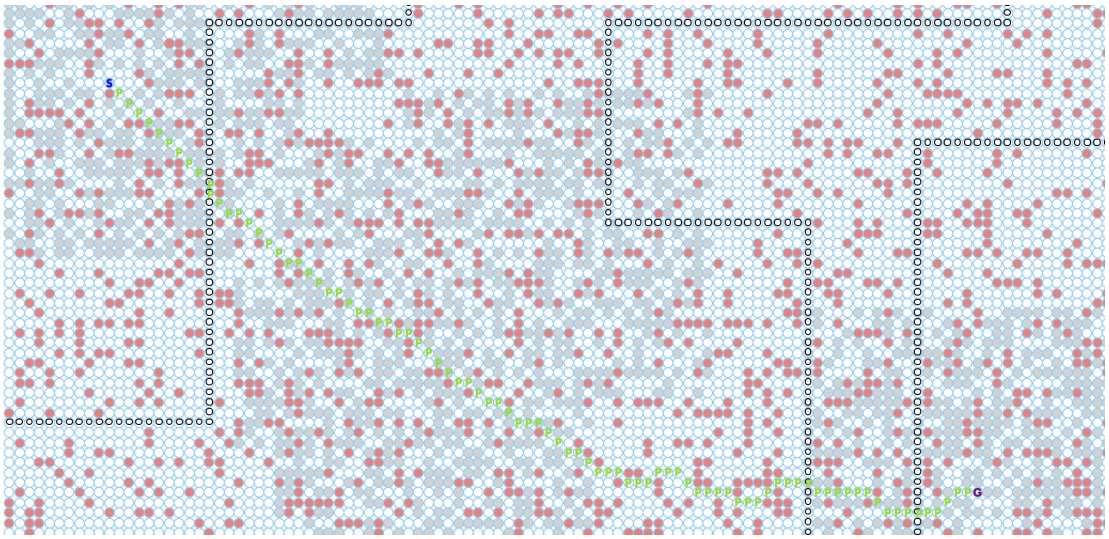
2. A * search(h=1):

Path length:81.08, Node Expanded:8155, Run Time: 0.287288



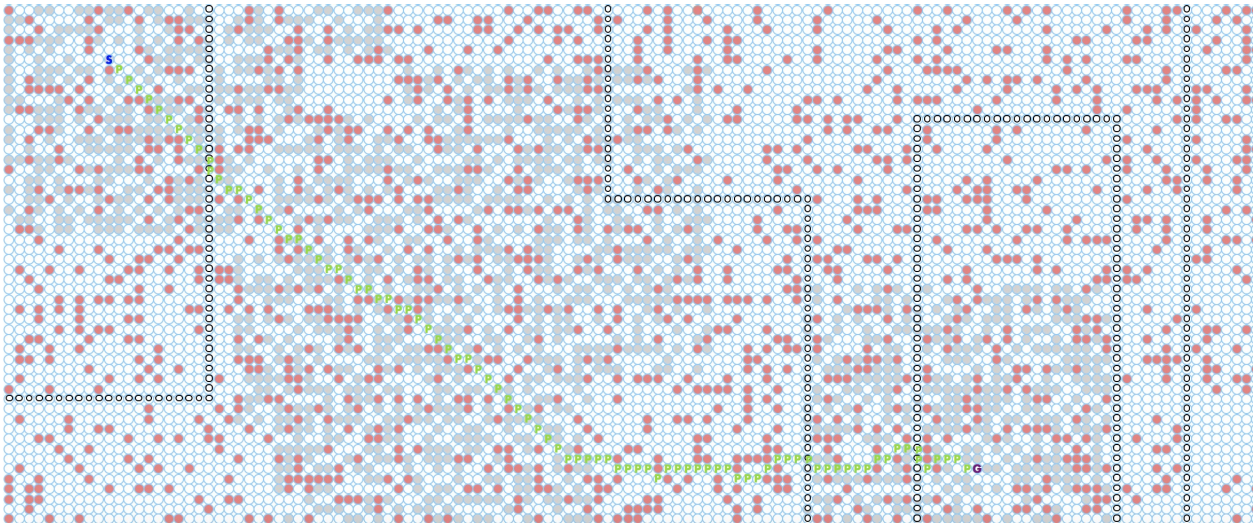
3. A * search(h=4):

Path length:137.33, Node Expanded:234, Run Time: 0.056771



4. Weighted A * search search:

Path length:144.03, Node Expanded:107, Run Time: 0.05289



C. Optimizations

To optimize the search algorithms, we used a Python dictionary as a heap, rather than a Python list (array), to implement the closed list more efficiently. Since we just needed to insert or check if a pair of coordinates existed inside the close list, a heap would speed up the check and may help the insertion. Overall, checking if a element exists takes O(n) for an array and O(1) for a heap.

For the heap, we initially used the sum of coordinate values to use as the key. However, this meant for every (x, y) there is a matching pair (y, x) with the same key. In addition, as the sums increased, there were a greater number of pairs that summed to the same value. As a result, we decided to distinguish (x, y) by multiplying x with a hash code value. With some testing, higher hash codes lowered the average size of a list for each key so we ended up using 91 as the hash code.

