

Devon Reing

Problem 1 Purpose: Compare the difference between the A* path results with the Greedy Best-First Search Path results.

A* Results:



A* Maze									
									g=inf h=0
g=1 h=17	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=2 h=16	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=3 h=15	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=4 h=14	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=5 h=13	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=6 h=12	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=7 h=11	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=8 h=10									g=inf h=0
g=9 h=9	g=10 h=8	g=11 h=7	g=12 h=6	g=13 h=5	g=14 h=4	g=15 h=3	g=16 h=2	g=17 h=1	g=18 h=0

A* Initial Path Finding Code:

```
91 #####
92 1 usage
93 def find_path(self):
94     open_set = PriorityQueue()
95
96     ##### Add the start state to the queue
97     open_set.put((0, self.agent_pos))
98
99     ##### Continue exploring until the queue is exhausted
100     while not open_set.empty():
101         current_cost, current_pos = open_set.get()
102         current_cell = self.cells[current_pos[0]][current_pos[1]]
103
104         ##### Stop if goal is reached
105         if current_pos == self.goal_pos:
106             self.reconstruct_path()
107             break
108
109         ##### Agent goes E, W, N, and S, whenever possible
110         for dx, dy in [(0, 1), (0, -1), (1, 0), (-1, 0)]:
111             new_pos = (current_pos[0] + dx, current_pos[1] + dy)
112
113             if 0 <= new_pos[0] < self.rows and 0 <= new_pos[1] < self.cols and not self.cells[new_pos[0]][
114                 new_pos[1]].is_wall:
115
116                 ##### The cost of moving to a new position is 1 unit
117                 new_g = current_cell.g + 1
118
119                 if new_g < self.cells[new_pos[0]][new_pos[1]].g:
120                     ##### Update the path cost g()
121                     self.cells[new_pos[0]][new_pos[1]].g = new_g
122
123                     ##### Update the heuristic h()
124                     self.cells[new_pos[0]][new_pos[1]].h = self.heuristic(new_pos)
125
126                     ##### Update the evaluation function for the cell n: f(n) = g(n) + h(n)
127                     self.cells[new_pos[0]][new_pos[1]].f = new_g + self.cells[new_pos[0]][new_pos[1]].h
128                     self.cells[new_pos[0]][new_pos[1]].parent = current_cell
129
130                     ##### Add the new cell to the priority queue
131                     open_set.put((self.cells[new_pos[0]][new_pos[1]].f, new_pos))
```

Greedy Best-First Search Results:

										g=inf h=0
g=0 h=17	g=0 h=16	g=0 h=15	g=0 h=14	g=0 h=13	g=0 h=12	g=0 h=11	g=0 h=10			g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=0 h=9			g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=0 h=8			g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=0 h=7			g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=0 h=6			g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=0 h=5			g=inf h=0
g=0 h=11	g=0 h=10	g=0 h=9	g=0 h=8	g=0 h=7	g=0 h=6	g=0 h=5	g=0 h=4			g=inf h=0
g=0 h=10										g=inf h=0
g=0 h=9	g=0 h=8	g=0 h=7	g=0 h=6	g=0 h=5	g=0 h=4	g=0 h=3	g=0 h=2	g=0 h=1	g=0 h=0	

