COM316: Artificial Intelligence

Problem 2: Heuristic Search

Derin Gezgin — Russell Kosovsky — Jay Nash

The Problem

Intro

The search in Problem 1 lacked direction. If we had some way of determining the goodness of our possible choices, we could better decide which one to try first.

Given

1. Same as Problem 1.
2. A function (h B C), which returns the goodness of block B in regards to reaching block C. This function is called a heuristic. You may define this function in any way you like as long as it does not use any other information than the positions of the two blocks.
3. The x,y position of the goal.

Problem

Find a more efficient way to find a path from the start to the goal than the one used in Problem 1. Have it find the shortest path or discuss how this could be done.

We met to discuss this problem on September 11th

Our Solution

Our initial idea for this problem is to create a heuristic to measure the *Euclidian Distance* between the nodes and return the distance, a function we defined in Scheme 1. This is the definition of our heuristic function:

Euclidean Heuristic Function

function HEURISTIC(B, C)
  $\Delta x = B_x - C_x$
  $\Delta y = B_y - C_y$
  return $\sqrt{\Delta x^2 + \Delta y^2}$
end function

In this case, when we start the program, we'll get the adjacent nodes of the starting node using the *adjacentv* function. After this step, we can call the heuristic function on each node and add them in a **priority queue**. We must use a priority queue to prioritize the nodes closer to our goal. We'll move to the node with the least distance and call the function again. While moving the robot, we'll also save the parent-child pair, as we discussed in the solution to Problem 1. Considering we use a priority queue, we'll always have the next node to explore with the least euclidean distance to the goal.

While this seems like it might work, this is an optimal solution in a ***completely empty*** grid.

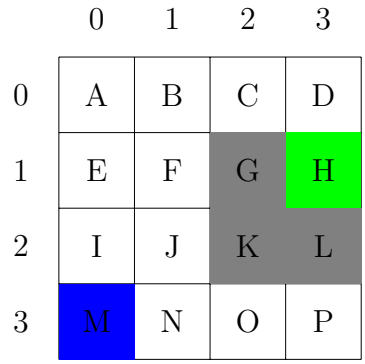
	0	1	2	3
0	A	B	C	D
1	E	F	G	H
2	I	J	K	L
3	M	N	O	P

Imagining this is the grid, our robot will start in the blue square, and the goal location is the green square. In this case, the robot will have a stair-like pattern until the green square.

The exploration of the robot would be like this:

1. Start with *Node M*.
2. Take the adjacent nodes, which would be I and N.
3. Node I has the euclidean distance from the goal node, $\sqrt{10}$ which is 3.16. On the other hand, node N has a distance of $\sqrt{8}$ which is 2.83. Our robot would choose N as it's closer to the goal.
4. At this point, the robot has two options: J and O. Both have an Euclidean distance of $\sqrt{5}$, which is 2.24. We'll assume it chooses O.
5. From the Node O our robot has two possible nodes to go: K and P. K has a distance of 1.41 and P has a distance of 2. Our robot would choose K as it's closer.
6. At this point the robot has two options: G and L both has an euclidean distance of 1. Our robot will choose either as they're equally close to the goal. We'll assume it chose G.
7. From here, our robot would directly go to the goal node H.

While our robot could find the goal easily by exploring fewer nodes than the Breadth-First search would do, if we had an obstacle at squares G, K, and L, our robot wouldn't be able to find the path as directly as this.



This is our new grid. The gray squares are obstacles and our robot can't go through them. For simplicity's sake, we won't explain what our robot will do step by step, but we'll show the squares it explores.

The path of exploration of our robot would be:

$$M \text{ (Starting node)} \rightarrow N \text{ (randomly chosen)} \rightarrow O \rightarrow P \text{ (We're stuck, go back to N)}$$
$$\rightarrow J \rightarrow F \rightarrow B \rightarrow C \rightarrow D \rightarrow H \text{ (Goal node)}$$

This is an example case for our solution that we're unable to guarantee that we can always find the shortest path in the shortest time possible. We choose O over J -nodes with the same heuristic compared to the goal- by randomness for the sake of example. Even though we moved extra in that part of the solution, this heuristic implementation still performs better than *Breadth First Search (BFS)*, which would also explore Node A.

In an extremely large grid with more obstacle combinations, this method can perform more like BFS, but it'll always perform better than BFS.

Another equivalent option for a Heuristic can be based on the *block-wise distance*. The pseudo-code for this option:

Blockwise Heuristic Function

```
function HEURISTIC(B, C)
    Δx = |B_x − C_x|
    Δy = |B_y − C_y|
    return Δx + Δy
end function
```