We also optimized the way the heuristic was applied to the grid. At first we applied the heuristic over the entire graph before running the search algorithm. However, this wasted resources when only a portion of the overall grid was searched. Instead, we compute the heuristic during the search, to ensure only relevant cells in the grid have the heuristic applied to them.

D. Heuristics

1. Euclidean Distance

Euclidean distance is also a admissible/consistent heuristic as it works on a subproblem of the grid problem where any degree of movement is allowed. We also cut the distance by 4 to mimic movement along a highway.

The formula for cell (x_1, y_1) and goal (x_2, y_2) is:

$$h((x_1, y_1)) = \frac{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}{4}$$

2. Manhattan Distance

This heuristic assumes every cell in the grid is an unblocked cell with a highway on it. Then it computes the shortest path using only horizontal and vertical steps. We assumed every cell is a highway because using only unblocked cells results in higher than usual h-values. Often a large portion of the optimal path is along a highway, so the heuristic should reflect this.

The formula for cell (x_1, y_1) and goal (x_2, y_2) is:

$$h((x_1, y_1)) = 0.25 * (dx + dy)$$

where $dx = |x_2 - x_1|$ and $dy = |y_2 - y_1|$. The 0.25 comes from the cost it takes to move horizontally or vertically in one step.

The heuristic is inadmissible for the grid problem because diagonal movements are normally allowed. As a result, Manhattan distance tends to overestimate the actual cost to the goal. Nevertheless, this metric can come in handy for sequential A* search because it overestimates the optimal distance slightly.

3. Diagonal Distance Heuristic

The best admissible/consistent heuristic we used was the diagonal distance heuristic. The diagonal distance heuristic is similar to the Manhattan distance heuristic, except diagonal steps are allowed.

The formula for cell $(x1, y1)$ and goal $(x2, y2)$ is:

$$h((x1, y1)) = 0.25 * (dx + dy) + (\frac{\sqrt{2}}{4} - 2(0.25)) * min(dx, dy)$$

where $dx = |x2 - x1|$ and $dy = |y2 - y1|$. The 0.25 comes from the cost to move horizontally and vertically between highway cells and $\frac{\sqrt{2}}{4}$ is the cost from moving diagonally between highway cells

Diagonal distance is better than Manhattan distance because it is admissible and Manhattan is not. Diagonal distance was preferred over Euclidean distance because the movements it allows is closer to the movements allowed on the actual grid. Therefore, the heuristic value will come closer to the actual optimal distance than the value from Euclidean distance.

4. Euclidean Squared

The formula for cell (x_1, y_1) and goal (x_2, y_2) is:

$$h((x_1, y_1)) = \frac{(x_2 - x_1)^2 + (y_2 - y_1)^2}{16}$$

This heuristic is similar to Euclidean distance, except it squares the output from Euclidean distance. As a result, it output very high values and it is not admissible. However, because the square root operation is not used, some computation time is saved. In general, this heuristic is pretty bad because its h-values will tend to be much higher than the g-values, but that may come in handy with sequential A* search.

5. Sample Heuristic

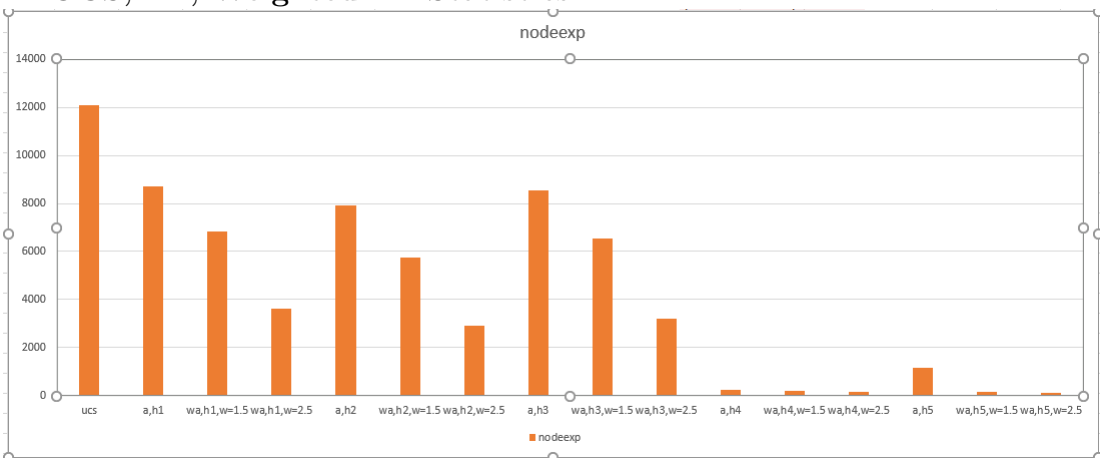
The heuristic given in the assignment instructions. May not be admissible or consistent for the grid problem.

