Assignment 3 Heuristic Search

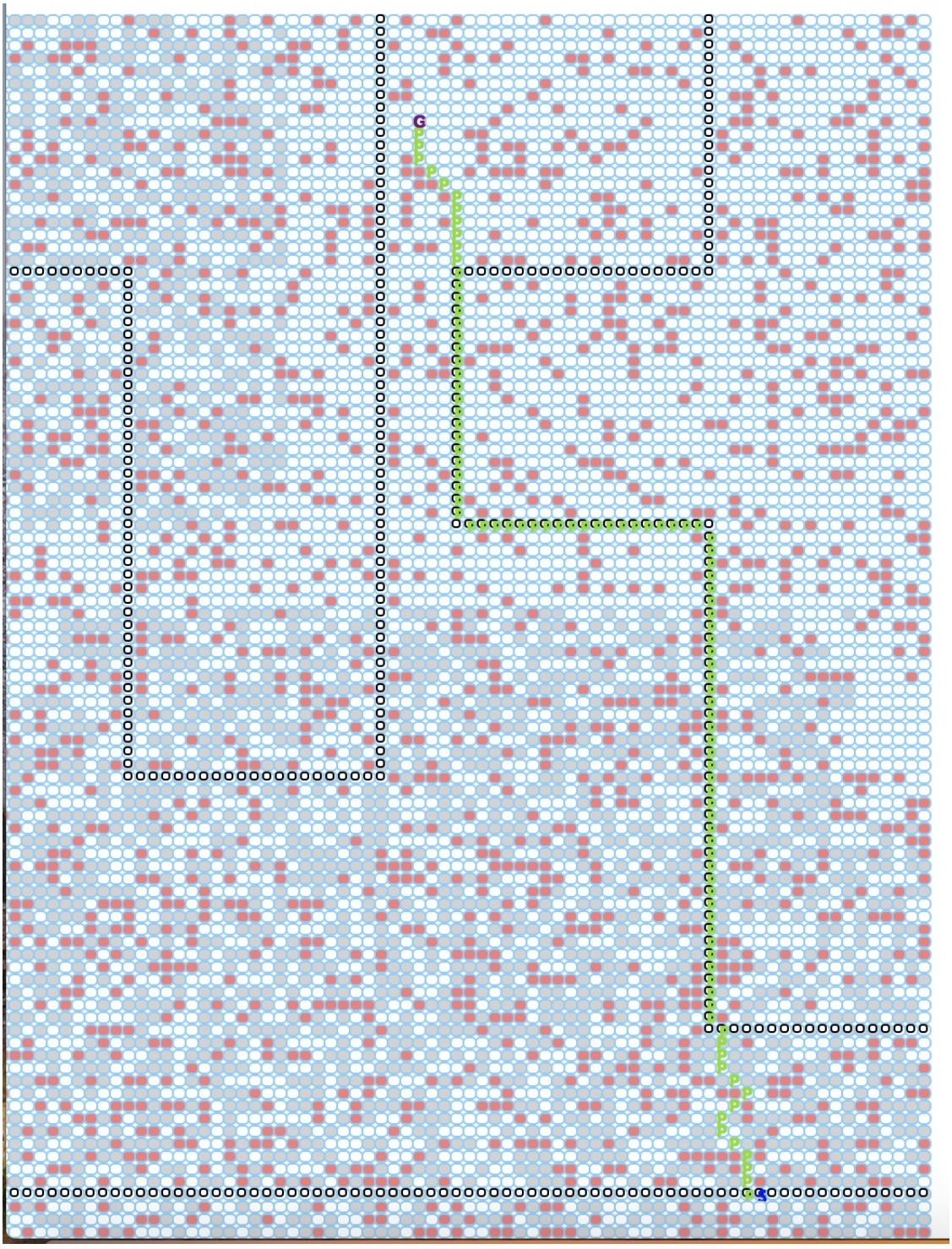
By Brandon Young and Ruicheng Wu

A. Interface

Here are three examples of the GUI. The first image shows the overall look of the GUI. The buttons located on the top-left of the interface are used to interact with the GUI. Clicking on a cell on the grid shows its h, g and f values below the buttons and above the grid.

Below is the control panel of the GUI. All of the I/O interactivity is located here. 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 dark blue "Upload" button is used to upload and show a grid from an existing file. The light blue "Cell Info" button is for uploading a file with the f,g and h values for cells in the grid.

This third image shows a closer look at the actual grid. 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.

