Jack Lowry, Christopher Chopping
jrl253, ctc111

# Homework 1

CS 440
February 10, 2022

## Problem A

The display function is as follows, in the file named parse.py:

def display_grid(cell_grid, path, start, goal, parent, closed)

The cell grid must be an eight neighbor array as described in the assignment, the path is a list of nodes generated by a A* Family algorithm, the start and goal are the vertexes at which A* should path between, the parent is the 2d grid of vertices that represent the parent of each node in the grid that was visited, which is generated by A* Family algorithms. Finally, closed is a the closed list generated by A* family algorithms.

## Problem B

A* :

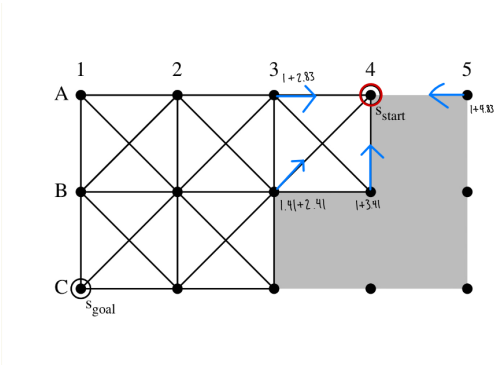


Figure 1: a

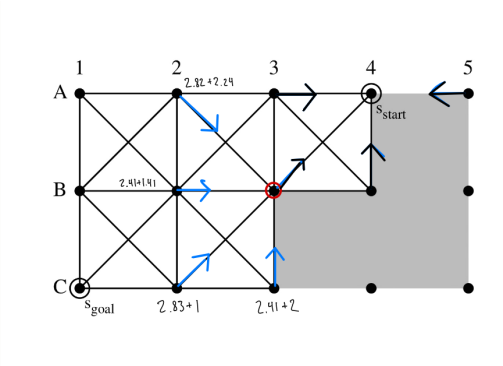


Figure 2: b

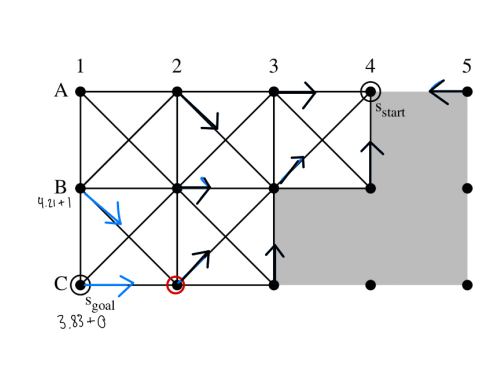


Figure 3: c

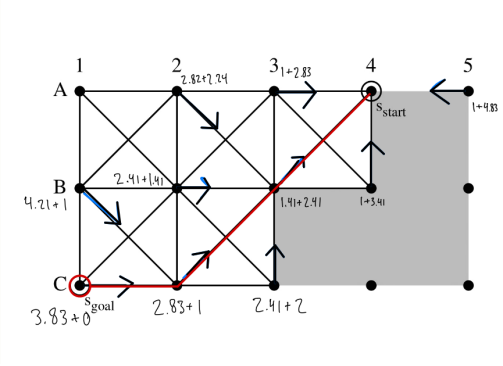


Figure 4: d

Theta* :

