Introduction To Artificial Intelligence
Assignment 1

Rahul Shome

July 23, 2018

Please ensure compliance with Rutgers academic integrity policies [1] in course of completing this assignment. Only one member of the group has to submit the completed assignment. Only electronic submissions will be accepted. Pictures of handwritten submissions will be penalized. Please make sure that the assignment submission includes the name of *every member of your group.*
Due date: August 02, 11:59pm.

Total Available Points: 100

1 Install the Testbed

Available Points: 10

Follow the instructions in the following page to install the testbed software.

`https://bitbucket.org/rahulshome/cs_440_assignment/wiki/Home`

In order to do so, you must have a bitbucket account to be able to download the repository to your machine. You must be able to run the following command at the end of the steps and see a red agent moving over a maze environment.

```
roslaunch prx_core assignment_1.launch \
env:="$PRACSYS_PATH/prx_core/launches/test_maze.txt"
```

`$PRACSYS_PATH` refers to the location where you installed the library.

2 Environment Generation

Available Points: 10

The following command can be used to generate environments:

```
python $PRACSYS_PATH/prx_core/launches/maze.py <ROWS> <COLS> <DENSITY>
```

`maze.py` is a script that accepts as arguments

- ROWS : number of rows in the maze
- COLS : number of columns in the maze
- DENSITY : a number between 0 and 100, that represents the percentage of cells that would be blocked

Once you run the script with the mentioned command line arguments, the output would be generated at `$PRACSYS_PATH/prx_core/launches/maze` in the following format, for say a maze with 10 rows and columns.

[1] http://academicintegrity.rutgers.edu/

```
10
10
1 1 1 1 1 1 1 1 1 1
1 0 1 1 1 1 1 1 1 1
1 0 1 1 1 1 1 1 1 1
1 0 1 1 1 1 1 0 1 1
1 1 1 1 1 1 1 0 1 1
0 0 0 0 0 1 1 0 1 1
1 1 1 1 1 1 1 0 1 1
1 1 1 1 0 1 1 0 1 1
1 1 1 0 1 1 1 1 1 1
1 1 0 1 1 1 1 1 1 1
```

In the file format, the first two lines save the #rows and #columns. The remainder of the file stores a 2D array or matrix corresponding to the cells of the maze. 1's refer to empty, or free cells. 0's refer to blocked cells. You need to generate 10 environments with the following conditions

1. 10×10 with a density of 10%
2. 10×10 with a density of 20%

These would be used in the evaluations in the next part of the assignment and the output maze file would replace `test_maze.txt` in the testbed command.

Include in a report, for each condition of environment generation, screengrabs of one instance of the output of the maze when you load the environment in the testbed i.e., 2 images of generated mazes.

3 A* Search

Available Points: 40

The file `$PRACSYS_PATH/prx_core/prx/utilities/applications/application.cpp` contains a function

```
std::vector< std::pair<int, int> > plan( int initial_i,
                                         int initial_j,
                                         int goal_i,
                                         int goal_j )
```

All (i,j) coordinates on the maze refer to a row-major ordering in the 2D matrix of the maze i.e., (0,0) refers to the top-left corner, and (r,c) corresponds to the bottom right corner for a maze with r rows and c columns.

The purpose of the *plan* function would be to give a *path* which is

– sequence of coordinates (i,j)

– beginning at (initial_i, initial_j)

– ending at (goal_i, goal_j)

– with every intermediate coordinate being an empty cell

– successive coordinates correspond to only **UP, DOWN, LEFT or RIGHT** motions.

The current implementation of the function does a naive Manhattan interpolation agnostic to the blocked cells. This is invoked with a random sequence of goal coordinates from the testbed. Once a solution is obtained, the agent follows the solution sequence and invokes the search repeatedly. You need to replace the content of this function with a call to your implementation of *A* search* that can find a solution to the search problem and return the *path*. If no such solution exists, the returned *path* should be empty.

You are free to write your search as an independent piece of code in a programming language of your choice but your solution must fill in the *path* variable inside this function. If you choose to write your code in `C++` an empty class `searcher.hpp` is provided to you to populate.

The credit for this part of the assignment would depend on the accuracy of your search i.e., your solutions should avoid the blocked cells and report the shortest paths during repeated calls from the testbed application. This would be evaluated in a demonstration.

4 Heuristics

Available Points: 40

In your code, keep track of the number of the number of expanded nodes in the search, and the number of steps in the solution *path*. The two different heuristics that need to be evaluated are:

1. A Manhattan Heuristic
2. A Heuristic that always returns 0

For each kind of heuristic try the following weighting in $f = g + w \times h$.

1. $f = g + h$
2. $f = g + (10 \times h)$

The weight w changes the priority of the heuristic in the search.

In your report, record the data

- For each of the two environment conditions $(10 \times 10, 10\%)$ and $(10 \times 10, 20\%)$
- For each of the 2 heuristics
- For each of the 2 weights
- In the 10 different maze instances of the environment, over 10 different calls to the search function, sum up the data to get
  - The total number of expanded nodes
  - The total number of steps in the solution path

You should end up with 8 different measures of #expansions and #steps. Can you explain the observations? Include in your report your thoughts on the data. What is the second heuristic similar to? Is it more beneficial to expand fewer nodes? What do you lose out on if you do so?

5 Submission

Submit a report on *Sakai* with the deliverables mentioned in the questions, including *only* your search implementation(and not the entire testbed repository).

6 Demonstration

A demonstration of your code would be required for grading the search method. The TA's would schedule time-slots for your group to demonstrate executions of your search on the testbed. You might be asked about details of your implementation, and explanations of the behavior seen from your solutions.

7 Extra Credit

Available Points: 20

Discuss in your report ways in which you can reuse information between repeated, chained searches in the same environment i.e., searches typically beginning from the goal of the previous search.