CMPUT 366 Assignment 1, Nicholas Serrano, 1508361

Analyzing plots (1st implementation)

In the first implementation of this assignment, we wrote a program that implements Dijkstra's and A* in order to solve path finding problems. The 2 algorithms were used in all test cases, and graphs were generated in order to compare the performance and solutions of each of these algorithms. Let's first look at the number of nodes expanded.

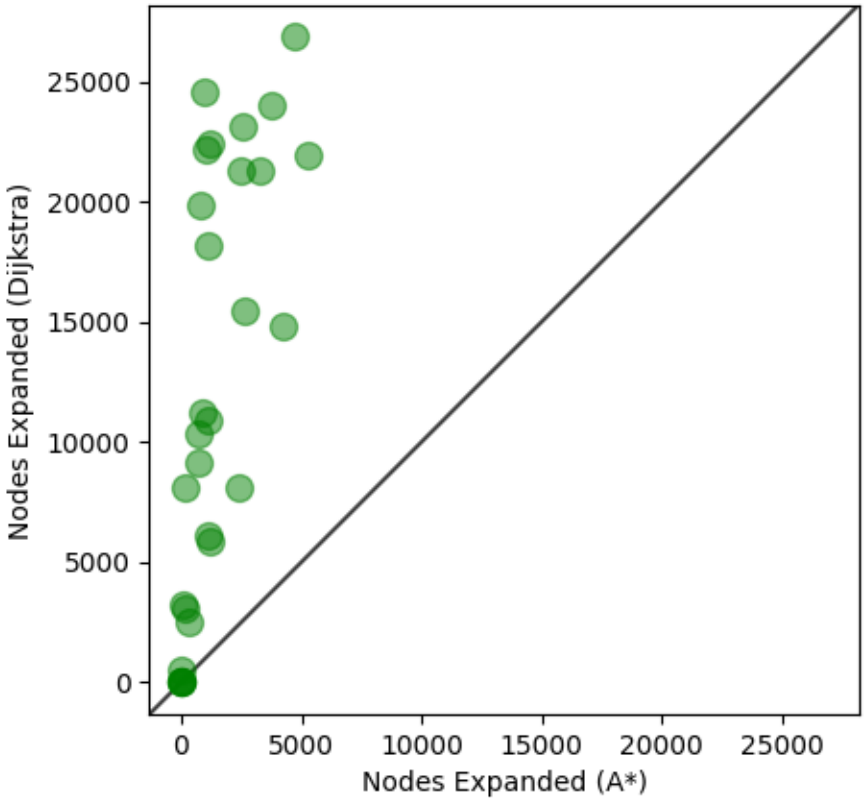

Figure 1: Nodes Expanded

Right away, we can see the nodes on the graph are heavily placed towards the left side of the graph. We can see that Dijkstra's requires a much larger number of nodes to be expanded in order to find a solution to the problem. The A* algorithm never went above 5000 nodes for all the test cases, while Dijkstra's had values that ranged from as high as above 25,000 nodes expanded. With the given heuristic that was provided for A*, we can determine that A* is much more efficient in terms of required number of nodes expansions when compared to Dijkstra's.

