CIIC 5015: Artificial Intelligence

Project 2

Prof. J. Fernando Vega Riveros
Christian D. Guzmán Torres
802-12-3240
March 26th, 2021

3.7 Consider the problem of finding the shortest path between two points on a plane that has convex polygonal objects as shown in Figure 3.31. This is an idealization of the problem that a robot has to solve to navigate a crowded environment. [1]

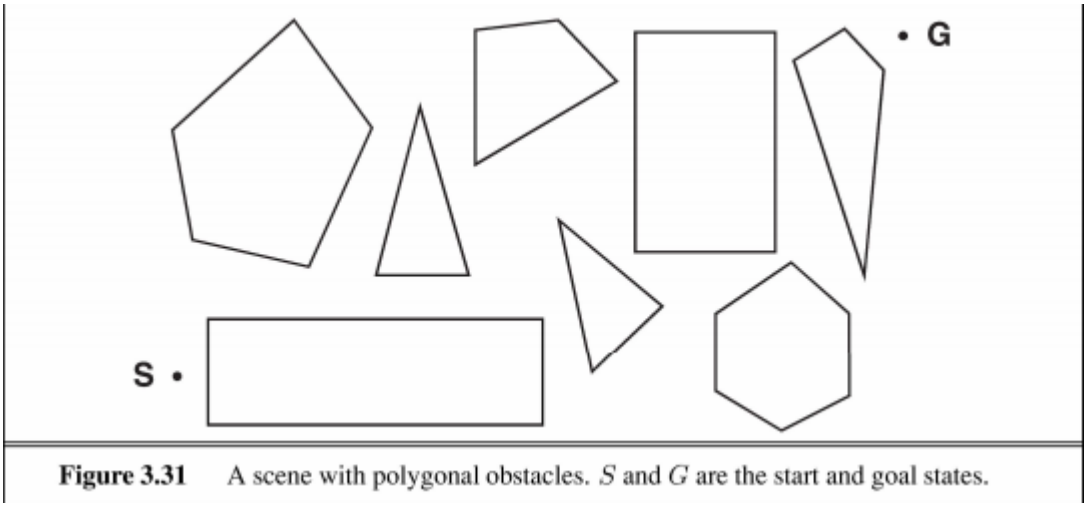


Figure 3.31 A scene with polygonal obstacles. S and G are the start and goal states.

a. Suppose the state space consists of all positions (x, y) in the plane. How many states are there? How many paths are there to the goal?

If we consider all points in the plane (including the ones occupied by polygonal objects) the number of states would be the area of the plane ($width \times height$). If we exclude the points occupied by the obstacles, then the number of states is reduced by sum of the area each obstacle occupies:

$$\#\ of\ states = Area\ of\ the\ plane - \sum_{all\ obstacles} Area\ of\ obstacle$$

This state space is relatively large and can include many paths to the goal. The total paths t consist of all possible combinations of positions (x, y) in the state space such that, the initial position is S, and the final position is G.

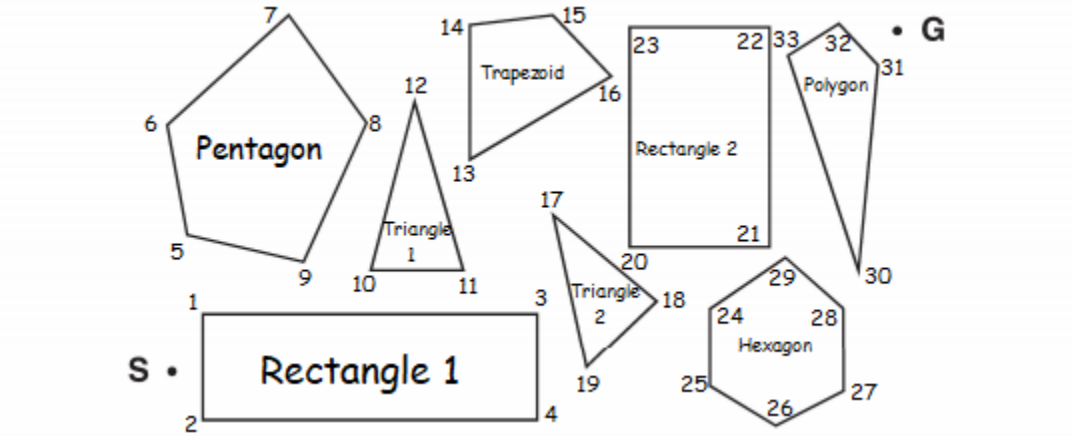
b. Explain briefly why the shortest path from one polygon vertex to any other in the scene must consist of straight-line segments joining some of the vertices of the polygons. Define a good state space now. How large is this state space?

