COM316: Artificial Intelligence
Problem 3: Real-Time Search

Derin Gezgin — Russell Kosovsky — Jay Nash

The Problem

Intro

If we were programming a robot with limited sensor range, we could not assume that it could find the best path first and then follow it. It needs to explore as it finds the best path.

Given

1. Same as Problem 2, except you can only check the status of nodes adjacent to the robot.
2. Now we need to keep track of the number of steps taken. The robot starts at the start spot and each move to a new frontier block costs the number of steps it takes.

Problem

Find an efficient way to get from the start to the goal. We can no longer expect to find the best way, but need to find a good way while minimizing our steps taken.

We met to discuss this problem on September 18th

Our Solution

We think the solution would be the mix of *Depth-First Search* and *Branch and Bound / A* Search.*

Considering our robot is limited to the nodes around it, it should select the nodes based on their heuristics. When it starts to search, it'll go through all the neighboring nodes. It'll choose the next node with the best heuristic and also in an *unvisited* state. When it moves to a new node, it'll mark the node as visited (for future reference) and repeat this process until it reaches the goal.

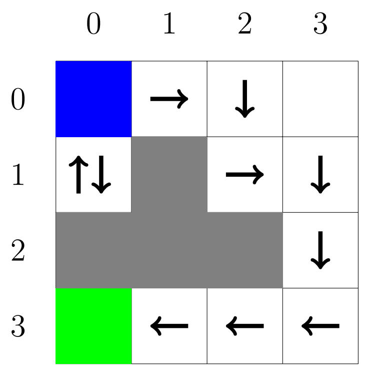
The problem is, when executing this approach, our node might get stuck in a dead end, as we initially commanded it to avoid the visited nodes. To prevent this flaw in our implementation, we can keep track of our current path in a *stack*. In a dead-end situation, we can *pop* from the stack until our current node has an unexplored neighbor. After this, we can continue from that node and apply the same procedure. While the text explanation might be enough, this is the step-by-step explanation of the implementation.

1. Start with the starting node. Push the starting node into the stack.
2. Get the adjacent nodes in a list. Among the unvisited nodes in this list, get the heuristic values and move the robot to the square with the minimum heuristic value. If multiple squares have the same heuristic value, the program will pick the node that is not adjacent to another node in the stack, otherwise randomly pick one. If there is no unvisited neighbor, move to *step 3*; otherwise, move to *step 4*.
3. We couldn't find any unvisited nodes in the neighboring nodes. This means that we're stuck in a dead end. Pop from the stack and check if the node has any unvisited neighbors. If so, go back to *step 2* and continue. If we're back in the start and don't have any neighbors to go to, this means that we explored the whole grid and couldn't find the goal:((
4. Check whether the new node we're in is the goal. If so, we found it!! If not, go back to step 2 and continue.

Pseudo-code implementation for this approach would be:

Pseudo Code 1: Real-Time Search

```
stack = (start);
start.makeVisited();
while stack is not null do
    currentNode = stack.top();
    if currentNode == goal then
        return goal;
    neighbors = getAdjacent(currentNode);
    unvisitedNeighbors = removeVisited(neighbors);
    if unvisitedNeighbors is not null then
        nextNeighbor = findNextNeighbor(getHeuristic(unvisitedNeighbors));
        moveRobot(nextNeighbor);
        stack.push(nextNeighbor);
        nextNeighbor.makeVisited();
    else
        stack.pop();
        moveRobot(stack.top());
        if stack is null then
            return "No more nodes to explore";
```



This is our grid with the heuristic value of each square. Our robot starts from the blue square and aims to go to the green square. 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.

In general, this approach allows our robot to reach its goal, but it doesn't guarantee the most optimal path. At the same time, in a very large grid, the robot can lose a lot of time in multiple dead ends and have to backtrack. As our stack removes the dead-end paths, when our robot is done with the search, it'll also have a path to the goal from the start.