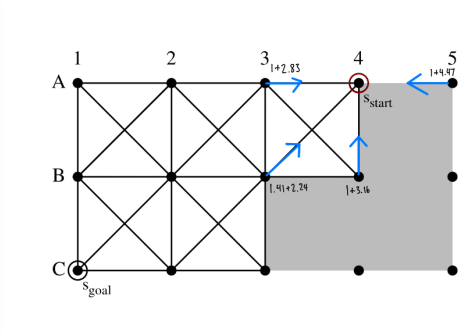


Figure 5: a

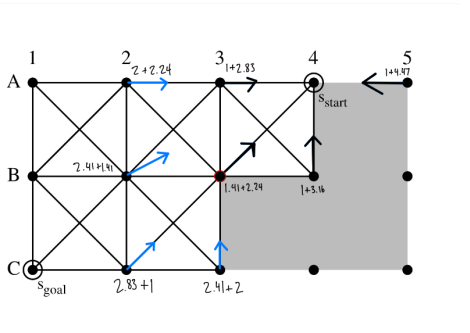


Figure 6: b

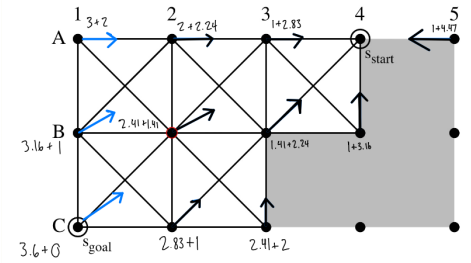


Figure 7: c

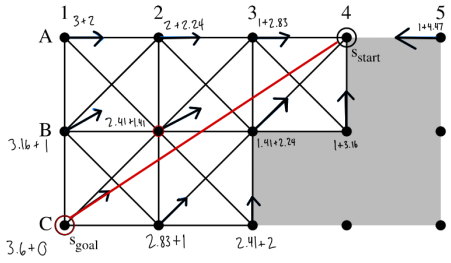


Figure 8: d

# Problem C

In order to implement A*, we used the psuedocode described in the assignment as a base. In order to begin using this, we first had to generate a list of vertices and an adjacency matrix from the list of filled and empty cells. Once this was done, we can run the typical A* algorithm starting at the goal vertex specified by the grid file. Each iteration of A*, you pop the node with the lowest f value, calculated from the cost to goal plus the heuristic value for that vertex, from the fringe (starting with the start node). Check if it's the goal (if it is, A* is done and you can return the generated path). If not, add it to the closed list and for each neighbor of the node, add it to the fringe and update the cost value of that vertex in the fringe. For A*, we updated the vertex by checking if the cost to goal from it's new found parent was better than it's old one. If it is, it's parent was updated to it's newly found parent, and it's cost was updated using the path with the newly found parent. This loop continues until either the goal is found, or all nodes are exhausted from the fringe, in which case no path was able to be found