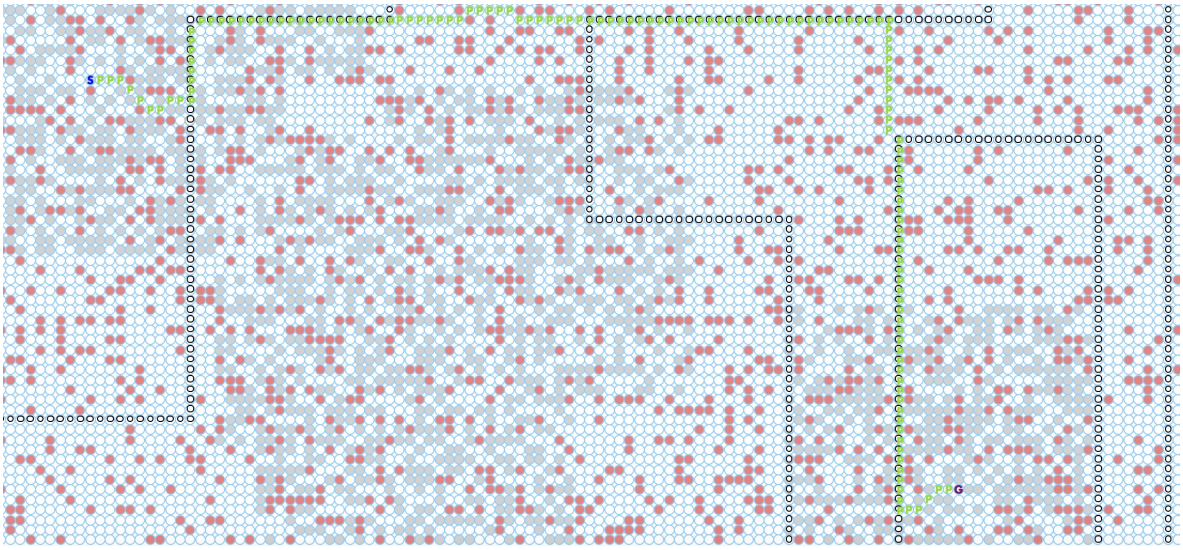

G
S

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.