Greedy Best-First Search Code Changes:

```

92 def find_path(self):
93     open_set = PriorityQueue()
94
95     ##### Add the start state to the queue
96     open_set.put((0, self.agent_pos))
97
98     ##### Continue exploring until the queue is exhausted
99     while not open_set.empty():
100         current_cost, current_pos = open_set.get()
101         current_cell = self.cells[current_pos[0]][current_pos[1]]
102
103         ##### Stop if goal is reached
104         if current_pos == self.goal_pos:
105             self.reconstruct_path()
106             break
107
108         ##### Agent goes E, W, N, and S, whenever possible
109         for dx, dy in [(0, 1), (0, -1), (1, 0), (-1, 0)]:
110             new_pos = (current_pos[0] + dx, current_pos[1] + dy)
111
112             if 0 <= new_pos[0] < self.rows and 0 <= new_pos[1] < self.cols and not self.cells[new_pos[0]][
113                 new_pos[1]].is_wall:
114
115                 ##### The cost of moving to a new position is 1 unit
116                 new_g = current_cell.g + 1
117
118                 if new_g < self.cells[new_pos[0]][new_pos[1]].g:
119                     ### Update the path cost g()
120                     self.cells[new_pos[0]][new_pos[1]].g = 0
121
122                     ### Update the heuristic h()
123                     self.cells[new_pos[0]][new_pos[1]].h = self.heuristic(new_pos)
124
125                     ### Update the evaluation function for the cell n: f(n) = g(n) + h(n)
126                     self.cells[new_pos[0]][new_pos[1]].f = new_g + self.cells[new_pos[0]][new_pos[1]].h
127                     self.cells[new_pos[0]][new_pos[1]].parent = current_cell
128
129                     ##### Add the new cell to the priority queue
130                     open_set.put((self.cells[new_pos[0]][new_pos[1]].f, new_pos))
131

```

In this problem, the Greedy Best-First Search and A* algorithms produce vastly different paths in their search for the best, or in this case, shortest path from the start to goal state. In the A* algorithm, the best path avoids the obstacles found in the form of the maroon walls and opts for a path that goes straight down until it reaches the last row and then across until it hits the goal state. Contrastingly, the Greedy Best-First Search algorithm removes the consideration of the actual path cost, resulting in a path that fails to consider the obstacles. Instead, this algorithm utilizes only the heuristics which result in the path initially attempting to move horizontally to reach the goal state faster. When that fails, it must backtrack resulting in a longer path in the end due to its greedy tendencies that fail to consider the actual cost of each step. Ultimately, the difference between the two algorithms consideration of the actual path costs result in differing final best paths found with the A* algorithm proving to be the better option.

Problem 2 Purpose: Discover how the Euclidean Distance affects the results of the path compared to the Manhattan Distance heuristic. Additionally, allow diagonal moves and choose the moves at random to compare the results with the original A* path found in problem 1.

Euclidean Distance Heuristic Used:

```
#####  
#### Euclidean distance  
#####  
3 usages  
def heuristic(self, pos):  
    dist1 = (abs(pos[0] - self.goal_pos[0]))**2  
    dist2 = abs(pos[1] - self.goal_pos[1])**2  
    return round(sqrt(dist1 + dist2), 2)
```

A* Results:

										g=inf h=0
g=inf h=0	g=1 h=11.31	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=inf h=0	g=inf h=0	g=2 h=9.9	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=3 h=8.49	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=4 h=7.07	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=5 h=7.21	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=inf h=0	g=inf h=0	g=6 h=7.62	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=inf h=0	g=7 h=8.25	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=8 h=9.06										g=inf h=0
g=inf h=0	g=9 h=8.0	g=10 h=7.0	g=11 h=6.0	g=12 h=5.0	g=13 h=4.0	g=14 h=3.0	g=15 h=2.0	g=16 h=1.0	g=17 h=0.0	

In this experiment, the addition of the ability to move diagonally coupled with the heuristic moving to be for the Euclidean Distance over the Manhattan Distance results in a different path found by the A* algorithm. In this new path, the path initially begins moving towards the goal state diagonally, but once it notices it will run into an obstacle, it begins to backtrack. This results in no additional moves being needed in comparison with the original A* path found in problem one. Additionally, the new ability to move diagonally allows the algorithm to cut down on one move once it reaches the corner of the last row in its pursuit of the goal state in the shortest possible path. The combination of the heuristics and the actual path cost being considered finds a new best path with these changes, ultimately cutting down on the path's total cost in the end.