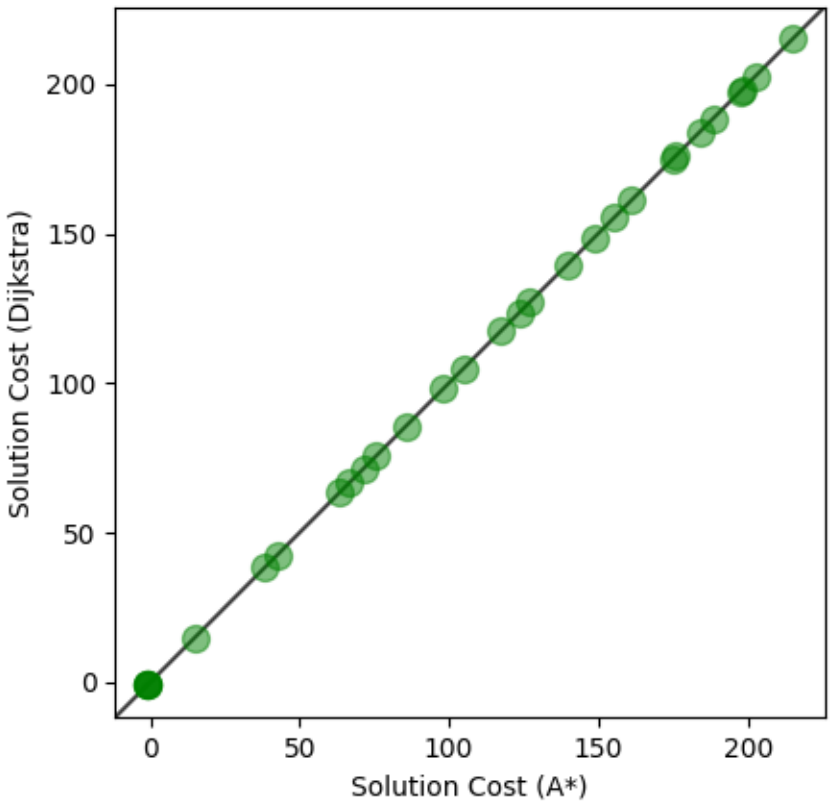

Figure 2: Solution Cost

Figure 2 shows us a very linear graph. What this tells us that is that A* and Dijkstra's were both able to find solutions with the same solution cost for each test case. This is because of the heuristic chosen for A*. Dijkstra's algorithm is guaranteed to find an optimal solution to a problem. Because the heuristic function of A* is an admissible heuristic (consistent implies admissibility), A* is also guaranteed to find the optimal solution, which explains why the solution cost in the graph above is the same for each algorithm.

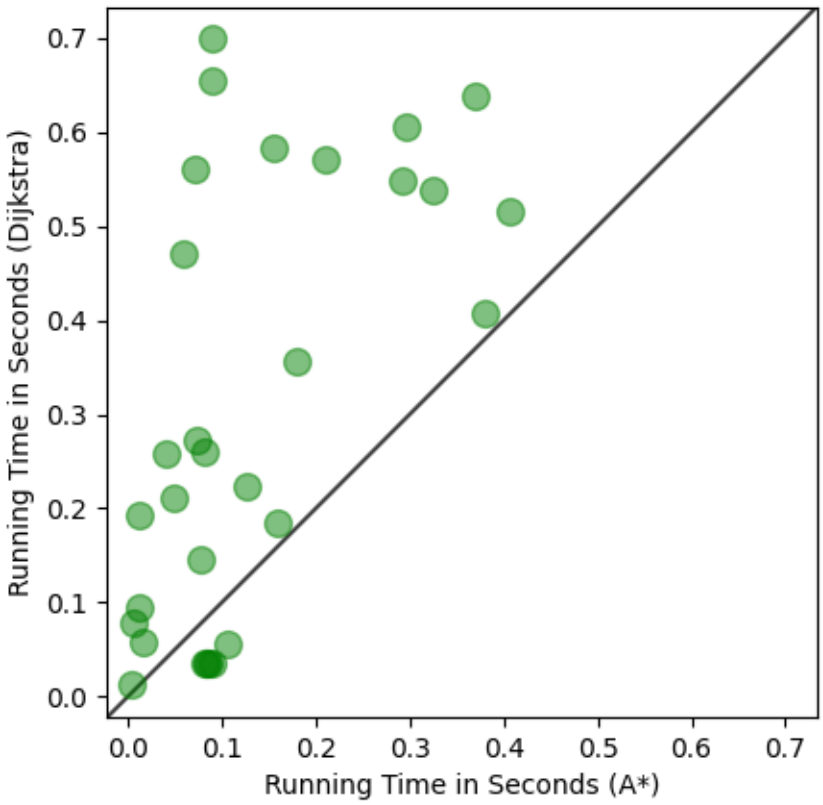

Figure 3: Run time

Now let's analyze the runtime. As shown in figure 3, we see a significantly larger amount of runtime on average for Dijkstra's algorithm when compared to A*. There are very few points on the far-right side of the graph. We saw in figure 1 that Dijkstra's algorithm expanded a significantly larger number of nodes when compared to A*. Therefore, it is expected that figure 3 should reflect this, and show a larger runtime for Dijkstra's due to the increased number of nodes expanded. Figure 3 indeed illustrates this exactly, and we can infer for the test cases provided and the heuristic function applied to A*, A* is on average the better choice in solving the path finding problems for these gives sets of maps. Why is this the case? Because Dijkstra's algorithm has no sense of direction where to go. There are maps in the tests cases with dead ends that are far away from the solution node. Dijkstra's will sometimes expand these far away paths leading to a larger number of node expansions, and thus larger runtime. The heuristic function in A* provides a sense of direction for where to go to find the shortest path, so A* is able to sometimes avoid these redundant paths, and in general find a better path more quickly in these particular maps.