A straight-line joining the vertices gives the straightest path and, therefore the shortest. This seems reasonable, since there are more points between any other path chosen. There are resources online that prove why this is true [1] and the concept is used as a heuristic to solve the Romania problem in the text [2].

Therefore, a good state space could consist of points (x, y) adjacent to the vertices of the polygon obstacles and the points S and G. This considerably reduces the number of states in the state space and simplifies the search problem. The number of states is equal to the number polygon vertices in the plane plus the two states for the points S and G. In figure 3.31 there are 33 vertices. Thus, the number of states is 35.

c. Define the necessary functions to implement the search problem, including an $ACTIONS$ function that takes a vertex as input and returns a set of vectors, each of which maps the current vertex to one of the vertices that can be reached in a straight line. (Do not forget the neighbors on the same polygon.) Use the straight-line distance for the heuristic function.

The implementation of this part is included in the 'exercise3_7.py' file handed in with the submission. The following labeled and numbered version of Figure 3.31 was used to code the implementation:



Each number represents a point adjacent to a vertex in the state space. A dictionary maps each identifier to the (x, y) pixel coordinates (obtained using paint). It is important to note that in paint the origin is at the top left and, x values increase to the right while, y values increase downward.

d. Apply one or more of the algorithms in this chapter to solve a range of problems in the domain, and comment on their performance.

In ‘exercise3_7.py’ the A* and Uniform Cost search algorithms are used to find the optimal solution. Additionally, Greedy best-first search is used to find an alternative, but not optimal, solution.

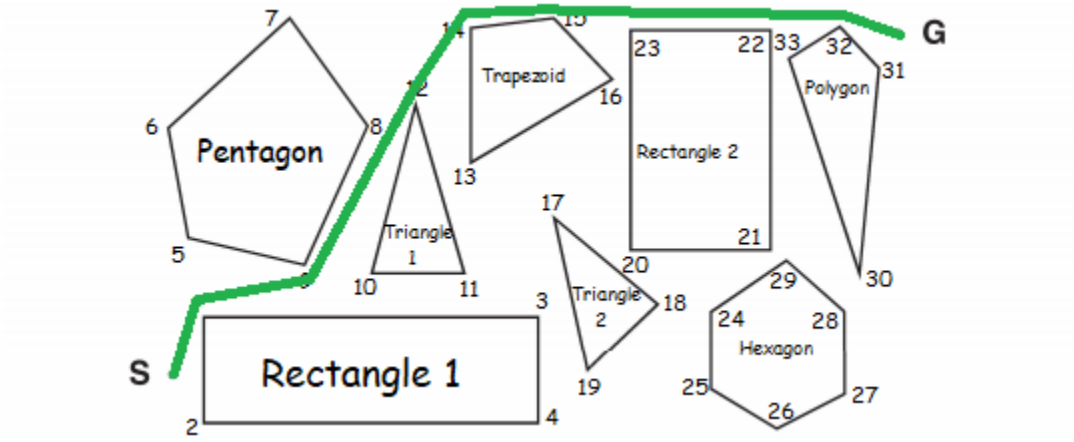
The code outputs the following:

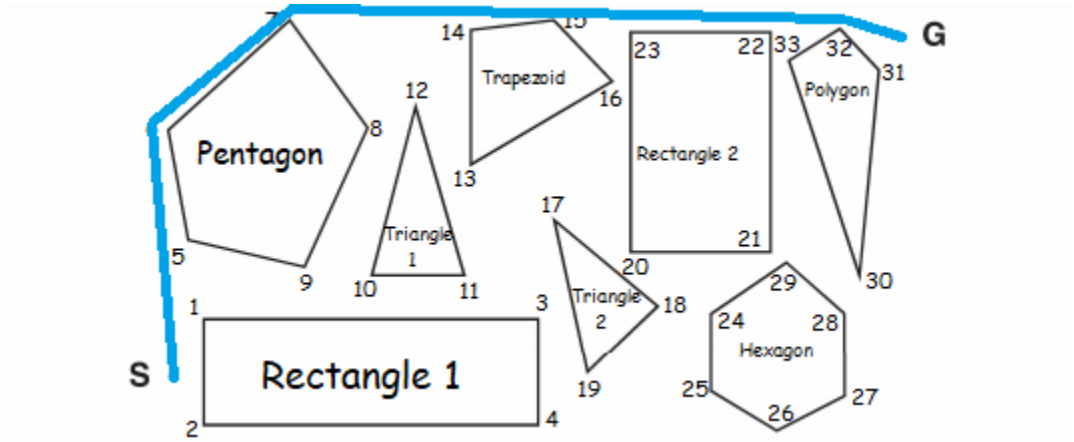
```
Using Informed Search Strategies


Performing A* search algorithm
18 paths have been expanded and 11 paths remain in the frontier
The resulting path of the search was: S 1 9 12 14 15 32 G
With a path cost of: 458.37613259984647 pixels traveled from the 536x218 pixels plane


Performing Greedy Best-First Search algorithm
5 paths have been expanded and 11 paths remain in the frontier
The resulting path of the search was: S 6 7 15 32 G
With a path cost of: 516.451441212899 pixels traveled from the 536x218 pixels plane


Using an Uninformed Search Strategy
Performing Uniform Cost Search algorithm
33 paths have been expanded and 1 paths remain in the frontier
The resulting path of the search was: S 1 9 12 14 15 32 G
With a path cost of: 458.37613259984647 pixels traveled from the 536x218 pixels plane


Process finished with exit code 0
```