Greedy Best-First Search Results:

A* Maze									
g=inf h=0	g=0 h=11.31	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0
g=inf h=0	g=inf h=0	g=0 h=9.9	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=0 h=8.49	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=0 h=7.07	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=0 h=5.66	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=0 h=4.24	g=inf h=0	g=inf h=0	g=inf h=0
g=inf h=0	g=0 h=8.25	g=0 h=7.28	g=0 h=6.32	g=0 h=5.39	g=0 h=4.47	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0
g=0 h=9.06									g=inf h=0
g=inf h=0	g=0 h=8.0	g=0 h=7.0	g=0 h=6.0	g=0 h=5.0	g=0 h=4.0	g=0 h=3.0	g=0 h=2.0	g=0 h=1.0	g=0 h=0.0

In this experiment, the Greedy Best-First Search utilizes the Euclidean Distance heuristic while simultaneously ignoring the actual path costs in order to create a new path that utilizes diagonal moves in order to attempt to reach the goal state in as short a path as possible. Unlike the A* algorithm that undergoes similar changes to the heuristics and allowed moves, this algorithm fails to recognize the obstacles that will soon become present through the walls and continues to move diagonally towards the goal state until it is eventually forced to begin backtracking. This is due to its blind nature towards the actual cost that the path or algorithm will end up incurring, as this algorithm only considers the heuristic that seeks to go straight to the goal state with no consideration for obstacles it may face. The heuristic seeks the shortest path in a perfect world with no obstacles, but ultimately ends up coming up short in its pursuit of this goal. This ultimately adds unnecessary extra moves to the best path that is found, once again resulting in the A* algorithm to find a better path.

Problem 3 Purpose: Discover how differing weights on the actual path costs and heuristics affect the path that the A* algorithm chooses.

Example of the weighted function in code:

```
#### Agent goes E, W, N, and S, whenever possible
for dx, dy in [(0, 1), (0, -1), (1, 0), (-1, 0)]:
    new_pos = (current_pos[0] + dx, current_pos[1] + dy)

    if 0 <= new_pos[0] < self.rows and 0 <= new_pos[1] < self.cols and not self.cells[new_pos[0]][
        new_pos[1]].is_wall:

        #### The cost of moving to a new position is 1 unit
        new_g = current_cell.g + 1

        if new_g < self.cells[new_pos[0]][new_pos[1]].g:
            ### Update the path cost g()
            self.cells[new_pos[0]][new_pos[1]].g = new_g

            ### Update the heuristic h()
            self.cells[new_pos[0]][new_pos[1]].h = self.heuristic(new_pos)

            ### Update the evaluation function for the cell n: f(n) = g(n) + h(n)
            self.cells[new_pos[0]][new_pos[1]].f = 6 * new_g + 2 * self.cells[new_pos[0]][new_pos[1]].h
            self.cells[new_pos[0]][new_pos[1]].parent = current_cell

        #### Add the new cell to the priority queue
        open_set.put((self.cells[new_pos[0]][new_pos[1]].f, new_pos))
```

1.