Analyzing plots (2nd implementation)

In our second implementation, we are now multiplying the heuristic function by 2. Before we show the graphs, lets think about what this might do first. A* with a consistent heuristic is similar to Dijkstra's, except instead of comparing g values we are comparing f values. What this means is that changing the heuristic function (used to also find f value) will indeed likely have an impact on A*. In our case, we are doubling the value of the heuristic. This means, we are putting more trust, or emphasis on our heuristic. We are confident in our heuristic and are more likely to follow a path that is heavily guided by the heuristic. We should also keep in mind that doubling the heuristic means that it is no longer guaranteed to be consistent.

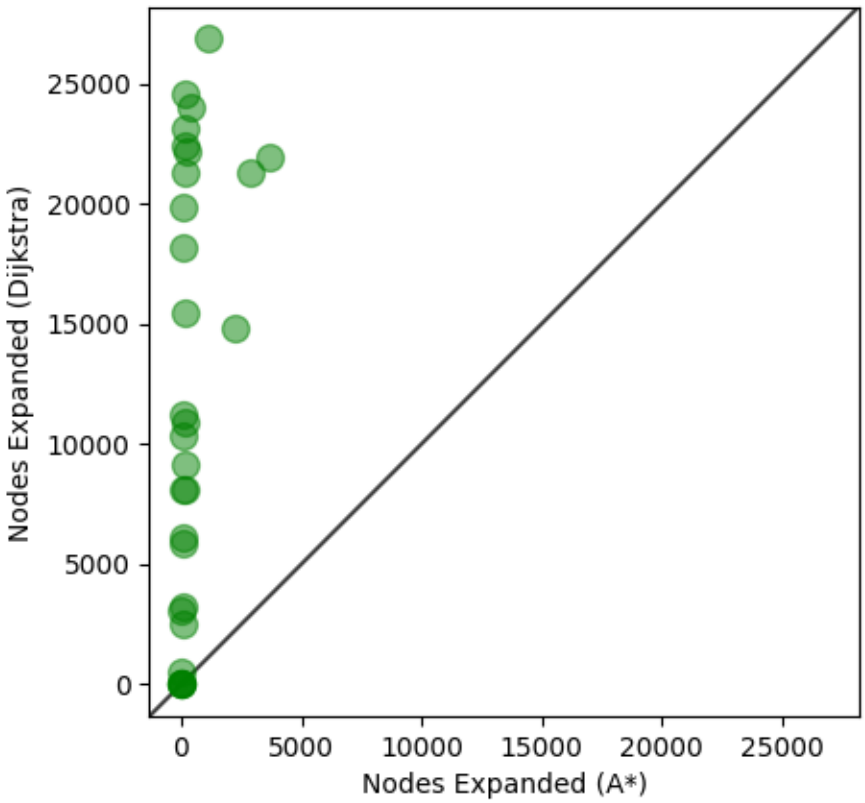

Figure 4: Nodes Expanded 2

Right away, in comparison to figure 1, we see a significant reduction in the number of nodes expanded for A*. What this means is that it is likely the paths in the map that were heavily enforced by the heuristic did not have too many walls blocking the direct path. We were able to find a more direct path to the solution thanks to the inflated heuristic (direct in that we didn't need to expand as many redundant nodes).