1. Uniform-cost search:

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

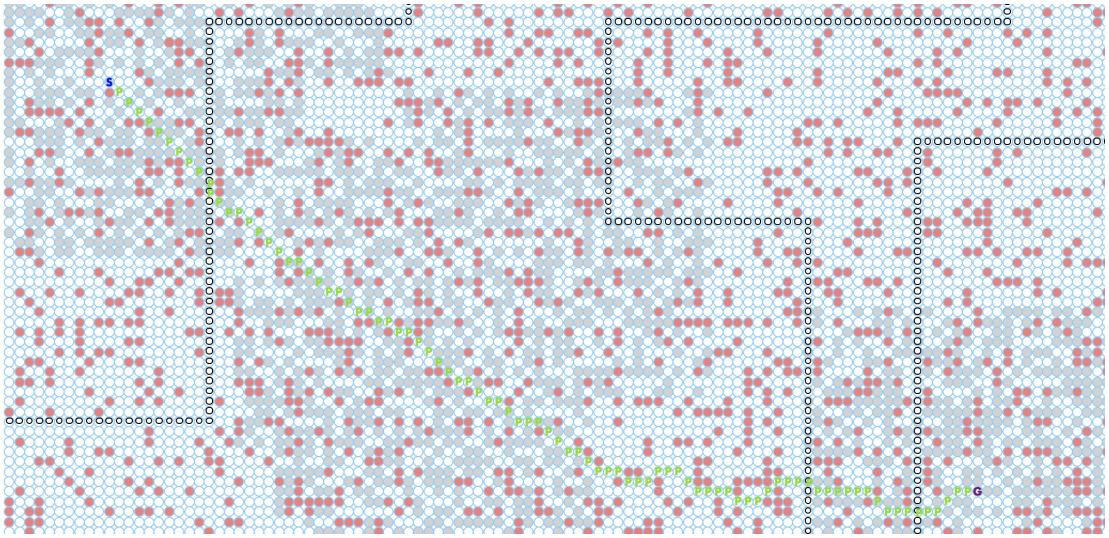


2. A * search(h=1):

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

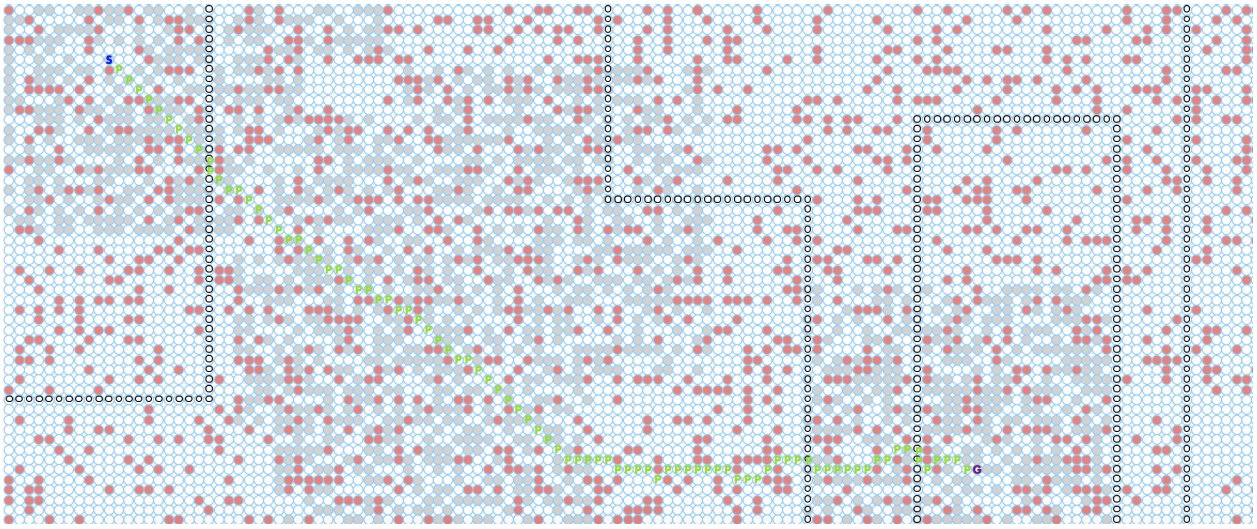
3. A * search(h=4):

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

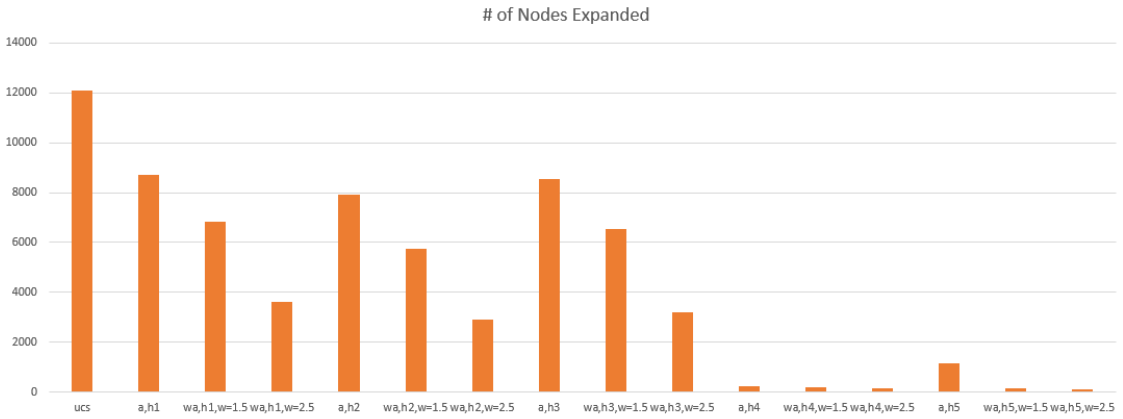
The heuristic given in the assignment instructions. May not be admissible or consistent for the grid problem.

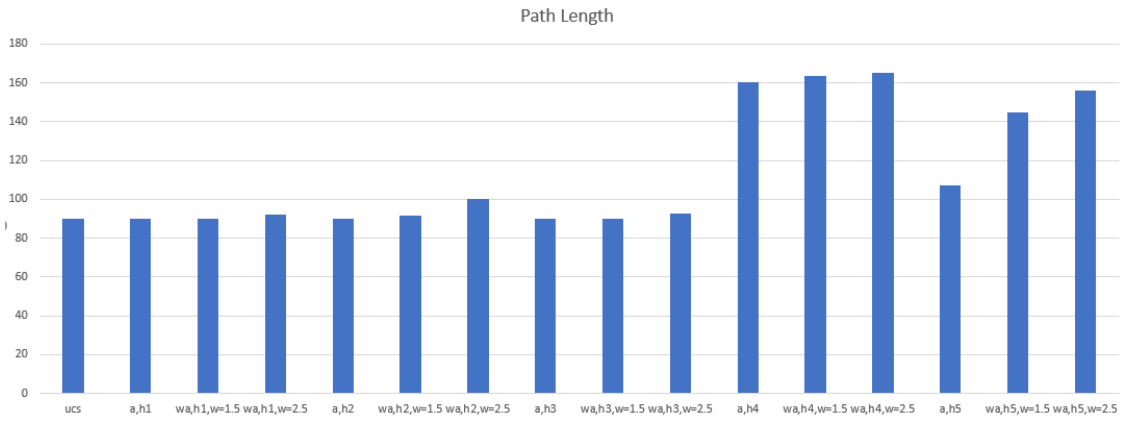
The formula for cell (x_1, y_1) and goal (x_2, y_2) is:

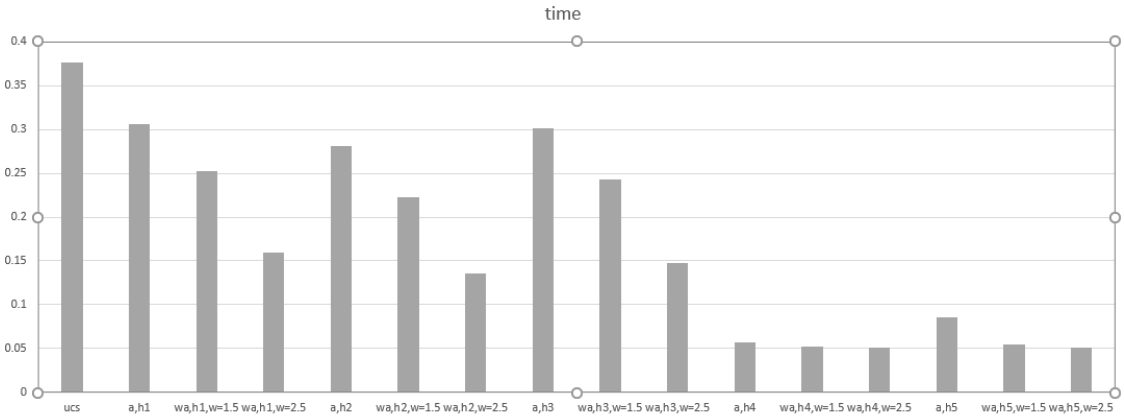
$$h((x_1, y_1)) = \sqrt{2} * min(dx, dy) + max(dx, dy) - min(dx, dy)$$

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*

At first glance, uniform-cost search (UCS) finds the shortest path, but it has the highest number of nodes expanded and time taken by far, compared to the other methods. This provides evidence how UCS sacrifices speed and memory for the guarantee of finding an optimal path.

In comparison, A* (heuristic search using the diagonal distance or euclidean distance heuristics), tended to run much quicker than UCS, while still finding the shortest path. A* also expanded roughly a third less nodes than UCS. When using inadmissible heuristics, the heuristic search ran very quickly and expanded very few nodes (like with heuristic 4, Euclidean squared), but at the expense of longer paths.

Weighted A* search is similar to heuristic search with the Euclidean squared heuristic in that its runtime was quicker than A*, as expected, but the path length suffers slightly. With the additional weight factor bias the search, the heuristic can search more greedily with a good heuristic, but may return a sub-optimal path. Just a slightly higher weight like 1.5 (weight = 1 is the same as A*) reduces the time and number of nodes expanded significantly, while the average path length increases slightly. With a larger weight, like 2.5, the effects of weighted A* search are even more noticeable. A higher weight leads to even fewer nodes expanded and an even shorter run time with only a slight increase from the optimal path. In general, for the "good" heuristics (heuristics 1, 2 and 3), the sacrifice is not as severe as for other heuristics (heuristics 4 and 5).

Between the heuristics themselves, heuristics 1 (Euclidean distance), 2 (Manhattan distance) and 3 (diagonal distance) performed the best in terms of path length. Heuristic 4 (Euclidean squared) and heuristic 5 (sample heuristic) performed the best in terms of time and nodes expanded.

The heuristic results were mostly expected, with diagonal distance performing slightly better than Euclidean distance. In heuristic search with no weights, diagonal distance and Euclidean distance both found the optimal path. Surprisingly, Manhattan distance's average path length comes really close to the optimal path length and, at the same time, it performs better in terms of nodes expanded and time in comparison to diagonal and Euclidean distance. With higher weights, Manhattan distance resulted in a noticeably worse path than the admissible heuristics with the same weights.

Euclidean squared and the sample heuristic perform the worst in terms of path length by almost a factor of 2. It should be noted that the sample

heuristic did not consider highways, which resulted in its inadmissibility. Moreover, with normal heuristic search, the sample heuristic's path comes fairly close to the optimal one. On the other hand, the Euclidean squared heuristic's path is consistently much longer than the optimal path, with or without weights. While these heuristic find longer paths, they also have huge saves in time and expand practically zero nodes relative to other searches.

Out of all of the heuristics, the sample heuristic is the only one whose average path length is significantly affected with weights 1.5 and 2.5. Other heuristics only have a slight change in average path length with the same weights.

In conclusion, Euclidean distance and diagonal distance are the best two heuristics to use among all of the five proposed ones. Diagonal distance outperforms Euclidean distance in nearly every aspect, except with weights involved, Euclidean distance's average path length fairs a bit better. To save on time, either use a slight weight (like 2.5) or use the Euclidean squared or sample heuristics.

G. Sequential A* Implementation

examples here

H. Sequential A* Analysis

analysis here

I. Sequential A* Proof

proof here