The results show that both A* and Uniform Cost strategies find the optimal solution. They differ in the number of nodes they expand. A* only had to expand 18 nodes to find the optimal solution while, Uniform Cost expanded all nodes. This illustrates how Uniform Cost is a complete strategy while A* is not.

The Greedy algorithm was able to find a solution after only expanding 5 paths and found a solution in less steps but, did not find the optimal one. The path cost of the solution it found was greater that that of the other strategies.

The following images illustrate the paths chosen. The first image shows the path chosen by both A* and Uniform Cost strategies and, the second shows the one chosen by the greed strategy.





3.9 The **missionaries and cannibals** problem is usually stated as follows. Three missionaries and three cannibals are on one side of a river, along with a boat that can hold one or two people. Find a way to get everyone to the other side without ever leaving a group of missionaries in one place outnumbered by cannibals in that place. This problem is famous in AI because it was the subject of the first paper that approached problem formulation from an analytical viewpoint. [1]

a. Formulate the problem precisely, making only those distinctions necessary to ensure a valid solution. Draw a diagram of the complete state space.

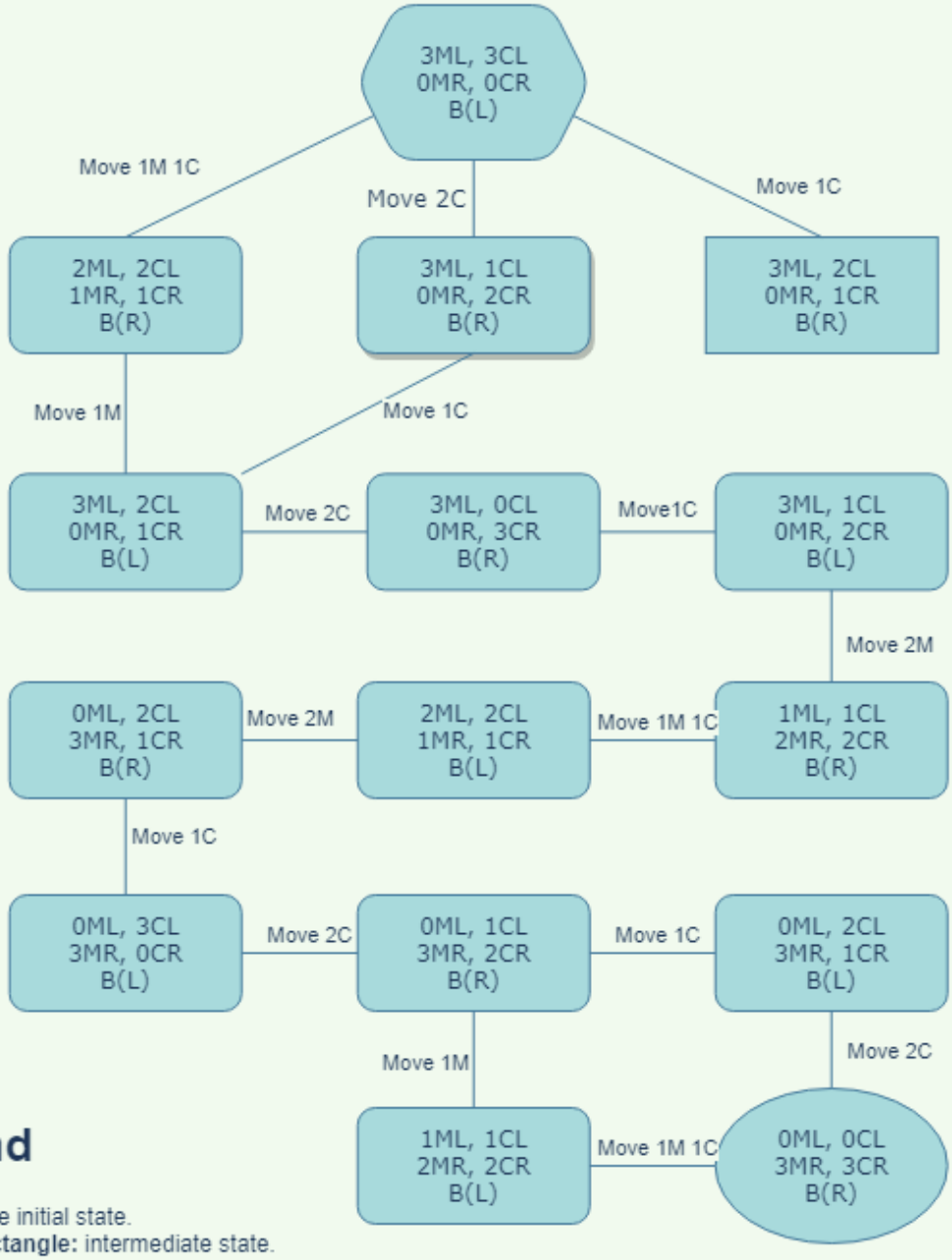
We formulate the above problem precisely by stating the following:

- In the *initial state* of the problem, the three missionaries, three cannibals, and boat are on the left side of the river.
- The agent has the following *actions available*. The action is applicable if executing it does not lead to an invalid state (having cannibals outnumber missionaries on either side of the river), it is mathematically possible (can't move more entities than those available), and it does not lead to the previous state again:
    - Move 1 missionary to the other side.
    - Move 2 missionaries to the other side.
    - Move 1 cannibal to the other side.
    - Move 2 cannibals to the other side.
    - Move 1 missionary and 1 cannibal to the other side.
- The *transition model* returns a new state in which the boat and, selected missionaries and/or cannibals are in the opposite side they were in the previous state.
- The *goal test* simply checks whether in the given state all three missionaries and all three cannibals are on the right side.
- The *path cost* keeps track of the amount of times the boat switches sides (regardless of how many people are on it). Therefore, the cost of each step is 1.

A diagram for the state space is included in the next page.

The missionaries and cannibals problem

The following diagram represents the state space for the missionaries and cannibals problem. It is assumed that in the initial state all missionaries and cannibals are on the left side of the river.



Legend

hexagon: the initial state.
rounded rectangle: intermediate state.
rectangle: dead end state.
oval: goal state.

M: missionaries
C: cannibals
B: boat
R: right
L: left

Note: All invalid states--where there are more cannibals than missionaries on either side of the river--are omitted.

b. Implement and solve the problem optimally using an appropriate search algorithm. Is it a good idea to check for repeated states?

The implementation can be found in the ‘exercise3_9.py’ file submitted. A Breath-Frist search algorithm is used to find the optimal solution. The code outputs the following:

```
Performing Breadth-First search algorithm
The resulting path of the search was:
(3, 3, 0, 0, 'L')
(3, 1, 0, 2, 'R')
(3, 2, 0, 1, 'L')
(3, 0, 0, 3, 'R')
(3, 1, 0, 2, 'L')
(1, 1, 2, 2, 'R')
(2, 2, 1, 1, 'L')
(0, 2, 3, 1, 'R')
(0, 3, 3, 0, 'L')
(0, 1, 3, 2, 'R')
(1, 1, 2, 2, 'L')
(0, 0, 3, 3, 'R')
With a path cost of: 11 boat rides


Process finished with exit code 0
```

It is a good idea to check for repeated states. There are loops within the state space which means, you can end up in an infinite loop if you fail to check for repeated states. Some code that illustrates this is left commented at the end of the file.

c. Why do you think people have a hard time solving this puzzle, given that the state space is so simple?

I think that people have a hard time solving this puzzle because they don’t formulate the problem precisely. It is difficult to solve a problem you have not fully described yet. At first, I was having a hard time figuring out how to solve the puzzle and was slightly overwhelmed. I decided to follow the steps outlined in the chapter and in part (a) of this exercise, which includes formulating the problem. Once I had a clear idea of what my

initial state was, what the available options were and set conditions for the actions to be applicable, finding a solution seemed an easier task. To complete the state space diagram, I went through each of the available actions and decided if they applied to the current state or not. It was then pretty evident that in many cases, only one new action applied. When the state space diagram was completed, I could see how the path to the solution was actually pretty simple!

References

[1] Russle S., and Norvig, P., in *Artificial Intelligence: A Modern Approach*, Pearson, 2009.

[2] A. Sinha, "How do I prove the shortest distance between two points is along the straight line joining them?," Quora, 17 November 2018. [Online]. Available: https://www.quora.com/How-do-I-prove-that-the-shortest-distance-between-two-points-is-along-the-straight-line-joining-them?share=1. [Accessed 22 March 2021].