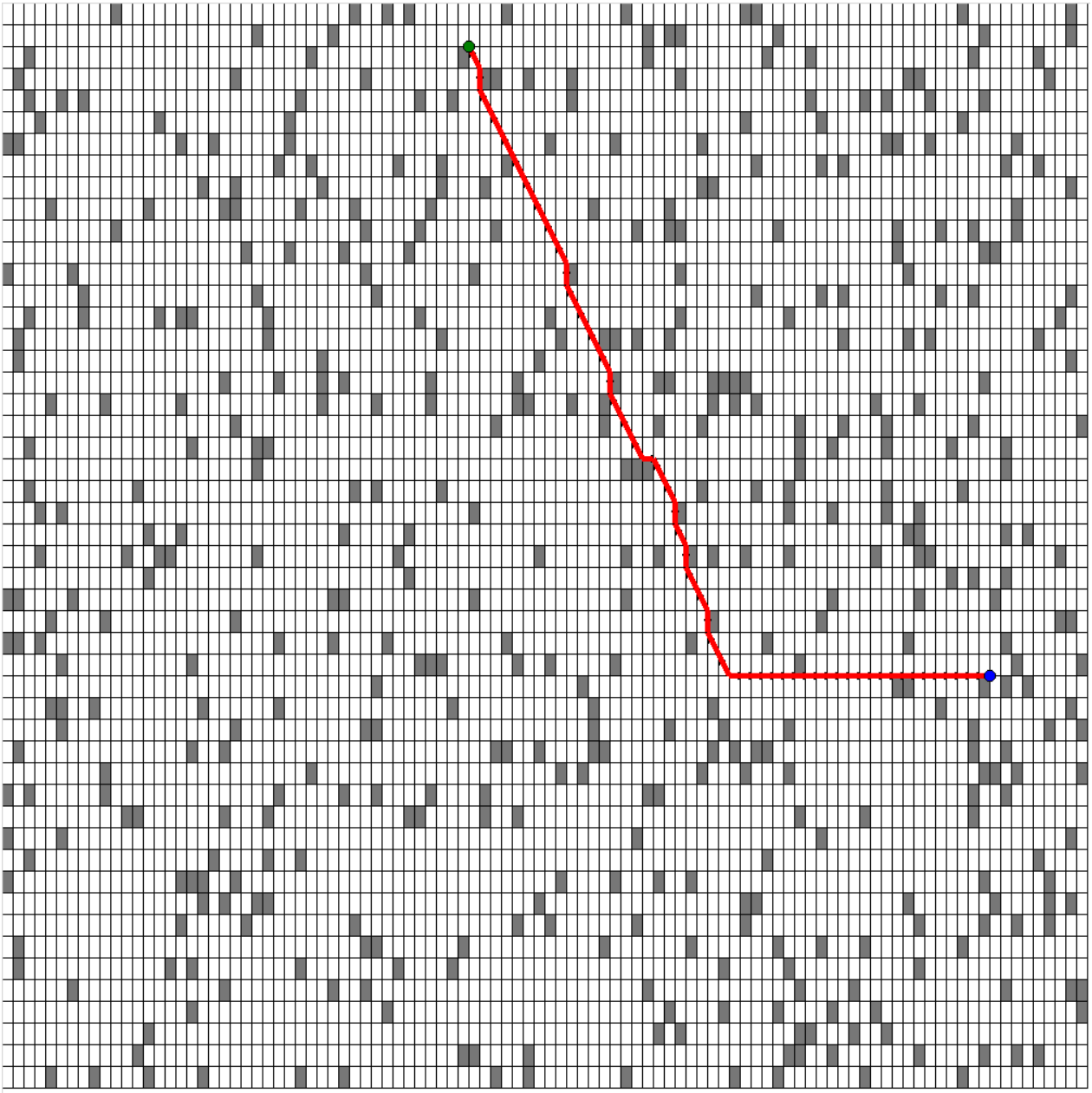

Figure 9: An example A* path

# Problem D

In order to implement A*, we used the same basic skeleton provided with the A* algorithm, with the only difference being in how we updated vertices. For Theta*, we add an extra check to see if there is a straightline path between the parent of the current vertex and the vertex being updated. This straightline path is calculated using Bresenham's line-drawing

algorithm, which iteratively checks each cell that a line drawn between the two vertexes would intersect, and returns whether or not the path composes of free cells. If the path is unblocked, then we update the vertex with the cost using that path to the parent of the current vertex, otherwise we simply use the same path that would be updated in the A* implementation. This has the effect of generating more efficient paths which wrap around obstacles more efficiently.