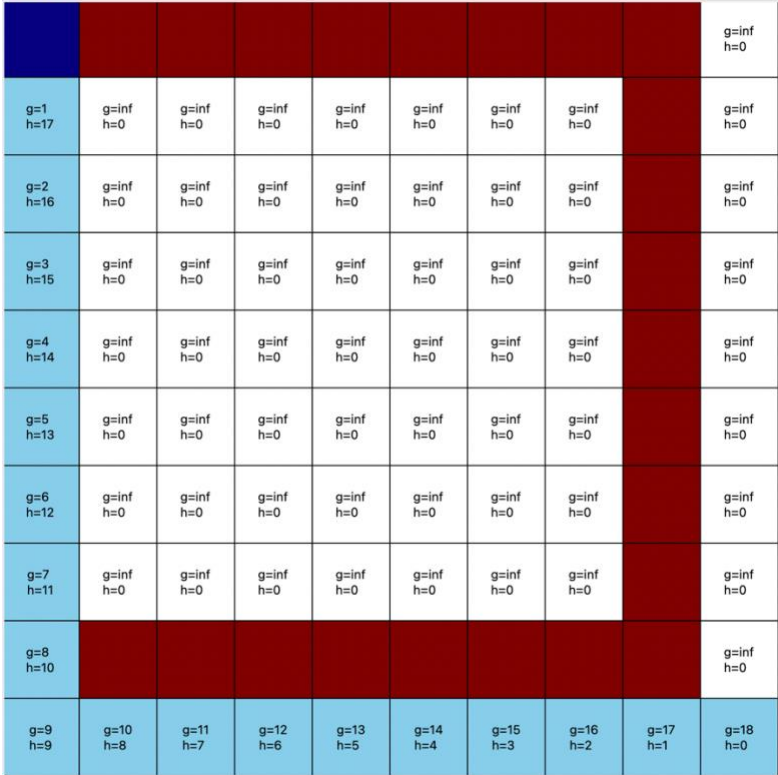
α	β	Observed Changes
2	6	

		<p>This weight begins to lean towards a Greedy Best-First path due to a higher weight being placed on the heuristic than the actual path cost. This causes the path to take a slightly longer way, but the slight weight still placed on the actual path cost keeps it from deviating from the shortest path too drastically.</p>																																																																																																				
3	5	<div><div><div>A* Maze</div><table><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>g=inf h=0</td></tr><tr><td>g=1 h=17</td><td>g=2 h=16</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td></td><td>g=inf h=0</td></tr><tr><td>g=inf h=0</td><td>g=3 h=15</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td></td><td>g=inf h=0</td></tr><tr><td>g=inf h=0</td><td>g=4 h=14</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td></td><td>g=inf h=0</td></tr><tr><td>g=inf h=0</td><td>g=5 h=13</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td></td><td>g=inf h=0</td></tr><tr><td>g=inf h=0</td><td>g=6 h=12</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td></td><td>g=inf h=0</td></tr><tr><td>g=inf h=0</td><td>g=7 h=11</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td></td><td>g=inf h=0</td></tr><tr><td>g=9 h=11</td><td>g=8 h=10</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td>g=inf h=0</td><td></td><td>g=inf h=0</td></tr><tr><td>g=10 h=10</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>g=inf h=0</td></tr><tr><td>g=11 h=9</td><td>g=12 h=8</td><td>g=13 h=7</td><td>g=14 h=6</td><td>g=15 h=5</td><td>g=16 h=4</td><td>g=17 h=3</td><td>g=18 h=2</td><td>g=19 h=1</td><td>g=20 h=0</td></tr></table></div><p>Similar to above, the higher weight being placed on the heuristic over the actual path cost leads to the path beginning on a similar path to the Greedy Best-First Search algorithm's path. However, the smaller difference between the weights in this equation leads to the path not progressing quite as far this time before beginning its return to a path similar to the path created by the A* algorithm.</p></div>										g=inf h=0	g=1 h=17	g=2 h=16	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0	g=inf h=0	g=3 h=15	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0	g=inf h=0	g=4 h=14	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0	g=inf h=0	g=5 h=13	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0	g=inf h=0	g=6 h=12	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0	g=inf h=0	g=7 h=11	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0	g=9 h=11	g=8 h=10	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0	g=10 h=10									g=inf h=0	g=11 h=9	g=12 h=8	g=13 h=7	g=14 h=6	g=15 h=5	g=16 h=4	g=17 h=3	g=18 h=2	g=19 h=1	g=20 h=0
									g=inf h=0																																																																																													
g=1 h=17	g=2 h=16	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0																																																																																													
g=inf h=0	g=3 h=15	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0																																																																																													
g=inf h=0	g=4 h=14	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0																																																																																													
g=inf h=0	g=5 h=13	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0																																																																																													
g=inf h=0	g=6 h=12	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0																																																																																													
g=inf h=0	g=7 h=11	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0																																																																																													
g=9 h=11	g=8 h=10	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0																																																																																													
g=10 h=10									g=inf h=0																																																																																													
g=11 h=9	g=12 h=8	g=13 h=7	g=14 h=6	g=15 h=5	g=16 h=4	g=17 h=3	g=18 h=2	g=19 h=1	g=20 h=0																																																																																													