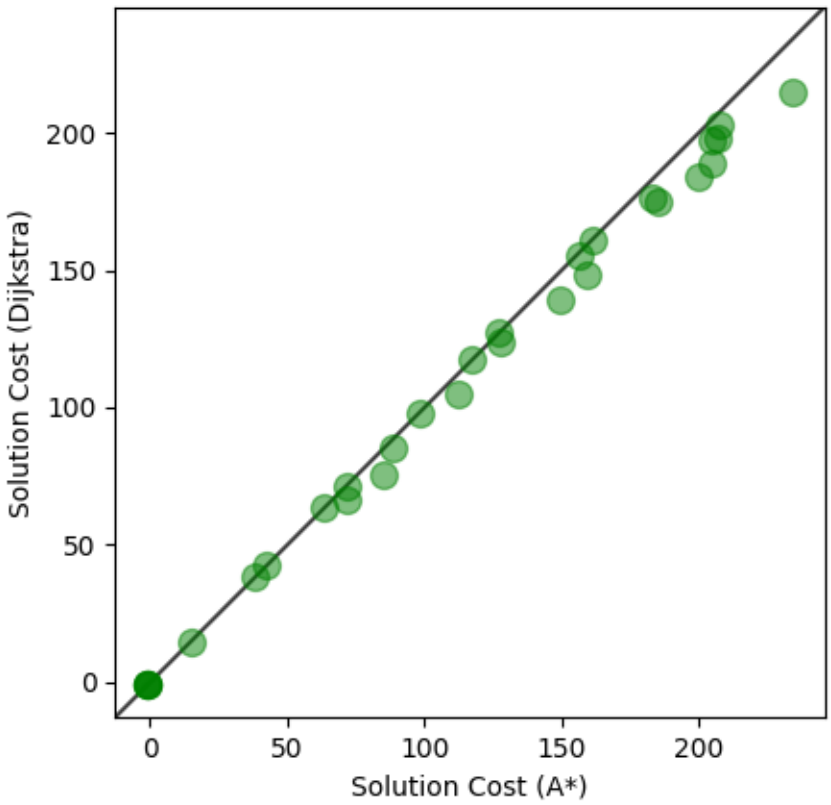

Figure 5: Solution Cost Part 2

In figure 5, we see A* is no longer perfectly 1:1 with Dijkstra's algorithm. That is, we are no longer always finding the optimal solution. This is to be expected, because the weighted heuristic is no longer guaranteed to be an admissible heuristic. In this case, it wasn't an admissible heuristic, and we can see A* found some solutions that were not optimal. Many points in the graph are now below the line.

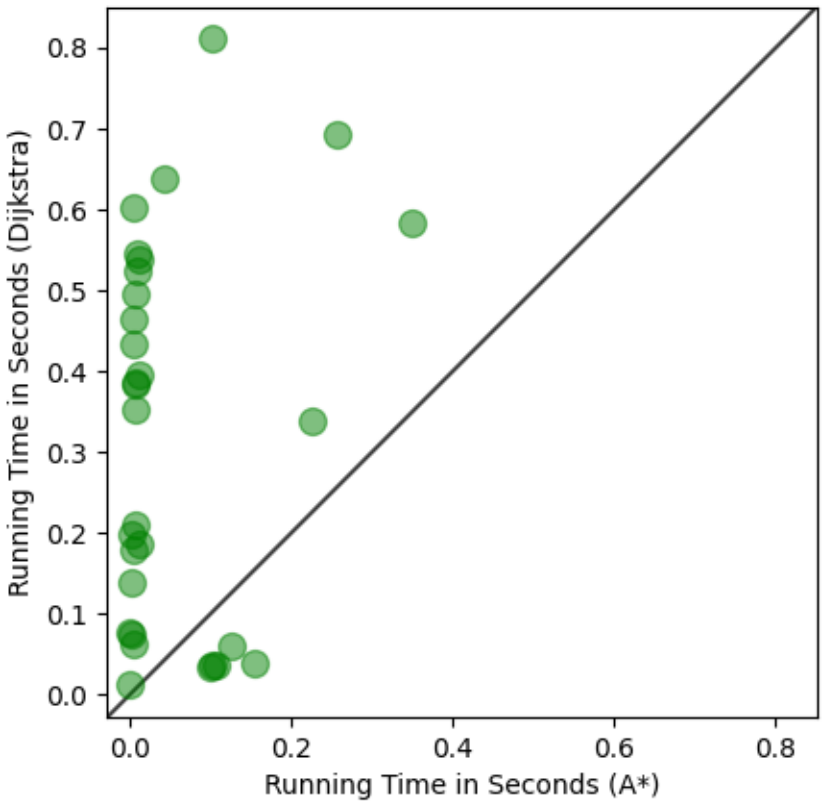

Figure 6: Runtime part 2

In figure 4, we saw a significant reduction in the number of nodes expanded for A*. Therefore, in figure 6, we should expect the overall runtime for A* to be better then what we saw in figure 3. Indeed, figure 6 shows an on average significant reduction in runtime. The plot points are heavily concentrated on the left side, indicating very short runtimes. Again, this is because we are putting more trust in the heuristic function to find the shortest path.

Overall, we have learned that for the right maps, a weighted heuristic function can help us find a solution in a faster amount of time. Although the solution isn't optimal, there are situations in the real works where sometimes we don't need the optimal solution, we just want to find a solution as quick as possible.