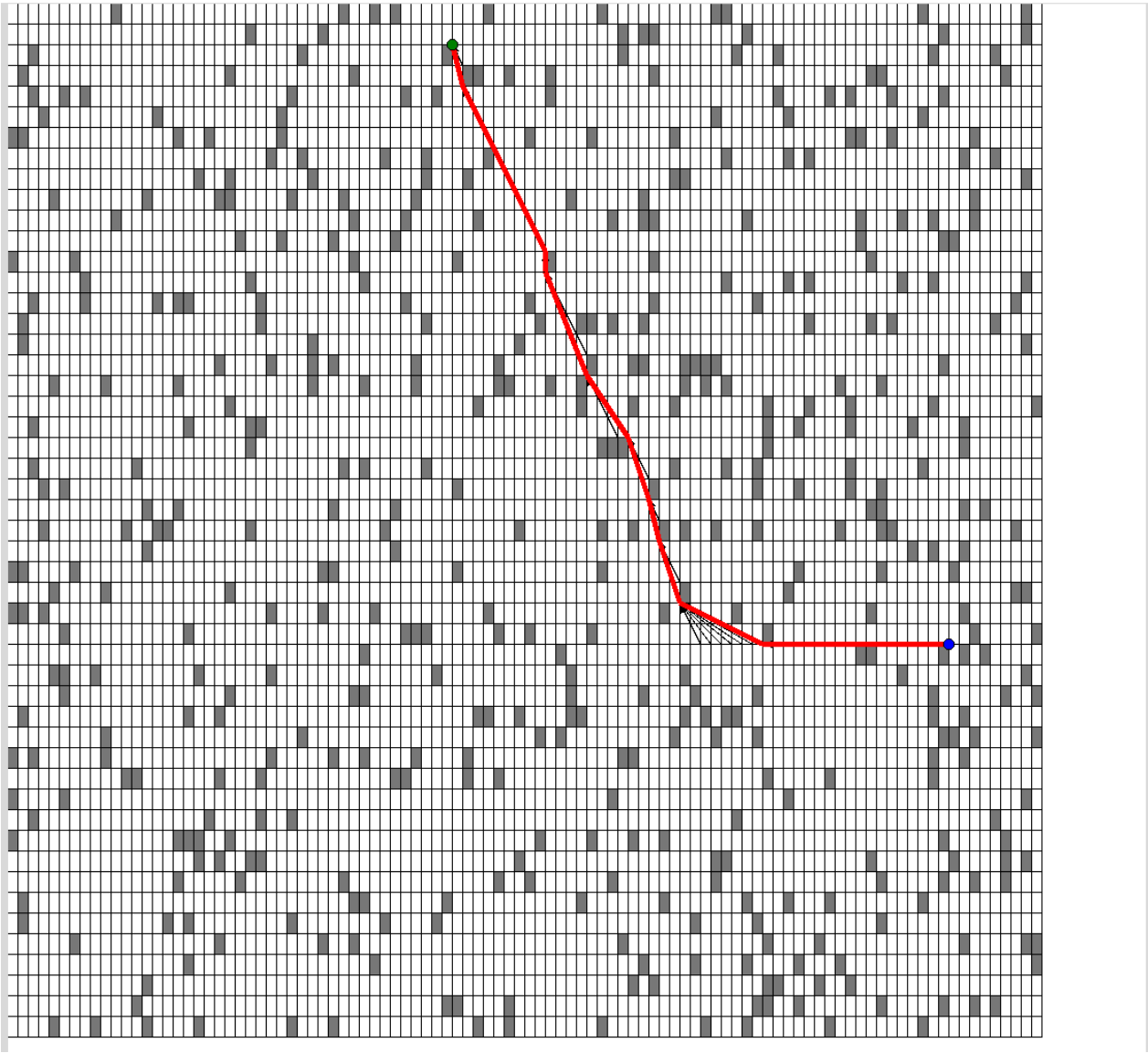

Figure 10: An example Theta* path

# Problem E

The A* algorithm will always find the shortest path in a graph given that is has an admissible heuristic function $h(n)$. A heuristic function is said to be admissible if the function never overestimates the cost of reaching the goal from it's current point. The heuristic function from Equation 1 in the assignment document is admissible because it follows this property.

Using the triangle inequality, we can prove that $h(s)$ is admissible:

$h(p(s)) \leq c(p(s), s) + h(s)$ where $p(n)$ is the parent of that node.
For any node, the heuristic represents the diagonal distance between the goal and the node, plus the remaining horizontal or vertical distance between where the diagonal reaches an axis of the the goal. Because the any movement between nodes is either on a diagonal of a cell, or the edge of a cell, the heuristic is always the quickest way to get to the goal assuming no obstacles. Thus, for any arbitrary node, it's parent node will always be less than or equal to the cost between the nodes.