4

4

A* Maze

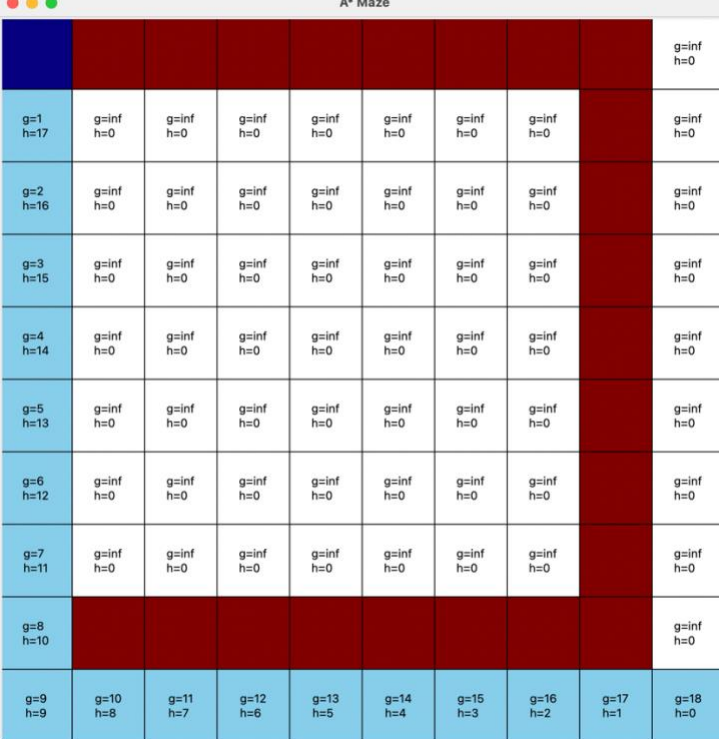


	0	1	2	3	4	5	6	7	8	9
0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0
1	g=1 h=17	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0
2	g=2 h=16	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0
3	g=3 h=15	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0
4	g=4 h=14	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0
5	g=5 h=13	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0
6	g=6 h=12	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0
7	g=7 h=11	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0
8	g=8 h=10	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0
9	g=9 h=9	g=10 h=8	g=11 h=7	g=12 h=6	g=13 h=5	g=14 h=4	g=15 h=3	g=16 h=2	g=17 h=1	g=18 h=0

The weights being equal in this equation leads to the path being the same as the A* algorithm. These weights allow for both the heuristic and actual path costs to be considered in order to find the shortest path from the start to goal state while also considering any obstacles.

5

3



A* Maze									
									g=inf h=0
g=1 h=17	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=2 h=16	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=3 h=15	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=4 h=14	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=5 h=13	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=6 h=12	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=7 h=11	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=8 h=10									g=inf h=0
g=9 h=9	g=10 h=8	g=11 h=7	g=12 h=6	g=13 h=5	g=14 h=4	g=15 h=3	g=16 h=2	g=17 h=1	g=18 h=0

The weights favoring the actual path costs in this equation results in the path following the same path created by the A* algorithm. This is most likely due to the fact that these weights allow for the algorithm to consider obstacles without being influenced by heuristics pushing the algorithm to go east first towards the goal state as it is currently set up.

6

2

A* Maze									
									g=inf h=0
g=1 h=17	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=2 h=16	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=3 h=15	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=4 h=14	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=5 h=13	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=6 h=12	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=7 h=11	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=8 h=10									g=inf h=0
g=9 h=9	g=10 h=8	g=11 h=7	g=12 h=6	g=13 h=5	g=14 h=4	g=15 h=3	g=16 h=2	g=17 h=1	g=18 h=0

The path follows the same pattern as above. Even as the distance increases between the weights for the actual path cost and the heuristic, the higher weight on the actual path cost continues to result in the same path being chosen as in the A* algorithm.