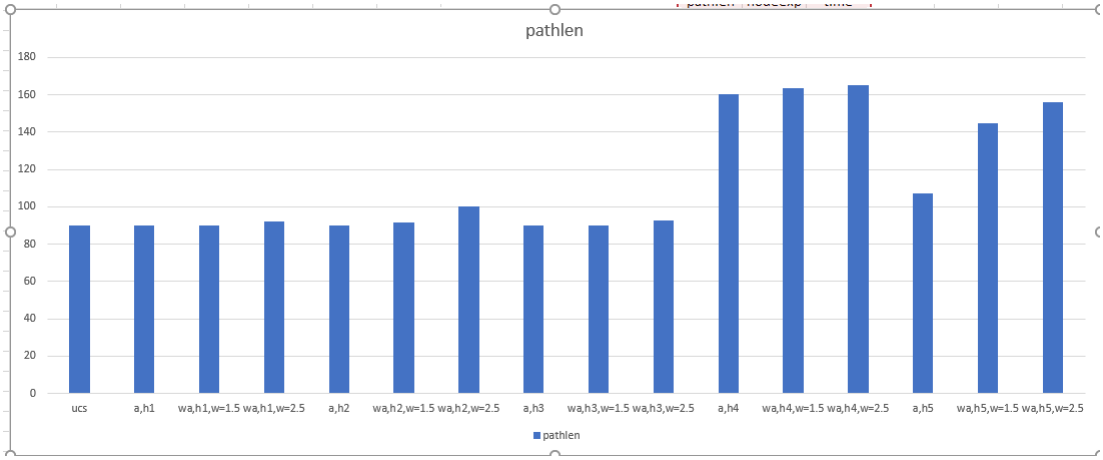
The formula for cell (x_1, y_1) and goal (x_2, y_2) is:

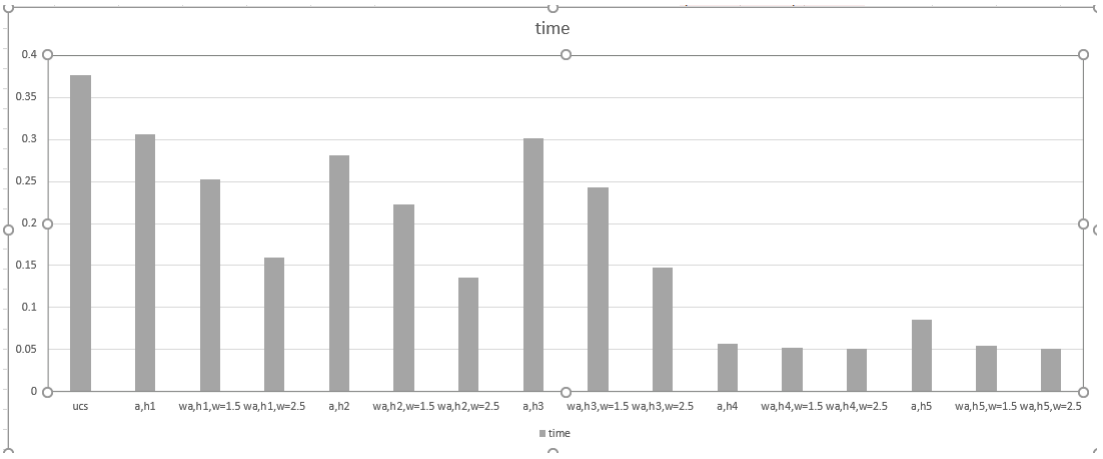
$$h((x_1, y_1)) = \sqrt{2} * min(dx, dy) + max(dx, dy) - min(dx, dy)$$

where $dx = |x_2 - x_1|$ and $dy = |y_2 - y_1|$.

E. UCS, A*, Weighted A* Statistics







F. Analysis of UCS, A* & Weighted A*

Between the heuristics, heuristic 3 (diagonal distance) performed the best in terms of path length and heuristic 4 (Euclidean squared) performed the best in terms of time. Heuristic 1 (H1) also performed better than H3 in terms of nodes expanded and slightly better in terms of time.

–Richard—(delete this line after Brandon confirmed and combined below statements)

UCS : it has the largest node to expand and highest running time, this proves its optimality based on sacrifice of speed and memory. But it guarantees an optimal path will be generated.

A* search: method depends on different type of heuristics.There are inadmissible ones, and so some A* family search used that heuristic (like h=4) expand much less nodes and run time than others because of this.Hence,the result path length is certainly worst sub-optimal among all heuristics.

Weighted A* search: now we have one more factor to add bias on the heuristic(pure A* treat this weight as 1).So clearly we can see, larger weight will have less node expansion and run time but for those "good" heuristics, the sacrifice for sub-optimality isn't that expensive. Like h=1,h=3,the weighted A* search with 2.5 weight saves a lot of time and memory to search but nearly get the exact same path length!

certainly ,h1 and h3 are best two heuristics to use among all 5 proposed ones. But h3 has better performance in every aspect except (finding optimal path length,h3 has slightly higher path length than h1)