2.

<

15

A* Maze

									g=inf h=0
g=1 h=17	g=2 h=16	g=3 h=15	g=4 h=14	g=5 h=13	g=6 h=12	g=inf h=0	g=inf h=0		g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=7 h=11	g=inf h=0	g=inf h=0		g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=8 h=10	g=inf h=0	g=inf h=0		g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=9 h=9	g=inf h=0	g=inf h=0		g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=10 h=8	g=inf h=0	g=inf h=0		g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=11 h=7	g=inf h=0	g=inf h=0		g=inf h=0
g=17 h=11	g=16 h=10	g=15 h=9	g=14 h=8	g=13 h=7	g=12 h=6	g=inf h=0	g=inf h=0		g=inf h=0
g=18 h=10									g=inf h=0
g=19 h=9	g=20 h=8	g=21 h=7	g=22 h=6	g=23 h=5	g=24 h=4	g=25 h=3	g=26 h=2	g=27 h=1	g=28 h=0

20

A* Maze

									g=inf h=0
g=1 h=17	g=2 h=16	g=3 h=15	g=4 h=14	g=5 h=13	g=6 h=12	g=inf h=0	g=inf h=0		g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=7 h=11	g=inf h=0	g=inf h=0		g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=8 h=10	g=inf h=0	g=inf h=0		g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=9 h=9	g=inf h=0	g=inf h=0		g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=10 h=8	g=inf h=0	g=inf h=0		g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=11 h=7	g=inf h=0	g=inf h=0		g=inf h=0
g=17 h=11	g=16 h=10	g=15 h=9	g=14 h=8	g=13 h=7	g=12 h=6	g=inf h=0	g=inf h=0		g=inf h=0
g=18 h=10									g=inf h=0
g=19 h=9	g=20 h=8	g=21 h=7	g=22 h=6	g=23 h=5	g=24 h=4	g=25 h=3	g=26 h=2	g=27 h=1	g=28 h=0

A* Maze									
									g=inf h=0
g=1 h=17	g=2 h=16	g=3 h=15	g=4 h=14	g=5 h=13	g=6 h=12	g=inf h=0	g=inf h=0		g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=7 h=11	g=inf h=0	g=inf h=0		g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=8 h=10	g=inf h=0	g=inf h=0		g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=9 h=9	g=inf h=0	g=inf h=0		g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=10 h=8	g=inf h=0	g=inf h=0		g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=11 h=7	g=inf h=0	g=inf h=0		g=inf h=0
g=17 h=11	g=16 h=10	g=15 h=9	g=14 h=8	g=13 h=7	g=12 h=6	g=inf h=0	g=inf h=0		g=inf h=0
g=18 h=10									g=inf h=0
g=19 h=9	g=20 h=8	g=21 h=7	g=22 h=6	g=23 h=5	g=24 h=4	g=25 h=3	g=26 h=2	g=27 h=1	g=28 h=0

The increased weight on the heuristic portion of this equation results in the path chosen to start to trend towards the path generated by the Greedy Best-First Search algorithm. This makes sense as this algorithm places all its value in the heuristic portion, and this equation places a higher value in the heuristics over the actual path cost while still somewhat considering the actual path cost to reduce the effects of only considering the heuristic portion. As the weight on the heuristic portion continues to increase while the weight on the actual cost stays the same, the increased weight on the heuristic portion of the function results in the function tending towards the Greedy Best-First Search algorithm's path for longer. Since the weight was increased even further increasing the distance between the weights on the actual path cost and heuristic cost, the new path progressed even further on the greedy path before beginning to move in a manner more similar to the A* path. This move back towards the A* path eventually each time is due to the actual path cost not being eliminated entirely, and rather its weight just being diminished.